Nichtmonotone, neuro-symbolische und begriffliche Wissensverarbeitung

Zusammenfassung wissenschaftlicher Arbeiten von

Dr. Pascal Hitzler

zum Zwecke der Habilitation
eingereicht bei der
Fakultät Informatik der Technischen Universität Dresden
im Dezember 2004

Vorwort

Diese Schrift dient dem Bericht über die Hauptaspekte meiner Forschertätigkeit an der TU Dresden seit der Promotion. In diesen dreieinhalb Jahren war es mein Bestreben, selbstständig aktuelle, theoretisch fundierte und anwendungsbezogene Fragestellungen zu entwickeln und zu verfolgen.

Naturgemäß war meine Forschung in dieser Zeit vor allem zu Anfang sehr explorativ. Meine Suche konzentrierte sich dabei auf das Herausarbeiten theoretischer Zusammenhänge. Ob ich einen Ansatz dann weiterverfolgte entschied sich anschließend aufgrund einer Evaluation aus angewandter Sicht.

Die in dieser Schrift dargestellten Forschungsansätze gedenke ich in Zukunft fortzuführen. Sie bilden die Keimzellen für langfristig angelegte angewandte und theoretische Untersuchungen. Ich werde in dieser Schrift ihr Potenzial ausführlich diskutieren und anhand der bereits abgeschlossenen Untersuchungen belegen.

In erster Linie sind meine Arbeiten der Wissensverarbeitung zuzuordnen. Methodisch führen sie jedoch weit über diese hinaus und bemühen unter anderem Teile der Theoretischen Informatik. Auf der Anwendungsebene zielen sie auf implementierbare und praktisch einsetzbare Systeme für verschiedene Bereiche der Künstlichen Intelligenz.

Einige der vorgelegten Arbeiten sind in Zusammenarbeit mit Koautoren, darunter einige von mir betreute Studenten, entstanden. Im Anhang habe ich dargelegt, worauf sich mein Anteil an Arbeiten mit mehreren Autoren jeweils erstreckt.

Danksagungen

Mein Dank gilt zuallererst Steffen Hölldobler. Die inhaltliche Arbeit zu meiner Habilitation entstand während der dreieinhalb Jahre, in denen ich zu seiner Arbeitsgruppe gehörte. Ich danke vor allem für den Freiraum, den er mir schuf, und in dem ich mein kreatives Forscherpotenzial meinen eigenen Interessen folgend entfalten konnte.

Meine Zeit dort wurde auch entscheidend geprägt von drei Studenten — Sebastian Bader, Markus Krötzsch, Matthias Wendt — nur einige der Früchte der vielen Diskussionen und Arbeitstreffen sind aus gemeinsamen Veröffentlichungen ersichtlich.

Ursula Hans — Uschi — mit der ich das Büro und auch einige Arbeit in der Lehre teilen durfte, trug ganz entscheidend zu meinem Wohlbefinden bei der Arbeit bei.

Es sind viele Menschen, die für meinen „Dresdner" Lebensabschnitt in privater oder beruflicher Hinsicht wichtig waren. Ich versuche mich an einer Aufzählung — und entschuldige mich schon jetzt für die Unvollständigkeit dieses Unterfangens: Franz Baader, Federico Banti, Howard Blair, Gerd Brewka, Kai Brünnler, Paola Bruscoli, Kathrin Dornbusch, Manfred Droste, Sylvia Epp, Matthias Fichtner, Bertram Fronhöfer, Bernhard Ganter, Horst Graupner, Axel Großmann, Sandra Großmann, Alessio Guglielmi, Miguel Gutierrez-Naranjo, Achim Jung, Reinhard Kahle, Uwe Kahler, Ozan Kahramanogullari, Matthias Knorr, Torsten Linss mit Poldi, Ai(mee) Liu, Carsten Lutz, Yves Martin, Judith und Reto Merges, Kristin Mitte, Rainer Osswald, Luis Pereira, Ralf Riethmüller, Kersten Schäfer, Dietlind Scharlach mit Jochen und Amelie, Sybille Schwarz, Tony and Martine Seda, Oxana Sergueeva, Olga Skvortsova, Mariana Stantcheva, Mike Stange, Charles Stewart, Rike, Friedrich und Moritz Stölzel, Hans-Peter Störr, Lutz Straßburger, Michael Thielscher, Pawel Waszkiewicz, Kerstin und Jürgen Weber, Maja von Wedelstedt, Guo-Qiang Zhang.

Seit August 2004 habe ich am AIFB, Institut für Angewandte Informatik und Formale Beschreibungsverfahren, Universität Karlsruhe, eine neue berufliche Heimat gefunden, die mich sehr bereichert. Für Diskussionen, Hilfe und Vertrauen danke ich Rudi Studer, Andreas Abecker, Andreas Eberhart, Lars Schmidt-Thieme, Steffen Staab, York Sure, Wolfgang Sperling, Sudhir Agarwal, Ernst Biesalski, Stephan Bloehdorn, Saartje Brockmans, Philipp Cimiano, Marc Ehrig, Stephan Grimm, Peter Haase, Sigfried Handschuh, Jens Hartmann, Mark Hefke, Steffen Lamparter, Guido Lindner, Daniel Oberle, Gisela Schillinger, Nenad und Ljiljana Stojanovic, Bernhard Tausch, Christoph Tempich, Zdenko (Denny) Vrandecic, Max Völkel, Johanna Völker, Susanne Winter, Valentin Zacharias.

Von Gerd Brewka, Bernhard Ganter und Steffen Hölldobler bekam ich hilfreiches Feedback zu einem ersten Entwurf dieser Schrift. Gudrun Eberhardt half meinem Deutsch auf die Sprünge. Für verbleibende inhaltliche, formelle und sprachliche Schwächen liegt jegliche Verantwortung bei mir.

Im Laufe meiner „Dresdner" Zeit erhielt ich finanzielle Unterstützung von folgenden Organisationen: Deutscher Akademischer Austauschdienst (DAAD), Boole Center for Research in Informatics Cork (BCRI), Centro de Inteligencia Artificial Lisboa (CENTRIA), Graduiertenkolleg Wissensrepräsentation Leipzig, Bonn International Graduate School (BIGS). Ein bewilligtes Stipendium der Foundation for Polish Science und des Mianowski Fund Warschau konnte ich leider aus terminlichen Gründen nicht antreten. Das International Quality Network *Rational Mobile Agents and Systems of Agents* (IQN) des Bundesministeriums für Bildung und Forschung (BMBF) und des DAAD machte Einladungen von Gastwissenschaftlern nach Dresden möglich. Dem DAAD, dem Graduiertenkolleg *Specification of discrete processes and systems of processes by operational models and logics* und dem Verein der Freunde und Förderer der TU Dresden danke ich für die finanzielle Hilfe für einige von mir betreute Studenten bei Auslandsaufenthalten und Konferenzbesuchen.

Auf meiner Stelle in Karlsruhe werde ich seit August 2004 finanziert vom Bundesministerium für Bildung und Forschung (BMBF) im SmartWeb Projekt und von der EU über das KnowledgeWeb Network of Excellence.

Ich danke meiner Mutter Renate Hitzler, ihren Geschwistern Hannelore Hauber und Rolf Hauber, meinem Schwager Friedemann Eberhardt, und meinen Schwiegereltern Gudrun und Raimund Eberhardt, für ihre uneingeschränkte Hilfe und Unterstützung in allen Aspekten des Lebens.

Und ich danke Anne. Für Hilfe und Ratschläge. Aber vor allem fürs dasein.

Inhaltsverzeichnis

Einführung

Der Künstlichen Intelligenz zugehörig befindet sich die Wissensverarbeitung traditionell im Spannungsfeld zwischen biologischer Motivation und Plausibilität einerseits und maschineller Umsetzung andererseits. Eine Abgrenzung zur einen oder anderen Seite mag für umgrenzte Forschungsprojekte adäquat sein, eine ideologische Verfestigung und Separierung verschiedener Forschungsgebiete jedoch wird weder der Entwicklung intelligenter Systeme noch der Erkenntnis um das Wesen der menschlichen Intelligenz dienen. Das Ringen um eine Integration biologischer Plausibilität mit technologischer Umsetzbarkeit gehört somit zu den zentralen Aufgaben der Wissensverarbeitung.

Wissensverarbeitung bedarf notwendigerweise der Wissensrepräsentation. Die Repräsentation von Wissen muss auf eine solche Weise geschehen, dass das Wissen einer maschinellen Verarbeitung zumindest grundsätzlich zugeführt werden kann. Die Wahl verschiedener Verarbeitungsmethoden zur selben syntaktischen Repräsentationsform mag wiederum verschiedene Interpretationen des syntaktisch repräsentierten Wissens nahelegen.

Die *Logik* ist die wohl älteste und in der Künstlichen Intelligenz verbreitetste Form der formalen Wissensrepräsentation. Durch sie wird die deklarative Beschreibung von Wissen in vom Menschen handhabbarer symbolischer Form möglich. Logische Kalküle erlauben außerdem den formalen Umgang mit Schlussfolgerungen, und damit eine Modellierung menschlicher kognitiver Fähigkeiten.

Die Logik in ihren klassischen Ausprägungen wie zum Beispiel der Prädikatenlogik erster Stufe, ist jedoch für die Modellierung von Wissen und Schlussfolgern und der maschinellen Umsetzung desselben nur bedingt direkt geeignet. Je nach Anwendungsdomäne wird eine direkte Modellierung sehr umständlich und deshalb unpraktisch und anwendungsfern, oder bleibt anderen, z.B. subymbolischen Ansätzen in der Leistung unterlegen. Seit jeher werden also in der Wissensverarbeitung Systeme entwickelt, analysisert und angewendet, die zwar symbolisch und im weitesten Sinne logikbasiert sind, aber doch den Rahmen der klassischen Logik in gewissem Maße verlassen. Andere in manchen Anwendungsdomänen erfolgreiche Systeme wiederum sind im Ansatz nicht logikbasiert, werden aber durch ein Studium ihrer logischen Aspekte auch in wissensbasierten Bereichen anwendbar.

Das Studium und die Anwendung logischer Aspekte verschiedener Ansätze zur Wissensverarbeitung ist das Thema dieser Schrift. Insbesondere werde ich Ergebnisse zu drei Arten von Wissensrepräsentation untersuchen, die in der Wissensverarbeitung eine Rolle spielen. Meine vorgestellten Arbeiten konzentrieren sich dabei auf den Vergleich und das Zusammenspiel verschiedener Methoden, auf die Herausarbeitung fundamentaler Zusammenhänge und zielen auf Umsetzungen in Anwendungsdomänen. Systematisch lassen sie sich drei Formen der Wissensverarbeitung zuordnen.

Die erste dieser Arten der Wissensverarbeitung ist die des nichtmonotonen Schließens. Entstanden um 1980 handelt es sich dabei um eine logikbasierte Methodik zur Modellierung der menschlichen Fähigkeit zur Inferenz aus unsicherer und unvollständiger Faktenlage. Eine Leitintuition dieses Fachgebietes ist die Annahme, dass der Mensch in Abwesenheit präziseren Wissens auf Grundregeln zugreift, die in den meisten Fällen zutreffend sind, zu denen es aber Ausnahmen geben kann. Solange also nicht bekannt ist, dass es sich um einen Ausnahmefall handelt, wird die Grundregel herangezogen.

Diese Form der Wissensrepräsentation bedient sich häufig der Syntax der Logikprogrammierung. Aus diesem syntaktischen Paradigma sind dann verschiedene Ansätze zur nichtmonotonen Wissensverarbeitung hervorgegangen, die miteinander konkurrieren aber auch verwandt sind. Diese Ansätze entstanden aus Anforderungen der Praxis, wo je nach Anwendungsbereich verschiedene Formalisierungen der Leitintuition angemessen erscheinen. Der erste Teil meiner Arbeit behandelt Beziehungen verschiedener solcher nichtmonotoner Semantiken für Logikprogramme, mit dem Ziel, die Eigenheiten der unterschiedlichen Formalisierungen herauszustellen und greifbar zu machen. Insbesondere entwickle ich einen Ansatz zur uniformen Beschreibung verschiedener Semantiken, der im Vergleich zu anderen solchen Methoden sehr viel flexibler ist und eine größere Klasse verschiedener Semantiken umfasst.

Die zweite Art der Wissensverarbeitung, die ich behandle, ist die der künstlichen neuronalen Netze. Diese in der Praxis der Künstlichen Intelligenz sehr erfolgreiche Methodik entstand aus der Abstraktion biologischer neuronaler Netze, wie sie im Nervensystem von Menschen und Tieren vorliegen. Seine praktischen Erfolge verdankt dieser Ansatz vor allem der Tatsache, dass künstliche neuronale Netzwerke auf effiziente Weise anhand von Beispielen trainiert werden können, komplexe Aufgaben wie z.B. Mustererkennung zu bewältigen.

In biologischen Systemen werden elektrische Potenziale durch ein gerichtetes Netzwerk propagiert und auf bisher nicht vollständig verstandene Weise massiv parallel verarbeitet. Für die künstlichen Gegenstücke werden Netzwerke und Verarbeitungsprinzipien abstrahiert und dadurch stark vereinfacht. Dennoch entzieht sich das Verständnis der immer noch komplexen Vorgänge in künstlichen Netzen unserem Verständnis im Sinne einer deklarativen oder logischen Lesbarkeit. Künstliche neuronale Netze stellen dadurch eine *subsymbolische* Art der Wissensverarbeitung dar, die von der logikbasierten stark verschieden ist. Letztere, insbesondere die auf Logikprogrammen basierende, ist deklarativ, entzieht sich aber vergleichbar erfolgreichen Lernmethoden. Ein weiterer Unterschied besteht in der *Robustheit* neuronaler Netzwerke, die — im Gegensatz zu Logikprogrammen — auch nach Ausfall eines Teils der repräsentierenden Strukturen in der Regel noch zufriedenstellend arbeiten. Anforderungen der Praxis machen eine integration logikbasierter und neuronaler Systeme wünschenswert. Systeme, die die Robustheit und Lernfähigkeit künstlicher neuronaler Netze mit der deklarativen Ausdrucksstärke logikbasierter Wissensverarbeitung kombinieren, sind von besonderem Interesse.

Im zweiten Teil dieser Schrift wende ich mich also Fragen der neurosymbolischen Integration zu. Dabei geht es um die Frage nach Möglichkeiten zur Integration logikbasierter Wissensverarbeitung mit solcher, basierend auf künstlichen neuronalen Netzen. Insbesondere widme ich mich der Frage einer Integration von Logik erster Stufe, die besondere Schwierigkeiten mit sich bringt. Die vorgestellten Ergebnisse beschreiben Repräsentationsmethoden für den semantischen Gehalt von Logikprogrammen mit Hilfe von Standardarchitekturen für künstliche neuronale Netze. Zum Teil werden dabei wiederum nichtmonotone Semantiken betrachtet. Die von meinen Koautoren und mir erarbeiteten Resultate konstituieren den aktuellen Stand der Forschung auf diesem Gebiet.

Die dritte Form der Wissensverarbeitung, der ich mich zuwende, basiert auf der abstrakten Beschreibung begrifflicher Inhalte und Ontologien, wie sie z.B. im Bereich des *Semantic Web* zum Einsatz kommen. Begriffliche Inhalte werden dabei zunächst durch Ordnungsstrukturen, genauer als Begriffshierarchien, dargestellt. Im Forschungsbereich *Semantic Web* stellt sich aber zunehmend heraus, dass zu einer für Anwendungen genügend reichhaltigen Repräsentation ontologischen Wissens Begriffshierarchien durch Regeln im Sinne der Logikprogrammierung erweiterbar sein müssen.

Im dritten Teil dieser Schrift wende ich mich daher Fragen des Zusammenspiels von Begriffshierarchien, Logikprogrammierung und nichtmonotonem Schließen zu. Genauer entwerfe ich ein generisches Paradigma zum integrierten nichtmonotonen Schließen auf Ordnungsstrukturen und zeige, dass es mit Standardansätzen aus den Bereichen nichtmonotones Schließen und Begriffsstrukturen voll verträglich ist.

Struktur dieser Schrift

Die drei Kapitel dieser Schrift entsprechen den genannten drei Formen der Wissensverarbeitung. Jedes Kapitel besteht aus fünf Teilen.

1. Eine knappen Zusammenfassung des Kapitels: Problemstellung, Lösung, Ausblick.

2. Eine Darstellung des Forschungskontextes, in dem meine Arbeiten angesiedelt sind.

3. Eine technisch gehaltene Hinführung zu einigen Grundbegriffe, die zum Verständnis der allgemeinen Diskussion meiner Resultate hilfreich sind. Es werden auch exemplarisch eigene Resultate formal beschrieben.

4. Ein allgemeiner gehaltener Überblick über eigene Resultate in den eingereichten Arbeiten.

5. Ein Ausblick auf Weiterführungen und Anwendungen aus meinen Arbeiten heraus.

Ein Leser mit Vorkenntnissen im Themenbereich eines Kapitels mag sich zunächst durch die Lektüre der Kurzfassung am Anfang des Kapitels sowie der Aufstellung der Resultate in den eingereichten Arbeiten orientieren.

Im Anhang findet sich eine Aufstellung meines Eigenanteils an den mit Koautoren publizierten Arbeiten. Die Arbeiten selbst sind beigefügt.

Kapitel 1

Semantik nichtmonotoner Logikprogrammierung

1.1 Kurzfassung

Es gibt viele verschiedene Ausprägungen nichtmonotonen Schließens, die sich in verschiedenen *Semantiken* für Logikprogramme niederschlagen. Diese modellieren verschiedene Intuitionen, die sich wiederum jeweils aus Anwendungsbeispielen motivieren.

Eine Systematisierung dieser verschiedenen Semantiken wird in jüngster Zeit von vielen Autoren versucht. Ziel dieser Untersuchungen ist zum einen ein vertieftes Verständnis verschiedener Semantiken für Fragen der Anwendung, und zum anderen die Suche nach geeigneten Semantiken für syntaktische Erweiterungen der Standardparadigmen, wie sie durch Anforderungen der Praxis vonnöten sind.

In meinen eigenen Arbeiten wird ein konzeptionell neuer Ansatz zur Systematisierung und vereinheitlichten Beschreibung verschiedener Semantiken — mit Hilfe von *Stufenfunktionen* — ausgeführt. Im Vergleich zu anderen Ansätzen können damit sehr viel mehr verschiedenartige Semantiken systematisch charakterisiert werden. Die Arbeiten zeigen außerdem, dass der Ansatz auf praxisrelevante syntaktische Erweiterungen übertragbar ist.

Zukünftige Arbeiten umfassen Methoden zum Herausarbeiten von praxisrelevanten Eigenschaften von Semantiken wie Berechenbarkeit und Komplexität, die Behandlung syntaktischer Erweiterungen für anwendungsnahe Modellierungen, sowie die Ableitung von Algorithmen zur automatischen Deduktion unter diesen Semantiken.

1.2 Forschungskontext: Logikprogrammierung und nichtmonotones Schließen

Logikprogrammierung ist — Programmieren mit Logik. Kurz gesagt beruht sie auf der Idee, Logik als Wissensrepräsentationssprache zu verwenden, um damit ein vorliegendes Problem zu spezifizieren und Lösungen mit Hilfe maschineller Deduktion herzuleiten. Die bahnbrechenden Arbeiten von Robert Kowalski [80], die wiederum auf der von Alan Robinson [106] entwickelten Resolutionsmethode als Grundlage für das Forschungsgebiet der *Deduktion* aufbauen, können als die Ursprünge der Logikprogrammierung angesehen werden. Kowalski entwickelte die SLD-Resolution als Verfeinerung der Resolutionsmethode, die es einem erlaubt, logische Formeln erster Stufe direkt als Programme für maschinelle Verarbeitung zu verwenden. Daraus entwickelte sich dann die Programmiersprache *Prolog*, die von Alain Colmerauer als erstem realisiert wurde [24].

Mittlerweile wurde die Logikprogrammierung zu einem der Standardprogrammierparadigmen und wurde in eine Vielzahl verschiedener Richtungen weiterentwickelt und zur Anwendung geführt. Als einige wenige Beispiele seien Natural Language Processing, (Deduktive) Datenbanken, maschinelle Deduktion, Wissensverarbeitung, Kognitive Robotik, Semantic Web, Rechtswesen und maschinelles Lernen genannt. Die Zahl der industriellen Anwendungen der zugrundeliegenden Techniken — vor allem Prolog, aber auch Constraint und Induktive Logikprogrammierung — wächst ständig, und es kann erwartet werden, dass der Trend sich fortsetzt. In [7] findet man einen hervorragenden Überblick über einige der wichtigsten Themen, die augenblicklich in der Logikprogrammierung behandelt werden. Standardreferenzen für die Prolog zugrundeliegende Theorie sind [85, 5].

Das Forschungsgebiet *Nichtmonotones Schließen* entstand aus der Idee, Aspekte des menschlichen Schließens zu formalisieren, die sich mit Hilfe klassischer logischer Methoden nur schwer ausdrücken lassen. Vor allem ging es um eine Formalisierung der Tatsache, dass Menschen in vielen Fällen dazu neigen, zwar logisch unscharfe, aber der Situation dennoch in praktischen Belangen höchst angemessene Schlüsse zu ziehen, eine wünschenswerte Eigenschaft, über die intelligente Systeme zur Zeit nur sehr beschränkt verfügen. Formaler betrachtet neigen Menschen dazu, aus einer gegebenen Wissensbasis mehr Schlüsse zu ziehen, als es mit den Mitteln der Prädikatenlogik erster Stufe eigentlich möglich wäre, was aber wiederum zur Folge hat, dass solche Schlussfolgerungen bei Bekanntwerden weiteren Wissens eventuell wieder zurückgenommen werden müssen. Im Gegensatz dazu sind klassische Logiken wie die Aussagen- oder die Prädikatenlogik *monoton* in dem Sinne, dass aus dem Schluss einer Formel F aus einer Theorie Γ folgt, dass F auch aus jeder Theorie folgt, die Γ enthält.

Es stellte sich jedoch heraus, dass sich diese nichtmonotonen Aspekte des menschlichen Schließens nur schwer auf befriedigende Weise formalisieren lassen. Frühe Arbeiten auf diesem Gebiet basierten im Wesentlichen auf drei grundsätzlich verschiedenen Ansätzen, zu denen z.B. in [38] eine exzellente Diskussion vorliegt: *Circumscription* von John McCarthy [91, 92], basierend auf Logik zweiter Stufe, *Autoepistemische Logik* von Robert Moore [94, 95] mit modallogischen Operatoren für geglaubte aber nicht notwendig gewusste Annahmen und die *Defaultlogik* von Ray Reiter [105], die auf der Idee basiert, dass manche Schlüsse immer dann (*by default*) gezogen werden sollten, wenn kein explizites Wissen gegen diese Schlüsse spricht, d.h. keine bekannte Ausnahme zum Schluss vorliegt.

In der zweiten Hälfte der 1980er Jahre erhielt die Forschung zum nichtmonotonen Schließen entscheidenden Aufwind durch Versuche, diese Methoden zur semantischen Analyse von Prolog und verwandten Logikprogrammierparadigmen einzusetzen. Tatsächlich enthält Prolog schon immer eine nichtmonotone Funktionalität: Wenn das System zeigen kann, dass ein Faktum A *nicht* aus einer gegebenen Wissensbasis — oder einem Programm — folgt, dann wird A als *falsch* angesehen, d.h. $\neg A$ ist eine Schlussfolgerung, die von Prolog gezogen wird. Nach Erweiterung der Wissensbasis um das Faktum A kann jedoch A abgeleitet werden, was die Rücknahme des vorhergehenden Schlusses $\neg A$ notwendig macht. Es sei bemerkt, dass die in $\neg A$ auftretende Negation nicht im Sinne z.B. der Prädikatenlogik erster Sufe interpretiert werden sollte. Vielmehr steht $\neg A$ für *Negation als Fehlschlag* (des Beweises von A).

Aus diesen und ähnlichen Gründen ist es nicht wirklich klar, ob *Negation als Fehlschlag* eine Negation in einem vernünftigen logischen Sinne ist oder vielmehr als eine extralogische Funktionalität von Prologsystemen aufgefasst werden sollte. Untersuchungen dieser Frage haben jedoch viele Ideen und Fragestellungen hervorgebracht, die die Forschung im Bereich Logikprogrammierung und Wissensverarbeitung nach wie vor inspirieren. Das obengenannte nichtmonotone Verhalten der *Negation als Fehlschlag* zum Beispiel veranlasste Untersuchungen, ob es nicht mit Hilfe etablierter nichtmonotoner Schlussparadigmen einer logischen Interpretation zugeführt werden könne. Diese Studien brachten zwar nur Teilerfolge hervor insofern *Negation als Fehlschlag* betroffen war. Doch sie führen auch zu der Beobachtung, dass Logikprogramme — möglicherweise ausgestattet mit zusätzlichen syntaktischen Erweiterungen — eine hervorragende Sprache für die Wissensverarbeitung unter Nichtmonotonie liefern.

Um 1990 verschob sich darum der Fokus in Richtung der Erforschung von Logikprogrammen als Wissensrepräsentationssprache für nichtmonotones Schließen, unter Inkorporation der allgemeineren Resultate und Einsichten, die bis zu diesem Zeitpunkt erzielt worden waren. Die entstehenden Paradigmen waren viel einfacher zu verstehen und aus maschineller Sicht handzuhaben und führten schließlich zu einer Reihe von Systemimplementierungen, bekannt als *Answer Set Programming* Systeme, deren Wichtigste zur Zeit DLV und SMODELS sind [30, 89, 113]. Diese Paradigmen und Systeme werden zur Zeit ausgebaut und verfeinert. Außerdem laufen ausgiebige Untersuchungen zur Anwendbarkeit in verschiedenen Bereichen der Künstlichen Intelligenz.

Answer Set Programming beruht dabei auf einer bestimmten semantischen Interpretation von Logikprogrammen, nämlich der durch *stabile Modelle* [45, 46] gegebenen. Diese ist wiederum nur eine der semantischen Lesarten, der Logikprogramme zugeführt werden können. Das Studium von Anwendungsbeispielen legt verschiedene Semantiken nahe, die auch tatsächlich vorgeschlagen und untersucht wurden. Die Semantik der stabilen Modelle hat sich Ende der 90er Jahre gegen eine Reihe von Konkurrenten als praktikabelste Lösung weitgehend durchgesetzt. Die verschiedenen vorgeschlagenen Semantiken sind jedoch zum großen Teil eng miteinander verwandt, und manche davon werden nach wie vor zur theoretischen Analyse oder zur Ableitung von Algorithmen herangezogen. Für syntaktische Erweiterungen von Answer Set Programming ist die Diskussion um geeignete semantische Lesarten weiterhin im Gange.

Eine Systematisierung dieser verschiedenen Semantiken wird in jüngster Zeit von vielen Autoren versucht, darunter [27, 28, 26, 36, 88, 86, 108]. Ziel dieser Untersuchungen ist zum einen ein vertieftes Verständnis verschiedener Semantiken für Fragen der Anwendung, und zum anderen die Suche nach geeigneten Semantiken für syntaktische Erweiterungen der Standardparadigmen, wie sie durch Anforderungen der Praxis vonnöten sind.

In meinen eigenen Arbeiten wird ein konzeptionell neuer Ansatz zur Systematisierung und vereinheitlichten Beschreibung verschiedener Semantiken — mit Hilfe von *Stufenfunktionen* — ausgeführt. Im Vergleich zu anderen Ansätzen können damit sehr viel mehr verschiedenartige Semantiken systematisch charakterisiert werden. Die Arbeiten zeigen außerdem, dass der Ansatz auf praxisrelevante syntaktische Erweiterungen übertragbar ist.

1.3 Technische Hinführung: Syntax und Semantik von Logikprogrammen

Zur vereinheitlichten Behandlung verschiedener Semantiken in der Logikprogrammierung bedienen wir uns des Hilfsmittels der *Stufenfunktionen* (engl. *Level Mappings*). Eine Stufenfunktion ist dabei eine Abbildung von Grundinstanzen von Atomen in eine wohlgeordnete Menge und dient informell der Beschreibung von rekursiven Abhängigkeiten innerhalb von Logikprogrammen. Wir führen zunächst Terminologie und Notation ein. Die meisten der folgenden Begrifflichkeiten werden auch in den restlichen Kapiteln der Schrift zur Verwendung kommen. Wir arbeiten im Folgenden über einer gegebenen Sprache der Prädikatenlogik erster Stufe und folgen im Wesentlichen [85].

Eine *Regel* ist eine Formel der Form

$$\forall X_1 \ldots \forall X_k (A \leftarrow A_1 \wedge \cdots \wedge A_m \wedge \neg B_1 \wedge \cdots \wedge \neg B_n)$$

der Prädikatenlogik erster Stufe, wobei $X_1, \ldots, X_n$ genau die in der Formel vorkommenden Variablen sind. Die Reihenfolge der Atome auf der rechten Seite des Implikationssymbols ist dabei unbedeutend. Eine solche Regel schreiben wir abgekürzt als

$$A \leftarrow A_1, \ldots, A_m, \neg B_1, \ldots, \neg B_n.$$

A wird als *Kopf* der Regel bezeichnet, $A_1, \ldots, A_m, \neg B_1, \ldots, \neg B_n$ als *Rumpf* der Regel. Eine Regel heißt ein *Faktum*, wenn $m = n = 0$ ist. Eine Regel heißt *definit*, wenn $n = 0$ ist. Ein (*normales*) *Logikprogramm* P besteht aus einer endlichen Menge von Regeln. Ein Logikprogramm heißt *definit*, wenn es nur aus definiten Regeln besteht. Wir gehen stets (ohne Beschränkung der Allgemeinheit) davon aus, dass die verwendete Sprache erster Stufe genau die in einem gegebenen Logikprogramm P vorkommenden Konstanten-, Funktions- und Prädikatensymbole beinhaltet. Die zugehörige *Herbrandbasis* B_P bezeichnet die Menge aller Grundatome über der gegebenen Sprache. Die Menge aller zu P gehörigen Grundinstanzen von Regeln bezeichnen wir mit $\mathrm{ground}(P)$.

Eine *Stufenfunktion* $l : B_P \to \alpha$ für ein Programm P ist eine Abbildung l von der Herbrandbasis B_P in eine (abzählbare) Ordinalzahl α.

Herbrandinterpretationen für ein Logikprogramm P können kanonisch mit Teilmengen von B_P identifiziert werden. Die Menge all dieser Teilmengen wird mit I_P bezeichnet. Die Menge I_P ist durch Teilmengeninklusion geordnet. Ist $I \subseteq B_P$, dann definieren wir $\neg I = \{\neg A \mid A \in I\}$. Der *Konsequenzoperator* T_P ist eine Abbildung auf I_P, wobei $T_P(I)$ für $I \in I_P$ die

Menge der $A \in B_P$ ist, für die eine Regel $A \leftarrow \texttt{body}$ in ground(P) existiert, für die `body` wahr unter I ist.

Wir betrachten auch dreiwertige Logiken, in denen neben *wahr* und *falsch* noch der Wahrheitswert *unbekannt* zur Verfügung steht. Negation, Konjunktion und Disjunktion werden auf naheliegende Weise (wie in Kleenes starker dreiwertiger Logik [32]) interpretiert: *nicht wahr* ergibt *falsch*, *nicht falsch* ergibt *wahr*, und *nicht unbekannt* ergibt *unbekannt*. Eine Konjunktion ist wahr genau dann, wenn alle ihre Komponenten wahr sind. Sie ist falsch genau dann, wenn mindestens eine ihrer Komponenten falsch ist. Eine Disjunktion ist wahr genau dann, wenn mindestens eine ihrer Komponenten wahr ist. Sie ist falsch genau dann, wenn alle ihre Komponenten falsch sind. Eine Implikation $p \rightarrow q$ ist wahr genau dann, wenn aus der Wahrheit von q die Wahrheit von p folgt. Letzteres genügt für unsere Darstellung, das sonstige Verhalten der Implikation ist nicht relevant.

Die (Herbrand-)Interpretationen in dreiwertiger Logik können mit den konsistenten Teilmengen von $B_P \cup \neg B_P$ identifiziert werden. Eine Teilmenge I von $B_P \cup \neg B_P$ heißt dabei *konsistent*, wenn für jedes $A \in B_P$ höchstens eines von A und $\neg A$ in I ist. Die Menge aller dreiwertigen Interpretationen bezeichnen wir mit $I_{P,3}$, sie ist durch Teilmengeninklusion geordnet.

Ein (zweiwertiges bzw. dreiwertiges) *Modell* für ein Logikprogramm P ist eine (zweiwertige bzw. dreiwertige) (Herbrand-)Interpretation, die alle Regeln in ground(P) (bezüglich zweiwertiger bzw. dreiwertiger Logik) wahr macht. Wird nicht explizit darauf verwiesen, dass eine Interpretation oder ein Modell dreiwertig ist, so ist ein klassisches zweiwertiges gemeint.

Semantiken für Logikprogramme sind in der Literatur auf sehr verschiedene Arten definiert und charakterisiert worden. Oft liegt dabei ein semantischer Operator wie der oben eingeführte Konsequenzoperator zugrunde. Für definite Programme zum Beispiel wird meist die Semantik betrachtet, die sich aus dem kleinsten (Herbrand-)Modell des Programms ergibt. Die in meinen Arbeiten vorgeschlagene vereinheitlichte Beschreibung von Semantiken ermöglicht Charakterisierungen verschiedenster Semantiken unter Verwendung von Stufenfunktionen als einzigem Hilfsmittel. Für definite Programme ist diese Charakterisierung wie folgt. Sie kann als prototypisch angesehen werden.

Satz 1.1 ([68])
Sei P ein definites Programm. Dann gibt es ein eindeutiges (zweiwertiges) Modell M für P, für das eine Stufenfunktion $l : B_P \rightarrow \alpha$ existiert, so dass die folgende Bedingung erfüllt ist: Für jedes $A \in M$ gibt es eine Regel $A \leftarrow A_1, \dots, A_n$ in ground(P) mit $A_i \in M$ und $l(A) > l(A_i)$ für alle i. Außerdem ist M dann das kleinste (zweiwertige) Modell für P.

Satz 1.1 enthält eine Charakterisierung des kleinsten Modells für definite Programme mit Hilfe von Stufenfunktionen. Die Charakterisierung ist *semi-syntaktischer* Natur: Die Stufenfunktion beschreibt syntaktische Bedingungen zur rekursiven Abhängigkeit von Atomen in ground(P). Durch das zu charakterisierende Modell selbst wird festgelegt, wann diese syntaktischen Bedingungen berücksichtigt werden müssen. Man vergleiche Satz 1.1 mit dem folgenden, welcher stabile Modelle [45] beschreibt und für andere Zwecke bewiesen wurde.

Satz 1.2 ([31])
Sei P ein normales Programm und M ein (zweiwertiges) Modell für P. Dann ist M genau dann ein stabiles Modell von P, wenn eine Stufenfunktion $l : B_P \to \alpha$ existiert, so dass die folgende Bedingung erfüllt ist: Für jedes $A \in M$ gibt es eine Regel $A \leftarrow A_1, \ldots, A_n, \neg B_1, \ldots, \neg B_m$ in ground(P) mit $A_i \in M$ und $l(A) > l(A_i)$ für alle i.

Die Sätze 1.1 und 1.2 unterscheiden sich nur geringfügig; vor allem ist die behandelte Programmklasse eine andere. Der formale Vergleich legt nahe, dass stabile Modelle für normale Programme eine ähnliche Rolle spielen wie das kleinste Modell für definite Programme. In der Tat haben sich stabile Modelle dafür durchgesetzt.

Der Beweis von Satz 1.1 folgt einem bestimmten Raster, welches auch für die meisten anderen Resultate unseres vereinheitlichten Ansatzes Verwendung finden kann. Zunächst ist bekannt, dass sich das kleinste Modell eines definiten Programms P als Vereinigung $\bigcup_{n\in\mathbb{N}} T_P^n(\emptyset)$ von Iterationen von $\emptyset$ unter T_P beschreiben lässt. Der Konsequenzoperator T_P ist in diesem Falle monoton, d.h. wir erhalten $T_P^n(\emptyset) \subseteq T_P^{n+1}(\emptyset)$ für alle $n \in \mathbb{N}$. Wir definieren nun die Menge $l^{-1}(n)$, für $n \in \mathbb{N}$, als $T_P^{n+1} \setminus T_P^n$. Man kann dann die charakterisierende Bedingung nachweisen. Umgekehrt zeigt man induktiv, dass aus der charakterisierenden Bedingung die Aussage $l^{-1}(n) \cap M \subseteq T_P^{n+1}$ für alle $n \in \mathbb{N}$ folgt.

Das Raster für andere Beweise unseres Ansatzes erhält man durch Veränderung des betrachteten Operators. Die Beweisdetails gestalten sich dennoch in vielen Fällen als schwierig.

Andere Ansätze zur vereinheitlichten Beschreibung verschiedener Semantiken verwenden abstraktere Ansätze zur Charakterisierung nichtmonotoner Logiken [27, 28, 88], oder basieren auf Biverbänden [26, 36, 86]. Erstere sind sehr abstrakt, zweitere sehr restriktiv. Vor allem aber zeigt sich, dass alle anderen versuchten Ansätze nicht ausreichen, um die vielfältigen syntaktische Erweiterungen zu erfassen, die zur Zeit diskutiert werden. Eine Ausnahme ist der Vorschlag in [108], das Verhältnis dieser Arbeit mit dem auf Stufenfunktionen basierenden Ansatz ist in einer der eingereichten Arbeiten ([60])

untersucht worden.

1.4 Eigene Resultate: Überblick über die eingereichten Arbeiten

In [68] wird der vereinheitlichende Ansatz erstmalig entwickelt und vorgeschlagen. Das oben genannte Beweisraster wird entwickelt. Die Flexibilität des Verfahrens wird demonstriert, indem die folgenden Semantiken charakterisiert werden: kleinstes Modell für definite Programme (Satz 1.1), Fittings Semantik [32], die wohlfundierte Semantik [119] und die schwach stratifizierte Semantik [103]. Mit Ausnahme der kleinsten Modellsemantik sind dies dreiwertige Semantiken. Das bereits zitierte Resultat von Fages (Satz 1.2) rundet die Darstellung ab.

Insbesondere durch die Behandlung der schwach stratifizierten Semantik zeigt sich, wie allgemein das beschriebene Verfahren ist. Andere Ansätze zur vereinheitlichten Beschreibung sind in der Regel nicht in der Lage, diese inzwischen weniger wichtig gewordene Semantik mit einzubeziehen. Die Darstellung lässt auch erwarten, dass unser Ansatz auf die meisten Semantiken, die sich mit monotonen Operatoren oder anderen monotonen Konstruktionen beschreiben lassen, übertragbar ist.

Das Hauptresultat der Veröffentlichung besteht in der Darstellung des konzeptionell neuen Ansatzes zur vereinheitlichten Beschreibung verschiedener Semantiken. Gleichzeitig ergeben sich aus den technischen Resultaten noch einige interessante Nebeneinsichten. Zum einen wird die Ähnlichkeit zwischen der kleinsten Modellsemantik für definite Programme und der stabilen Modellsemantik für normale Programme formalisiert und deutlich — wie bereits oben argumentiert. Eine ähnliche Parallele ergibt sich aus den Resultaten zu Fittings Semantik und der wohlfundierten Semantik: Letztere wird als eine *stratifizierte Version* (im Sinne von [6, 104]) der ersteren erkannt. Interessanterweise war es eigentlich die schwach stratifizierte Semantik, die aus dieser Motivation heraus konstruiert wurde. Die Resultate zeigen aber, dass sie — im Gegensatz zu der aus anderen Motiven entstandenen wohlfundierten Semantik — diesen Anspruch nicht erfüllt.

Die Veröffentlichung ist eine stark überarbeitete und erweiterte Version von [66].

In [54] werden Fragen behandelt, die durch die in [68] hergeleiteten Charakterisierungen verschiedener Semantiken aufgeworfen werden. Insbesondere zeigt sich in [68], dass die wohlfundierte Semantik aus Fittings Semantik abgeleitet werden kann, indem bestimmte Selbstbezüglichkeiten, die in Fittings Semantik zum Wahrheitswert *unbekannt* führen, als *falsch* ausgewertet werden. Diese Beobachtung deckt sich mit der sehr viel älteren, aber bis dahin hauptsächlich informellen Einsicht, dass die wohlfundierte Semantik *Falschheit* bevorzugt [35].

Der vereinheitlichte Ansatz erlaubt nun durch einfache formale Veränderungen der Charakterisierungen bekannter Semantiken, die Bevorzugung von *Falschheit* zugunsten von *Wahrheit* oder *Unbestimmtheit* aufzugegen. In [54] wird gezeigt, dass eine solche Behandlung zu einer Theorie führt, die völlig analog zur üblichen aufgebaut ist. Die Resultate zeigen, dass eine Entwicklung neuer und interessanter Semantiken mit Hilfe von Charakterisierungen durch Stufenfunktionen grundsätzlich möglich ist. Es ist natürlich nicht zu erwarten, dass die für diese Demonstration neu entwickelten Semantiken direkt praktische Relevanz haben werden; erst für die Behandlung noch nicht so eingehend studierter syntaktischer Erweiterungen sind entsprechende Ergebnisse zu erwarten.

Diese Veröffentlichung ist leicht überarbeitet beim *Journal of Logic and Computation* eingereicht worden.

In [60] wird unser Ansatz zur vereinheitlichten Charakterisierung von Semantiken mit einem anderen, in [108] entwickelten Ansatz ähnlicher Zielgebung, verglichen. Es wird bewiesen, dass sämtliche mit Hilfe von [108] behandelbaren Semantiken auch mit Hilfe von Stufenfunktionen charakterisiert werden können. Der formal anspruchsvolle Beweis wirft außerdem eine Reihe von Korollaren ab. In diesen werden verschiedene Semantiken für *disjunktive* Logikprogramme charakterisiert. Letztere sind eine syntaktische Erweiterung normaler Logikprogramme, indem statt eines einzigen Atoms im Kopf von Regeln auch Disjunktionen von Atomen zugelassen werden.

Die Resultate demonstrieren die Stärke und Allgemeinheit des auf Stufenfunktionen beruhenden Ansatzes.

Diese Veröffentlichung ist beim 19. Workshop on (Constraint) Logic Programming eingereicht worden.

1.5 Ausblick: Weiterführungen und Anwendungen

Ein vereinheitlichter Ansatz zur Charakterisierung verschiedener Semantiken führt direkt zu einem vertieften Verständnis der Modellierungsprinzipien in der Logikprogrammierung. Es ist zu erwarten, dass durch eine Weiterentwicklung Einsichten gewonnen werden können, die zu konkreten Anwendungen führen.

Zunächst ist zu vermuten, dass unser Ansatz durch die Verwendung von Wohlordnungen Relevanz für Entscheidbarkeitsfragen hat. Tatsächlich ist das im Bereich nichtmonotonen Schließens zur Zeit vorherrschende Paradigma *Answer Set Programming* nur für aussagenlogische oder vergleichbar endliche Problemstellungen implementiert, was Ausdrucksstärke und Anwendbarkeit für die Wissensverarbeitung stark einschränkt. Trotz hohen Bedarfs wird an diesem Punkt kaum geforscht, wohl vor allem aus Mangel an innovativen Ideen. Die prominenteste Ausnahme sind die Arbeiten von Bonatti [17]. Da das Schließen mit nichtmonotonen Semantiken wie der der stabilen oder der wohlfundierten Modelle im Allgemeinen nicht semi-entscheidbar ist, ist die Identifizierung entscheidbarer oder semi-entscheidbarer Fragmente eine naheliegende Vorgehensweise. Mit Hilfe der Charakterisierungen durch Stufenfunktionen können solche Fragmente in etwa wie folgt gewonnen werden: Zunächst charakterisiert man eine Semantik mit Hilfe von Stufenfunktionen. Dann beschreibt man Bedingungen, unter denen die Herleitung des Wahrheitswertes von Atomen der Stufe $n + 1$ aus denen der Stufe n entscheidbar ist und terminiert. Es folgt dann, dass die Wahrheitswerte aller Atome bis zur Stufe ω entscheidbar sind.[1] Sind entscheidbare Fragmente solcherart charakterisiert, ist eine Algorithmisierung entsprechender Beweis- oder Entscheidungsverfahren unproblematisch. Eine vergleichbare Konstruktion wurde in [83] für SLD-Resolution verfolgt.

Eine andere Forschungsrichtung, die sich aus dem vereinheitlichten Ansatz eröffnet, ist die semantische Untersuchung syntaktischer Erweiterungen für die Logikprogrammierung. Vor allem sind dazu disjunktive Programme zu untersuchen, was in der von mir betreuten Arbeit [79] schon im Ansatz geschehen ist. Von besonderem Interesse ist dabei allerdings die Untersuchung der verschiedenen konkurrierenden Vorschläge für wohlfundierte Semantiken für disjunktive Programme. Mit Hilfe unseres Ansatzes lassen sich diese dann vergleichen und Anhaltspunkte herausarbeiten, welche Semantiken in welchem Kontext zu bevorzugen sind. Andere syntaktische Erweiterungen, die ähnlichen Analysen zugeführt werden sollen, sind geordnete Disjunktion [18], Präferenzen [19], quantitative Erweiterungen [118, 77, 122, 90, 87] und andere.

Ist eine kritische Masse an Beschreibungen verschiedener Semantiken mit unserem Verfahren erst vorhanden, dann liegt natürlich auch die Entwicklung einer *Metatheorie* auf der Hand, mit deren Hilfe z.B. Komplexitätsklassen und andere Eigenschaften direkt aus den vereinheitlichten Charakterisierungen ablesbar sein könnten.

Zuletzt sei erwähnt, dass auch im Zusammenhang approximativer Deduk-

[1] ω ist die kleinste unendliche Ordinalzahl.

tionsverfahren Charakterisierungen mit Hilfe von Stufenfunktionen hilfreich sein können. In [59] zum Beispiel wird ein Verfahren zur approximativen Deduktion mit Hilfe von Stufenfunktionen semantisch beschrieben.

Kapitel 2

Neuro-symbolische Integration

2.1 Kurzfassung

Neuro-symbolische Integration befasst sich mit der Entwicklung intelligenter Systeme, die wünschenswerte Eigenschaften logikbasierter und konnektionistischer Wissensverarbeitung vereinen. Diese Forschung ist motiviert aus Anforderungen der Praxis, in der die Entscheidung für eines der beiden Paradigmen meist eine Entscheidung *gegen* die wünschenswerten Eigenschaften des anderen notwendig macht. Während erste Systeme für die konnektionistische Behandlung von Aussagenlogik bereits vorliegen, gestaltet sich die Integration mit der Prädikatenlogik als eine sehr schwierige Herausforderung für die Forschung.

Meine Arbeiten befassen sich mit der Integration von Logikprogrammen erster Stufe mit künstlichen neuronalen Netzen in Standardarchitekturen. In ihnen werden die zur Zeit einzigen vorliegenden Techniken beschrieben, wie solche Logikprogramme konnektionistisch repräsentiert werden können. Konkret behandelt werden Repräsentationen mit dreischichtigen Perzeptronen, mit RBF-Netzwerken, und mit *fibred* Netzwerken, wobei die konkreten Umsetzungen mit jeweils anderen Methoden erfolgen.

Zukünftige Arbeiten betreffen die Erweiterung dieser Techniken zu intelligenten Systemen mit Lern- und Erklärungsfähigkeiten, sowie die konkrete Umsetzung derselben und das Studium von Anwendungsszenarien.

2.2 Forschungskontext: Logik und künstliche neuronale Netzwerke

Intelligente Systeme, die auf künstlichen neuronalen Netzen (auch *konnektionistische Systeme* genannt) basieren, unterscheiden sich grundlegend von logikbasierten. Logikprogramme zum Beispiel sind normalerweise stark rekursiv und sind deklarativ gut verstanden. Die zugrundeliegende Sprache ist die der Prädikatenlogik, die es aufgrund ihrer symbolischen Natur einfach macht, Programmspezifikationen mehr oder weniger direkt als Programme aufzufassen. Der Erfolg von künstlichen neuronalen Netzen ist darin begründet, dass sie mit Hilfe von Rohdaten trainiert werden können, und sich in wichtigen Anwendungsgebieten die erlernten Funktionalitäten als höchst nützlich und anwendbar herausstellen — selbst wenn die Rohdaten verrauscht sind. Erfolgreiche Netzarchitekturen verwenden jedoch kaum rekurrente (d.h. rekursive) Strukturen. Ausserdem ist Wissen in einem trainierten Netz nur sehr implizit repräsentiert, und adäquate Verfahren zur Extraktion dieses Wissens in symbolischer Form konnten bis heute nicht entwickelt werden.

Eine Integration der robusten auf neuronalen Netzen basierenden Lernverfahren mit symbolischen Wissensverarbeitungsverfahren wie der Logikprogrammierung ist daher erstrebenswert, insbesondere wenn die jeweiligen Stärken der beiden Paradigmen erhalten werden können. Der aktuelle Stand der Forschung auf diesem Gebiet ist jedoch von diesem Ziel noch weit entfernt. Eines der Hauptprobleme bei der Entwicklung integrierter Systeme ist die Frage, wie symbolisches Wissen mit Hilfe künstlicher neuronaler Netze repräsentiert werden kann. Zufriedenstellende Antworten auf diese Frage werden auf natürliche Weise zu Wissensextraktionsverfahren und anwendbaren integrierten neuro-symbolischen Systemen führen.

Bisher waren die Ansätze zur Integration logischer und konnektionistischer Systeme hauptsächlich aussagenlogischer Natur oder beschränkt auf prädikatenlogische Logikprogramme ohne Funktionssymbole oder auf ähnliche endliche Einschränkungen, die sich auf eine aussagenlogische Behandlung zurückführen lassen. Diese Ansätze reichen zurück bis zu den bahnbrechenden Arbeiten von McCulloch und Pitts [93] und führten in den 80er und 90er Jahren zu einer Reihe von Systemen wie KBANN [117], SHRUTI [112, 111], BUR [73] und anderen Arbeiten wie z.B. [84, 97, 70, 41, 44]. Überblicksarbeiten zu diesem Gebiet sind z.B. [20, 42, 49].

Die Entwicklung integrierter neuro-symbolischer Systeme wird sehr viel schwieriger, wenn man die Einschränkung auf endliche Systeme zu überwinden sucht. Ein prinzipielles Hindernis besteht dabei in der Notwendigkeit, eine im Grunde unendliche Sprache (z.B. der Prädikatenlogik) mit

Hilfe endlich vieler Netzwerkknoten zu behandeln. Neben den Arbeiten zu Autoassoziativspeichern (RAAM), die von Pollack initiiert wurden [100] und lediglich das Lernen von Termen erster Ordnung betreffen, und den verwandten *holographic reduced representations* [98] gibt es dazu im Grunde nur einen einzigen Ansatz, der auf [72] zurückgeht, welcher wiederum von der aussagenlogischen Arbeit [71] motiviert ist. Er basiert auf der Idee, Logikprogramme mit Hilfe ihres Konsequenzoperators zu repräsentieren. Der Operator wird dazu in eine reellwertige Funktion übertragen, die unter gewissen Bedingungen durch neuronale Netze mit Standardarchitekturen berechnet oder approximiert werden kann.

Von zentraler Wichtigkeit für diesen Ansatz ist ein Approximationssatz von Funahashi [37], welcher grob gesprochen aussagt, dass sich jede auf einem Kompaktum definierte stetige reellwertige Funktion durch Eingabe-Ausgabe-Funktionen von neuronalen Netzwerken in Standardarchitektur — mehrschichtige Perzeptronen mit sigmoidalen Aktivierungsfunktionen — beliebig genau in der Maximumsnorm approximieren lässt. In [72] wird gezeigt, dass sich dieses Resultat verwenden lässt, um in diesem Sinne die Approximierbarkeit einer eingeschränkten Programmklasse, nämlich der von Programmen, die azyklisch bezüglich einer bijektiven Stufenfunktion sind, zu zeigen. Der Beweis in [72] ist jedoch ein reiner Existenzbeweis, d.h. konkrete Angaben, wie ein approximierendes Netzwerk gefunden werden kann, können daraus nicht direkt abgeleitet werden.

Schon in meiner Dissertation [51], bzw. in [63], findet sich eine erste geringfügige Verallgemeinerung dieser Resultate. Meine Anstrengungen seither verfolgten zum einen das Ziel, die in [72] behandelte sehr eingeschränkte Programmklasse zu verallgemeinern, und zum anderen auf konstruktive — und daher praktisch anwendbare — Repräsentationen hinzuarbeiten. In meinen Arbeiten werden die zur Zeit einzigen vorliegenden Techniken beschrieben, wie solche Logikprogramme konnektionistisch repräsentiert werden können. Konkret behandelt werden Repräsentationen mit dreischichtigen Perzeptronen, mit RBF-Netzwerken, und mit *fibred* Netzwerken, wobei die konkreten Umsetzungen mit jeweils anderen Methoden erfolgen.

2.3 Technische Hinführung: Konnektionistische Repräsentationen von Logikprogrammen erster Stufe

Biologische neuronale Netze bestehen aus einer Menge von *Neuronen*, die aus *Soma*, *Axon* und *Dendriten* bestehen und miteinander verbunden sind. Elektrische Potenziale können durch molekulare Mechanismen entlang der Dendriten zum Soma gelangen und ein elektrisches *Erregungspotenzial* auslösen, welches dann entlang des Axons zu den Dendriten anderer Neuronen transportiert wird.

Künstliche neuronale Netzwerke sind Abstraktionen der biologischen. Die Neuronen entsprechen einer Menge von Knoten in einem gerichteten Graphen; die Kanten des Graphen entsprechen den Verbindungen der Neuronen durch Axone und Dendriten. Anstelle elektrischer Potenziale werden in künstlichen Netzen in der Regel reelle Zahlen entlang der Verbindungen weitergegeben. Zu jedem Zeitpunkt wird also mit jedem Knoten oder Neuron eine reelle Zahl assoziiert, die wir die *Erregung* des Knotens nennen. Die Kanten sind zudem mit reellwertigen sogenannten *Gewichten* ausgestattet, deren Funktion noch erklärt werden wird. Mit jedem Knoten wird außerdem eine reellwertige *Aktivierungsfunktion* assoziiert.

Die Weiterleitung von Erregungen geschieht nun wie folgt. Sind $N_1, \ldots, N_n$ die Vorgängerknoten eines Knotens N, und ist x_i die Erregung von N_i für alle i, dann errechnet sich die Erregung x von N nach der Formel

$$x = \phi\left(\sum_{i=1}^{n} w_i x_i\right),$$

wobei w_i, für alle i, das reellzahlige Gewicht der Verbindung von N_i nach N ist, und $\phi : \mathbb{R} \to \mathbb{R}$ die Aktivierungsfunktion von N. Die Weiterleitung von Erregung im Netzwerk ist meist synchronisiert, d.h. die Aktualisierung der Erregung von allen oder von Gruppen von Knoten geschieht gleichzeitig in diskreten Zeitschritten. Ist der zugrundeliegende Graph azyklisch, dann sprechen wir von einem *vorwärtsgerichteten* Netz; in diesem Falle sind die Knoten meist in Schichten angeordnet, Verbindungen gehen immer nur von einer Schicht zur nächsten, und die Schichten werden nacheinander aktualisiert. Ist der zugrundeliegende Graph nicht azyklisch, sprechen wir von einem *rekurrenten* Netz. In vielen Fällen werden außerdem gewisse Knoten als *Eingangsknoten* und andere als *Ausgangsknoten* betrachtet. Mit ihrer Hilfe werden Eingaben ins Netzwerk gemacht und Ausgaben abgelesen. In vorwärtsgerichteten Netzen sind meist die Knoten der ersten Schicht die Eingangsknoten und die der letzten Schicht die Ausgangsknoten. In einem solchen Fall assoziiert man mit einem Netzwerk eine *Eingabe-Ausgabefunktion*, die folgendermaßen bestimmt ist: Sind $x_1, \ldots, x_n$ die Erregungen der n Eingangsknoten, und sind $y_1, \ldots, y_m$ die Erregungen der m Ausgangsknoten, so bildet die Eingabe-Ausgabefunktion des Netzwerks das Tupel $(x_1, \ldots, x_n)$ auf das Tupel $(y_1, \ldots, y_m)$ ab, sie ist also eine Funktion von $\mathbb{R}^n$ nach $\mathbb{R}^m$.

Eine Vielzahl verschiedener Weiterleitungsmechanismen und Architekturen werden in der Theorie behandelt und in der Praxis angewandt; nicht alle von ihnen fallen unter das vorgestellte Schema, das aber für unsere Diskussion genügen wird. Eine der meistverbreiteten Architekturen ist die des *dreischichtigen Perzeptrons*. Es handelt sich dabei um ein vorwärtsgerichtetes Netz mit drei Schichten, einer Eingabeschicht, einer versteckten Schicht und einer Ausgabeschicht. Die Aktivierungsfunktion ist *sigmoidal*, d.h. sie ist nichtkonstant, beschränkt, monoton steigend und stetig. Meist wird die Funktion

$$\phi : x \mapsto \frac{1}{1+e^{-x}}$$

oder eine ähnliche dafür verwendet.

Der auf [72] zurückgehende Ansatz zur Repräsentation logischer Programme durch künstliche neuronale Netze verwendet folgendes Resultat:

Satz 2.1 ([37])
Sei $\phi : \mathbb{R} \to \mathbb{R}$ eine nichtkonstante, beschränkte, monoton steigende und stetige Funktion, $K \subseteq \mathbb{R}^n$ kompakt, $f : K \to \mathbb{R}$ stetig und $\varepsilon > 0$. Dann existiert ein dreischichtiges Perzeptron mit Aktivierungsfunktion ϕ, dessen Eingabe-Ausgabefunktion $\bar{f} : K \to \mathbb{R}$ die Bedingung $\max_{x \in K} d(f(x), \bar{f}(x)) < \varepsilon$ erfüllt, wobei d eine die natürliche Topologie auf $\mathbb{R}$ erzeugende Metrik ist.

Zur Repräsentation eines Logikprogrammes P verwenden wir nun den zugehörigen Konsequenzoperator $T_P : I_P \to I_P$ und repräsentieren ihn mit Hilfe der Eingabe-Ausgabefunktion $F : K \to \mathbb{R}$ mit $K \subseteq \mathbb{R}$ kompakt, eines mehrschichtigen Perzeptrons. Dazu muss zunächst die Menge I_P aller Interpretationen als eine kompakte Teilmenge von $\mathbb{R}$ verstanden werden. Dazu bedienen wir uns einer bijektiven Stufenfunktion $l : B_P \to \mathbb{N}$, wählen eine natürliche Zahl $B > 2$ und definieren die Einbettung

$$\iota : I_P \to \mathbb{R} : I \mapsto \sum_{n=0}^{\infty} B^{-n} I(l^{-1}(n)).$$

Dabei ist $I(A) = 1$ wenn $I \models A$, und $I(A) = 0$ wenn $I \not\models A$.

Die Bilder $\iota(I_P)$ sind wohlbekannte Teilmengen von $[0, 1]$, nämlich *Cantormengen*, die zum einen in der mathematischen *Topologie* (z.B. [124]), zum anderen in der Theorie der *Fraktale* (z.B. [13]) und anderswo Verwendung finden.

Mit Hilfe der Einbettung ι könnnen wir nun den Operator T_P auf die reellen Zahlen übertragen: Wir definieren

$$\iota(T_P) : \iota(I_P) \to \iota(I_P) : x \mapsto \iota(T_P(\iota^{-1}(x))).$$

Die Funktion $\iota(T_P)$ ist nun nach Funahashis Satz durch dreischichtige Perzeptronen approximierbar, wenn sie stetig ist. Versieht man I_P mit der durch ι vermittelten Initialtopologie[2] Q, dann ist $\iota(T_P)$ approximierbar, wenn T_P stetig bezüglich Q ist.

Die Cantortopologie Q auf I_P hat unabhängig davon in die Theorie der Logikprogrammierung zur Behandlung nichtmonotoner Konsequenzoperatoren Eingang gefunden (z.B. [14, 110, 65]), ist also eine in der Logikprogrammierung natürlich auftretende Struktur. Zur Charakterisierung der Stetigkeit von T_P bezüglich Q sei das folgende Resultat angeführt.

Satz 2.2 (Spezialisierung eines Resultats aus [57])
Sei P ein Logikprogramm. Dann ist T_P genau dann stetig in der Cantortopologie, wenn für alle $A \in B_P$ und $I \in I_P$ mit $A \notin I$ eine endliche Menge $S \subseteq B_P$ existiert, so dass für alle $J \in I_P$, die mit I auf S übereinstimmen, gilt: $A \in T_P(J)$ genau dann, wenn $A \in T_P(I)$.

Zusammenfassend erhalten wir folgenden Satz, der als Ausgangspunkt für die Diskussion meiner Beiträge zu diesem Forschungsgebiet dient.

Satz 2.3 (Spezialisierung eines Resultats aus [57])
Sei P ein Logikprogramm, so dass T_P die Bedingungen aus Satz 2.2 erfüllt. Dann ist T_P im Sinne von Satz 2.1 durch Eingabe-Ausgabefunktionen von dreischichtigen Perzeptronen approximierbar.

Die durch Satz 2.3 behandelbaren Programme sind z.B. alle aussagenlogischen, sowie alle *überdeckten*; das sind solche, bei denen jede in einem Rumpf auftretende Variable auch im zugehörigen Kopf auftritt. Nach [50] ist diese Einschränkung für definite Programme unerheblich.

2.4 Eigene Resultate: Überblick über die eingereichten Arbeiten

In [57] wird der in [72] behandelte Spezialfall zur Repräsentation von Logikprogrammen mit dreischichtigen Perzeptronen bedeutend erweitert. Dies wird ermöglicht durch eine sehr viel umfassendere Behandlung des Themas aus topologischer Sicht. Dadurch wird dieses Forschungsthema eingebettet in den größeren Forschungskontext der Arbeiten um topologische Methoden in der Logikprogrammierung, zu denen z.B. [14, 34, 110, 16, 101, 65] gehören.

In [57] werden neben dem üblichen Konsequenzoperator T_P eine ganze Klasse von verwandten semantischen Operatoren behandelt, die in der Lo-

[2] [124]

gikprogrammierung auftreten. Darunter fällt zum Beispiel der üblicherweise mit Fittings Semantik assoziierte Operator aus [32], aber auch andere, wie in [33, 61, 36].

Gleichzeitig wird in [57] die Behandlung aller Programme mit Cantor-stetigem semantischem Operator ermöglicht, und Stetigkeitscharakterisierungen für diese erarbeitet. Es wird außerdem gezeigt, dass unter gewissen Bedingungen an das Logikprogramm auch das Iterationsverhalten semantischer Operatoren mit Hilfe der approximierenden Netzwerke simuliert werden kann. Dafür werden unter Anderem genaue Fehlerabschätzungen für die Approximationen abgeleitet.

Es wird auch kurz eine Alternative zum Satz von Funahashi betrachtet, die auf [74] zurückgeht. In diesem Approximationsresultat wird für die zu approximierende Funktion nur Messbarkeit vorausgesetzt, die Approximation selbst ist dann jedoch auch nur bezüglich einer aus dem zugrundeliegenden Maß abgeleiteten Metrik zu gewährleisten. In [57] wird gezeigt, dass die entsprechend in $\mathbb{R}$ eingebetteten semantischen Operatoren *für alle Programme* stets messbar sind. Aufgrund der genannten Einschränkungen bezüglich der zu gewährleistenden Approximation bleibt die Anwendbarkeit dieser Einsicht jedoch zunächst zweifelhaft.

In der Veröffentlichung werden außerdem ausführlich aussagenlogische Resultate behandelt, die auf [71] zurückgehen.

Die Veröffentlichung ist eine Zusammenführung, Überarbeitung und Erweiterung von [71, 64].

In [11] wird das Problem der Repräsentation beziehungsweise Approximation von $\iota(T_P)$ mit Hilfe von künstlichen neuronalen Netzen auf grundsätzlich andere Weise und unter Umgehung des Satzes von Funahashi behandelt. Ausgangspunkt ist die Beobachtung, dass näherungsweise Darstellungen des Graphen der Funktion $\iota(T_P)$ für beliebige P *selbstähnlich* im Sinne der Fraktal- und Chaostheorie erscheinen. Solche selbstähnlichen Strukturen treten als Attraktoren von iterierten Funktionensystemen im Sinne von [13] auf.

Da sich iterierte Funktionensysteme recht einfach mit Hilfe rekurrenter Netze bestimmter Architektur darstellen lassen, liegt es auf der Hand, ein Verfahren zu entwickeln, mit dem sich aus einem gegebenen Programm P ein iteriertes Funktionensystem ableiten lässt, dessen Attraktor $\iota(T_P)$ ist oder approximiert. Durch Überführung in ein rekurrentes Netz erhält man dann eine konnektionistische Darstellung von P.

Es stellt sich nun heraus, dass Stetigkeit von $\iota(T_P)$ für eine solche Behandlung nur bedingt ausreicht. Gewährleisten lässt sich die Existenz eines iterierten Funktionensystems, dessen Attraktor der Graph von $\iota(T_P)$ ist, nur unter der Bedingung, dass $\iota(T_P)$ *Lipschitz-stetig* [124] ist. Das gewonnene Resultat ist jedoch stark genug, um die experimentell gewonnene Beobachtung der Selbstähnlichkeit der Graphen zu begründen.

Das genannte Resultat liefert jedoch keine befriedigende Darstellung eines approximierenden iterierten Funktionensystems. In einer technisch sehr aufwändigen Darstellung wird daher gezeigt, wie unter der genannten Bedingung der Lipschitz-stetigkeit zu P eine Folge von iterierten Funktionensystemen gewonnen werden kann, deren Folge von Attraktoren gegen den Graphen von T_P konvergiert. Die verwendete Technik ist eine Abwandlung der *fraktalen Interpolation* aus [13].

Abschließend wird in der Veröffentlichung dargelegt, wie konnektionistische Systeme aus den erhaltenen iterierten Funktionensystemen gewonnen werden können.

Die Arbeit führt die Forschung um die konnektionistische Behandlung von prädikatenlogischen Logikprogrammen insofern entscheidend weiter, als in ihr erstmalig konkrete Algorithmen zur Konstruktion approximierender Netzwerke vorstellt werden.

[55] ist eine Weiterführung der unter meiner Betreuung entstandenen Arbeit [123]. In letzterer wird ein Ergebnis über die *Fixpunktvervollständigung* $\mathrm{fix}(P)$ eines Programms P, wie in [29] eingeführt, vorgetragen. Es besagt unter anderem, dass der die stabile Modellsemantik charakterisierende zu P gehörige *Gelfond-Lifschitz-Operator* mit $T_{\mathrm{fix}(P)}$ identisch ist, d.h. die Behandlung des ersteren Operators lässt sich auf den einfacheren letzteren zurückführen. Gleichzeitig wird auch einer der zur wohlfundierten Semantik gehörenden Operatoren auf den einfacheren zu Fittings Semantik gehörenden Operator zurückgeführt.

In [55] werden verschiedene Korollare aus diesen Resultaten gezogen. Obwohl der Gelfond-Lifschitz-Operator nicht unter die in [57] behandelten Operatoren fällt, werden mit Hilfe des in [123] erzielten Resultats die Ergebnisse dennoch übertragbar, und ebenso die Ergebnisse aus [11]. Gleiches gilt auch für den genannten, zur wohlfundierten Semantik gehörenden Operator, der sich mit den in [57] genannten Methoden nicht direkt behandeln lässt — der zu Fittings Semantik gehörende Operator aber schon.

In der Veröffentlichung werden außerdem weitere Korollare behandelt, die auf ähnliche Weise aus einigen in meiner Dissertation gezeigten Ergebnissen abgeleitet werden können.

In dieser Arbeit wird erstmalig vorgestellt, wie prädikatenlogische Logikprogramme unter stabiler Modellsemantik konnektionistisch behandelt werden können.

[12] ist eine Übersichtsarbeit, in der der Stand der aktuellen Forschung zu neuro-symbolischer Integration jenseits der Aussagenlogik dargestellt wird. Dabei wird ein Fragenkatalog vorgestellt, der aktuelle Herausforderungen in diesem Gebiet aufzeigt und zukünftige vorausgreift.

Von dieser Veröffentlichung ist eine Zeitschriftversion in Vorbereitung.

In [10] wird die Suche nach geeigneten konnektionistischen Darstellungsformen für Logikprogramme weitergeführt. Ausgangspunkt sind neueste Arbeiten zu sogenannten *fibring* Netzwerken [43], bei denen Erregungen von Knoten die Gewichte oder Aktivierungsfunktionen anderswo im Netzwerk kontrolliert verändern können.

In [10] wird exemplarisch vorgestellt, wie Programme mit Hilfe von einfachen fibring Netzwerken dargestellt werden können. Die Vorgehensweise ist konstruktiv, d.h. es werden Verfahren angegeben, mit denen konkret die entsprechenden Netzwerke gewonnen werden können.

Diese Veröffentlichung ist bei der FLAIRS 2005 im Special Track on Integrated Intelligent Systems eingereicht worden.

2.5 Ausblick: Weiterführung und Anwendungen

In jüngster Zeit wurden auf dem Gebiet der neuro-symbolischen Integration Fortschritte erzielt, die eine Überführung dieser Technologie in den Anwendungsbereich attraktiv machen. Ein Anwendungsgebiet, das ich in absehbarer Zeit dafür erschließen möchte, ist das automatische Erlernen ontologischen Wissens, wie es zur Zeit im Zusammenhang mit dem Semantic Web erforscht wird. Dabei geht es um die Bereitstellung von Hintergrundwissen in Form von Ontologien, um das Internet, dessen Inhalte bis heute nur von Menschen erschlossen werden können, für die intelligente maschinelle Verarbeitung zugänglich zu machen.

Beim Lernen von Ontologien geht es um die Generierung dieses ontologischen Hintergrundwissens mit Hilfe von maschinellen Lernverfahren, motiviert durch die vernünftige Annahme, dass auch in Zukunft viele Webseiten, wenn nicht die meisten, ohne entsprechende Annotation mit Ontologien im Internet bereitgestellt, also automatisch erschlossen werden müssen. Es kann angenommen werden, dass sich die für Ontologien verwendeten Wissensrepräsentationssprachen, insbesondere das auf Beschreibungslogiken basierende OWL [96, 4], für eine konnektionistische Behandlung eignen. Bei OWL handelt es sich z.B. um ein entscheidbares Fragment der Prädikatenlogik erster Stufe und es sollte deshalb mit endlichen Netzen abbildbar sein. Einen Ausgangspunkt für solche Untersuchungen bietet das in [48, 121] beschriebene

Fragment. Das zur Zeit entstehende Gebiet des Lernens von Ontologien ist ausserdem noch in der Phase der Konsolidierung, in der geeignete Techniken zum maschinellen Lernen gesucht werden, und die erfolgreichen Techniken der künstlichen neuronalen Netze können nicht ohne eine neuro-symbolische Brücke für diese Zwecke genutzt werden.

Parallel zur praktischen Umsetzung der bekannten Techniken der neuro-symbolischen Integration stellen sich auch noch einige grundsätzliche Fragen, die durch die oben diskutierten Arbeiten aufgeworfen werden. Zum einen fehlen noch immer in vielen Fällen konkrete Algorithmen zur Konstruktion von approximierenden Netzen. Zum anderen ist die behandelbare Programmklasse immer noch eingeschränkt und sollte erweitert werden, in etwa durch geeignete Transformation der Programme vor der Approximation durch Netzwerke.

Ein weiterer naheliegender nächster Schritt ist die Bereitstellung von Wissensextraktionsverfahren aus neuronalen Netzen, basierend auf den diskutierten Repräsentationen, sowie die Entwicklung von Lernalgorithmen, z.B. Modifikationen des *Backpropagation*-Algorithmus (siehe [15]), die repräsentiertes symbolisches Wissen respektieren. Diese Arbeiten sollten jedoch in enger Verknüpfung mit der Entwicklung angewandter Verfahren vorgenommen werden.

Kapitel 3

Schließen über begrifflichem Wissen

3.1 Kurzfassung

Anforderungen aus der Praxis im Umfeld des Semantic Web zeigen, dass eine Integration begrifflichen Wissens — z.B. in Form von Ontologien — mit regelbasierten Systemen — z.B. der Logikprogrammierung — erforderlich ist. Wie diese Integration am besten zu bewerkstelligen ist, ist zur Zeit Gegenstand intensiver Forschungsanstrengungen.

In meinen Arbeiten verfolge ich systematisch die Frage der Integration von Begriffshierarchien und regelbasiertem nichtmonotonem Schließen. Meine Resultate liefern konkrete normative Charakterisierungen der erstrebenswerten Systeme und ihrer Semantiken. Sie zeichnen sich dadurch aus, dass sie die einzigen Arbeiten sind, die diese Frage von einem systematischen und strukturorientierten Standpunkt aus betrachten und eine Integration aus grundlegenden Prinzipien ableiten.

Zukünftige Arbeiten befassen sich mit der konkreten Algorithmisierung und praktischen Umsetzung der Integration auf Anwendungsszenarien im Bereich der Formalen Begriffsanalyse und des Semantic Web.

3.2 Forschungskontext: Logik und begriffliches Wissen

Die Darstellung begrifflichen Wissens nimmt gerade in jüngster Zeit in der Wissensverarbeitung an Bedeutung zu. Insbesondere im Umfeld der Semantic Web-Forschung entstanden und entstehen Repräsentationssprachen, die speziell auf explizite oder implizite Darstellung und Verarbeitung begrifflichen Wissens ausgelegt sind [115]. Prominente Vertreter sind Beschreibunslogiken [8], insbesondere OWL [96, 4], F-Logik [76, 3], aber auch Begriffsgraphen und andere. Die entsprechenden begrifflichen Wissensbasen werden meist als *Ontologien* bezeichnet. In der Semantic Web-Forschung interessiert unter anderem, welche Repräsentationsmittel grundsätzlich für die Praxis am geeignetsten sind.

Die entsprechenden Repräsentationssprachen sind logischer Natur, obgleich dieser Sachverhalt nicht immer im Vordergrund steht. Tatsächlich lassen sie sich oft als Fragmente der Prädikatenlogik erster Stufe auffassen. Gleichzeitig werden aber auch Repräsentationsmittel benötigt, die über diese hinausgehen und zum Beispiel arithmetische Aspekte oder nichtmonotones Schließen möglich machen. Die Diskussion um geeignete Mittel dazu ist im Augenblick in vollem Gange.

Begriffliches Wissen handelt von Begriffen und ihren Beziehungen zueinander, zum Beispiel in Form von Taxonomien. Durch die natürlichen Beziehungen zwischen *Unterbegriffen* und *Oberbegriffen* ist begriffliches Wissen daher in erster Linie hierarchisch organisiert, d.h. kann mit Hilfe von Ordnungsstrukturen dargestellt werden. Die *Formale Begriffsanalyse* [40] stellt mathematische Methoden zur Erzeugung von Begriffshierarchien aus Rohdaten zur Verfügung und findet dazu immer weitere Verbreitung in der Informatik und im Bereich des Semantic Web [116]. Durch die ihr zugrunde liegende reichhaltige Theorie der Verbandsstrukturen eröffnen sich Begriffshierarchien somit einer formalen Analyse mit diesen Mitteln. Da die logische Lesart von Begriffshierarchien für die Wissensverarbeitung von entscheidender Bedeutung ist, ergibt sich außerdem ein Zusammenspiel von Ordnungsstrukturen und Logik in diesem Gebiet, das formal zwangsläufig mit Themen und Methoden aus dem Bereich der *Stone-Dualität* [75] eng verwandt ist.

Für die Informatik relevante Aspekte der Stone-Dualität und — allgemeiner — des Zusammenspiels zwischen Ordnungsstrukturen und Logik werden in der *Domänentheorie* [2, 47] untersucht, die die formalen Grundlagen zur Untersuchung von denotationellen Semantiken für Programmiersprachen zur Verfügung stellt. Sie befasst sich mit Ordnungsstrukturen, die zur Modellierung von Aspekten der Berechenbarkeit verwendet werden können. Elemente einer Ordnungsstruktur werden dabei als nach ihrem *Informationsgehalt* geordnet verstanden und formal in maschinell repräsentierbare und approximierbare unterteilt.

In dem mit Stone-Dualität eng verwandten Teilbereich der *Domänenlogiken* werden logische Repräsentationsformen für in der Domänentheorie auftretende Ordnungsstrukturen untersucht. Diese Vorgehensweise wurde schon von Scott [109] initiiert, dessen Arbeiten auch maßgeblich für die Entstehung der Domänentheorie als Forschungsgebiet waren. Arbeiten, die diese Gedanken fortführen, sind z.B. [120, 1, 126, 114, 25]. Die auftretenden logischen Formalismen erinnern dabei oft an regelbasierte Systeme, wie sie in der Logikprogrammierung vorkommen. Die Verwendung domänentheoretischer Methoden im Bereich des nichtmonotonen Schließens mit Logikprogrammen liegt daher auf der Hand und wurde ansatzweise ebenfalls untersucht, z.B. in [78, 62, 107, 128, 65, 127].

Meine jüngsten Beiträge zu diesem Gebiet zielen auf die Erarbeitung von theoretisch sauber fundierten Grundlagen zur Behandlung von hierarchisch strukturiertem begrifflichem Wissen, die als normativ für die Entwicklung von anwendbaren Verfahren gelten können. Mein Interesse gilt dabei insbesondere der Verbindung von regelbasiertem nichtmonotonem Schließen und hierarchischem Wissen, aber auch Methoden zur Verknüpfung von heterogenen ontologischen Wissensbasen. Meine Resultate liefern konkrete normative Charakterisierungen der erstrebenswerten Systeme und ihrer Semantiken. Sie zeichnen sich dadurch aus, dass sie die einzigen Arbeiten sind, die diese Frage von einem systematischen und strukturorientierten Standpunkt aus betrachten und eine Integration aus grundlegenden Prinzipien ableiten.

3.3 Technische Hinführung: Formale Begriffsanalyse und Domänentheorie

Zur formalen Behandlung begrifflichen Wissens bedienen wir uns der Formalen Begriffsanalyse [40]. Sie entstand aus philosophischem Gedankengut [125] und dient in Anwendungsgebieten vor allem der Erzeugung und visuellen Darstellung von Begriffshierarchien aus Rohdaten, die in der Form von Gegenstands-Merkmals-Beziehungen vorliegen.

Ein *formaler Kontext* (G, M, I) besteht aus einer Menge G von *Gegenständen*, einer Menge M von *Merkmalen* und einer infix notierten *Inzidenzrelation* $I \subseteq G \times M$. Für $A \subseteq G$ und $B \subseteq M$ definieren wir

$$\begin{aligned}\alpha(A) &= \{m \in M \mid aIm \text{ für alle } a \in A\} \qquad \text{und}\\ \omega(B) &= \{g \in G \mid gIb \text{ für alle } b \in B\}.\end{aligned}$$

Ein Paar (A, B) mit $A \subseteq G$ und $B \subseteq M$ heißt ein *formaler Begriff*, wenn $\alpha(A) = B$ und $\omega(B) = A$ ist. Wir nennen A den *Extent* und B den *Intent* des Begriffs (A, B). Die Menge $\mathcal{B}(G, M, I)$ aller formalen Begriffe des formalen Kontexts (G, M, I) lässt sich auch äquivalent beschreiben als die Menge aller Paare $(\omega(B), \alpha(\omega(B)))$ für $B \in M$. Für unsere Diskussion können wir deshalb formale Begriffe mit ihren Intenten identifizieren. Für zwei Begriffe (d.h. Beggriffsintenten) B und C schreiben wir $B \preceq C$, falls $C \subseteq B$. Wir betrachten $\mathcal{B}(G, M, I)$ als geordnet durch $\preceq$. Es stellt sich heraus, dass die so zu erhaltenden Ordnungsstrukturen genau die vollständigen Verbände sind. $(\mathcal{B}(G, M, I), \preceq)$ heißt entsprechend der zu (G, M, I) gehörige *Begriffsverband*. Die geordnete Teilmenge, die aus $(\mathcal{B}(G, M, I), \preceq)$ durch Einschränkung auf alle Elemente der Form $\alpha(g)$ oder $\alpha(\omega(m))$, für $g \in G$ bzw. $m \in M$, entsteht, heißt die *Galois-Teilhierarchie* von $\mathcal{B}(G, M, I)$.

In Anwendungsszenarien in der Informatik, den Natur- und Sozialwissenschaften (siehe z.B. [116, 81, 102]) stellen sich die durch formale Begriffsanalysen gewonnenen Begriffshierarchien meist für den menschlichen Experten als intuitiv einsichtige Form der Wissensrepräsentation dar. Durch ihre dadurch begründete hohe Anwendungsrelevanz und ihre saubere formale Grundlegung in der Mathematik ist sie daher für das Studium begrifflichen Wissens bestens geeignet. Entsprechend gibt es bereits zahlreiche Arbeiten z.B. im Kontext der Beschreibungslogiken [9] und anderer Bereiche der Wissensrepräsentation mit Ontologien (z.B. [23, 22, 21]).

Auch logische Aspekte der formalen Begriffsanalyse wurden schon untersucht (z.B. [39]). Wir suchen allerdings die Nähe zu den für die Informatik relevanten Strukturen im Umfeld der Stone-Dualität. Als das geeignete Werkzeug stellt sich die ursprünglich für die Charakterisierung von Smyth-Powerdomänen entwickelte Domänenlogik *RZ* heraus. Wir benötigen zunächst etwas Begriffsbildung. Sie folgt [107].

Eine *kohärente algebraische Cpo* ist eine Ordnungsstruktur $(D, \sqsubseteq)$, die folgenden Bedingungen genügt:

(i) $(D, \sqsubseteq)$ ist eine partiell geordnete Menge mit einem kleinsten Element $\bot$.

(ii) D enthält das Supremum jeder gerichteten Teilmenge.

(iii) Für alle $d \in D$ gilt $d = \sup(\downarrow d \cap \mathsf{K}(D))$, wobei $\downarrow d = \{c \in D \mid c \sqsubseteq d\}$ ist und $\mathsf{K}(D)$ die Menge aller $c \in D$ ist, so dass für jede gerichtete Menge $A \subseteq D$ mit $c \sqsubseteq \sup A$ ein $a \in A$ mit $c \sqsubseteq a$ existiert. $\mathsf{K}(D)$ heißt die *Menge der kompakten Elemente* von D.

(iv) Die Schnittmenge endlich vieler Scott-kompakt-offener Teilmengen von D ist wieder Scott-kompakt-offen. *Scott-kompakt-offen* bezieht sich dabei auf die *Scott-Topologie* auf $(D, \sqsubseteq)$, zu der eine Basis durch die Mengen der Form $\{y \mid x \sqsubseteq y\}$ für $x \in \mathsf{K}(D)$ gegeben ist.

Intuitiv erschließt sich die gerade gegebene Definition in etwa folgendermaßen. Die Menge D besteht aus *Informationseinheiten*, die nach

Informations- oder Wissensgehalt geordnet sind. Das kleinste Element $\perp$ ist das Element ohne Information. Hat man eine verträgliche (gerichtete) Teilmenge an Informationseinheiten, so existiert auch eine Einheit (das Supremum), die die Gesamtinformation dieser Teilmenge repräsentiert (Bedingung (ii)). Kompakte Elemente stehen für *maschinell repräsentierbare*, d.h. in einem gewissen Sinne *endliche* Einheiten. In D lässt sich somit jede Information durch repräsentierbare Informationen annähern. Die letzte Bedingung (iv) ist eine technische Bedingung, die einen zusätzlichen finitistischen Charakter von D erzwingt. Übersetzt im Sinne von [107] steht es in etwa für die Aussage: Endliche Disjunktionen repräsentierbarer Einheiten sind wieder repräsentierbar.

Ein Hauptbeispiel für eine kohärente algebraische Cpo ist die Menge $(I_{P,3}, \subseteq)$ aus Abschnitt 1.3. Die Menge $\mathsf{K}(I_{P,3})$ besteht dabei wie intuitiv zu erwarten genau aus den *endlichen* Teilmengen von $I_{P,3}$, und es ist $\perp = \emptyset$. Die Menge $(I_{P,3}, \subseteq)$ ist auch als $\mathbb{T}^\omega$ in der Domänentheorie bekannt [99].

Außerdem ist jede endliche partiell geordnete Menge mit kleinstem Element eine kohärente algebraische Cpo.

Die *Logik RZ* wird nun folgendermaßen definiert, folgend [107].

Sei $(D, \sqsubseteq)$ eine kohärente algebraische Cpo. Eine *Klausel* oder *Disjunktion* über D ist eine endliche Teilmenge von $\mathsf{K}(D)$. Wir nennen $w \in D$ ein *Modell* für eine Klausel X und schreiben $d \models X$, wenn es ein $x \in X$ gibt mit $x \sqsubseteq d$. Eine *Theorie* T ist eine Menge von Klauseln, und wir schreiben $w \models T$, wenn $w \models X$ für alle $X \in T$ gilt. Wir schreiben $T \models X$, wenn aus $w \models T$ immer $w \models X$ folgt, und nennen die Klausel X in diesem Fall eine *logische Konsequenz* aus T.

Die gerade gegebene Definition wird unmittelbar einsichtig, wenn man sich Beispiele aus $(I_{P,3}, \subseteq)$ vor Augen führt. Zum Beispiel gelten $\{\{\{p,q\}\}\} \models \{\{p\}\}$ und $\{\{\{p\},\{\neg q\}\},\{\{q\}\}\} \models \{\{p\}\}$. Ersteres entspräche in etwa der Implikation von p aus $p \wedge q$. Das Zweite sagt aus, dass aus $(p \vee \neg q) \wedge q$ die Aussage p folgt.

Das Verhältnis zwischen Formaler Begriffsanalyse und der Logik RZ erschließt sich nun z.B. aus folgendem Resultat:

Satz 3.1 ([56])
Sei $(D, \sqsubseteq)$ eine kohärente algebraische Cpo, (G, M, I) ein formaler Kontext und $(L, \leq)$ die Galois-Teilhierarchie von $\mathcal{B}(G, M, I)$. Sei außermdem $\iota : L \to D$ eine ordnungsumkehrende injektive Funktion mit $\mathsf{K}(D) \subseteq \iota(L)$. Ist nun $A = \{m_1, \ldots, m_n\} \subseteq M$ eine Menge von Merkmalen mit $\iota(m_i) \in \mathsf{K}(D)$ für alle i, dann gilt

$$\alpha(\omega(A)) = \{m \mid \{\{\iota(m_1)\}, \ldots, \{\iota(m_n)\}\} \models \{\iota(m)\}\}.$$

Informell ausgedrückt sagt Satz 3.1, dass (endlicher) Begriffsabschluss in der Formalen Begriffsanalyse dem konjunktiven Fragment der Logik RZ entspricht.

3.4 Eigene Resultate: Überblick über die eingereichten Arbeiten

In [56] werden die Formale Begriffsanalyse und Answer Set Programming als nichtmonotones Wissensverarbeitungsparadigma zum ersten Mal mit Hilfe der Domänenlogik RZ verknüpft. Satz 3.1 bietet dabei den Ausgangspunkt. Die Logik RZ wird zu einem disjunktiven Logikprogrammierparadigma auf kohärenten algebraischen Cpos erweitert und mit nichtmonotonen Aspekten im Sinne des Answer Set Programming ausgestattet. Mittels der Logik RZ wird jedem solchen RZ-Logikprogramm eine Semantik zugeordnet.

Das so erhaltene Logikprogrammierparadigma ist eine Verallgemeinerung von Answer Set Programming in folgendem Sinne: Jedes Answer-Set-Programm im Sinne von [46] kann direkt in ein RZ-Logikprogramm über $\mathbb{T}^\omega$ übersetzt werden, so dass die bezüglich RZ erhaltene Semantik genau der stabilen Modellsemantik des ursprünglichen Programms entspricht. In diesem Sinne sind RZ-Logikprogramme eine konservative Verallgemeinerung von Answer Set Programming auf (geeignete) Ordnungsstrukturen. Die Einschränkung der Ordnungsstrukturen auf kohärente algebraische Cpos ist dabei nicht bedenklich, da in der Praxis der Wissensverarbeitung auftretende Hierarchien dieser Bedingung meist trivialerweise genügen.

Das soeben diskutierte grundlegende Resultat liefert nun zum einen ein Paradigma zur Fragebeantwortung über begrifflichem Wissen im Sinne der Formalen Begriffsanalyse. Dabei können RZ-Logikprogramme wahlweise als komplexe Anfragen oder als regelbasierte Erweiterung der hierarchischen Wissensbasis verstanden werden. Eine entsprechende nicht veröffentlichte Prototypimplementierung in Prolog hat gezeigt, dass die erhaltenen Antworten wie zu erwarten intuitiv einleuchten. Es sei auch bemerkt, dass die zugrundeliegende Ordnungsstruktur keineswegs aus einem formalen Kontext gewonnen sein muss — Klassenhierarchien über OWL-Ontologien können genauso dafür verwendet werden wie Taxonomien und anderes ontologisches Wissen.

Zum anderen und Wesentlicheren ist das diskutierte Resultat aber fundamentaler und normativer Natur. Die erfolgreiche und kohärente Verknüpfung fortgeschrittener Techniken und Resultate aus verschiedenen Disziplinen ist ein starker Hinweis darauf, dass ein sauberes Verfahren zum Answer Set

Programming über begrifflichem Wissen *semantisch* dem vorgestellten entsprechen sollte. Die Entwicklung von konkreten Umsetzungen dieser Einsicht steht allerdings noch aus.

Von dieser Veröffentlichung ist eine Zeitschriftenversion in Vorbereitung.

In [58] wird das Verhältnis von Formaler Begriffsanalyse und Domänenlogik ausgiebig beleuchtet. Zentral ist dabei der neu eingeführte Begriff der *approximierbaren Begriffsverbände*. Ein *approximierbarer Begriff* ist dabei eine Menge B von Merkmalen, so dass für jede endliche Teilmenge $E \subseteq B$ die Beziehung $\alpha(\omega(E)) \subseteq B$ gilt. Für approximierbare Begriffe B, C definiert man wieder $B \preceq C$ wenn $C \subseteq B$ gilt. Jeder (herkömmliche) formale Begriff ist approximierbar. Die aus formalen Kontexten erhältlichen approximierbaren Begriffsverbände sind genau die vollständigen algebraischen Verbände.

Auf der domänenlogischen Seite kommen spezielle *Scott-Informationssysteme* [109] zum Einsatz, die sich als deduktive Systeme über Konjunktionen von Aussagen verstehen lassen. Formal wird gezeigt, dass die Kategorie der Scott-Informationssyteme mit den üblichen Morphismen und trivialer Konsistenzrelation äquivalent zur Kategorie **Cxt** der formalen Kontexte ist, wobei letztere mit Hilfe von zu approximierbaren Begriffen passenden, intuitiv einleuchtenden und natürlichen Morphismen gebildet wird. Aus bekannten Resultaten folgt damit sofort, dass **Cxt** äquivalent zur Kategorie der in der Domänentheorie wichtigen vollständigen algebraischen Verbände ist.

In der Veröffentlichung wird dieser Sachverhalt von logischer und algebraischer Seite weiter beleuchtet durch Bezugnahme und Einbindung in das Umfeld der Stone-Dualität und durch eine Diskussion von Satz 3.1 in diesem Kontext.

Diese Veröffentlichung ist eine sehr stark überarbeitete und erweiterte Version von [69]. Sie wurde bei *Theoretical Computer Science* eingereicht.

In [52] werden im Wesentlichen zwei Resultate bewiesen. Zum einen wird die zur Logik RZ gehörende Beweistheorie vereinfacht. Zum anderen wird gezeigt, unter welchen Nebenbedingungen sich logische Konsequenz in RZ durch ein Resolutionsverfahren berechnen lässt. Diese Ansätze bieten Potenzial für Anwendungen, wurden aber noch nicht weiterentwickelt.

Eine Kurzversion dieser Veröffentlichung erschien in [53].

In [67] wird der zu Satz 3.1 gehörige Fall *endlicher* Kontexte und Ordnungsstrukturen durchleuchtet. Die formalen Resultate sind in diesem Fall ähnlich, aber wie zu erwarten sehr viel stärker. Da in Anwendungen der Formalen Begriffsanalyse meist nur endliche Kontexte auftreten, ist diese Beschränkung für die Praxis kaum von Relevanz.

3.5 Ausblick: Schließen über dem Semantic Web

Aus meinen Arbeiten ergeben sich mindestens die folgenden drei Ansätze zur Umsetzung in Anwendungsgebieten.

Zum Ersten liefern RZ-Logikprogramme ein Paradigma für Regelerweiterungen begrifflicher Wissensbasen, zusammen mit einer komplexen Anfragesprache für dieselben. Vor allem im Umfeld erfolgreicher Anwendungsszenarien der Formalen Begriffsanalyse erscheinen entsprechende Fallstudien lohnenswert. Insbesondere kann dadurch das implizit in großen als formale Kontexte vorliegenden Rohdatenmengen enthaltene Wissen ohne vorherige aufwändige Berechnung des Begiffsverbandes erschlossen werden. Die algorithmische Umsetzung kann durch Kombination der Beweistheorie der Logik RZ mit erfolgreichen Methoden des Answer Set Programming erfolgen.

Zum Zweiten geben die in [56] angestellten Untersuchungen Hinweise darauf, wie eine formal saubere Verknüpfung von hierarchischem Wissen und regelbasiertem nichtmonotonem Schließen auf semantischer Ebene zu gestalten ist. Um konkrete Paradigmen zu entwickeln ist eine Übertragung auf die im Umfeld des Semantic Web verwendeten Wissensrepräsentationssprachen vorzunehmen und sind entsprechende Deduktionsalgorithmen zu entwerfen.

Zum Dritten berührt [58] Fragen des Zusammenführens von ontologischen Wissensbasen. Wie z.B. in [82] ausführlich dargelegt, lassen sich solche Verfahren aus kategorientheoretischen Konstruktionen herleiten. Die Identifikation geeigneter Kategorien von für die Semantic-Web-Forschung relevanten Wissensrepräsentationsformalismen zielt daher auf Anwendungen in diesem Bereich.

Anteil des Autors an den eingereichten Arbeiten

Pascal Hitzler and Matthias Wendt. A uniform approach to logic programming semantics. *Theory and Practice of Logic Programming*, 5(1–2), 2005, 123–159. Im Druck. [68]. Stark überarbeitete und erweiterte Fassung von [66].

Mein Koautor war zum Zeitpunkt der Entstehung dieser Arbeit Student im Hauptstudium an der Fakultät für Informatik der TU Dresden. Unter meiner Anleitung erarbeitete er Spezialfälle der Resultate, die ich dann verallgemeinerte und in einen größeren Kontext einbettete. Die Fragestellung stammt von mir.

Pascal Hitzler. Towards a systematic account of different logic programming semantics. In *Proceedings of the 26th German Conference on Artificial Intelligence, KI2003, Hamburg, September 2003*, Lecture Notes in Artificial Intelligence. Springer, Berlin, 2003. [54]. Leicht überarbeitet beim *Journal of Logic and Computation* eingereicht.

Ich bin alleiniger Autor dieser Arbeit.

Pascal Hitzler and Sibylle Schwarz. Level mapping characterizations of selector-generated models for logic programs. Technical Report WV–04–04, Knowledge Representation and Reasoning Group, Department of Computer Science, Dresden University of Technology, 2004. [60]. Beim *19. Workshop on (Constraint) Logic Programming* eingereicht.

Meine Koautorin war zur Zeit der Entstehung dieser Arbeit Doktorandin unter Prof. Herre an der Fakultät für Mathematik und Informatik an der Universität Leipzig. Die Arbeit entstand in enger Kooperation und ist beiden Autoren zu gleichen Teilen zuzurechnen.

Pascal Hitzler, Steffen Hölldobler, and Anthony K. Seda. Logic programs and connectionist networks. *Journal of Applied Logic*, 2(3), 2004, 245-272, Special Issue on Neural-Symbolic Systems. [57]. Zusammenführung und Überarbeitung von [71, 64].

Die Arbeit besteht im Wesentlichen aus einem aussagenlogischen und einem prädikatenlogischen Teil. Am aussagenlogischen Teil war ich kaum beteiligt. Die Fragestellung und die Resultate zum prädikatenlogischen Teil — mit Ausnahme einer kurzen Diskussion von Ergebnissen aus [72] — wurden von mir erdacht, entworfen und in erster Näherung ausgeführt. Die detaillierte Ausarbeitung dieses Teils wurde zusammen mit Anthony K. Seda vorgenommen.

Sebastian Bader and Pascal Hitzler. Logic programs, iterated function systems, and recurrent radial basis function networks. *Journal of Applied Logic* 2(3), 2004, 273-300, Special Issue on Neural-Symbolic Systems. [11].

Zum Zeitpunkt der Entstehung dieser Arbeit arbeitete mein Koautor unter meiner Anleitung an seiner Masterarbeit im Studiengang Computational Logic an der TU Dresden. Die Fragestellung entstand bei ausführlichen Diskussionen zur Thematik, wobei Sebastian Bader dabei die massgeblichen Ideen einbrachte. In die Ausführung der zum Teil sehr anspruchsvollen Beweise war ich an vielen Punkten lenkend und korrigierend involviert und ebenso bei der Ausarbeitung und Verallgemeinerung der Resultate zu einem für eine gehobene Zeitschrift angemessenen Beitrag.

Pascal Hitzler. Corollaries on the fixpoint completion: studying the stable semantics by means of the Clark completion. In D. Seipel, M. Hanus, U. Geske, and O. Bartenstein, editors, *Proceedings of the 15th International Conference on Applications of Declarative Programming and Knowledge Management and the 18th Workshop on Logic Programming, Potsdam, Germany, March 4-6, 2004*, Technichal Report 327, pages 13–27. Bayerische Julius-Maximilians-Universität Würzburg, Institut für Informatik, 2004. [55].

Ich bin alleiniger Autor dieser Arbeit.

Sebastian Bader, Pascal Hitzler and Steffen Hölldobler. The integration of connectionism and knowledge representation and reasoning as a challenge for artificial intelligence. In: L. Li and K.K. Yen, *Proceedings of the Third International Conference on Information, Tokyo, Japan, November/December 2004*, pages 22–33. ISBN 4-901329-02-2, International Information Institute, 2004. Eine Zeitschriftversion ist in Vorbereitung.

Der Erstautor war zum Zeitpunkt der Entstehung dieser Arbeit Doktorand unter meiner Anleitung. Sie entstand aus einem ähnlichen älteren Entwurf von Steffen Hölldobler, den ich substantiell ergänzte, auf den neuesten Stand brachte und nach Diskussionen mit meinen Koautoren erweiterte.

Sebastian Bader, Artur S. d'Avila Garcez and Pascal Hitzler. Computing first-order logic programs by fibring artificial neural networks. *Technical*

Report, Institute AIFB, University of Karlsruhe, 2004. [10]. Eingereicht bei *FLAIRS 2005*, Special Session on Integrated Systems.

Der Erstautor war zum Zeitpunkt der Entstehung dieser Arbeit Doktorand unter meiner Anleitung. Sie entstand aus einer Diskussion der ersten beiden Autoren und wurde vom Erstautor und mir ausgeführt.

Pascal Hitzler. Default reasoning over domains and concept hierarchies. In *Biundo, Frühwirth and Palm, Proceedings of the 27th German conference on Artificial Intelligence, KI'2004, Ulm, Germany, September 2004*, Lecture Notes in Artificial Intelligence 3238, pages 351–365. Springer, Berlin, 2004. [56]. Eine Zeitschriftversion ist in Vorbereitung.

Ich bin alleiniger Autor dieser Arbeit.

Pascal Hitzler, Markus Krötzsch, and Guo-Qiang Zhang. A categorical view on algebraic lattices in formal concept analysis. *Technischer Bericht, AIFB, Universität Karlsruhe.* [58]. Bei *Theoretical Computer Science* eingereicht. Sie ist eine stark überarbeitete und erweiterte Fassung von [69].

Markus Krötzsch arbeitete zum Zeitpunkt der Entstehung dieser Arbeit an seiner Masterarbeit im Studiengang Computational Logic an der TU Dresden unter der Anleitung der anderen beiden Autoren. Die Arbeit ist eine Ausarbeitung von [69], in der die wesentlichen neuen Resultate enthalten sind. Zu diesen stammt die Fragestellung von Guo-Qiang Zhang, die Ausarbeitung im Wesentlichen von mir. Die Erweiterungen zu [58] wurden in erster Linie von Markus Krötzsch mit Ergänzungen von mir eingebracht.

Pascal Hitzler. A generalized resolution theorem. *Journal of Electrical Engineering, Slovak Academy of Sciences*, 55(1–2):25–30, 2003. [52]. Eine Kurzversion erschien in [53].

Ich bin alleiniger Autor dieser Arbeit.

Pascal Hitzler and Matthias Wendt. Formal concept analysis and resolution in algebraic domains. In Aldo de Moor and Bernhard Ganter, editors, *Using Conceptual Structures — Contributions to ICCS 2003*, pages 157–170. Shaker Verlag, Aachen, 2003. [67].

Zum Zeitpunkt der Entstehung dieser Arbeit war mein Koautor Student im Masterstudiengang Computational Logic an der TU Dresden. Die Fragestellung und ersten Ergebnisse stammen von mir, wurden von meinem Koautor verallgemeinert und dann in enger Zusammenarbeit ausgearbeitet.

Literaturverzeichnis

[1] Samson Abramsky. Domain theory in logical form. *Annals of Pure and Applied Logic*, 51:1–77, 1991.

[2] Samson Abramsky and Achim Jung. Domain theory. In Samson Abramsky, Dov Gabbay, and Thomas S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 3. Clarendon, Oxford, 1994.

[3] Jürgen Angele and Georg Lausen. Ontologies in F-logic. In Steffen Staab and Rudi Studer, editors, *Handbook on Ontologies*, pages 29–50. Springer, 2004.

[4] Grigoris Antoniou and Frank van Harmelen. Web Ontology Language: OWL. In Steffen Staab and Rudi Studer, editors, *Handbook on Ontologies*, pages 67–92. Springer, 2004.

[5] Krzysztof R. Apt. *From Logic Programming to Prolog.* International Series in Computer Science. Prentice Hall, 1997.

[6] Krzysztof R. Apt, Howard A. Blair, and Adrian Walker. Towards a theory of declarative knowledge. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 89–148. Morgan Kaufmann, Los Altos, CA, 1988.

[7] Krzysztof R. Apt, V. Wiktor Marek, Miroslav Truszczyński, and David S. Warren, editors. *The Logic Programming Paradigm: A 25-Year Perspective.* Springer, Berlin, 1999.

[8] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications.* Cambridge University Press, 2003.

[9] Franz Baader and Ralf Molitor. Building and structuring description logic knowledge bases using least common subsumers and concept

analysis. In B. Ganter and G. Mineau, editors, *Conceptual Structures: Logical, Linguistic, and Computational Issues – Proceedings of the 8th International Conference on Conceptual Structures (ICCS2000)*, volume 1867 of *Lecture Notes in Artificial Intelligence*, pages 290–303. Springer Verlag, 2000.

[10] Sebastian Bader, Artur S. d'Avila Garcez, and Pascal Hitzler. Computing first-order logic programs by fibring artificial neural networks. Technical report, Institute AIFB, University of Karlsruhe, September 2004. Submitted to FLAIRS 2005, Special Track on Integrated Systems.

[11] Sebastian Bader and Pascal Hitzler. Logic programs, iterated function systems, and recurrent radial basis function networks. *Journal of Applied Logic*, 2(3):273–300, 2004.

[12] Sebastian Bader, Pascal Hitzler, and Steffen Hölldobler. The integration of connectionism and knowledge representation and reasoning as a challenge for artificial intelligence. In L. Li and K.K. Yen, editors, *Proceedings of the Third International Conference on Information, Tokyo, Japan, November/December 2004*, pages 22–33. International Information Institute, 2004. ISBN 4-901329-02-2.

[13] Michael Barnsley. *Fractals Everywhere.* Academic Press, San Diego, CA, USA, 1993.

[14] Aida Batarekh and V.S. Subrahmanian. Topological model set deformations in logic programming. *Fundamenta Informaticae*, 12:357–400, 1989.

[15] Christopher M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, 1995.

[16] Howard A. Blair, Fred Dushin, David W. Jakel, Angel J. Rivera, and Metin Sezgin. Continuous models of computation for logic programs. In Krzysztof R. Apt, V. Wiktor Marek, Miroslav Truszczyński, and David S. Warren, editors, *The Logic Programming Paradigm: A 25-Year Persepective*, pages 231–255. Springer, Berlin, 1999.

[17] Piero A. Bonatti. Reasoning with infinite stable models. *Artificial Intelligence*, 156(1):75–111, 2004.

[18] Gerhard Brewka. Logic programming with ordered disjunction. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence, July/August, 2002, Edmonton, Alberta, Canada*, pages 100–105. AAAI Press, 2002.

[19] Gerhard Brewka and Thomas Eiter. Preferred answer sets for extended logic programs. *Artificial Intelligence*, 109:297–356, 1999.

[20] Anthony Browne and Ron Sun. Connectionist inference models. *Neural Networks*, 14(10):1331–1355, 2001.

[21] Philipp Cimiano, Andreas Hotho, and Steffen Staab. Clustering ontologies from text. In *Proceedings of the Conference on Lexical Resources and Evaluation (LREC)*, 2004.

[22] Philipp Cimiano, Gerd Stumme, Andreas Hotho, and Julien Tane. Conceptual knowledge processing with formal concept analysis and ontologies. In *Proceedings of the The Second International Conference on Formal Concept Analysis (ICFCA 04)*, 2004.

[23] Richard J. Cole, Peter W. Eklund, and Gerd Stumme. Document retrieval for email search and discovery using formal concept analysis. *Journal of Applied Artificial Intelligence (AAI)*, 17(3):257–280, 2003.

[24] Alain Colmerauer and Philippe Roussel. The birth of Prolog. In *ACM SIGPLAN Notices*, volume 28(3), pages 37–52. ACM Press, 1993.

[25] Thierry Coquand and Guo-Qiang Zhang. Sequents, frames, and completeness. In *14th International Workshop on Computer Science Logic, Fischbachau, Germany, August 2000*, volume 1862 of *Lecture Notes in Computer Science*, pages 277–291. Springer, 2000.

[26] Marc Denecker, V. Wiktor Marek, and Miroslaw Truszczynski. Approximating operators, stable operators, well-founded fixpoints and applications in non-monotonic reasoning. In Jack Minker, editor, *Logic-based Artificial Intelligence*, chapter 6, pages 127–144. Kluwer Academic Publishers, Boston, 2000.

[27] Jürgen Dix. A classification theory of semantics of normal logic programs: I. Strong properties. *Fundamenta Informaticae*, 22(3):227–255, 1995.

[28] Jürgen Dix. A classification theory of semantics of normal logic programs: II. Weak properties. *Fundamenta Informaticae*, 22(3):257–288, 1995.

[29] Phan Minh Dung and Kanchana Kanchanasut. A fixpoint approach to declarative semantics of logic programs. In Ewing L. Lusk and Ross A. Overbeek, editors, *Logic Programming, Proceedings of the North American Conference 1989, NACLP'89, Cleveland, Ohio*, pages 604–625. MIT Press, 1989.

[30] Thomas Eiter, Nicola Leone, Christinel Mateis, Gerald Pfeifer, and Francesco Scarcello. A deductive system for nonmonotonic reasoning. In Jürgen Dix, Ulrich Furbach, and Anil Nerode, editors, *Proceedings of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'97*, volume 1265 of *Lecture Notes in Artificial Intelligence*. Springer, Berlin, 1997.

[31] François Fages. Consistency of Clark's completion and existence of stable models. *Journal of Methods of Logic in Computer Science*, 1:51–60, 1994.

[32] Melvin Fitting. A Kripke-Kleene-semantics for general logic programs. *The Journal of Logic Programming*, 2:295–312, 1985.

[33] Melvin Fitting. Bilattices and the semantics of logic programming. *The Journal of Logic Programming*, 11:91–116, 1991.

[34] Melvin Fitting. Metric methods: Three examples and a theorem. *The Journal of Logic Programming*, 21(3):113–127, 1994.

[35] Melvin Fitting. A theory of truth that prefers falsehood. *Journal of Philosophical Logic*, 26:477–500, 1997.

[36] Melvin Fitting. Fixpoint semantics for logic programming — A survey. *Theoretical Computer Science*, 278(1–2):25–51, 2002.

[37] Ken-Ichi Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2:183–192, 1989.

[38] Dov M. Gabbay, C.J. Hogger, and J.A. Robinson. *Nonmonotonic Reasoning and Uncertain Reasoning*, volume 3 of *Handbook of Logic in Artificial Intelligence and Logic Programming.* Clarendon Press, Oxford, 1994.

[39] Bernhard Ganter and Rudolf Wille. Contextual attribute logic. In William M. Tepfenhart and Walling R. Cyre, editors, *Conceptual Structures: Standards and Practices. Proceedings of the 7th International Conference on Conceptual Structures, ICCS '99, July 1999, Blacksburgh, Virginia, USA*, volume 1640 of *Lecture Notes in Artificial Intelligence*, pages 377–388. Springer, Berlin, 1999.

[40] Bernhard Ganter and Rudolf Wille. *Formal Concept Analysis — Mathematical Foundations.* Springer, Berlin, 1999.

[41] Artur S. d'Avila Garcez, Krysia Broda, and Dov M. Gabbay. Symbolic knowledge extraction from trained neural networks: A sound approach. *Artificial Intelligence*, 125:155–207, 2001.

[42] Artur S. d'Avila Garcez, Krysia B. Broda, and Dov M. Gabbay. *Neural-Symbolic Learning Systems — Foundations and Applications.* Perspectives in Neural Computing. Springer, Berlin, 2002.

[43] Artur S. d'Avila Garcez and Dov M. Gabbay. Fibring neural networks. In *In Proceedings of the 19th National Conference on Artificial Intelligence (AAAI 04). San Jose, California, USA, July 2004.* AAAI Press, 2004. To appear.

[44] Artur S. d'Avila Garcez and Gerson Zaverucha. The connectionist inductive learning and logic programming system. *Applied Intelligence, Special Issue on Neural networks and Structured Knowledge*, 11(1):59–77, 1999.

[45] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert A. Kowalski and Kenneth A. Bowen, editors, *Logic Programming. Proceedings of the 5th International Conference and Symposium on Logic Programming*, pages 1070–1080. MIT Press, 1988.

[46] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.

[47] G. Gierz, K.H. Hofmann, K. Keimel, J.D. Lawson, M. Mislove, and D.S. Scott. *Continuous Lattices and Domains*, volume 93 of *Encyclopedia of Mathematics and its Applications.* Cambridge University Press, 2003.

[48] Benjamin Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. Description logic programs: Combining logic programs with description

logics. In *Proc. of WWW 2003, Budapest, Hungary, May 2003*, pages 48–57. ACM, 2003.

[49] Hans W. Güsgen and Steffen Hölldobler. Connectionist inference systems. In Bertram Fronhöfer and Graham Wrightson, editors, *Parallelization in Inference Systems*, volume 590 of *Lecture Notes in Artificial Intelligence*, pages 82–120. Springer, Berlin, 1992.

[50] Michael Hanus. On extra variables in (Equational) Logic Programming. In Leon Sterling, editor, *Logic Programming, Proceedings of the Twelfth International Conference on Logic Programming, June 1995, Tokyo, Japan*, pages 665–679. MIT Press, 1995.

[51] Pascal Hitzler. *Generalized Metrics and Topology in Logic Programming Semantics.* PhD thesis, Department of Mathematics, National University of Ireland, University College Cork, 2001.

[52] Pascal Hitzler. A generalized resolution theorem. *Journal of Electrical Engineering, Slovak Academy of Sciences*, 55(1–2):25–30, 2003.

[53] Pascal Hitzler. A resolution theorem for algebraic domains. In Georg Gottlob and Toby Walsh, editors, *Proceedings of the 18th International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 2003*, pages 1339–1340. Morgan Kaufmann Publishers, 2003.

[54] Pascal Hitzler. Towards a systematic account of different logic programming semantics. In Andreas Günter, Rudolf Kruse, and Bernd Neumann, editors, *Proceedings of the 26th German Conference on Artificial Intelligence, KI2003, Hamburg, September 2003*, volume 2821 of *Lecture Notes in Artificial Intelligence*, pages 355–369. Springer, Berlin, 2003.

[55] Pascal Hitzler. Corollaries on the fixpoint completion: studying the stable semantics by means of the clark completion. In D. Seipel, M. Hanus, U. Geske, and O. Bartenstein, editors, *Proceedings of the 15th International Conference on Applications of Declarative Programming and Knowledge Management and the 18th Workshop on Logic Programming, Potsdam, Germany, March 4-6, 2004*, volume 327 of *Technichal Report*, pages 13–27. Bayerische Julius-Maximilians-Universität Würzburg, Institut für Informatik, 2004.

[56] Pascal Hitzler. Default reasoning over domains and concept hierarchies. In Susanne Biundo, Thom Frühwirth, and Günther Palm, editors,

Proceedings of the 27th German conference on Artificial Intelligence, KI'2004, Ulm, Germany, September 2004, volume 3238 of *Lecture Notes in Artificial Intelligence*, pages 351–365. Springer, Berlin, 2004.

[57] Pascal Hitzler, Steffen Hölldobler, and Anthony K. Seda. Logic programs and connectionist networks. *Journal of Applied Logic*, 3(2):245–272, 2004.

[58] Pascal Hitzler, Markus Krötzsch, and Guo-Qiang Zhang. A categorical view on algebraic lattices in formal concept analysis. Technical report, AIFB, Universität Karlsruhe, 2004. Submitted to Theoretical Computer Science.

[59] Pascal Hitzler and Boris Motik. Towards resolution-based approximate reasoning for OWL-DL. In: Perry Groot, Pascal Hitzler, Boris Motik, Holger Wache, Methods for Approximate Reasoning. EU Knowledge-Web Network of Excellence deliverable D2.1.2, 2004. To appear.

[60] Pascal Hitzler and Sibylle Schwarz. Level mapping characterizations of selector-generated models for logic programs. Technical Report WV–04–04, Knowledge Representation and Reasoning Group, Department of Computer Science, Dresden University of Technology, 2004. Submitted to the 19th Workshop on (Constraint) Logic Programming.

[61] Pascal Hitzler and Anthony K. Seda. Characterizations of classes of programs by three-valued operators. In Michael Gelfond, Nicola Leone, and Gerald Pfeifer, editors, *Logic Programming and Nonmonotonic Reasoning, Proceedings of the 5th International Conference on Logic Programming and Non-Monotonic Reasoning, LPNMR'99, El Paso, Texas, USA*, volume 1730 of *Lecture Notes in Artificial Intelligence*, pages 357–371. Springer, Berlin, 1999.

[62] Pascal Hitzler and Anthony K. Seda. Some issues concerning fixed points in computational logic: Quasi-metrics, multivalued mappings and the Knaster-Tarski theorem. In *Proceedings of the 14th Summer Conference on Topology and its Applications: Special Session on Topology in Computer Science, New York*, volume 24 of *Topology Proceedings*, pages 223–250, 1999.

[63] Pascal Hitzler and Anthony K. Seda. A note on relationships between logic programs and neural networks. In Paul Gibson and David Sinclair, editors, *Proceedings of the Fourth Irish Workshop on Formal Methods,*

IWFM'00, Electronic Workshops in Comupting (eWiC). British Computer Society, 2000.

[64] Pascal Hitzler and Anthony K. Seda. Continuity of semantic operators in logic programming and their approximation by artificial neural networks. In Andreas Günter, Rudolf Kruse, and Bernd Neumann, editors, *Proceedings of the 26th German Conference on Artificial Intelligence, KI2003*, volume 2821 of *Lecture Notes in Artificial Intelligence*, pages 105–119. Springer, 2003.

[65] Pascal Hitzler and Anthony K. Seda. Generalized metrics and uniquely determined logic programs. *Theoretical Computer Science*, 305(1–3):187–219, 2003.

[66] Pascal Hitzler and Matthias Wendt. The well-founded semantics is a stratified Fitting semantics. In Matthias Jarke, Jana Koehler, and Gerhard Lakemeyer, editors, *Proceedings of the 25th Annual German Conference on Artificial Intelligence, KI2002, Aachen, Germany, September 2002*, volume 2479 of *Lecture Notes in Artificial Intelligence*, pages 205–221. Springer, Berlin, 2002.

[67] Pascal Hitzler and Matthias Wendt. Formal concept analysis and resolution in algebraic domains. In Aldo de Moor and Bernhard Ganter, editors, *Using Conceptual Structures — Contributions to ICCS 2003*, pages 157–170. Shaker Verlag, Aachen, 2003.

[68] Pascal Hitzler and Matthias Wendt. A uniform approach to logic programming semantics. *Theory and Practice of Logic Programming*, 5(1–2):123–159, 2005. To appear.

[69] Pascal Hitzler and Guo-Qiang Zhang. A cartesian closed category of approximable concept structures. In Karl-Erich Wolff, Heather D. Pfeiffer, and Harry S. Delugach, editors, *Proceedings of the International Conference On Conceptual Structures, Huntsville, Alabama, USA*, Lecture Notes in Computer Science, pages 170–185. Springer, July 2004.

[70] Steffen Hölldobler. *Automated Inferencing and Connectionist Models.* Fakultät Informatik, Technische Hochschule Darmstadt, 1993. Habilitationsschrift.

[71] Steffen Hölldobler and Yvonne Kalinke. Towards a massively parallel computational model for logic programming. In *Proceedings ECAI94 Workshop on Combining Symbolic and Connectionist Processing*, pages 68–77. ECCAI, 1994.

[72] Steffen Hölldobler, Yvonne Kalinke, and Hans-Peter Störr. Approximating the semantics of logic programs by recurrent neural networks. *Applied Intelligence*, 11:45–58, 1999.

[73] Steffen Hölldobler, Yvonne Kalinke, and Jörg Wunderlich. A recursive neural network for reflexive reasoning. In Stefan Wermter and Ron Sun, editors, *Hybrid Neural Systems.* Springer, Berlin, 1999.

[74] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[75] Peter T. Johnstone. *Stone Spaces.* Number 3 in Cambridge studies in advanced mathematics. Cambridge University Press, 1982.

[76] Michael Kifer, Georg Lausen, and James Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42:741–843, 1995.

[77] Michael Kifer and V.S. Subrahmanian. Theory of generalized annotated logic programming and its applications. *The Journal of Logic Programming*, 1993.

[78] Eric Klavins, William C. Rounds, and Guo-Qiang Zhang. Experimenting with power default reasoning. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference, AAAI 98, IAAI 98, July 1998, Madison, Wisconsin, USA*, pages 846–852. AAAI Press / The MIT Press, 1998.

[79] Matthias Knorr. Level mapping characterizations for quantitative and disjunctive logic programs. Bachelor's Thesis, Department of Computer Science, Technische Universität Dresden, Germany, 2003.

[80] Robert A. Kowalski. Predicate logic as a programming language. In *Proceedings IFIP'74*, pages 569–574. North-Holland, 1974.

[81] Sabine Krolak-Schwerdt and Bernhard Ganter. Cognitive organization of person attributes: Measurement procedures and statistical models. In *Exploratory Data Analysis in Empirical Research*, volume 22 of *Studies in Classification, Data Analysis, and Knowledge Organization*, pages 472–482. Springer, 2002.

[82] Markus Krötzsch, Pascal Hitzler, Marc Ehrig, and York Sure. What is ontology merging? — a category-theoretical perspective using pushouts. Technical report, AIFB, Universität Karlsruhe, 2004. Submitted to the 2nd European Semantic Web Conference.

[83] Kenneth Kunen. Negation in logic programming. *The Journal of Logic Programming*, 4:289–308, 1987.

[84] Trent E. Lange and Michael G. Dyer. Frame selection in a connectionist model of high-level inferencing. In *COGSCI*, pages 706–713, 1989.

[85] John W. Lloyd. *Foundations of Logic Programming*. Springer, Berlin, 1988.

[86] Yann Loyer, Nicolas Spyratos, and Daniel Stamate. Parametrized semantics of logic programs — a unifying framework. *Theoretical Computer Science*, 308(1–3):429–447, 2003.

[87] Thomas Lukasiewicz. Fixpoint characterizations for many-valued disjunctive logic programs with probabilistic semantics. In *Proceedings of the 6th International Conference on Logic Programming and Non-Monotonic Reasoning, Vienna, Austria*, September 2001.

[88] David Makinson. Bridges between classical and nonmonotonic logic. *Logic Journal of the IGPL*, 11(1):69–96, 2003.

[89] V. Wiktor Marek and Miroslav Truszczyński. Stable models and an alternative logic programming paradigm. In Krzysztof R. Apt, V. Wiktor Marek, Miroslav Truszczyński, and David S. Warren, editors, *The Logic Programming Paradigm: A 25-Year Persepective*, pages 375–398. Springer, Berlin, 1999.

[90] Cristinel Mateis. Quantitative disjunctive logic programming: Semantics and computation. *AI communications*, 13(4):225–248, 2000.

[91] John McCarthy. Epistemological problems of artificial intelligence. In *Proceedings of IJCAI-77*, pages 1038–1044, 1977.

[92] John McCarthy. Circumscription — a form of non-monotonic reasoning. *Artificial Intelligence*, 13(1):27–39, 1980.

[93] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[94] Robert Moore. Possible-worlds semantics for autoepistemic logic. In *Proceedings of the 1984 Non-monotonic Reasoning Workshop*. AAAI, Menlo Park, CA, 1984.

[95] Robert Moore. Semantical considerations on nonmonotonic logic. *Artificial Intelligence*, 25(1), 1985.

[96] Web ontology language (OWL). www.w3.org/2004/OWL/, 2004.

[97] Gadi Pinkas. Propositional non-monotonic reasoning and inconsistency in symmetric neural networks. In John Mylopoulos and Raymond Reiter, editors, *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 525–530. Morgan Kaufmann, 1991.

[98] Tony A. Plate. Holographic reduced representations. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 30–35, 1991.

[99] Gordon Plotkin. T^ω as a universal domain. *Journal of Computer and System Sciences*, 17:209–236, 1978.

[100] Jordan B. Pollack. Recursive distributed representations. *AIJ*, 46:77–105, 1990.

[101] Sibylla Prieß-Crampe and Paolo Ribenboim. Ultrametric spaces and logic programming. *The Journal of Logic Programming*, 42:59–70, 2000.

[102] Uta Priss. Linguistic applications of formal concept analysis. In *Proceedings of ICFCA 2003*, 2003. To appear.

[103] Halina Przymusinska and Teodor C. Przymusinski. Weakly stratified logic programs. *Fundamenta Informaticae*, 13:51–65, 1990.

[104] Teodor C. Przymusinski. On the declarative semantics of deductive databases and logic programs. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 193–216. Morgan Kaufmann, Los Altos, CA, 1988.

[105] Raymond Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.

[106] J. Alan Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, 1965.

[107] William C. Rounds and Guo-Qiang Zhang. Clausal logic and logic programming in algebraic domains. *Information and Computation*, 171(2):156–182, 2001.

[108] Sibylle Schwarz. *Selektor-erzeugte Modelle verallgemeinerter logischer Programme*. PhD thesis, Universität Leipzig, 2004.

[109] Dana S. Scott. Domains for denotational semantics. In Magens Nielsen and Erik M. Schmidt, editors, *Automata, Languages and Programming, 9th Colloquium, July 1982, Aarhus, Denmark, Proceedings*, volume 140 of *Lecture Notes in Computer Science*, pages 577–613. Springer, Berlin, 1982.

[110] Anthony K. Seda. Topology and the semantics of logic programs. *Fundamenta Informaticae*, 24(4):359–386, 1995.

[111] Lokenda Shastri. Advances in Shruti — A neurally motivated model of relational knowledge representation and rapid inference using temporal synchrony. *Applied Intelligence*, 11:78–108, 1999.

[112] Lokendra Shastri and Venkat Ajjanagadde. From associations to systematic reasoning: A connectionist representation of rules, variables and dynamic bindings using temporal synchrony. *Behavioural and Brain Sciences*, 16(3):417–494, September 1993.

[113] Patrik Simons, Ilkka Niemelä, and Timo Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1–2):181–234, 2002.

[114] Michael B. Smyth. Topology. In Samson Abramsky, Dov M. Gabbay, and Thomas S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 1, pages 641–761. Clarendon, Oxford, 1994.

[115] Steffen Staab and Rudi Studer, editors. *Handbook on Ontologies*. International Handbooks on Information Systems. Springer, 2004.

[116] Gerd Stumme. Formal concept analysis on its way from mathematics to computer science. In U. Priss, D. Corbett, and G. Angelova (eds.), editors, *Conceptual Structures: Integration and Interfaces, Proc. ICCS 2002*, LNAI, pages 2–19. Springer, 2002.

[117] Geoffrey G. Towell and Jude W. Shavlik. Knowledge-based artificial neural networks. *Artificial Intelligence*, 70(1–2):119–165, 1994.

[118] Maarten H. van Emden. Quantitative deduction and its fixpoint theory. *The Journal of Logic Programming*, 1:37–53, 1986.

[119] Allen van Gelder, Kenneth A. Ross, and John S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.

[120] Steven Vickers. *Topology via Logic.* Cambridge University Press, Cambridge, UK, 1989.

[121] Raphael Volz. *Web Ontology Reasoning with Logic Databases.* PhD thesis, AIFB, University of Karlsruhe, 2004.

[122] Gerd Wagner. Negation in fuzzy and possibilistic logic programs. In Trevor Martin and Francesca Arcelli, editors, *Logic Programming and Soft Computing.* Research Studies Press, 1998.

[123] Matthias Wendt. Unfolding the well-founded semantics. *Journal of Electrical Engineering, Slovak Academy of Sciences*, 53(12/s):56–59, 2002. (Proceedings of the 4th Slovakian Student Conference in Applied Mathematics, Bratislava, April 2002).

[124] Stephen Willard. *General Topology.* Addison-Wesley, Reading, MA, 1970.

[125] Rudolf Wille. Restructuring lattice theory: An approach based on hierarchies of concepts. In Ivan Rival, editor, *Ordered Sets*, pages 445–470. Reidel, Dordrecht-Boston, 1982.

[126] Guo-Qiang Zhang. *Logic of Domains.* Birkhauser, Boston, 1991.

[127] Guo-Qiang Zhang and William Rounds. Reasoning with power defaults. *Theoretical Computer Science*, 323(1–3):321–350, 2004.

[128] Guo-Qiang Zhang and William C. Rounds. Semantics of logic programs and representation of Smyth powerdomains. In Klaus Keimel et al., editors, *Domains and Processes*, pages 151–179. Kluwer, 2001.

Vorgelegte Veröffentlichungen

Theory and Practice of Logic Programming 5(1-2) pp 93-121, 2005, in print.

A uniform approach to logic programming semantics

Pascal Hitzler and Matthias Wendt
Knowledge Representation and Reasoning Group, Artificial Intelligence Institute
Department of Computer Science, Dresden University of Technology
Dresden, Germany
(*e-mail:* {phitzler,mw177754}@inf.tu-dresden.de)

submitted 1 January 2003; revised 1 January 2003; accepted 1 January 2003

Abstract

Part of the theory of logic programming and nonmonotonic reasoning concerns the study of fixed-point semantics for these paradigms. Several different semantics have been proposed during the last two decades, and some have been more successful and acknowledged than others. The rationales behind those various semantics have been manifold, depending on one's point of view, which may be that of a programmer or inspired by commonsense reasoning, and consequently the constructions which lead to these semantics are technically very diverse, and the exact relationships between them have not yet been fully understood. In this paper, we present a conceptually new method, based on level mappings, which allows to provide uniform characterizations of different semantics for logic programs. We will display our approach by giving new and uniform characterizations of some of the major semantics, more particular of the least model semantics for definite programs, of the Fitting semantics, and of the well-founded semantics. A novel characterization of the weakly perfect model semantics will also be provided.

KEYWORDS: Level mapping, Fitting semantics, well-founded semantics, least model semantics, stable semantics, weak stratification

Contents

1 Introduction

Negation in logic programming differs from the negation of classical logic. Indeed, the quest for a satisfactory understanding of negation in logic programming is still inconclusive — although the issue has cooled down a bit recently — and has proved to be very stimulating for research activities in computational logic, and in particular amongst knowledge representation and reasoning researchers concerned with commonsense and nonmonotonic reasoning. During the last two decades, different interpretations of negation in logic programming have lead to the development of a variety of *declarative semantics*, as they are called. Some early research efforts for establishing a satisfactory declarative semantics for negation as failure and its variants, as featured by the resolution-based Prolog family of logic programming systems, have later on been merged with nonmonotonic frameworks for commonsense reasoning, culminating recently in the development of so-called answer set programming systems, like SMODELS or DLV (Eiter et al. 1997; Marek and Truszczyński 1999; Lifschitz 2002; Simons et al. 200x).

Systematically, one can understand Fitting's proposal (Fitting 1985) of a *Kripke-Kleene semantics* — also known as *Fitting semantics* — as a cornerstone which plays a fundamental rôle both for resolution-based and nonmonotonic reasoning inspired logic programming. Indeed, his proposal, which is based on a monotonic semantic operator in Kleene's strong three-valued logic, has been pursued in both communities, for example by Kunen (Kunen 1987) for giving a semantics for pure Prolog, and by Apt and Pedreschi (Apt and Pedreschi 1993) in their fundamental paper on termination analysis of negation as failure, leading to the notion of *acceptable program*. On the other hand, however, Fitting himself (Fitting 1991a; Fitting 2002), using a bilattice-based approach which was further developed by Denecker, Marek, and Truszczynski (Denecker et al. 2000), tied his semantics closely to the major semantics inspired by nonmonotonic reasoning, namely the *stable model semantics* due to Gelfond and Lifschitz (Gelfond and Lifschitz 1988), which is based on a nonmonotonic semantic operator, and the *well-founded semantics* due to van Gelder, Ross, and Schlipf (van Gelder et al. 1991), originally defined using a different monotonic operator in three-valued logic together with a notion of unfoundedness.

Another fundamental idea which was recognised in both communities was that of *stratification*, with the underlying idea of restricting attention to certain kinds of programs in which recursion through negation is prevented. Apt, Blair, and Walker (Apt et al. 1988) proposed a variant of resolution suitable for these programs, while Przymusinski (Przymusinski 1988) and van Gelder (van Gelder 1988) generalized the notion to *local* stratification. Przymusinski (Przymusinski 1988) developed the *perfect model semantics* for locally stratified programs, and together with Przymusinska (Przymusinska and Przymusinski 1990) generalized it later to a three-valued setting as the *weakly perfect model semantics*.

The semantics mentioned so far are defined and characterized using a variety of different techniques and constructions, including monotonic and nonmonotonic semantic operators in two- and three-valued logics, program transformations, level mappings, restrictions to suitable subprograms, detection of cyclic dependencies etc. Relationships between the semantics have been established, but even a simple comparison of the respective models in restricted cases could be rather tedious. So, in this paper, we propose a methodology which allows to obtain uniform characterizations of all semantics previously mentioned, and we believe that it will scale up well to most semantics based on monotonic operators, and also to some nonmonotonic operators, and to extensions of the logic programming paradigm including disjunctive conclusions and uncertainty. The characterizations will allow immediate comparison between the semantics, and once obtained we will easily be able to make some new and interesting observations, including the fact that the well-founded semantics can formally be understood as a Fitting semantics augmented with a form of stratification. Indeed we will note that from this novel perspective the well-founded semantics captures the idea of stratification much better than the weakly perfect model semantics, thus providing a formal explanation for the historic fact that the latter has not received as much attention as the former.

The main tool which will be employed for our characterizations is the notion of *level mapping*. Level mappings are mappings from Herbrand bases to ordinals, i.e. they induce orderings on the set of all ground atoms while disallowing infinite descending chains. They have been a technical tool in a variety of contexts, including termination analysis for resolution-based logic programming as studied by Bezem (Bezem 1989), Apt and Pedreschi (Apt and Pedreschi 1993), Marchiori (Marchiori 1996), Pedreschi, Ruggieri, and Smaus (Pedreschi et al. 2002), and others, where they appear naturally since ordinals are well-orderings. They have been used for defining classes of programs with desirable semantic properties, e.g. by Apt, Blair, and Walker (Apt et al. 1988), Przymusinski (Przymusinski 1988) and Cavedon (Cavedon 1991), and they are intertwined with topological investigations of fixed-point semantics in logic programming, as studied e.g. by Fitting (Fitting 1994; Fitting 2002), and by Hitzler and Seda (Seda 1995; Seda 1997; Hitzler 2001; Hitzler and Seda 2003b). Level mappings are also relevant to some aspects of the study of relationships between logic programming and artificial neural networks, as studied by Hölldobler, Kalinke, and Störr (Hölldobler et al. 1999) and by Hitzler and Seda (Hitzler and Seda 2000; Hitzler and Seda 2003a). In our novel approach to uniform characterizations of different semantics, we will use them as a technical tool for capturing dependencies between atoms in a program.

The paper is structured as follows. Section 2 contains preliminaries which are needed to make the paper relatively self-contained. The subsequent sections contain the announced uniform characterizations of the least model semantics for definite programs and the stable model semantics in Section 3, of the Fitting semantics in Section 4, of the well-founded semantics in Section 5, and of the weakly perfect model semantics in Section 6. Related work will be discussed in Section 7, and we close with conclusions and a discussion of further work in Section 8.

Part of this paper was presented at the 25th German Conference on Artificial Intelligence, KI2002, Aachen, Germany, September 2002 (Hitzler and Wendt 2002).

Acknowledgement. We thank Tony Seda for pointing out some flaws in a previous version of the proof of Theorem 5.2.

2 Preliminaries and Notation

A (*normal*) *logic program* is a finite set of (universally quantified) *clauses* of the form $\forall(A \leftarrow A_1 \wedge \ldots \wedge A_n \wedge \neg B_1 \wedge \ldots \wedge \neg B_m)$, commonly written as $A \leftarrow A_1, \ldots, A_n, \neg B_1, \ldots, \neg B_m$, where A, A_i, and B_j, for $i = 1, \ldots, n$ and $j = 1, \ldots, m$, are atoms over some given first order language. A is called the *head* of the clause, while the remaining atoms make up the *body* of the clause, and depending on context, a body of a clause will be a set of literals (i.e. atoms or negated atoms) or the conjunction of these literals. Care will be taken that this identification does not cause confusion. We allow a body, i.e. a conjunction, to be empty, in which case it always evaluates to true. A clause with empty body is called a *unit clause* or a *fact*. A clause is called *definite*, if it contains no negation symbol. A program is called *definite* if it consists only of definite clauses. We will usually denote atoms with A or B, and literals, which may be atoms or negated atoms, by L or K.

Given a logic program P, we can extract from it the components of a first order language. The corresponding set of ground atoms, i.e. the *Herbrand base* of the program, will be denoted by B_P. For a subset $I \subseteq B_P$, we set $\neg I = \{\neg A \mid A \in I\}$. The set of all ground instances of P with respect to B_P will be denoted by $\mathsf{ground}(P)$. For $I \subseteq B_P \cup \neg B_P$ we say that A is *true with respect to* (or *in*) I if $A \in I$, we say that A is *false with respect to* (or *in*) I if $\neg A \in I$, and if neither is the case, we say that A is *undefined with respect to* (or *in*) I. A (*three-valued* or *partial*) *interpretation* I for P is a subset of $B_P \cup \neg B_P$ which is *consistent*, i.e. whenever $A \in I$ then $\neg A \notin I$. A body, i.e. a conjunction of literals, is true in an interpretation I if every literal in the body is true in I, it is false in I if one of its literals is false in I, and otherwise it is undefined in I. For a negative literal $L = \neg A$ we will find it convenient to write $\neg L \in I$ if $A \in I$ and say that L is false in I etc. in this case. By I_P we denote the set of all (three-valued) interpretations of P. It is a complete partial order (cpo) via set-inclusion, i.e. it contains the empty set as least element, and every ascending chain has a supremum, namely its union. A *model* of P is an interpretation $I \in I_P$ such that for each clause $A \leftarrow \mathtt{body}$ we have that $\mathtt{body}$ true in I implies A true in I, and $\mathtt{body}$ undefined in I implies A true or undefined in I. A *total* interpretation is an interpretation I such that no $A \in B_P$ is undefined in I.

For an interpretation I and a program P, an *I-partial level mapping* for P is a partial mapping $l : B_P \rightarrow \alpha$ with domain $\mathsf{dom}(l) = \{A \mid A \in I \text{ or } \neg A \in I\}$, where α is some (countable) ordinal. We extend every level mapping to literals by setting $l(\neg A) = l(A)$ for all $A \in \mathsf{dom}(l)$. A (*total*) *level mapping* is a total mapping $l : B_P \rightarrow \alpha$ for some (countable) ordinal α.

Given a normal logic program P and some $I \subseteq B_P \cup \neg B_P$, we say that $U \subseteq B_P$ is an *unfounded set* (*of* P) *with respect to* I if each atom $A \in U$ satisfies the following condition: For each clause $A \leftarrow \mathtt{body}$ in $\mathsf{ground}(P)$ (at least) one of the following holds.

(Ui) Some (positive or negative) literal in $\mathtt{body}$ is false in I.
(Uii) Some (non-negated) atom in $\mathtt{body}$ occurs in U.

Given a normal logic program P, we define the following operators on $B_P \cup \neg B_P$.

$T_P(I)$ is the set of all $A \in B_P$ such that there exists a clause $A \leftarrow \mathsf{body}$ in $\mathsf{ground}(P)$ such that body is true in I. $F_P(I)$ is the set of all $A \in B_P$ such that for all clauses $A \leftarrow \mathsf{body}$ in $\mathsf{ground}(P)$ we have that body is false in I. Both T_P and F_P map elements of I_P to elements of I_P. Now define the operator $\Phi_P : I_P \to I_P$ by

$$\Phi_P(I) = T_P(I) \cup \neg F_P(I)\cdot$$

This operator is due to Fitting (Fitting 1985) and is monotonic on the cpo I_P, hence has a least fixed point by the Tarski fixed-point theorem, and we can obtain this fixed point by defining, for each monotonic operator F, that $F \uparrow 0 = \emptyset$, $F \uparrow (\alpha + 1) = F(F \uparrow \alpha)$ for any ordinal α, and $F \uparrow \beta = \bigcup_{\gamma<\beta} F \uparrow \gamma$ for any limit ordinal β, and the least fixed point $\mathsf{lfp}(F)$ of F is obtained as $F \uparrow \alpha$ for some ordinal α. The least fixed point of Φ_P is called the *Kripke-Kleene model* or *Fitting model* of P, determining the *Fitting semantics* of P.

Example 2.1
Let P be the program consisting of the two clauses $p \leftarrow p$ and $q \leftarrow \neg r$. Then $\Phi_P \uparrow 1 = \{\neg r\}$, and $\Phi_P \uparrow 2 = \{q, \neg r\} = \Phi_P \uparrow 3$ is the Fitting model of P.

Now, for $I \subseteq B_P \cup \neg B_P$, let $U_P(I)$ be the greatest unfounded set (of P) with respect to I, which always exists due to van Gelder, Ross, and Schlipf (van Gelder et al. 1991). Finally, define

$$W_P(I) = T_P(I) \cup \neg U_P(I)$$

for all $I \subseteq B_P \cup \neg B_P$. The operator W_P, which operates on the cpo $B_P \cup \neg B_P$, is due to van Gelder et al. (van Gelder et al. 1991) and is monotonic, hence has a least fixed point by the Tarski fixed-point theorem, as above for Φ_P. It turns out that $W_P \uparrow \alpha$ is in I_P for each ordinal α, and so the least fixed point of W_P is also in I_P and is called the *well-founded model* of P, giving the *well-founded semantics* of P.

Example 2.2
Let P be the program consisting of the following clauses.

$$\begin{array}{rcl} s & \leftarrow & q \\ q & \leftarrow & \neg p \\ p & \leftarrow & p \\ r & \leftarrow & \neg r \end{array}$$

Then $\{p\}$ is the largest unfounded set of P with respect to $\emptyset$ and we obtain

$$\begin{array}{rcl} W_P \uparrow 1 & = & \{\neg p\}, \\ W_P \uparrow 2 & = & \{\neg p, q\}, \qquad \text{and} \\ W_P \uparrow 3 & = & \{\neg p, q, s\} \\ & = & W_P \uparrow 4\cdot \end{array}$$

Given a program P, we define the operator T_P^+ on subsets of B_P by $T_P^+(I) = T_P(I \cup \neg(B_P \setminus I))$. It is well-known that for definite programs this operator is monotonic on the set of all subsets of B_P, with respect to subset inclusion. Indeed it is Scott-continuous (Lloyd 1988; Abramsky and Jung 1994; Stoltenberg-Hansen et al. 1994) and, via Kleene's fixed-point theorem, achieves its least fixed point M as the supremum of the iterates $T_P^+ \uparrow n$ for $n \in \mathbb{N}$. So $M = \mathsf{lfp}(T_P^+) = T_P^+ \uparrow \omega$ is *the least two-valued model* of P. In turn, we can identify M with the total interpretation $M \cup \neg(B_P \setminus M)$, which we will call the *definite (partial) model* of P.

Example 2.3
Let P be the program consisting of the clauses

$$\begin{array}{lll} p(0) & \leftarrow & \\ p(s(X)) & \leftarrow & p(X), \end{array}$$

where X denotes a variable and 0 a constant symbol. Write $s^n(0)$ for the term $s(\cdots s(0)\cdots)$ in which the symbol s appears n times. Then

$$T_P^+ \uparrow n = \left\{ p\left(s^k(0)\right) \mid k < n \right\}$$

for all $n \in \mathbb{N}$ and $\{p(s^n(0)) \mid n \in \mathbb{N}\}$ is the least two-valued model of P.

In order to avoid confusion, we will use the following terminology: the notion of *interpretation* will by default denote consistent subsets of $B_P \cup \neg B_P$, i.e. interpretations in three-valued logic. We will sometimes emphasize this point by using the notion *partial interpretation*. By *two-valued interpretations* we mean subsets of B_P. Given a partial interpretation I, we set $I^+ = I \cap B_P$ and $I^- = \{A \in B_P \mid \neg A \in I\}$. Each two-valued interpretation I can be identified with the partial interpretation $I' = I \cup \neg(B_P \setminus I)$. Both, interpretations and two-valued interpretations, are ordered by subset inclusion. We note however, that these two orderings differ: If $I \subseteq B_P$, for example, then I' is always a maximal element in the ordering for partial interpretations, while I is in general not maximal as a two-valued interpretation. The two orderings correspond to the knowledge- and the truth-ordering due to Fitting (Fitting 1991a).

There is a semantics using two-valued logic, the stable model semantics due to Gelfond and Lifschitz (Gelfond and Lifschitz 1988), which is intimately related to the well-founded semantics. Let P be a normal program, and let $M \subseteq B_P$ be a set of atoms. Then we define P/M to be the (ground) program consisting of all clauses $A \leftarrow A_1, \ldots, A_n$ for which there is a clause $A \leftarrow A_1, \ldots, A_n, \neg B_1, \ldots, \neg B_m$ in $\mathsf{ground}(P)$ with $B_1, \ldots, B_m \notin M$. Since P/M does no longer contain negation, it has a least two-valued model $T_{P/M}^+ \uparrow \omega$. For any two-valued interpretation I we can therefore define the operator $\mathrm{GL}_P(I) = T_{P/I}^+ \uparrow \omega$, and call M a *stable model* of the normal program P if it is a fixed point of the operator GL_P, i.e. if $M = \mathrm{GL}_P(M) = T_{P/M}^+ \uparrow \omega$. As it turns out, the operator GL_P is in general not monotonic for normal programs P. However it is *antitonic*, i.e. whenever $I \subseteq J \subseteq B_P$ then $\mathrm{GL}_P(J) \subseteq \mathrm{GL}_P(I)$. As a consequence, the operator GL_P^2, obtained by applying GL_P twice, is monotonic and hence has a least fixed point L_P and a greatest fixed point G_P. Van Gelder (van Gelder 1989) has shown that $\mathrm{GL}_P(L_P) = G_P$, $L_P = \mathrm{GL}_P(G_P)$, and that $L_P \cup \neg(B_P \setminus G_P)$ coincides with the well-founded model of P. This is called the *alternating fixed point characterization* of the well-founded semantics.

Example 2.4
Consider the program P from Example 2.2. The subprogram Q consisting of the first three clauses of the program P has stable model $M = \{s, q\}$, which can be verified by noting that Q/M consists of the clauses

$$\begin{array}{lcl} s & \leftarrow & q \\ q & \leftarrow & \\ p & \leftarrow & p, \end{array}$$

and has M as its least two-valued model.

For the program P we obtain

$$\begin{array}{lcll} \mathrm{GL}_P(\emptyset) & = & \{q, s, r\}, & \\ \mathrm{GL}_P(\{q, s, r\}) & = & \{q, s\} & \\ & = & \mathrm{GL}_P^2(\{q, s\}), & \text{and} \\ \mathrm{GL}_P(B_P) & = & \emptyset. & \end{array}$$

So $L_P = \{q, s\}$ while $G_P = \{q, s, r\}$, and $L_P \cup \neg(B_P \setminus G_P) = \{q, s, \neg p\}$ is the well-founded model of P.

3 Least and Stable Model Semantics

The most fundamental semantics in logic programming is based on the fact mentioned above that the operator T_P^+ has a least fixed point $M = T_P^+ \uparrow \omega$ whenever P is definite. The two-valued interpretation M turns out to be the least two-valued model of the program, and is therefore canonically the model which should be considered for definite programs. Our first result characterizes the least model using level mappings, and serves to convey the main ideas underlying our method. It is a straightforward result but has, to the best of our knowledge, not been noted before.

Theorem 3.1
Let P be a definite program. Then there is a unique two-valued model M of P for which there exists a (total) level mapping $l : B_P \to \alpha$ such that for each atom $A \in M$ there exists a clause $A \leftarrow A_1, \dots, A_n$ in $\mathsf{ground}(P)$ with $A_i \in M$ and $l(A) > l(A_i)$ for all $i = 1, \dots, n$. Furthermore, M is the least two-valued model of P.

Proof
Let M be the least two-valued model $T_P^+ \uparrow \omega$, choose $\alpha = \omega$, and define $l : B_P \to \alpha$ by setting $l(A) = \min\{n \mid A \in T_P^+ \uparrow (n + 1)\}$, if $A \in M$, and by setting $l(A) = 0$, if $A \notin M$. From the fact that $\emptyset \subseteq T_P^+ \uparrow 1 \subseteq \dots \subseteq T_P^+ \uparrow n \subseteq \dots \subseteq T_P^+ \uparrow \omega = \bigcup_m T_P^+ \uparrow m$, for each n, we see that l is well-defined and that the least model $T_P^+ \uparrow \omega$ for P has the desired properties.

Conversely, if M is a two-valued model for P which satisfies the given condition for some mapping $l : B_P \to \alpha$, then it is easy to show, by induction on $l(A)$, that $A \in M$ implies $A \in T_P^+ \uparrow (l(A) + 1)$. This yields that $M \subseteq T_P^+ \uparrow \omega$, and hence that $M = T_P^+ \uparrow \omega$ by minimality of the model $T_P^+ \uparrow \omega$. $\square$

Example 3.2
For the program P from Example 2.3 we obtain $l(p(s^n(0))) = n$ for the level mapping l defined in the proof of Theorem 3.1.

The proof of Theorem 3.1 can serve as a blueprint for obtaining characterizations if the semantics under consideration is based on the least fixed point of a monotonic operator F, and indeed our results for the Fitting semantics and the well-founded semantics, Theorems 4.2 and 5.2, together with their proofs, follow this scheme. In one direction, levels are assigned to atoms A according to the least ordinal α such that A is not undefined in $F\uparrow(\alpha+1)$, and dependencies between atoms of some level and atoms of lower levels are captured by the nature of the considered operator, which will certainly vary from case to case. In Theorem 3.1, the condition thus obtained suffices for uniquely determining the least model, whereas in other cases which we will study later, so for the Fitting semantics and the well-founded semantics, the level mapping conditions will not suffice for unique characterization of the desired model. However, the desired model will in each case turn out to be the greatest among all models satisfying the given conditions. So in these cases it will remain to show, by transfinite induction on the level of some given atom A, that the truth value assigned to A by any model satisfying the given conditions is also assigned to A by $F\uparrow(l(A)+1)$, which at the same time proves that $\mathsf{lfp}(F)$ is the greatest model satisfying the given conditions. For the proof of Theorem 3.1, the proof method just described can be applied straightforwardly, however for more sophisticated operators may become technically challenging on the detailed level.

We now turn to the stable model semantics, which in the case of programs with negation has come to be the major semantics based on two-valued logic. The following characterization is in the spirit of our proposal, and is due to Fages (Fages 1994). It is striking in its similarity to the characterization of the least model for definite programs in Theorem 3.1. For completeness of our exhibition, we include a proof of the fact.

Theorem 3.3
Let P be normal. Then a two-valued model $M \subseteq B_P$ of P is a stable model of P if and only if there exists a (total) level mapping $l : B_P \to \alpha$ such that for each $A \in M$ there exists $A \leftarrow A_1, \dots, A_n, \neg B_1, \dots, \neg B_m$ in $\mathsf{ground}(P)$ with $A_i \in M$, $B_j \notin M$, and $l(A) > l(A_i)$ for all $i = 1, \dots, n$ and $j = 1, \dots, m$.

Proof
Let M be a stable model of P, i.e. $\mathrm{GL}_P(M) = T^+_{P/M}\uparrow\omega = M$. Then M is the least model for P/M, hence is also a model for P, and, by Theorem 3.1, satisfies the required condition with respect to any level mapping l with $l(A) = \min\{n \mid A \in T_{P/M}\uparrow(n+1)\}$ for each $A \in M$. Conversely, let M be a model which satisfies the condition in the statement of the theorem. Then, for every $A \in M$, there is a clause C in $\mathsf{ground}(P)$ of the form $A \leftarrow A_1, \dots, A_n, \neg B_1, \dots, \neg B_k$ such that the body of C is true in M and satisfies $l(A) > l(A_i)$ for all $i = 1, \dots, n$. But then, for every $A \in M$, there is a clause $A \leftarrow A_1, \dots, A_n$ in P/M whose body is true in M and

such that $l(A) > l(A_i)$ for all $i = 1, \dots, n$. By Theorem 3.1, this means that M is the least model for P/M, that is, $M = T^+_{P/M}{\uparrow}\omega = \mathrm{GL}(M)$. $\square$

The proof of Theorem 3.3 just given partly follows the proof scheme discussed previously, by considering the monotonic operator $T^+_{P/M}$, which is used for defining stable models.

Example 3.4
Recall the program P from Example 2.2, and consider the program Q consisting of the first three clauses of P. We already noted in Example 2.4 that Q has stable model $\{s, q\}$. A corresponding level mapping, as defined in the proof of Theorem 3.3, satisfies $l(q) = 0$ and $l(s) = 1$, while $l(p)$ can be an arbitrary value.

4 Fitting Semantics

We next turn to the Fitting semantics. Following the proof scheme which we described in Section 3, we expect levels $l(A)$ to be assigned to atoms A such that $l(A)$ is the least α such that A is not undefined in $\Phi_P\uparrow(\alpha + 1)$. An analysis of the operator Φ_P eventually yields the following conditions.

Definition 4.1
Let P be a normal logic program, I be a model of P, and l be an I-partial level mapping for P. We say that P *satisfies* (F) *with respect to I and l*, if each $A \in \mathsf{dom}(l)$ satisfies one of the following conditions.

(Fi) $A \in I$ and there exists a clause $A \leftarrow L_1, \dots, L_n$ in $\mathsf{ground}(P)$ with $L_i \in I$ and $l(A) > l(L_i)$ for all i.
(Fii) $\neg A \in I$ and for each clause $A \leftarrow L_1, \dots, L_n$ in $\mathsf{ground}(P)$ there exists i with $\neg L_i \in I$ and $l(A) > l(L_i)$.

If $A \in \mathsf{dom}(l)$ satisfies (Fi), then we say that A *satisfies* (Fi) *with respect to I and l*, and similarly if $A \in \mathsf{dom}(l)$ satisfies (Fii).

We note that condition (Fi) is stronger than the condition used for characterizing stable models in Theorem 3.3. The proof of the next theorem closely follows our proof scheme.

Theorem 4.2
Let P be a normal logic program with Fitting model M. Then M is the greatest model among all models I, for which there exists an I-partial level mapping l for P such that P satisfies (F) with respect to I and l.

Proof
Let M_P be the Fitting model of P and define the M_P-partial level mapping l_P as follows: $l_P(A) = \alpha$, where α is the least ordinal such that A is not undefined in $\Phi_P \uparrow (\alpha + 1)$. The proof will be established by showing the following facts: (1) P satisfies (F) with respect to M_P and l_P. (2) If I is a model of P and l an I-partial level mapping such that P satisfies (F) with respect to I and l, then $I \subseteq M_P$.

(1) Let $A \in \mathsf{dom}(l_P)$ and $l_P(A) = \alpha$. We consider two cases.

(Case i) If $A \in M_P$, then $A \in T_P(\Phi_P \uparrow \alpha)$, hence there exists a clause $A \leftarrow \mathtt{body}$ in $\mathsf{ground}(P)$ such that $\mathtt{body}$ is true in $\Phi_P \uparrow \alpha$. Thus, for all $L_i \in \mathtt{body}$ we have that $L_i \in \Phi_P \uparrow \alpha$, and hence $l_P(L_i) < \alpha$ and $L_i \in M_P$ for all i. Consequently, A satisfies (Fi) with respect to M_P and l_P.

(Case ii) If $\neg A \in M_P$, then $A \in F_P(\Phi_P \uparrow \alpha)$, hence for all clauses $A \leftarrow \mathtt{body}$ in $\mathsf{ground}(P)$ there exists $L \in \mathtt{body}$ with $\neg L \in \Phi_P \uparrow \alpha$ and $l_P(L) < \alpha$, hence $\neg L \in M_P$. Consequently, A satisfies (Fii) with respect to M_P and l_P, and we have established that fact (1) holds.

(2) We show via transfinite induction on $\alpha = l(A)$, that whenever $A \in I$ (respectively, $\neg A \in I$), then $A \in \Phi_P \uparrow (\alpha + 1)$ (respectively, $\neg A \in \Phi_P \uparrow (\alpha + 1)$). For the base case, note that if $l(A) = 0$, then $A \in I$ implies that A occurs as the head of a fact in $\mathsf{ground}(P)$, hence $A \in \Phi_P \uparrow 1$, and $\neg A \in I$ implies that there is no clause with head A in $\mathsf{ground}(P)$, hence $\neg A \in \Phi_P \uparrow 1$. So assume now that the induction hypothesis holds for all $B \in B_P$ with $l(B) < \alpha$. We consider two cases.

(Case i) If $A \in I$, then it satisfies (Fi) with respect to I and l. Hence there is a clause $A \leftarrow \mathtt{body}$ in $\mathsf{ground}(P)$ such that $\mathtt{body} \subseteq I$ and $l(K) < \alpha$ for all $K \in \mathtt{body}$. Hence $\mathtt{body} \subseteq M_P$ by induction hypothesis, and since M_P is a model of P we obtain $A \in M_P$.

(Case ii) If $\neg A \in I$, then A satisfies (Fii) with respect to I and l. Hence for all clauses $A \leftarrow \mathtt{body}$ in $\mathsf{ground}(P)$ we have that there is $K \in \mathtt{body}$ with $\neg K \in I$ and $l(K) < \alpha$. Hence for all these K we have $\neg K \in M_P$ by induction hypothesis, and consequently for all clauses $A \leftarrow \mathtt{body}$ in $\mathsf{ground}(P)$ we obtain that $\mathtt{body}$ is false in M_P. Since $M_P = \Phi_P(M_P)$ is a fixed point of the Φ_P-operator, we obtain $\neg A \in M_P$. This establishes fact (2) and concludes the proof. $\square$

Example 4.3
Consider the program P from Example 2.1. Then the level mapping l, as defined in the proof of Theorem 4.2, satsifies $l(r) = 0$ and $l(q) = 1$.

It is interesting to consider the special case where the Fitting model is total. Programs with this property are called Φ-*accessible* (Hitzler and Seda 1999; Hitzler and Seda 2003b), and include e.g. the acceptable programs due to Apt and Pedreschi (Apt and Pedreschi 1993).

Corollary 4.4
A normal logic program P has a total Fitting model if and only if there is a total model I of P and a (total) level mapping l for P such that P satisfies (F) with respect to I and l.

The result follows immediately as a special case of Theorem 4.2, and is closely related to results reported in (Hitzler and Seda 1999; Hitzler and Seda 2003b). The reader familiar with acceptable programs will also note the close relationship between Corollary 4.4 and the defining conditions for acceptable programs. Indeed, the theorem due to Apt and Pedreschi (Apt and Pedreschi 1993), which says that every acceptable program has a total Fitting model, follows without any effort from our result. It also follows immediately, by comparing Corollary 4.4 and Theorem 3.3, that a total Fitting model is always stable, which is a well-known fact.

5 Well-Founded Semantics

The characterization of the well-founded model again closely follows our proof scheme. Before discussing this, though, we will take a short detour which will eventually reveal a surprising fact about the well-founded semantics: From our new perspective the well-founded semantics can be understood as a stratified version of the Fitting semantics.

Let us first recall the definition of a (locally) stratified program, due to Apt, Blair, Walker, and Przymusinski (Apt et al. 1988; Przymusinski 1988): A normal logic program is called *locally stratified* if there exists a (total) level mapping $l : B_P \to \alpha$, for some ordinal α, such that for each clause $A \leftarrow A_1, \dots, A_n, \neg B_1, \dots, \neg B_m$ in $\mathsf{ground}(P)$ we have that $l(A) \geq l(A_i)$ and $l(A) > l(B_j)$ for all $i = 1, \dots, n$ and $j = 1, \dots, m$.

The notion of (locally) stratifed program, as already mentioned in the introduction, was developed with the idea of *preventing recursion through negation*, while allowing recursion through positive dependencies. There exist locally stratified programs which do not have a total Fitting model and vice versa. Indeed, the program consisting of the single clause $p \leftarrow p$ is locally stratified but p remains undefined in the Fitting model. Conversely, the program consisting of the two clauses $q \leftarrow$ and $q \leftarrow \neg q$ is not locally stratified but its Fitting model assigns to q the truth value *true*.

By comparing Definition 4.1 with the definition of locally stratified programs, we notice that condition (Fii) requires a strict decrease of level between the head and a literal in the rule, independent of this literal being positive or negative. But, on the other hand, condition (Fii) imposes no further restrictions on the remaining body literals, while the notion of local stratification does. These considerations motivate the substitution of condition (Fii) by the condition (WFii), as given in the following definition.

Definition 5.1
Let P be a normal logic program, I be a model of P, and l be an I-partial level mapping for P. We say that *P satisfies* (WF) *with respect to I and l*, if each $A \in \mathsf{dom}(l)$ satisfies one of the following conditions.

(WFi) $A \in I$ and there exists a clause $A \leftarrow L_1, \dots, L_n$ in $\mathsf{ground}(P)$ with $L_i \in I$ and $l(A) > l(L_i)$ for all i.

(WFii) $\neg A \in I$ and for each clause $A \leftarrow A_1, \dots, A_n, \neg B_1, \dots, \neg B_m$ in $\mathsf{ground}(P)$ (at least) one of the following conditions holds:

(WFiia) There exists $i \in \{1, \dots, n\}$ with $\neg A_i \in I$ and $l(A) \geq l(A_i)$.
(WFiib) There exists $j \in \{1, \dots, m\}$ with $B_j \in I$ and $l(A) > l(B_j)$.

If $A \in \mathsf{dom}(l)$ satisfies (WFi), then we say that *A satisfies* (WFi) *with respect to I and l*, and similarly if $A \in \mathsf{dom}(l)$ satisfies (WFii).

We note that conditions (Fi) and (WFi) are identical. Indeed, replacing (WFi) by a stratified version such as the following seems not satisfactory.

(SFi) $A \in I$ and there exists a clause $A \leftarrow A_1, \dots, A_n, \neg B_1, \dots, \neg B_m$ in $\mathsf{ground}(P)$ with $A_i, B_j \in I$, $l(A) \geq l(A_i)$, and $l(A) > l(B_j)$ for all i and j.

If we replace condition (WFi) by condition (SFi), then it is not guaranteed that for any given program there is a greatest model satisfying the desired properties: Consider the program consisting of the two clauses $p \leftarrow p$ and $q \leftarrow \neg p$, and the two (total) models $\{p, \neg q\}$ and $\{\neg p, q\}$, which are incomparable, and the level mapping l with $l(p) = 0$ and $l(q) = 1$. A detailed analysis of condition (SFi) in the context of our approach can be found in (Hitzler 2003).

So, in the light of Theorem 4.2, Definition 5.1 should provide a natural "stratified version" of the Fitting semantics. And indeed it does, and furthermore, the resulting semantics coincides with the well-founded semantics, which is a very satisfactory result. The proof of the fact again follows our proof scheme, but is slightly more involved due to the necessary treatment of unfounded sets.

Theorem 5.2
Let P be a normal logic program with well-founded model M. Then M is the greatest model among all models I, for which there exists an I-partial level mapping l for P such that P satisfies (WF) with respect to I and l.

Proof
Let M_P be the well-founded model of P and define the M_P-partial level mapping l_P as follows: $l_P(A) = \alpha$, where α is the least ordinal such that A is not undefined in $W_P \uparrow (\alpha+1)$. The proof will be established by showing the following facts: (1) P satisfies (WF) with respect to M_P and l_P. (2) If I is a model of P and l an I-partial level mapping such that P satisfies (WF) with respect to I and l, then $I \subseteq M_P$.

(1) Let $A \in \mathsf{dom}(l_P)$ and $l_P(A) = \alpha$. We consider two cases.

(Case i) If $A \in M_P$, then $A \in T_P(W_P \uparrow \alpha)$, hence there exists a clause $A \leftarrow \mathtt{body}$ in $\mathsf{ground}(P)$ such that $\mathtt{body}$ is true in $W_P \uparrow \alpha$. Thus, for all $L_i \in \mathtt{body}$ we have that $L_i \in W_P \uparrow \alpha$. Hence, $l_P(L_i) < \alpha$ and $L_i \in M_P$ for all i. Consequently, A satisfies (WFi) with respect to M_P and l_P.

(Case ii) If $\neg A \in M_P$, then $A \in U_P(W_P \uparrow \alpha)$, i.e. A is contained in the greatest unfounded set of P with respect to $W_P \uparrow \alpha$. Hence for each clause $A \leftarrow \mathtt{body}$ in $\mathsf{ground}(P)$, at least one of (Ui) or (Uii) holds for this clause with respect to $W_P \uparrow \alpha$ and the unfounded set $U_P(W_P \uparrow \alpha)$. If (Ui) holds, then there exists some literal $L \in \mathtt{body}$ with $\neg L \in W_P \uparrow \alpha$. Hence $l_P(L) < \alpha$ and condition (WFiib) holds relative to M_P and l_P if L is an atom, or condition (WFiia) holds relative to M_P and l_P if L is a negated atom. On the other hand, if (Uii) holds, then some (non-negated) atom B in $\mathtt{body}$ occurs in $U_P(W_P \uparrow \alpha)$. Hence $l_P(B) \leq l_P(A)$ and A satisfies (WFiia) with respect to M_P and l_P. Thus we have established that fact (1) holds.

(2) We show via transfinite induction on $\alpha = l(A)$, that whenever $A \in I$ (respectively, $\neg A \in I$), then $A \in W_P \uparrow (\alpha + 1)$ (respectively, $\neg A \in W_P \uparrow (\alpha + 1)$). For the base case, note that if $l(A) = 0$, then $A \in I$ implies that A occurs as the head

of a fact in $\mathsf{ground}(P)$. Hence, $A \in W_P \uparrow 1$. If $\neg A \in I$, then consider the set U of all atoms B with $l(B) = 0$ and $\neg B \in I$. We show that U is an unfounded set of P with respect to $W_P \uparrow 0$, and this suffices since it implies $\neg A \in W_P \uparrow 1$ by the fact that $A \in U$. So let $C \in U$ and let $C \leftarrow \mathtt{body}$ be a clause in $\mathsf{ground}(P)$. Since $\neg C \in I$, and $l(C) = 0$, we have that C satisfies (WFiia) with respect to I and l, and so condition (Uii) is satisfied showing that U is an unfounded set of P with respect to I. Assume now that the induction hypothesis holds for all $B \in B_P$ with $l(B) < \alpha$. We consider two cases.

(Case i) If $A \in I$, then it satisfies (WFi) with respect to I and l. Hence there is a clause $A \leftarrow \mathtt{body}$ in $\mathsf{ground}(P)$ such that $\mathtt{body} \subseteq I$ and $l(K) < \alpha$ for all $K \in \mathtt{body}$. Hence $\mathtt{body} \subseteq W_P \uparrow \alpha$, and we obtain $A \in T_P(W_P \uparrow \alpha)$ as required.

(Case ii) If $\neg A \in I$, consider the set U of all atoms B with $l(B) = \alpha$ and $\neg B \in I$. We show that U is an unfounded set of P with respect to $W_P \uparrow \alpha$, and this suffices since it implies $\neg A \in W_P \uparrow (\alpha + 1)$ by the fact that $A \in U$. So let $C \in U$ and let $C \leftarrow \mathtt{body}$ be a clause in $\mathsf{ground}(P)$. Since $\neg C \in I$, we have that C satisfies (WFii) with respect to I and l. If there is a literal $L \in \mathtt{body}$ with $\neg L \in I$ and $l(L) < l(C)$, then by the induction hypothesis we obtain $\neg L \in W_P \uparrow \alpha$, so condition (Ui) is satisfied for the clause $C \leftarrow \mathtt{body}$ with respect to $W_P \uparrow \alpha$ and U. In the remaining case we have that C satisfies condition (WFiia), and there exists an atom $B \in \mathtt{body}$ with $\neg B \in I$ and $l(B) = l(C)$. Hence, $B \in U$ showing that condition (Uii) is satisfied for the clause $C \leftarrow \mathtt{body}$ with respect to $W_P \uparrow \alpha$ and U. Hence U is an unfounded set of P with respect to $W_P \uparrow \alpha$. $\square$

Example 5.3
Consider the program P from Example 2.2. With notation from the proof of Theorem 5.2, we obtain $l(p) = 0$, $l(q) = 1$, and $l(s) = 2$.

As a special case, we consider programs with total well-founded model. The following corollary follows without effort from Theorem 5.2.

Corollary 5.4
A normal logic program P has a total well-founded model if and only if there is a total model I of P and a (total) level mapping l such that P satisfies (WF) with respect to I and l.

As a further example for the application of our proof scheme, we use Theorem 5.2 in order to prove a result by van Gelder (van Gelder 1989) which we mentioned in the introduction, concerning the alternating fixed-point characterization of the well-founded semantics. Let us first introduce some temporary notation, where P is an arbitrary program.

$$\begin{array}{lcll}
L_0 & = & \emptyset & \\
G_0 & = & B_P & \\
L_{\alpha+1} & = & \mathrm{GL}_P(G_\alpha) & \text{for any ordinal } \alpha \\
G_{\alpha+1} & = & \mathrm{GL}_P(L_\alpha) & \text{for any ordinal } \alpha \\
L_\alpha & = & \bigcup_{\beta<\alpha} L_\beta & \text{for limit ordinal } \alpha \\
G_\alpha & = & \bigcap_{\beta<\alpha} G_\beta & \text{for limit ordinal } \alpha
\end{array}$$

Since $\emptyset \subseteq B_P$, we obtain $L_0 \subseteq L_1 \subseteq G_1 \subseteq G_0$ and, by transfinite induction, it can easily be shown that $L_\alpha \subseteq L_\beta \subseteq G_\beta \subseteq G_\alpha$ whenever $\alpha \leq \beta$. In order to apply our proof scheme, we need to detect a monotonic operator, or at least some kind of monotonic construction, underlying the alternative fixed-point characterization. The assignment $(L_\alpha, G_\alpha) \mapsto (L_{\alpha+1}, G_{\alpha+1})$, using the temporary notation introduced above, will serve for this purpose. The proof of the following theorem is based on it and our general proof scheme, with modifications where necessary, for example for accomodating the fact that $G_{\alpha+1}$ is not defined using G_α, but rather L_α, and that we work with the complements $B_P \setminus G_\alpha$ instead of the sets G_α.

Theorem 5.5
Let P be a normal program. Then $M = L_P \cup \neg(B_P \setminus G_P)$ is the well-founded model of P.

Proof
First, we define an M-partial level mapping l. For convenience, we will take as image set of l, pairs (α, n) of ordinals, where $n \leq \omega$, with the lexicographic ordering. This can be done without loss of generality because any set of pairs of ordinals, lexicographically ordered, is certainly well-ordered and therefore order-isomorphic to an ordinal. For $A \in L_P$, let $l(A)$ be the pair (α, n), where α is the least ordinal such that $A \in L_{\alpha+1}$, and n is the least ordinal such that $A \in T_{P/G_\alpha}\uparrow(n+1)$. For $B \notin G_P$, let $l(B)$ be the pair (β, ω), where β is the least ordinal such that $B \notin G_{\beta+1}$. We show next by transfinite induction that P satisfies (WF) with respect to M and l.

Let $A \in L_1 = T_{P/B_P} \uparrow \omega$. Since P/B_P consists of exactly all clauses from $\mathsf{ground}(P)$ which contain no negation, we have that A is contained in the least two-valued model for a definite subprogram of P, namely P/B_P, and (WFi) is satisfied by Theorem 3.1. Now let $\neg B \in \neg(B_P \setminus G_P)$ be such that $B \in (B_P \setminus G_1) = B_P \setminus T_{P/\emptyset}\uparrow\omega$. Since $P/\emptyset$ contains all clauses from $\mathsf{ground}(P)$ with all negative literals removed, we obtain that each clause in $\mathsf{ground}(P)$ with head B must contain a positive body literal $C \notin G_1$, which, by definition of l, must have the same level as B, hence (WFiia) is satisfied.

Assume now that, for some ordinal α, we have shown that A satisfies (WF) with respect to M and l for all $n \leq \omega$ and all $A \in B_P$ with $l(A) \leq (\alpha, n)$.

Let $A \in L_{\alpha+1} \setminus L_\alpha = T_{P/G_\alpha} \uparrow \omega \setminus L_\alpha$. Then $A \in T_{P/G_\alpha} \uparrow n \setminus L_\alpha$ for some $n \in \mathbb{N}$; note that all (negative) literals which were removed by the Gelfond-Lifschitz transformation from clauses with head A have level less than $(\alpha, 0)$. Then the assertion that A satisfies (WF) with respect to M and l follows again by Theorem 3.1.

Let $A \in (B_P \setminus G_{\alpha+1}) \cap G_\alpha$. Then $A \notin T_{P/L_\alpha} \uparrow \omega$. Now for any clause $A \leftarrow A_1, \ldots, A_k, \neg B_1, \ldots, \neg B_m$ in $\mathsf{ground}(P)$, if $B_j \in L_\alpha$ for some j, then $l(A) > l(B_j)$. Otherwise, since $A \notin T_{P/L_\alpha}\uparrow\omega$, we have that there exists A_i with $A_i \notin T_{P/L_\alpha}\uparrow\omega$, and hence $l(A) \geq l(A_i)$, and this suffices.

This finishes the proof that P satisfies (WF) with respect to M and l. It therefore only remains to show that M is greatest with this property.

So assume that $M_1 \neq M$ is the greatest model such that P satisfies (WF) with respect to M_1 and some M_1-partial level mapping l_1.

Assume $L \in M_1 \setminus M$ and, without loss of generality, let the literal L be chosen such that $l_1(L)$ is minimal. We consider the following two cases.

(Case i) If $L = A$ is an atom, then there exists a clause $A \leftarrow \mathtt{body}$ in $\mathsf{ground}(P)$ such that $l_1(L) < l_1(A)$ for all literals L in $\mathtt{body}$, and such that $\mathtt{body}$ is true in M_1. Hence, $\mathtt{body}$ is true in M and $A \leftarrow \mathtt{body}$ transforms to a clause $A \leftarrow A_1, \ldots, A_n$ in P/G_P with $A_1, \ldots, A_n \in L_P = T_{P/G_P} \uparrow \omega$. But this implies $A \in M$, contradicting $A \in M_1 \setminus M$.

(Case ii) If $L = \neg A \in M_1 \setminus M$ is a negated atom, then $\neg A \in M_1$ and $A \in G_P = T_{P/L_P} \uparrow \omega$, so $A \in T_{P/L_P} \uparrow n$ for some $n \in \mathbb{N}$. We show by induction on n that this leads to a contradiction, to finish the proof.

If $A \in T_{P/L_P} \uparrow 1$, then there is a unit clause $A \leftarrow$ in P/L_P, and any corresponding clause $A \leftarrow \neg B_1, \ldots, \neg B_k$ in $\mathsf{ground}(P)$ satisfies $B_1, \ldots, B_k \notin L_P$. Since $\neg A \in M_1$, we also obtain by Theorem 5.2 that there is $i \in \{1, \ldots, k\}$ such that $B_i \in M_1$ and $l_1(B_i) < l_1(A)$. By minimality of $l_1(A)$, we obtain $B_i \in M$, and hence $B_i \in L_P$, which contradicts $B_i \notin L_P$.

Now assume that there is no $\neg B \in M_1 \setminus M$ with $B \in T_{P/L_P} \uparrow k$ for any $k < n + 1$, and let $\neg A \in M_1 \setminus M$ with $A \in T_{P/L_P} \uparrow (n + 1)$. Then there is a clause $A \leftarrow A_1, \ldots, A_m$ in P/L_P with $A_1, \ldots, A_m \in T_{P/L_P} \uparrow n \subseteq G_P$, and we note that we cannot have $\neg A_i \in M_1 \setminus M$ for any $i \in \{1, \ldots, m\}$, by our current induction hypothesis. Furthermore, it is also impossible for $\neg A_i$ to belong to M for any i, otherwise we would have $A_i \in B_P \setminus G_P$. Thus, we conclude that we cannot have $\neg A_i \in M_1$ for any i. Moreover, there is a corresponding clause $A \leftarrow A_1, \ldots, A_m, \neg B_1, \ldots, \neg B_{m_1}$ in $\mathsf{ground}(P)$ with $B_1, \ldots, B_{m_1} \notin L_P$. Hence, by Theorem 5.2, we know that there is $i \in \{1, \ldots, m_1\}$ such that $B_i \in M_1$ and $l_1(B_i) < l_1(A)$. By minimality of $l_1(A)$, we conclude that $B_i \in M$, so that $B_i \in L_P$, and this contradicts $B_i \notin L_P$. □

Example 5.6
Consider again the program P from Examples 2.2, 2.4, and 5.3. With notation from the proof of Theorem 5.5 we get $l(q) = (1, 0)$, $l(s) = (1, 1)$, and $l(p) = (0, \omega)$.

6 Weakly Perfect Model Semantics

By applying our proof scheme, we have obtained new and uniform characterizations of the Fitting semantics and the well-founded semantics, and argued that the well-founded semantics is a stratified version of the Fitting semantics. Our argumentation is based on the key intuition underlying the notion of stratification, that recursion should be allowed through positive dependencies, but be forbidden through negative dependencies. As we have seen in Theorem 5.2, the well-founded semantics provides this for a setting in three-valued logic. Historically, a different semantics, given by the so-called weakly perfect model associated with each program, was proposed by Przymusinska and Przymusinski (Przymusinska and Przymusinski 1990) in order to carry over the intuition underlying the notion of stratification to a three-valued setting. In the following, we will characterize weakly perfect models via level mappings, in the spirit of our approach. We will thus have obtained uniform characterizations of the Fitting semantics, the well-founded semantics, and the weakly perfect model semantics, which makes it possible to easily compare them.

Definition 6.1
Let P be a normal logic program, I be a model of P and l be an I-partial level mapping for P. We say that P *satisfies* (WS) *with respect to* I *and* l, if each $A \in \mathsf{dom}(l)$ satisfies one of the following conditions.

(WSi) $A \in I$ and there exists a clause $A \leftarrow L_1, \dots, L_n \in \mathsf{ground}(P)$ such that $L_i \in I$ and $l(A) > l(L_i)$ for all $i = 1, \dots, n$.

(WSii) $\neg A \in I$ and for each clause $A \leftarrow A_1, \dots, A_n, \neg B_1, \dots, \neg B_m \in \mathsf{ground}(P)$ (at least) one of the following three conditions holds.

- (WSiia) There exists i such that $\neg A_i \in I$ and $l(A) > l(A_i)$.
- (WSiib) For all k we have $l(A) \geq l(A_k)$, for all j we have $l(A) > l(B_j)$, and there exists i with $\neg A_i \in I$.
- (WSiic) There exists j such that $B_j \in I$ and $l(A) > l(B_j)$.

We observe that the condition (WSii) in the above theorem is more general than (Fii), and more restrictive than (WFii).

We will see below in Theorem 6.4, that Definition 6.1 captures the weakly perfect model, in the same way in which Definitions 4.1 and 5.1 capture the Fitting model, respectively the well-founded model.

In order to proceed with this, we first need to recall the definition of weakly perfect models due to Przymusinska and Przymusinski (Przymusinska and Przymusinski 1990), and we will do this next. For ease of notation, it will be convenient to consider (countably infinite) propositional programs instead of programs over a first-order language. This is both common practice and no restriction, because the ground instantiation $\mathsf{ground}(P)$ of a given program P can be understood as a propositional program which may consist of a countably infinite number of clauses. Let us remark that our definition below differs slightly from the original one, and we will return to this point later. It nevertheless leads to exactly the same notion of weakly stratified program.

Let P be a (countably infinite propositional) normal logic program. An atom $A \in B_P$ *refers to* an atom $B \in B_P$ if B or $\neg B$ occurs as a body literal in a clause $A \leftarrow \mathsf{body}$ in P. A *refers negatively to* B if $\neg B$ occurs as a body literal in such a clause. We say that A *depends on* B if the pair (A, B) is in the transitive closure of the relation *refers to*, and we write this as $B \leq A$. We say that A *depends negatively on* B if there are $C, D \in B_P$ such that C refers negatively to D and the following hold: (1) $C \leq A$ or $C = A$ (the latter meaning identity). (2) $B \leq D$ or $B = D$. We write $B < A$ in this case. For $A, B \in B_P$, we write $A \sim B$ if either $A = B$, or A and B depend negatively on each other, i.e. if $A < B$ and $B < A$ both hold. The relation $\sim$ is an equivalence relation and its equivalence classes are called *components* of P. A component is *trivial* if it consists of a single element A with $A \not< A$.

Let C_1 and C_2 be two components of a program P. We write $C_1 \prec C_2$ if and only if $C_1 \neq C_2$ and for all $A_1 \in C_1$ there is $A_2 \in C_2$ with $A_1 < A_2$. A component C_1 is called *minimal* if there is no component C_2 with $C_2 \prec C_1$.

Given a normal logic program P, the *bottom stratum* $S(P)$ of P is the union of all minimal components of P. The *bottom layer* of P is the subprogram $L(P)$ of P which consists of all clauses from P with heads belonging to $S(P)$.

Given a (partial) interpretation I of P, we define the *reduct of P with respect to I* as the program P/I obtained from P by performing the following reductions. (1) Remove from P all clauses which contain a body literal L such that $\neg L \in I$ or whose head belongs to I. (2) Remove from all remaining clauses all body literals L with $L \in I$. (3) Remove from the resulting program all non-unit clauses, whose heads appear also as unit clauses in the program.

Definition 6.2
The *weakly perfect model* M_P of a program P is defined by transfinite induction as follows. Let $P_0 = P$ and $M_0 = \emptyset$. For each (countable) ordinal $\alpha > 0$ such that programs P_δ and partial interpretations M_δ have already been defined for all $\delta < \alpha$, let

$$\begin{array}{lcl} N_\alpha & = & \bigcup_{0<\delta<\alpha} M_\delta, \\ P_\alpha & = & P/N_\alpha, \\ R_\alpha & \text{is} & \text{the set of all atoms which are undefined in } N_\alpha \\ & & \text{and were eliminated from } P \text{ by reducing it with respect to } N_\alpha, \\ S_\alpha & = & S(P_\alpha), \text{ and} \\ L_\alpha & = & L(P_\alpha) \cdot \end{array}$$

The construction then proceeds with one of the following three cases. (1) If P_α is empty, then the construction stops and $M_P = N_\alpha \cup \neg R_\alpha$ is the (*total*) *weakly perfect model* of P. (2) If the bottom stratum S_α is empty or if the bottom layer L_α contains a negative literal, then the construction also stops and $M_P = N_\alpha \cup \neg R_\alpha$ is the (*partial*) *weakly perfect model* of P. (3) In the remaining case L_α is a definite program, and we define $M_\alpha = H \cup \neg R_\alpha$, where H is the definite (partial) model of L_α, and the construction continues.

For every α, the set $S_\alpha \cup R_\alpha$ is called the *α-th stratum* of P and the program L_α is called the *α-th layer* of P.

A *weakly stratified program* is a program with a total weakly perfect model. The set of its strata is then called its *weak stratification*.

Example 6.3
Consider the program P which consists of the following six clauses.

$$\begin{array}{lcl} a & \leftarrow & \neg b \\ b & \leftarrow & c, \neg a \\ b & \leftarrow & c, \neg d \\ c & \leftarrow & b, \neg e \\ d & \leftarrow & e \\ e & \leftarrow & d \end{array}$$

Then $N_1 = M_1 = \{\neg d, \neg e\}$ and P/N_1 consists of the clauses

$$\begin{array}{rcl} a & \leftarrow & \neg b \\ b & \leftarrow & c, \neg a \\ b & \leftarrow & c \\ c & \leftarrow & b. \end{array}$$

Its least component is $\{a, b, c\}$. The corresponding bottom layer, which is all of P/N_1, contains a negative literal, so the construction stops and $M_2 = N_1 = \{\neg d, \neg e\}$ is the (partial) weakly perfect model of P.

Let us return to the remark made earlier that our definition of weakly perfect model, as given in Definition 6.2, differs slightly from the version introduced by Przymusinska and Przymusinski (Przymusinska and Przymusinski 1990). In order to obtain the original definition, points (2) and (3) of Definition 6.2 have to be replaced as follows: (2) If the bottom stratum S_α is empty or if the bottom layer L_α *has no least two-valued model*, then the construction stops and $M_P = N_\alpha \cup \neg R_\alpha$ is the (partial) weakly perfect model of P. (3) In the remaining case L_α *has a least two-valued model*, and we define $M_\alpha = H \cup \neg R_\alpha$, where H is the partial model of L_α corresponding to its least two-valued model, and the construction continues.

The original definition is more general due to the fact that every definite program has a least two-valued model. However, while the least two-valued model of a definite program can be obtained as the least fixed point of the monotonic (and even Scott-continuous) operator T_P^+, we know of no similar result, or general operator, for obtaining the least two-valued model, if existent, of progams which are not definite. The original definition therefore seems to be rather awkward, and indeed, for the definition of weakly stratified programs (Przymusinska and Przymusinski 1990), the more general version was dropped in favour of requiring definite layers. So Definition 6.2 is an adaptation taking the original notion of weakly stratified program into account, and appears to be more natural. In the following, the notion of *weakly perfect model* will refer to Definition 6.2.

To be pedantic, there is another difference, namely that we have made explicit the sets R_α of Definition 6.2, which were only implicitly treated in the original definition. The result is the same.

We show next that Definition 6.1 indeed captures the weakly perfect model. The proof basically follows our proof scheme, with some alterations, and the monotonic construction which defines the weakly perfect model serves in place of a monotonic operator. The technical details of the proof are very involved.

Theorem 6.4
Let P be a normal logic program with weakly perfect model M_P. Then M_P is the greatest model among all models I, for which there exists an I-partial level mapping l for P such that P satisfies (WS) with respect to I and l.

We prepare the proof of Theorem 6.4 by introducing some notation, which will make the presentation much more transparent. As for the proof of Theorem 5.5, we will consider level mappings which map into pairs (β, n) of ordinals, where $n \leq \omega$.

Let P be a normal logic program with (partial) weakly perfect model M_P. Then define the M_P-partial level mapping l_P as follows: $l_P(A) = (\beta, n)$, where $A \in S_\beta \cup R_\beta$ and n is least with $A \in T^+_{L_\beta} \uparrow (n+1)$, if such an n exists, and $n = \omega$ otherwise. We observe that if $l_P(A) = l_P(B)$ then there exists α with $A, B \in S_\alpha \cup R_\alpha$, and if $A \in S_\alpha \cup R_\alpha$ and $B \in S_\beta \cup R_\beta$ with $\alpha < \beta$, then $l(A) < l(B)$.

The following definition is again technical and will help to ease notation and arguments.

Definition 6.5
Let P and Q be two programs and let I be an interpretation.

1. If $C_1 = (A \leftarrow L_1, \dots, L_m)$ and $C_2 = (B \leftarrow K_1, \dots, K_n)$ are two clauses, then we say that C_1 *subsumes* C_2, written $C_1 \preccurlyeq C_2$, if $A = B$ and $\{L_1, \dots, L_m\} \subseteq \{K_1, \dots, K_n\}$.
2. We say that P *subsumes* Q, written $P \preccurlyeq Q$, if for each clause C_1 in P there exists a clause C_2 in Q with $C_1 \preccurlyeq C_2$.
3. We say that P *subsumes* Q *model-consistently* (*with respect to* I), written $P \preccurlyeq_I Q$, if the following conditions hold. (i) For each clause $C_1 = (A \leftarrow L_1, \dots, L_m)$ in P there exists a clause $C_2 = (B \leftarrow K_1, \dots, K_n)$ in Q with $C_1 \preccurlyeq C_2$ and $(\{K_1, \dots, K_n\} \setminus \{L_1, \dots, L_m\}) \subseteq I$. (ii) For each clause $C_2 = (B \leftarrow K_1, \dots, K_n)$ in Q with $\{K_1, \dots, K_n\} \in I$ and $B \notin I$ there exists a clause C_1 in P such that $C_1 \preccurlyeq C_2$.

A clause C_1 subsumes a clause C_2 if both have the same head and the body of C_2 contains at least the body literals of C_1, e.g. $p \leftarrow q$ subsumes $p \leftarrow q, \neg r$. A program P subsumes a program Q if every clause in P can be generated this way from a clause in Q, e.g. the program consisting of the two clauses $p \leftarrow q$ and $p \leftarrow r$ subsumes the program consisting of $p \leftarrow q, \neg s$ and $p \leftarrow r, p$. This is also an example of a model-consistent subsumption with respect to the interpretation $\{\neg s, p\}$. Concerning Example 6.3, note that $P/N_1 \preccurlyeq_{N_1} P$, which is no coincidence. Indeed, Definition 6.5 facilitates the proof of Theorem 6.4 by employing the following lemma.

Lemma 1
With notation from Definiton 6.2, we have $P/N_\alpha \preccurlyeq_{N_\alpha} P$ for all α.

Proof
Condition 3(i) of Definition 6.5 holds because every clause in P/N_α is obtained from a clause in P by deleting body literals which are contained in N_α. Condition 3(ii) holds because for each clause in P with head $A \notin N_\alpha$ whose body is true under N_α, we have that $A \leftarrow$ is a fact in P/N_α. $\square$

The next lemma establishes the induction step in part (2) of the proof of Theorem 6.4.

Lemma 2
If I is a non-empty model of a (infinite propositional normal) logic program P' and l an I-partial level mapping such that P' satisfies (WS) with respect to I and l, then the following hold for $P = P'/\emptyset$.

1. The bottom stratum $S(P)$ of P is non-empty and consists of trivial components only.
2. The bottom layer $L(P)$ of P is definite.
3. The definite (partial) model N of $L(P)$ is consistent with I in the following sense: we have $I' \subseteq N$, where I' is the restriction of I to all atoms which are not undefined in N.
4. P/N satisfies (WS) with respect to $I \setminus N$ and l/N, where l/N is the restriction of l to the atoms in $I \setminus N$.

Proof
(a) Assume there exists some component $C \subseteq S(P)$ which is not trivial. Then there must exist atoms $A, B \in C$ with $A < B$, $B < A$, and $A \neq B$. Without loss of generality, we can assume that A is chosen such that $l(A)$ is minimal. Now let A' be any atom occuring in a clause with head A. Then $A > B > A \geq A'$, hence $A > A'$, and by minimality of the component we must also have $A' > A$, and we obtain that all atoms occuring in clauses with head A must be contained in C. We consider two cases.

(Case i) If $A \in I$, then there must be a fact $A \leftarrow$ in P, since otherwise by (WSi) we had a clause $A \leftarrow L_1, \dots, L_n$ (for some $n \geq 1$) with $L_1, \dots, L_n \in I$ and $l(A) > l(L_i)$ for all i, contradicting the minimality of $l(A)$. Since $P = P'/\emptyset$ we obtain that $A \leftarrow$ is the only clause in P with head A, contradicting the existence of $B \neq A$ with $B < A$.

(Case ii) If $\neg A \in I$, and since A was chosen minimal with respect to l, we obtain that condition (WSiib) must hold for each clause $A \leftarrow A_1, \dots, A_n, \neg B_1, \dots, \neg B_m$ with respect to I and l, and that $m = 0$. Furthermore, all A_i must be contained in C, as already noted above, and $l(A) \geq l(A_i)$ for all i by (WSiib). Also from (Case i) we obtain that no A_i can be contained in I. We have now established that for all A_i in the body of any clause with head A, we have $l(A) = l(A_i)$ and $\neg A_i \in I$. The same argument holds for all clauses with head A_i, for all i, and the argument repeats. Now from $A > B$ we obtain that there are $D, E \in C$ with $A \geq E$ (or $A = E$), $D \geq B$ (or $D = B$), and E refers negatively to D. As we have just seen, we obtain $\neg E \in I$ and $l(E) = l(A)$. Since E refers negatively to D, there is a clause with head E and $\neg D$ contained in the body of this clause. Since (WSii) holds for this clause, there must be a literal L in the body with level less than $l(E)$, hence $l(L) < l(A)$ and $L \in C$ which is a contradiction. We thus have established that all components are trivial.

We show next that the bottom stratum is non-empty. Indeed, let A be an atom such that $l(A)$ is minimal. We will show that $\{A\}$ is a component. So assume it is not, i.e. that there is B with $B < A$. Then there exist $D_1, \dots, D_k$, for some $k \in \mathbb{N}$, such that $D_1 = A$, D_j refers to D_{j+1} for all $j = 1, \dots, k-1$, and D_k refers negatively to some B' with $B' \geq B$ (or $B' = B$).

We show next by induction that for all $j = 1, \ldots, k$ the following statements hold: $\neg D_j \in I$, $B < D_j$, and $l(D_j) = l(A)$. Indeed note that for $j = 1$, i.e. $D_j = A$, we have that $B < D_j = A$ and $l(D_j) = l(A)$. Assuming $A \in I$, we obtain by minimality of $l(A)$ that $A \leftarrow$ is the only clause in $P = P'/\emptyset$ with head A, contradicting the existence of $B < A$. So $\neg A \in I$, and the assertion holds for $j = 1$. Now assume the assertion holds some $j < k$. Then obviously $D_{j+1} > B$. By $\neg D_j \in I$ and $l(D_j) = l(A)$, we obtain that (WSii) must hold, and by the minimality of $l(A)$ we infer that (WSiib) must hold and that no clause with head D_j contains negated atoms. So $l(D_{j+1}) = l(D_j) = l(A)$ holds by (WSiib) and minimality of $l(A)$. Furthermore, the assumption $D_{j+1} \in I$ can be rejected by the same argument as for A above, because then $D_{j+1} \leftarrow$ would be the only clause with head D_{j+1}, by minimality of $l(D_{j+1}) = l(A)$, contradicting $B < D_{j+1}$. This concludes the inductive proof.

Summarizing, we obtain that D_k refers negatively to B', and that $\neg D_k \in I$. But then there is a clause with head D_k and $\neg B'$ in its body which satisfies (WSii), contradicting the minimality of $l(D_k) = l(A)$. This concludes the proof of statement (a).

(b) According to (Przymusinska and Przymusinski 1990) we have that whenever all components are trivial, then the bottom layer is definite. So the assertion follows from (a).

(c) Let $A \in I'$ be an atom with $A \notin N$, and assume without loss of generality that A is chosen such that $l(A)$ is minimal with these properties. Then there must be a clause $A \leftarrow \texttt{body}$ in P such that all literals in `body` are true with respect to I', hence with respect to N by minimality of $l(A)$. Thus `body` is true in N, and since N is a model of $L(P)$ we obtain $A \in N$, which contradicts our assumption.

Now let $A \in N$ be an atom with $A \notin I'$, and assume without loss of generality that A is chosen such that n is minimal with $A \in T^+_{L(P)} \uparrow (n+1)$. But then there is a definite clause $A \leftarrow \texttt{body}$ in $L(P)$ such that all atoms in `body` are true with respect to $T^+_{L(P)} \uparrow n$, hence also with respect to I', and since I' is a model of $L(P)$ we obtain $A \in I'$, which contradicts our assumption.

Finally, let $\neg A \in I'$. Then we cannot have $A \in N$ since this implies $A \in I'$. So $\neg A \in N$ since N is a total model of $L(P)$.

(d) From Lemma 1, we know that $P/N \preccurlyeq_N P$. We distinguish two cases.

(Case i) If $A \in I \setminus N$, then there must exist a clause $A \leftarrow L_1, \ldots, L_k$ in P such that $L_i \in I$ and $l(A) > l(L_i)$ for all i. Since it is not possible that $A \in N$, there must also be a clause in P/N which subsumes $A \leftarrow L_1, \ldots, L_k$, and which therefore satisfies (WSi). So A satisfies (WSi).

(Case ii) If $\neg A \in I \setminus N$, then for each clause $A \leftarrow \texttt{body1}$ in P/N there must be a clause $A \leftarrow \texttt{body}$ in P which is subsumed by the former, and since $\neg A \in I$, we obtain that condition (WSii) must be satisfied by A, and by the clause $A \leftarrow \texttt{body}$. Since reduction with respect to N removes only body literals which are true in N, condition (WSii) is still met. □

We can now proceed with the proof.

Proof of Theorem 6.4
The proof will be established by showing the following facts: (1) P satisfies (WS) with respect to M_P and l_P. (2) If I is a model of P and l an I-partial level mapping such that P satisfies (WS) with respect to I and l, then $I \subseteq M_P$.

(1) Let $A \in \mathsf{dom}(l_P)$ and $l_P(A) = (\alpha, n)$. We consider two cases.

(Case i) If $A \in M_P$, then $A \in T^+_{L_\alpha} \uparrow (n+1)$. Hence there exists a definite clause $A \leftarrow A_1, \dots, A_k$ in L_α with $A_1, \dots, A_k \in T^+_{L_\alpha} \uparrow n$, so $A_1, \dots, A_k \in M_P$ with $l_P(A) > l_P(A_i)$ for all i. Since $P/N_\alpha \preccurlyeq_{N_\alpha} P$ by Lemma 1, there must exist a clause $A \leftarrow A_1, \dots, A_k, L_1, \dots, L_m$ in P with literals $L_1, \dots, L_m \in N_\alpha \subseteq M_P$, and we obtain $l_P(L_j) < l_P(A)$ for all $j = 1, \dots, m$. So (WSi) holds in this case.

(Case ii) If $\neg A \in M_P$, then let $A \leftarrow A_1, \dots, A_k, \neg B_1, \dots, \neg B_m$ be a clause in P, noting that (WSii) is trivially satisfied in case no such clause exists. We consider the following two subcases.

(Subcase ii.a) Assume A is undefined in N_α and was eliminated from P by reducing it with respect to N_α, i.e. $A \in R_\alpha$. Then, in particular, there must be some $\neg A_i \in N_\alpha$ or some $B_j \in N_\alpha$, which yields $l_P(A_i) < l_P(A)$, respectively $l_P(B_j) < l_P(A)$, and hence one of (WSiia), (WSiic) holds.

(Subcase ii.b) Assume $\neg A \in H$, where H is the definite (partial) model of L_α. Since P/N_α subsumes P model-consistently with respect to N_α, we obtain that there must be some A_i with $\neg A_i \in H$, and by definition of l_P we obtain $l_P(A) = l_P(A_i) = (\alpha, \omega)$, and hence also $l_P(A_{i'}) \leq l_P(A_i)$ for all $i' \neq i$. Furthermore, since P/N_α is definite, we obtain that $\neg B_j \in N_\alpha$ for all j, hence $l_P(B_j) < l_P(A)$ for all j. So condition (WSiib) is satisfied.

(2) First note that for all models M, N of P with $M \subseteq N$ we have $(P/M)/N = P/(M \cup N) = P/N$ and $(P/N)/\emptyset = P/N$.

Let I_α denote I restricted to the atoms which are not undefined in $N_\alpha \cup R_\alpha$. It suffices to show the following: For all $\alpha > 0$ we have $I_\alpha \subseteq N_\alpha \cup R_\alpha$, and $I \setminus M_P = \emptyset$.

We next show by induction that if $\alpha > 0$ is an ordinal, then the following statements hold. (a) The bottom stratum of P/N_α is non-empty and consists of trivial components only. (b) The bottom layer of P/N_α is definite. (c) $I_\alpha \subseteq N_\alpha \cup R_\alpha$. (d) $P/N_{\alpha+1}$ satisfies (WS) with respect to $I \setminus N_{\alpha+1}$ and $l/N_{\alpha+1}$.

Note first that P satisfies the hypothesis of Lemma 2, hence also its consequences. So $P/N_1 = P/\emptyset$ satisfies (WS) with respect to $I \setminus N_1$ and l/N_1, and by application of Lemma 2 we obtain that statements (a) and (b) hold. For (c), note that no atom in R_1 can be true in I, because no atom in R_1 can appear as head of a clause in P, and apply Lemma 2 (c). For (d), apply Lemma 2, noting that $P/N_2 \preccurlyeq_{N_2} P$.

For α being a limit ordinal, we can show exactly as in the proof of Lemma 2 (d), that P satisfies (WS) with respect to $I \setminus N_\alpha$ and l/N_α. So Lemma 2 is applicable and statements (a) and (b) follow. For (c), let $A \in R_\alpha$. Then every clause in P with head A contains a body literal which is false in N_α. By induction hypothesis, this implies that no clause with head A in P can have a body which is true in I. So $A \notin I$. Together with Lemma 2 (c), this proves statement (c). For (d), apply again Lemma 2 (d), noting that $P/N_{\alpha+1} \preccurlyeq_{N_{\alpha+1}} P$.

For $\alpha = \beta + 1$ being a successor ordinal, we obtain by induction hypothesis that P/N_β satisfies the hypothesis of Lemma 2, so again statements (a) and (b) follow immediately from this lemma, and (c), (d) follow as in the case for α being a limit ordinal.

It remains to show that $I \setminus M_P = \emptyset$. Indeed by the transfinite induction argument just given we obtain that P/M_P satisfies (WS) with respect to $I \setminus M_P$ and l/M_P. If $I \setminus M_P$ is non-empty, then by Lemma 2 the bottom stratum $S(P/M_P)$ is non-empty and the bottom layer $L(P/M_P)$ is definite with definite (partial) model M. Hence by definition of the weakly perfect model M_P of P we must have that $M \subseteq M_P$ which contradicts the fact that M is the definite model of $L(P/M_P)$. Hence $I \setminus M_P$ must be empty which concludes the proof. $\square$

Of independent interest is again the case, where the model in question is total. We see immediately, for example, in the light of Theorem 3.3, that the model is then stable.

Corollary 6.6
A normal logic program P is weakly stratified, i.e. has a total weakly perfect model, if and only if there is a total model I of P and a (total) level mapping l for P such that P satisfies (WS) with respect to I and l.

We also obtain the following corollary as a trivial consequence of our uniform characterizations by level mappings.

Corollary 6.7
Let P be a normal logic progam with Fitting model M_{F}, weakly perfect model M_{WP}, and well-founded model M_{WF}. Then $M_{\mathrm{F}} \subseteq M_{\mathrm{WP}} \subseteq M_{\mathrm{WF}}$.

Example 6.8
Consider the program P from Example 6.3. Then $M_{\mathrm{F}} = \emptyset$, $M_{\mathrm{WP}} = \{\neg d, \neg e\}$, and $M_{\mathrm{WF}} = \{a, \neg b, \neg c, \neg d, \neg e\}$.

7 Related Work

As already mentioned in the introduction, level mappings have been used for studying semantic aspects of logic programs in a number of different ways. Our presentation suggests a novel application of level mappings, namely for providing uniform characterizations of different fixed-point semantics for logic programs with negation. Although we believe our perspective to be new in this general form, there nevertheless have been results in the literature which are very close in spirit to our characterizations.

A first noteable example of this is Fages' characterization of stable models (Fages 1994), which we have stated in Theorem 3.3. Another result which uses level mappings to characterize a semantics is by Lifschitz, Przymusinski, Stärk, and McCain (Lifschitz et al. 1995, Lemma 3). We briefly compare their characterization of the well-founded semantics and ours. In fact, this discussion can be based upon two different characterizations of the least fixed point of a monotonic operator F. On the one hand, this least fixed point is of course the *least of all fixed points* of F, and on the other hand, this least fixed point is the limit of the sequence of powers $(F \uparrow \alpha)_\alpha$, and in this latter sense is the *least iterate* of F *which is also a fixed point*. Our characterizations of definite, Fitting, well-founded, and weakly stratified semantics use the latter approach, which is reflected in our general proof scheme, which defines level mappings according to powers, or iterates, of the respective operators. The results by Fages (Fitting 1994) and Lifschitz et al. (Lifschitz et al. 1995) hinge upon the former approach, i.e. they are based on the idea of characterizing the fixed points of an operator — GL_P, respectively Ψ_P (Przymusinski 1989; Bonnier et al. 1991) — and so the sought fixed point turns out to be the least of those. Consequently, as can be seen in the proof of Theorem 3.3, the level mapping in Fages' characterization, and likewise in the result by Lifschitz et al., arises only indirectly from the operator — GL_P, respectively Ψ_P — whose fixed point is sought. Indeed, the level mapping by Fages is defined according to iterates of $T_{P/I}$, which is the operator for obtaining $\mathrm{GL}_P(I)$, for any I. The result by Lifschitz et al. is obtained similarly based on a three-valued operator Ψ_P.

Unforunately, these characterizations by Fages, in Theorem 3.3, respectively by Lifschitz et al. (Lifschitz et al. 1995), seem to be applicable only to operators which are defined by least fixed points of other operators, as is the case for GL_P and Ψ_P, and it seems that the approach by Lifschitz et al. is unlikely to scale to other semantics. For example, we attempted a straightforward characterization of the Fitting semantics in the spririt of Lifschitz et al. which failed.

On a more technical level, a difference between our result, Theorem 5.2, and the characterization by Lifschitz et al. (Lifschitz et al. 1995) of the well-founded semantics is this: In our characterization, the model is described using conditions on atoms which are true or false (i.e. *not undefined*) in the well-founded model, whereas in theirs the conditions are on those atoms which are true or undefined (i.e. *not false*) in the well-founded model. The reason for this is that we consider iterates of W_P, where $W_P \uparrow 0 = \emptyset$, while they use the fact that each fixed point of Ψ_P is a least fixed point of $\Phi_{P/I}$ with respect to the truth ordering on interpretations (note that in this case P/I denotes a three-valued generalization of the Gelfond-Lifschitz transformation due to Przymusinski (Przymusinski 1989)). In this ordering we have $\Phi_{P/I} \uparrow 0 = \neg B_P$. It is nevertheless nice to note that in the special case of the well-founded semantics there exist two complementary characterizations using level mappings.

Since our proposal emphasizes uniformity of characterizations, it is related to the large body of work on uniform approaches to logic programming semantics, of which we will discuss two in more detail: the algebraic approach via bilattices due to Fitting, and the work of Dix.

Bilattice-based semantics has a long tradition in logic programming theory, starting out from the four-valued logic of Belnap (Belnap 1977). The underlying set of truth values, a four-element lattice, was recognized to admit two ordering relations which can be interpreted as truth- and knowledge-order. As such it has the structure of a bilattice, a term due to Ginsberg (Ginsberg 1986), who was the first to note the importance of bilattices for inference in artificial intelligence (Ginsberg 1992). This general approach was imported into logic programming theory by Fitting (Fitting 1991a). Although multi-valued logics had been used for logic programming semantics before (Fitting 1985), bilattices provided an interesting approach to semantics as they are capable of incorporating both reasoning about truth and reasoning about knowledge, and, more technically, because they have nice algebraic behaviour. Using this general framework Fitting was able to show interesting relationships between the stable and the well-founded semantics (Fitting 1991b; Fitting 1993; Fitting 2002).

Without claiming completeness we note two current developments in the bilattice-based approach to logic programming: Fitting's framework has been extended to an algebraic approach for approximating operators by Denecker, Marek, and Truszczynski (Denecker et al. 2000). The inspiring starting point of this work was the noted relationship between the stable model semantics and the well-founded semantics, the latter approximating the former. The other line of research was pursued mainly by Arieli and Avron (Arieli and Avron 1994; Arieli and Avron 1998; Arieli 2002), who use bilattices for paraconsistent reasoning in logic programming. The above outline of the historical development of bilattices in logic programming theory suggests a similar kind of uniformity as we claim for our approach. The exact relationship between both approaches, however, is still to be investigated. On the one hand, bilattices can cope with paraconsistency — an issue of logic programming and deductive databases, which is becoming more and more important — in a very convenient way. On the other hand, our approach can deal with semantics based on multi-valued logics, whose underlying truth structure is not a bilattice. A starting point for investigations in this direction could be the obvious meeting point of both theories: the well-founded semantics for which we can provide a characterization and which is a special case of the general approximation theory of Denecker et al. (Denecker et al. 2000).

Another very general, and uniform, approach to logic programming pursues a different point of view, namely logic programming semantics as nonmonotonic inference. The general theory of nonmonotonic inference and a classification of properties of nonmonotonic operators was developed by Kraus, Lehmann, and Magidor (Kraus et al. 1990), leading to the notion *KLM-axioms* for these properties, and developed further by Makinson (Makinson 1994). These axioms were adopted to the terminology of logic programming and extended to a general theory of logic programming semantics by Dix (Dix 1995a; Dix 1995b). In this framework, different known semantics are classified according to strong properties — the KLM-axioms which hold for the semantics – and weak properties — specific properties which deal with the irregularities of negation-as-failure. As such Dix' framework is indeed a general and uniform approach to logic programming, its main focus being on semantic properties of logic programs. Our approach in turn could be called *semi-syntactic* in that definitions that employ level mappings naturally take the structure of the logic program into account. As in the case of the bilattice-based approaches, it is not yet completely clear whether these two approaches can be amalgamated in the sense of a correspondence between properties of level mappings, e.g. strict or semi-strict descent of the level, etc., on the one hand, and KLM-properties of the logic program on the other. However, we believe that it is possible to develop a proof scheme for nonmonotonic properties of logic programs in the style of the proof scheme presented in the paper, which can be used to cast semantics based on monotonic operators into level mapping form.

We finally mention the work by Hitzler and Seda (Hitzler and Seda 1999), which was the root and starting point for our investigations. This framework aims at the characterization of program classes, such as (locally) stratified programs (Apt et al. 1988; Przymusinski 1988), acceptable programs (Apt and Pedreschi 1993), or Φ-accessible programs (Hitzler and Seda 1999). Such program classes appear naturally whenever a semantics is not defined for all logic programs. In these cases one tries to characterize those programs, for which the semantics is well-defined or well-behaved. Their main tool were monotonic operators in three-valued logic, in the spirit of Fitting's Φ_P, rather than level mappings. With each operator comes a least fixed point, hence a semantics, and it is easily checked that these semantics can be characterized using our approach, again by straightforward application of our proof scheme. Indeed, preliminary steps in this direction already led to an independent proof of a special case of Corollary 6.7 (Hitzler and Seda 2001).

8 Conclusions and Further Work

We have proposed a novel approach for obtaining uniform characterizations of different semantics for logic programs. We have exemplified this by giving new alternative characterizations of some of the major semantics from the literature. We have developed and presented a methodology for obtaining characterizations from monotonic semantic operators or related constructions, and a proof scheme for showing correctness of the obtained characterizations. We consider our contribution to be fundamental, with potential for extension in many directions.

Our approach employs level mappings as central tool. The uniformity with which our characterizations were obtained and proven to be correct suggests that our method should be of wider applicability. In fact, since it builds upon the well-known Tarski fixed point theorem, it should scale well to most, if not all semantics, which are defined by means of a monotonic operator. The main contribution of this paper is thus, that we have developed a novel way of presenting logic programming semantics in some kind of *normal* or *standard form*. This can be used for easy comparison of semantics with respect to the syntactic structures that can be used with a certain semantics, i.e. to what extent the semantics is able to 'break up' positive or negative dependencies or loops between atoms in the program, as in Corollary 6.7.

However, there are many more requirements which a general and uniform approach to logic program semantics should eventually be able to meet, including (i) a better understanding of known semantics, (ii) proof schemes for deriving properties of semantics, (iii) extendability to new programming constructs, and (iv) support for designing new semantics for special purposes.

Requirement (i) is met to some extent by our appoach, since it enables easy comparison of semantics, as discussed earlier. However, in order to meet the other requirements, i.e. to set up a *meta-theory* of level-mapping-based semantics, a lot of further research is needed. We list some topics to be pursued in the future, some of which are under current investigation by the authors. There are many properties which are interesting to know about a certain semantics, depending on one's perspective. For the nonmonotonic reasoning aspect of logic programming it would certainly be interesting to have a proof scheme as flexible and uniform as the one presented in this paper. Results and proofs in the literature (Fages 1994; Dix 1995a; Turner 2001) suggest that there is a strong dependency between notions of ordering on the Herbrand base, as expressed by level mappings, and KLM-properties satisfied by a semantics, which constitutes some evidence that a general proof scheme for proving KLM-properties from level mapping definitions can be developed. Other interesting properties are e.g. the computational complexity of a semantics, but also logical characterizations of the behaviour of negation in logic programs, a line of research initiated by Pearce (Pearce 1997).

For (iii), it would be desirable to extend our characterizations also to disjunctive programs, which could perhaps contribute to the discussion about appropriate generalizations of semantics of normal logic programs to the disjunctive case.

We finally want to mention that the elegant mathematical framework of level mapping definitions naturally gives rise to the design of new semantics. However, at the time being this is only a partial fulfillment of (iv): As long as a meta-theory for level-mapping-based semantics is missing, one still has to apply conventional methods for extracting properties of the respective semantics from its definition.

References

Abramsky, S. and Jung, A. 1994. Domain theory. In *Handbook of Logic in Computer Science*, S. Abramsky, D. Gabbay, and T. S. Maibaum, Eds. Vol. 3. Clarendon, Oxford, 1–168.

Apt, K. R., Blair, H. A., and Walker, A. 1988. Towards a theory of declarative knowledge. In *Foundations of Deductive Databases and Logic Programming*, J. Minker, Ed. Morgan Kaufmann, Los Altos, CA, 89–148.

Apt, K. R. and Pedreschi, D. 1993. Reasoning about termination of pure Prolog programs. *Information and Computation 106*, 109–157.

Arieli, O. 2002. Paraconsistent declarative semantics for extended logic programs. *Annals of Mathematics and Artificial Intelligence 36(4)*, 381–417.

Arieli, O. and Avron, A. 1994. Logical bilattices and inconsistent data. In *Proceedings of the 9th Annual IEEE Symposium on Logic in Computer Science*. IEEE Press, 468–476.

Arieli, O. and Avron, A. 1998. The value of the four values. *Artificial Intelligence 102,* 1, 97–141.

Belnap, N. D. 1977. A useful four-valued logic. In *Modern Uses of Multiple-Valued Logic*, J. M. Dunn and G. Epstein, Eds. Reidel, Dordrecht, 5–37.

Bezem, M. 1989. Characterizing termination of logic programs with level mappings. In *Proceedings of the North American Conference on Logic Programming*, E. L. Lusk and R. A. Overbeek, Eds. MIT Press, Cambridge, MA, 69–80.

Bonnier, S., Nilsson, U., and Näslund, T. 1991. A simple fixed point characterization of three-valued stable model semantics. *Information Processing Letters 40,* 2, 73–78.

Cavedon, L. 1991. Acyclic programs and the completeness of SLDNF-resolution. *Theoretical Computer Science 86*, 81–92.

Denecker, M., Marek, V. W., and Truszczynski, M. 2000. Approximating operators, stable operators, well-founded fixpoints and applications in non-monotonic reasoning. In *Logic-based Artificial Intelligence*, J. Minker, Ed. Kluwer Academic Publishers, Boston, Chapter 6, 127–144.

Dix, J. 1995a. A classification theory of semantics of normal logic programs: I. Strong properties. *Fundamenta Informaticae 22,* 3, 227–255.

Dix, J. 1995b. A classification theory of semantics of normal logic programs: II. Weak properties. *Fundamenta Informaticae 22,* 3, 257–288.

Eiter, T., Leone, N., Mateis, C., Pfeifer, G., and Scarcello, F. 1997. A deductive system for nonmonotonic reasoning. In *Proceedings of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'97)*, J. Dix, U. Furbach, and A. Nerode, Eds. Lecture Notes in Artificial Intelligence, vol. 1265. Springer, Berlin, 364–375.

Fages, F. 1994. Consistency of Clark's completion and existence of stable models. *Journal of Methods of Logic in Computer Science 1*, 51–60.

Fitting, M. 1985. A Kripke-Kleene-semantics for general logic programs. *The Journal of Logic Programming 2*, 295–312.

Fitting, M. 1991a. Bilattices and the semantics of logic programming. *The Journal of Logic Programming 11*, 91–116.

Fitting, M. 1991b. Well-founded semantics, generalized. In *Logic Programming, Proceedings of the 1991 International Symposium.* MIT Press, Cambridge, MA, 71–84.

Fitting, M. 1993. The family of stable models. *Journal of Logic Programming 17*, 197–225.

Fitting, M. 1994. Metric methods: Three examples and a theorem. *The Journal of Logic Programming 21,* 3, 113–127.

Fitting, M. 2002. Fixpoint semantics for logic programming — A survey. *Theoretical Computer Science 278,* 1–2, 25–51.

Gelfond, M. and Lifschitz, V. 1988. The stable model semantics for logic programming. In *Logic Programming. Proceedings of the 5th International Conference and Symposium on Logic Programming*, R. A. Kowalski and K. A. Bowen, Eds. MIT Press, 1070–1080.

Ginsberg, M. L. 1986. Bilattices. Tech. Rep. 86-72, Stanford University, KSL.

Ginsberg, M. L. 1992. Multivalued logics: A uniform approach to inference in artificial intelligence. *Computational Intelligence 4,* 3, 256–316.

Hitzler, P. 2001. Generalized metrics and topology in logic programming semantics. Ph.D. thesis, Department of Mathematics, National University of Ireland, University College Cork.

Hitzler, P. 2003. Towards a systematic account of different logic programming semantics. In *Proceedings of the 26th German Conference on Artificial Intelligence, KI2003, Hamburg, September 2003*, A. Günter, R. Krause, and B. Neumann, Eds. Lecture Notes in Artificial Intelligence, vol. 2821. Springer, Berlin, 355–369.

Hitzler, P. and Seda, A. K. 1999. Characterizations of classes of programs by three-valued operators. In *Logic Programming and Nonmonotonic Reasoning, Proceedings of the 5th International Conference on Logic Programming and Non-Monotonic Reasoning, LPNMR'99, El Paso, Texas, USA*, M. Gelfond, N. Leone, and G. Pfeifer, Eds. Lecture Notes in Artificial Intelligence, vol. 1730. Springer, Berlin, 357–371.

Hitzler, P. and Seda, A. K. 2000. A note on relationships between logic programs and neural networks. In *Proceedings of the Fourth Irish Workshop on Formal Methods, IWFM'00*, P. Gibson and D. Sinclair, Eds. Electronic Workshops in Comupting (eWiC). British Computer Society.

Hitzler, P. and Seda, A. K. 2001. Unique supported-model classes of logic programs. *Information 4,* 3, 295–302.

Hitzler, P. and Seda, A. K. 2003a. Continuity of semantic operators in logic programming and their approximation by artificial neural networks. In *Proceedings of the 26th German Conference on Artificial Intelligence, KI2003*, A. Günter, R. Krause, and B. Neumann, Eds. Lecture Notes in Artificial Intelligence, vol. 2821. Springer, 105–119.

Hitzler, P. and Seda, A. K. 2003b. Generalized metrics and uniquely determined logic programs. *Theoretical Computer Science 305*, 1–3, 187–219.

Hitzler, P. and Wendt, M. 2002. The well-founded semantics is a stratified Fitting semantics. In *Proceedings of the 25th Annual German Conference on Artificial Intelligence, KI2002, Aachen, Germany, September 2002*, M. Jarke, J. Koehler, and G. Lakemeyer, Eds. Lecture Notes in Artificial Intelligence, vol. 2479. Springer, Berlin, 205–221.

Hölldobler, S., Kalinke, Y., and Störr, H.-P. 1999. Approximating the semantics of logic programs by recurrent neural networks. *Applied Intelligence 11*, 45–58.

Kraus, S., Lehmann, D., and Magidor, M. 1990. Nonmonotonic reasoning, preferential models and cumulative logics. *Artificial Intelligence 44,* 1, 167–207.

Kunen, K. 1987. Negation in logic programming. *The Journal of Logic Programming 4*, 289–308.

Lifschitz, V. 2002. Answer set programming and plan generation. *Artificial Intelligence 138*, 39–54.

Lifschitz, V., McCain, N., Przymusinski, T. C., and Stärk, R. F. 1995. Loop checking and the well-founded semantics. In *Logic Programming and Non-monotonic Reasoning, Proceedings of the 3rd International Conference, LPNMR'95, Lexington, KY, USA, June 1995*, V. W. Marek and A. Nerode, Eds. Lecture Notes in Computer Science, vol. 928. Springer, 127–142.

Lloyd, J. W. 1988. *Foundations of Logic Programming*. Springer, Berlin.

Makinson, D. 1994. General patterns of nonmonotonic reasoning. In *Handbook of Logic in Artificial Intelligence and Logic Programming, Vol. 3, Nonmonotonic and Uncertain Reasoning*, D. M. Gabbay, C. J. Hogger, and J. A. Robinson, Eds. Oxford University Press.

Marchiori, E. 1996. On termination of general logic programs with respect to constructive negation. *The Journal of Logic Programming 26,* 1, 69–89.

Marek, V. W. and Truszczyński, M. 1999. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm: A 25-Year Persepective*, K. R. Apt, V. W. Marek, M. Truszczyński, and D. S. Warren, Eds. Springer, Berlin, 375–398.

Pearce, D. 1997. A new logical characterisation of stable models and answer sets. In *Non-Monotonic Extensions of Logic Programming, NMELP '96*, J. Dix, L. M. Pereira, and T. C. Przymusinski, Eds. Lecture Notes in Computer Science, vol. 1216. Springer, 57–70.

Pedreschi, D., Ruggieri, S., and Smaus, J.-G. 2002. Classes of terminating logic programs. *Theory and Practice of Logic Programs 2,* 3, 369–418.

Przymusinska, H. and Przymusinski, T. C. 1990. Weakly stratified logic programs. *Fundamenta Informaticae 13*, 51–65.

Przymusinski, T. C. 1988. On the declarative semantics of deductive databases and logic programs. In *Foundations of Deductive Databases and Logic Programming*, J. Minker, Ed. Morgan Kaufmann, Los Altos, CA, 193–216.

Przymusinski, T. C. 1989. Well-founded semantics coincides with three-valued stable semantics. *Fundamenta Informaticae 13,* 4, 445–464.

Seda, A. K. 1995. Topology and the semantics of logic programs. *Fundamenta Informaticae 24,* 4, 359–386.

Seda, A. K. 1997. Quasi-metrics and the semantics of logic programs. *Fundamenta Informaticae 29,* 1, 97–117.

Simons, P., Niemelä, I., and Soininen, T. 200x. Extending and implementing the stable model semantics. *Artificial Intelligence*. To appear.

Stoltenberg-Hansen, V., Lindström, I., and Griffor, E. R. 1994. *Mathematical Theory of Domains*. Cambridge University Press.

Turner, H. 2001. Order-consistent programs are cautiously monotonic. *Journal of Theory and Practice of Logic Programming 1,* 4, 487–495.

van Gelder, A. 1988. Negation as failure using tight derivations for general logic programs. In *Foundations of Deductive Databases and Logic Programming*, J. Minker, Ed. Morgan Kaufmann, Los Altos, CA, 149–176.

van Gelder, A. 1989. The alternating fixpoint of logic programs with negation. In *Proceedings of the Eighth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Philadelphia, Pennsylvania*. ACM Press, 1–10.

van Gelder, A., Ross, K. A., and Schlipf, J. S. 1991. The well-founded semantics for general logic programs. *Journal of the ACM 38,* 3, 620–650.

Towards a Systematic Account of Different Logic Programming Semantics

Pascal Hitzler

Department of Computer Science, Dresden University of Technology
`www.wv.inf.tu-dresden.de/~pascal/`
`phitzler@inf.tu-dresden.de`

Abstract. In [1,2], a new methodology has been proposed which allows to derive uniform characterizations of different declarative semantics for logic programs with negation. One result from this work is that the well-founded semantics can formally be understood as a stratified version of the Fitting (or Kripke-Kleene) semantics. The constructions leading to this result, however, show a certain asymmetry which is not readily understood. We will study this situation here with the result that we will obtain a coherent picture of relations between different semantics.

1 Introduction

Within the past twenty years, many different declarative semantics for logic programs with negation have been developed. Different perspectives on the question what properties a semantics should foremost satisfy, have led to a variety of diverse proposals. From a knowledge representation and reasoning point of view it appears to be important that a semantics captures established non-monotonic reasoning frameworks, e.g. Reiters default logic [3], and that they allow intuitively appealing, i.e. "common sense", encodings of AI problems. The semantics which, due to common opinion by researchers in the field, satisfy these requirements best, are the least model semantics for definite programs [4], and for normal programs the stable [5] and the well-founded semantics [6]. Of lesser importance, albeit still acknowledged in particular for their relation to resolution-based logic programming, are the Fitting semantics [7] and approaches based on stratification [8,9].

The semantics just mentioned are closely connnected by a number of well- (and some lesser-) known relationships, and many authors have contributed to this understanding. Fitting [10] provides a framework using Belnap's four-valued logic which encompasses supported, stable, Fitting, and well-founded semantics. His work was recently extended by Denecker, Marek, and Truszczynski [11]. Przymusinsky [12] gives a version in three-valued logic of the stable semantics, and shows that it coincides with the well-founded one. Van Gelder [13] constructs the well-founded semantics unsing the Gelfond-Lifschitz operator originally associated with the stable semantics. Dung and Kanchanasut [14] define the notion of fixpoint completion of a program which provides connections between the supported and the stable semantics, as well as between the Fitting and the well-founded semantics, studied by Fages [15] and Wendt [16]. Hitzler and Wendt [1, 2] have recently provided a unifying framework using level mappings, and results which amongst other things give further support to the point of view that the stable semantics is a formal and natural extension to normal programs of the least model semantics for definite programs. Furthermore, it was shown that the well-founded semantics can be understood, formally, as a stratified version of the Fitting semantics.

This latter result, however, exposes a certain asymmetry in the construction leading to it, and it is natural to ask the question as to what exactly is underlying it. This is what we will study in the sequel. In a nutshell, we will see that formally this asymmetry is due to the well-known preference of falsehood in logic programming semantics. More importantly, we will also see that a "dual" theory, obtained from prefering truth, can be stablished which is in perfect analogy to the close and well-known relationships between the different semantics mentioned above. We want to make it explicit from the start that we do not intend to provide new semantics for practical purposes[1]. We rather want to focus on the deepening of the theoretical insights into the relations between different semantics, by painting a coherent and complete picture of the dependencies and interconnections. We find the richness of the theory very appealing, and strongly supportive of the opinion that the major semantics studied in the field are founded on a sound theoretical base.

The plan of the paper is as follows. In Section 2 we will introduce notation and terminology needed for proving the results in the main body of the paper. We will also review in detail those results from [1,2] which triggered and motivated our investigations. In Section 3 we will provide a variant of the stable semantics which prefers truth, and in Section 4 we will do likewise for the well-founded semantics. Throughout, our definitions will be accompanied by results which complete the picture of relationships between different semantics.

2 Preliminaries and Notation

A (*normal*) *logic program* is a finite set of (universally quantified) *clauses* of the form $\forall(A \leftarrow A_1 \wedge \cdots \wedge A_n \wedge \neg B_1 \wedge \cdots \wedge \neg B_m)$, commonly written as $A \leftarrow A_1, \dots, A_n, \neg B_1, \dots, \neg B_m$, where A, A_i, and B_j, for $i = 1, \dots, n$ and $j = 1, \dots, m$, are atoms over some given first order language. A is called the *head* of the clause, while the remaining atoms make up the *body* of the clause, and depending on context, a body of a clause will be a set of literals (i.e. atoms or negated atoms) or the conjunction of these literals. Care will be taken that this identification does not cause confusion. We allow a body, i.e. a conjunction, to be empty, in which case it always evaluates to true. A clause with empty body is called a *unit clause* or a *fact*. A clause is called *definite*, if it contains no negation symbol. A program is called *definite* if it consists only of definite

[1] Although there may be some virtue to this perspective, see [17].

clauses. We will usually denote atoms with A or B, and literals, which may be atoms or negated atoms, by L or K.

Given a logic program P, we can extract from it the components of a first order language, and we always make the mild assumption that this language contains at least one constant symbol. The corresponding set of ground atoms, i.e. the *Herbrand base* of the program, will be denoted by B_P. For a subset $I \subseteq B_P$, we set $\neg I = \{\neg A \mid A \in B_P\}$. The set of all ground instances of P with respect to B_P will be denoted by $\mathsf{ground}(P)$. For $I \subseteq B_P \cup \neg B_P$, we say that A is *true with respect to* (or *in*) I if $A \in I$, we say that A is *false with respect to* (or *in*) I if $\neg A \in I$, and if neither is the case, we say that A is *undefined with respect to* (or *in*) I. A (*three-valued* or *partial*) *interpretation* I for P is a subset of $B_P \cup \neg B_P$ which is *consistent*, i.e. whenever $A \in I$ then $\neg A \notin I$. A body, i.e. a conjunction of literals, is true in an interpretation I if every literal in the body is true in I, it is false in I if one of its literals is false in I, and otherwise it is undefined in I. For a negated literal $L = \neg A$ we will find it convenient to write $\neg L \in I$ if $A \in I$. By I_P we denote the set of all (three-valued) interpretations of P. Both I_P and $B_P \cup \neg B_P$ are complete partial orders (cpos) via set-inclusion, i.e. they contain the empty set as least element, and every ascending chain has a supremum, namely its union. A *model* of P is an interpretation $I \in I_P$ such that for each clause $A \leftarrow \mathsf{body}$ we have that $\mathsf{body} \subseteq I$ implies $A \in I$. A *total* interpretation is an interpretation I such that no $A \in B_P$ is undefined in I.

For an interpretation I and a program P, an *I-partial level mapping* for P is a partial mapping $l : B_P \to \alpha$ with domain $\mathsf{dom}(l) = \{A \mid A \in I \text{ or } \neg A \in I\}$, where α is some (countable) ordinal. We extend every level mapping to literals by setting $l(\neg A) = l(A)$ for all $A \in \mathsf{dom}(l)$. A (*total*) *level mapping* is a total mapping $l : B_P \to \alpha$ for some (countable) ordinal α.

Given a normal logic program P and some $I \subseteq B_P \cup \neg B_P$, we say that $U \subseteq B_P$ is an *unfounded set* (*of P*) *with respect to I* if each atom $A \in U$ satisfies the following condition: For each clause $A \leftarrow \mathsf{body}$ in $\mathsf{ground}(P)$ (at least) one of the following holds.

(Ui) Some (positive or negative) literal in body is false in I.
(Uii) Some (non-negated) atom in body occurs in U.

Given a normal logic program P, we define the following operators on $B_P \cup \neg B_P$. $T_P(I)$ is the set of all $A \in B_P$ such that there exists a clause $A \leftarrow \mathsf{body}$ in $\mathsf{ground}(P)$ such that body is true in I. $F_P(I)$ is the set of all $A \in B_P$ such that for all clauses $A \leftarrow \mathsf{body}$ in $\mathsf{ground}(P)$ we have that body is false in I. Both T_P and F_P map elements of I_P to elements of I_P. Now define the operator $\Phi_P : I_P \to I_P$ by

$$\Phi_P(I) = T_P(I) \cup \neg F_P(I).$$

This operator is due to [7] and is well-defined and monotonic on the cpo I_P, hence has a least fixed point by the Knaster-Tarski[2] fixed-point theorem, and we can obtain this fixed point by defining, for each monotonic operator F, that

[2] We follow the terminology from [18]. The Knaster-Tarski theorem is sometimes called Tarski theorem and states that every monotonic function on a cpo has a least fixed

$F\uparrow 0 = \emptyset$, $F\uparrow(\alpha+1) = F(F\uparrow\alpha)$ for any ordinal α, and $F\uparrow\beta = \bigcup_{\gamma<\beta} F\uparrow\gamma$ for any limit ordinal β, and the least fixed point of F is obtained as $F\uparrow\alpha$ for some ordinal α. The least fixed point of Φ_P is called the *Kripke-Kleene model* or *Fitting model* of P, determining the *Fitting semantics* of P.

Now, for $I \subseteq B_P \cup \neg B_P$, let $U_P(I)$ be the greatest unfounded set (of P) with respect to I, which always exists due to [6]. Finally, define

$$W_P(I) = T_P(I) \cup \neg U_P(I)$$

for all $I \subseteq B_P \cup \neg B_P$. The operator W_P, which operates on the cpo $B_P \cup \neg B_P$, is due to [6] and is monotonic, hence has a least fixed point by the Knaster-Tarski[2] fixed-point theorem, as above for Φ_P. It turns out that $W_P\uparrow\alpha$ is in I_P for each ordinal α, and so the least fixed point of W_P is also in I_P and is called the *well-founded model* of P, giving the *well-founded semantics* of P.

In order to avoid confusion, we will use the following terminology: the notion of *interpretation*, and I_P will be the set of all those, will by default denote consistent subsets of $B_P \cup \neg B_P$, i.e. interpretations in three-valued logic. We will sometimes emphasize this point by using the notion *partial interpretation.* By *two-valued interpretations* we mean subsets of B_P. Both interpretations and two-valued interpretations are ordered by subset inclusion. Each two-valued interpretation I can be identified with the partial interpretation $I' = I \cup \neg(B_P \setminus I)$. Note however, that in this case I' is always a maximal element in the ordering for partial interpretations, while I is in general not maximal as a two-valued interpretation[3]. Given a partial interpretation I, we set $I^+ = I \cap B_P$ and $I^- = \{A \in B_P \mid \neg A \in I\}$.

Given a program P, we define the operator T_P^+ on subsets of B_P by $T_P^+(I) = T_P(I \cup \neg(B_P \setminus I))$. The pre-fixed points of T_P^+, i.e. the two-valued interpretations $I \subseteq B_P$ with $T_P^+(I) \subseteq I$, are exactly the models, in the sense of classical logic, of P. Post-fixed points of T_P^+, i.e. $I \subseteq B_P$ with $I \subseteq T_P^+(I)$ are called *supported interpretations* of P, and a supported model of P is a model P which is a supported interpretation. The supported models of P thus coincide with the fixed points of T_P^+. It is well-known that for definite programs P the operator T_P^+ is monotonic on the set of all subsets of B_P, with respect to subset inclusion. Indeed it is Scott-continuous [4,20] and, via the Tarski-Kantorovich[2] fixed-point theorem, achieves its least pre-fixed point M, which is also a fixed point, as the supremum of the iterates $T_P^+\uparrow n$ for $n \in \mathbb{N}$. So $M = \mathsf{lfp}\left(T_P^+\right) = T_P^+\uparrow\omega$ is *the least two-valued model* of P. Likewise, since the set of all subsets of B_P is

point, which can be obtained by transfinitely iterating the bottom element of the cpo. The Tarski-Kantorovitch theorem is sometimes refered to as the Kleene theorem or the Scott theorem (or even as "the" fixed-point theorem) and states that if the function is additionally Scott (or order-) continuous, then the least fixed point can be obtained by an iteration which is not transfinite, i.e. closes off at ω, the least infinite ordinal. In both cases, the least fixed point is also the least pre-fixed point of the function.

[3] These two orderings in fact correspond to the knowledge and truth orderings as discussed in [19].

a complete lattice, and therefore has greatest element B_P, we can also define $T_P^+ \downarrow 0 = B_P$ and inductively $T_P^+ \downarrow (\alpha+1) = T_P^+(T_P^+ \downarrow \alpha)$ for each ordinal α and $T_P^+ \downarrow \beta = \bigcap_{\gamma<\beta} T_P^+ \downarrow \gamma$ for each limit ordinal β. Again by the Knaster-Tarski fixed-point theorem, applied to the superset inclusion ordering (i.e. reverse subset inclusion) on subsets of B_P, it turns out that T_P^+ has a greatest fixed point, $\mathsf{gfp}\left(T_P^+\right)$.

The stable model semantics due to [5] is intimately related to the well-founded semantics. Let P be a normal program, and let $M \subseteq B_P$ be a set of atoms. Then we define P/M to be the (ground) program consisting of all clauses $A \leftarrow A_1, \dots, A_n$ for which there is a clause $A \leftarrow A_1, \dots, A_n, \neg B_1, \dots, \neg B_m$ in $\mathsf{ground}(P)$ with $B_1, \dots, B_m \notin M$. Since P/M does no longer contain negation, it has a least two-valued model $T_{P/M}^+ \uparrow \omega$. For any two-valued interpretation I we can therefore define the operator $\mathrm{GL}_P(I) = T_{P/I}^+ \uparrow \omega$, and call M a *stable model* of the normal program P if it is a fixed point of the operator GL_P, i.e. if $M = \mathrm{GL}_P(M) = T_{P/M}^+ \uparrow \omega$. As it turns out, the operator GL_P is in general not monotonic for normal programs P. However it is *antitonic*, i.e. whenever $I \subseteq J \subseteq B_P$ then $\mathrm{GL}_P(J) \subseteq \mathrm{GL}_P(I)$. As a consequence, the operator GL_P^2, obtained by applying GL_P twice, is monotonic, and hence has a least fixed point L_P and a greatest fixed point G_P. In [13] it was shown that $\mathrm{GL}_P(L_P) = G_P$, $L_P = \mathrm{GL}_P(G_P)$, and that $L_P \cup \neg(B_P \setminus G_P)$ coincides with the well-founded model of P. This is called the *alternating fixed point characterization* of the well-founded semantics.

Some Results

The following is a straightforward result which has, to the best of our knowledge, not been noted before. It follows the general approach put forward in [1,2].

Theorem 1. *Let P be a definite program. Then there is a unique two-valued model M of P for which there exists a (total) level mapping $l : B_P \to \alpha$ such that for each atom $A \in M$ there exists a clause $A \leftarrow A_1, \dots, A_n$ in $\mathsf{ground}(P)$ with $A_i \in M$ and $l(A) > l(A_i)$ for all $i = 1, \dots, n$. Furthermore, M is the least two-valued model of P.*

Proof. Let M be the least two-valued model $T_P^+ \uparrow \omega$, choose $\alpha = \omega$, and define $l : B_P \to \alpha$ by setting $l(A) = \min\{n \mid A \in T_P^+ \uparrow (n+1)\}$, if $A \in M$, and by setting $l(A) = 0$, if $A \notin M$. From the fact that $\emptyset \subseteq T_P^+ \uparrow 1 \subseteq \ldots \subseteq T_P^+ \uparrow n \subseteq \ldots \subseteq T_P^+ \uparrow \omega = \bigcup_m T_P^+ \uparrow m$, for each n, we see that l is well-defined and that the least model $T_P^+ \uparrow \omega$ for P has the desired properties.

Conversely, if M is a two-valued model for P which satisfies the given condition for some mapping $l : B_P \to \alpha$, then it is easy to show, by induction on $l(A)$, that $A \in M$ implies $A \in T_P^+ \uparrow (l(A)+1)$. This yields that $M \subseteq T_P^+ \uparrow \omega$, and hence that $M = T_P^+ \uparrow \omega$ by minimality of the model $T_P^+ \uparrow \omega$.

The following result is due to [15], and is striking in its similarity to Theorem 1.

Theorem 2. *Let P be normal. Then a two-valued model $M \subseteq B_P$ of P is a stable model of P if and only if there exists a (total) level mapping $l : B_P \to \alpha$ such that for each $A \in M$ there exists $A \leftarrow A_1, \dots, A_n \neg B_1, \dots, \neg B_m$ in* $\mathsf{ground}(P)$ *with $A_i \in M$, $B_j \notin M$, and $l(A) > l(A_i)$ for all $i = 1, \dots, n$ and $j = 1, \dots, m$.*

We next recall the following alternative characterization of the Fitting model, due to [1,2].

Definition 1. *Let P be a normal logic program, I be a model of P, and l be an I-partial level mapping for P. We say that P* satisfies *(F)* with respect to I and *l, if each $A \in$* $\mathsf{dom}(l)$ *satisfies one of the following conditions.*

(Fi) *$A \in I$ and there exists a clause $A \leftarrow L_1, \dots, L_n$ in* $\mathsf{ground}(P)$ *such that $L_i \in I$ and $l(A) > l(L_i)$ for all i.*
(Fii) *$\neg A \in I$ and for each clause $A \leftarrow L_1, \dots, L_n$ in* $\mathsf{ground}(P)$ *there exists i with $\neg L_i \in I$ and $l(A) > l(L_i)$.*

Theorem 3. *Let P be a normal logic program with Fitting model M. Then M is the greatest model among all models I, for which there exists an I-partial level mapping l for P such that P satisfies (F) with respect to I and l.*

Let us recall next the definition of a (locally) stratified program, due to [8,9]: A normal logic program is called *locally stratified* if there exists a (total) level mapping $l : B_P \to \alpha$, for some ordinal α, such that for each clause $A \leftarrow A_1, \dots, A_n, \neg B_1, \dots, \neg B_m$ in $\mathsf{ground}(P)$ we have that $l(A) \geq l(A_i)$ and $l(A) > l(B_j)$ for all $i = 1, \dots, n$ and $j = 1, \dots, m$. The notion of (locally) stratifed program was developed with the idea of preventing *recursion through negation*, while allowing recursion through positive dependencies. (Locally) stratified programs have total well-founded models.

There exist locally stratified programs which do not have a total Fitting semantics and vice versa — just consider the programs consisting of the single clauses $p \leftarrow p$, respectively, $p \leftarrow \neg p, q$. In fact, condition (Fii) requires a strict decrease of level between the head and a literal in the rule, independent of this literal being positive or negative. But, on the other hand, condition (Fii) imposes no further restrictions on the remaining body literals, while the notion of local stratification does. These considerations motivate the substitution of condition (Fii) by the condition (Cii), as done for the following definition.

Definition 2. *Let P be a normal logic program, I be a model of P, and l be an I-partial level mapping for P. We say that P* satisfies *(WF)* with respect to I and *l, if each $A \in$* $\mathsf{dom}(l)$ *satisfies (Fi) or the following condition.*

(Cii) *$\neg A \in I$ and for each clause $A \leftarrow A_1, \dots, A_n, \neg B_1, \dots, \neg B_m$ contained in* $\mathsf{ground}(P)$ *(at least) one of the following conditions holds:*
(Ciia) *There exists $i \in \{1, \dots, n\}$ with $\neg A_i \in I$ and $l(A) \geq l(A_i)$.*
(Ciib) *There exists $j \in \{1, \dots, m\}$ with $B_j \in I$ and $l(A) > l(B_j)$.*

So, in the light of Theorem 3, Definition 2 should provide a natural "stratified version" of the Fitting semantics. And indeed it does, and furthermore, the resulting semantics coincides with the well-founded semantics, which is a very satisfactory result from [1,2].

Theorem 4. *Let P be a normal logic program with well-founded model M. Then M is the greatest model among all models I, for which there exists an I-partial level mapping l for P such that P satisfies (WF) with respect to I and l.*

For completeness, we remark that an alternative characterization of the weakly perfect model semantics [21] can also be found in [1,2].

The approach which led to the results just mentioned, originally put forward in [1,2], provides a methodology for obtaining uniform characterizations of different semantics for logic programs.

3 Maximally Circular Stable Semantics

We note that condition (Fi) has been reused in Definition 2. Thus, Definition 1 has been "stratified" only with respect to condition (Fii), yielding (Cii), but not with respect to (Fi). Indeed, also replacing (Fi) by a stratified version such as the following seems not satisfactory at first sight.

(Ci) $A \in I$ and there exists a clause $A \leftarrow A_1, \dots, A_n, \neg B_1, \dots, \neg B_m$ in $\mathsf{ground}(P)$ such that $A_i, \neg B_j \in I$, $l(A) \geq l(A_i)$, and $l(A) > l(B_j)$ for all i and j.

If we replace condition (Fi) by condition (Ci) in Definition 2, then it is not guaranteed that for any given program there is a greatest model satisfying the desired properties, as the following example from [1,2] shows.

Example 1. Consider the program consisting of the two clauses $p \leftarrow p$ and $q \leftarrow \neg p$, and the two (total) models $M_1 = \{p, \neg q\}$ and $M_2 = \{\neg p, q\}$, which are incomparable, and the level mapping l with $l(p) = 0$ and $l(q) = 1$.

In order to arrive at an understanding of this asymmetry, we consider the setting with conditions (Ci) and (Fii), which is somehow "dual" to the well-founded semantics which is characterized by (Fi) and (Cii).

Definition 3. *Let P be a normal logic program, I be a model of P, and l be an I-partial level mapping for P. We say that P* satisfies *(CW)* with respect to *I* and *l, if each $A \in \mathsf{dom}(l)$ satisfies (Ci) or (Fii).*

By virtue of Definition 3 we will be able to develop a theory which complements the restults from Section 2. We will first characterize the greatest model of a definite program analogously to Theorem 1.

Theorem 5. *Let P be a definite program. Then there is a unique two-valued supported interpretation M of P for which there exists a (total) level mapping $l : B_P \to \alpha$ such that for each atom $A \notin M$ and for all clauses $A \leftarrow A_1, \ldots, A_n$ in $\mathsf{ground}(P)$ there is some $A_i \notin M$ with $l(A) > l(A_i)$. Furthermore, M is the greatest two-valued model of P.*

Proof. Let M be the greatest two-valued model of P, and let α be the least ordinal such that $M = T_P^+ \downarrow \alpha$. Define $l : B_P \to \alpha$ by setting $l(A) = \min\{\gamma \mid A \notin T_P^+ \downarrow (\gamma + 1)\}$ for $A \notin M$, and by setting $l(A) = 0$ if $A \in M$. The mapping l is well-defined because $A \notin M$ with $A \notin T_P^+ \downarrow \gamma = \bigcap_{\beta<\gamma} T_P^+ \downarrow \beta$ for some limit ordinal γ implies $A \notin T_P^+ \downarrow \beta$ for some $\beta < \gamma$. So the least ordinal β with $A \notin T_P^+ \downarrow \beta$ is always a successor ordinal. Now assume that there is $A \notin M$ which does not satisfy the stated condition. We can furthermore assume without loss of generality that A is chosen with this property such that $l(A)$ is minimal. Let $A \leftarrow A_1, \ldots, A_n$ be a clause in $\mathsf{ground}(P)$. Since $A \notin T_P^+ \left(T_P^+ \downarrow l(A)\right)$ we obtain $A_i \notin T_P^+ \downarrow l(A) \supseteq M$ for some i. But then $l(A_i) < l(A)$ which contradicts minimality of $l(A)$.

Conversely, let M be a two-valued model for P which satisfies the given condition for some mapping $l : B_P \to \alpha$. We show by transfinite induction on $l(A)$ that $A \notin M$ implies $A \notin T_P^+ \downarrow (l(A) + 1)$, which suffices because it implies that for the greatest two-valued model $T_P^+ \downarrow \beta$ of P we have that $T_P^+ \downarrow \beta \subseteq M$, and therefore $T_P^+ \downarrow \beta = M$. For the inductive proof consider first the case where $l(A) = 0$. Then there is no clause in $\mathsf{ground}(P)$ with head A and consequently $A \notin T_P^+ \downarrow 1 = T_P^+(B_P)$. Now assume that the statement to be proven holds for all $B \notin M$ with $l(B) < \alpha$, where α is some ordinal, and let $A \notin M$ with $l(A) = \alpha$. Then each clause in $\mathsf{ground}(P)$ with head A contains an atom B with $l(B) = \beta < \alpha$ and $B \notin M$. Hence $B \notin T_P^+ \downarrow (\beta + 1)$ and consequently $A \notin T_P^+ \downarrow (\alpha + 1)$.

The following definition and theorem are analogous to Theorem 2.

Definition 4. *Let P be normal. Then $M \subseteq B_P$ is called a* maximally circular stable model *(*maxstable model*) of P if it is a two-valued supported interpretation of P and there exists a (total) level mapping $l : B_P \to \alpha$ such that for each atom $A \notin M$ and for all clauses $A \leftarrow A_1, \ldots, A_n, \neg B_1, \ldots, \neg B_m$ in $\mathsf{ground}(P)$ with $B_1, \ldots, B_m \notin M$ there is some $A_i \notin M$ with $l(A) > l(A_i)$.*

Theorem 6. *$M \subseteq B_P$ is a maxstable model of P if and only if $M = \mathsf{gfp}\left(T_{P/M}^+\right)$.*

Proof. First note that every maxstable model is a a supported model. Indeed supportedness follows immediately from the definition. Now assume that M is maxstable but is not a model, i.e. there is $A \notin M$ but there is a clause $A \leftarrow A_1, \ldots, A_n$ in $\mathsf{ground}(P)$ with $A_i \in M$ for all i. But by the definition of maxstable model we must have that there is $A_i \notin M$, which contradicts $A_i \in M$.

Now let M be a maxstable model of P. Let $A \notin M$ and let $T^+_{P/M}\downarrow\alpha = \mathsf{gfp}\left(T^+_{P/M}\right)$. We show by transfinite induction on $l(A)$ that $A \notin T^+_{P/M}\downarrow(l(A)+1)$ and hence $A \notin T^+_{P/M}\downarrow\alpha$. For $l(A)=0$ there is no clause with head A in P/M, so $A \notin T^+_{P/M}\downarrow 1$. Now let $l(A)=\beta$ for some ordinal β. By assumption we have that for all clauses $A \leftarrow A_1,\dots,A_n,\neg B_1,\dots,\neg B_m$ with $B_1,\dots,B_m \notin M$ there exists $A_i \notin M$ with $l(A) > l(A_i)$, say $l(A_i)=\gamma<\beta$. Hence $A_i \notin T^+_{P/M}\downarrow(\gamma+1)$, and consequently $A \notin T^+_{P/M}\downarrow(\beta+1)$, which shows that $\mathsf{gfp}\left(T^+_{P/M}\right)\subseteq M$.

So let again M be a maxstable model of P and let $A \notin \mathsf{gfp}\left(T^+_{P/M}\right) = T^+_{P/M}\downarrow\alpha$ and $l(A)=\beta$. Then for each clause $A \leftarrow A_1,\dots,A_n$ in P/M there is A_i with $A_i \notin T^+_{P/M}\downarrow\alpha$ and $l(A) > l(A_i)$. Now assume $A \in M$. Without loss of generality we can furthermore assume that A is chosen such that $l(A)=\beta$ is minimal. Hence $A_i \notin M$, and we obtain that for each clause in P/M with head A one of the corresponding body atoms is false in M. By supportedness of M this yields $A \notin M$, which contradicts our assumption. Hence $A \notin M$ as desired.

Conversely, let $M = \mathsf{gfp}\left(T^+_{P/M}\right)$. Then as an immediate consequence of Theorem 5 we obtain that M is maxstable.

4 Maximally Circular Well-Founded Semantics

Maxstable models are formally analogous[4] to stable models in that the former are fixed points of the operator $I \mapsto \mathsf{gfp}\left(T^+_{P/I}\right)$, while the latter are fixed points of the operator $I \mapsto \mathsf{lfp}\left(T^+_{P/I}\right)$. Further, in analogy to the alternating fixed point characterization of the well-founded model, we can obtain a corresponding variant of the well-founded semantics, which we will do next. Theorem 6 suggests the defininition of the following operator.

Definition 5. *Let P be a normal program and I be a two-valued interpretation. Then define* $\mathrm{CGL}_P(I) = \mathsf{gfp}\left(T^+_{P/I}\right)$.

Using the operator CGL_P, we can define a "maximally circular" version of the alternating fixed-point semantics.

Proposition 1. *Let P be a normal program. Then the following hold.*

(i) CGL_P *is antitonic and* CGL_P^2 *is monotonic.*
(ii) $\mathrm{CGL}_P\left(\mathsf{lfp}\left(\mathrm{CGL}_P^2\right)\right) = \mathsf{gfp}\left(\mathrm{CGL}_P^2\right)$ *and* $\mathrm{CGL}_P\left(\mathsf{gfp}\left(\mathrm{CGL}_P^2\right)\right) = \mathsf{lfp}\left(\mathrm{CGL}_P^2\right)$.

Proof. (i) If $I \subseteq J \in B_P$, then $P/J \subseteq P/I$ and consequently $\mathrm{CGL}_P(J) = \mathsf{gfp}\left(T^+_{P/J}\right) \subseteq \mathsf{gfp}\left(T^+_{P/I}\right) = \mathrm{CGL}_P(I)$. Monotonicity of CGL_P^2 then follows trivially.

[4] The term *dual* seems not to be entirely adequate in this situation, although it is intuitionally appealing.

(ii) Let $L_P = \mathsf{lfp}\left(\mathrm{CGL}_P^2\right)$ and $G_P = \mathsf{gfp}\left(\mathrm{CGL}_P^2\right)$. Then we can calculate $\mathrm{CGL}_P^2(\mathrm{CGL}_P(L_P)) = \mathrm{CGL}_P\left(\mathrm{CGL}_P^2(L_P)\right) = \mathrm{CGL}_P(L_P)$, so $\mathrm{CGL}_P(L_P)$ is a fixed point of CGL_P^2, and hence $L_P \subseteq \mathrm{CGL}_P(L_P) \subseteq G_P$. Similarly, $L_P \subseteq \mathrm{CGL}_P(G_P) \subseteq G_P$. Since $L_P \subseteq G_P$ we get from the antitonicity of CGL_P that $L_P \subseteq \mathrm{CGL}_P(G_P) \subseteq \mathrm{CGL}_P(L_P) \subseteq G_P$. Similarly, since $\mathrm{CGL}_P(L_P) \subseteq G_P$, we obtain $\mathrm{CGL}_P(G_P) \subseteq \mathrm{CGL}_P^2(L_P) = L_P \subseteq \mathrm{CGL}_P(G_P)$, so $\mathrm{CGL}_P(G_P) = L_P$, and also $G_P = \mathrm{CGL}_P^2(G_P) = \mathrm{CGL}_P(L_P)$.

We will now define an operator for the maximally circular well-founded semantics. Given a normal logic program P and some $I \in I_P$, we say that $S \subseteq B_P$ is a *self-founded set (of P) with respect to I* if $S \cup I \in I_P$ and each atom $A \in S$ satisfies the following condition: There exists a clause $A \leftarrow \mathsf{body}$ in $\mathsf{ground}(P)$ such that one of the following holds.

(Si) body is true in I.
(Sii) Some (non-negated) atoms in body occur in S and all other literals in body are true in I.

Self-founded sets are analogous[5] to unfounded sets, and the following proposition holds.

Proposition 2. *Let P be a normal program and let $I \in I_P$. Then there exists a greatest self-founded set of P with respect to I.*

Proof. If $(S_i)_{i\in\mathcal{I}}$ is a family of sets each of which is a self-founded set of P with respect to I, then it is easy to see that $\bigcup_{i\in\mathcal{I}} S_i$ is also a self-founded set of P with respect to I.

Given a normal program P and $I \in I_P$, let $S_P(I)$ be the greatest self-founded set of P with respect to I, and define the operator CW_P on I_P by

$$\mathrm{CW}_P(I) = S_P(I) \cup \neg F_P(I).$$

Proposition 3. *The operator CW_P is well-defined and monotonic.*

Proof. For well-definedness, we have to show that $S_P(I) \cap F_P(I) = \emptyset$ for all $I \in I_P$. So assume there is $A \in S_P(I) \cap F_P(I)$. From $A \in F_P(I)$ we obtain that for each clause with head A there is a corresponding body literal L which is false in I. From $A \in S_P(I)$, more precisely from (Sii), we can furthermore conclude that L is an atom and $L \in S_P(I)$. But then $\neg L \in I$ and $L \in S_P(I)$ which is impossible by definition of self-founded set which requires that $S_P(I) \cup I \in I_P$. So $S_P(I) \cap F_P(I) = \emptyset$ and CW_P is well-defined.

For monotonicity, let $I \subseteq J \in I_P$ and let $L \in \mathrm{CW}_P(I)$. If $L = \neg A$ is a negated atom, then $A \in F_P(I)$ and all clauses with head A contain a body literal which is false in I, hence in J, and we obtain $A \in F_P(J)$. If $L = A$ is an atom, then $A \in S_P(I)$ and there exists a clause $A \leftarrow \mathsf{body}$ in $\mathsf{ground}(P)$ such

[5] Again, it is not really a duality.

that (at least) one of (Si) or (Sii) holds. If (Si) holds, then body is true in I, hence in J, and $A \in S_P(J)$. If (Sii) holds, then some non-negated atoms in body occur in S and all other literals in body are true in I, hence in J, and we obtain $A \in S_P(J)$.

The following theorem relates our previous observations to Definition 3, in perfect analogy to the correspondence between the stable model semantics, Theorem 1, Fages's characterization from Theorem 2, the well-founded semantics, and the alternating fixed point characterization.

Theorem 7. *Let P be a normal program and $M_P = \mathsf{lfp}(\mathrm{CW}_P)$. Then the following hold.*

(i) M_P is the greatest model among all models I of P such that there is an I-partial level mapping l for P such that P satisfies (CW) with respect to I and l.
(ii) $M_P = \mathsf{lfp}\left(\mathrm{CGL}_P^2\right) \cup \neg\left(B_P \setminus \mathsf{gfp}\left(\mathrm{CGL}_P^2\right)\right)$.

Proof. (i) Let $M_P = \mathsf{lfp}(\mathrm{CW}_P)$ and define the M_P-partial level mapping l_P as follows: $l_P(A) = \alpha$, where α is the least ordinal such that A is not undefined in $\mathrm{CW}_P\!\uparrow\!(\alpha+1)$. The proof will be established by showing the following facts: (1) P satisfies (CW) with respect to M_P and l_P. (2) If I is a model of P and l is an I-partial level mapping such that P satisfies (CW) with respect to I and l, then $I \subseteq M_P$.

(1) Let $A \in \mathsf{dom}(l_P)$ and $l_P(A) = \alpha$. We consider two cases.

(Case i) If $A \in M_P$, then $A \in S_P(\mathrm{CW}_P \uparrow \alpha)$, hence there exists a clause $A \leftarrow \mathsf{body}$ in $\mathsf{ground}(P)$ such that (Si) or (Sii) holds with respect to $\mathrm{CW}_P\!\uparrow\!\alpha$. If (Si) holds, then all literals in body are true in $\mathrm{CW}_P\!\uparrow\!\alpha$, hence have level less than $l_P(A)$ and (Ci) is satisfied. If (Sii) holds, then some non-negated atoms from body occur in $S_P(\mathrm{CW}_P\!\uparrow\!\alpha)$, hence have level less than or equal to $l_P(A)$, and all remaining literals in body are true in $\mathrm{CW}_P\!\uparrow\!\alpha$, hence have level less than $l_P(A)$. Consequently, A satisfies (Ci) with respect to M_P and l_P.

(Case ii) If $\neg A \in M_P$, then $A \in F_P(\mathrm{CW}_P\!\uparrow\!\alpha)$, hence for all clauses $A \leftarrow \mathsf{body}$ in $\mathsf{ground}(P)$ there exists $L \in \mathsf{body}$ with $\neg L \in \mathrm{CW}_P\!\uparrow\!\alpha$ and $l_P(L) < \alpha$, hence $\neg L \in M_P$. Consequently, A satisfies (Fii) with respect to M_P and l_P, and we have established that fact (1) holds.

(2) We show via transfinite induction on $\alpha = l(A)$, that whenever $A \in I$ (respectively, $\neg A \in I$), then $A \in \mathrm{CW}_P \uparrow (\alpha+1)$ (respectively, $\neg A \in \mathrm{CW}_P \uparrow (\alpha+1)$). For the base case, note that if $l(A) = 0$, then $\neg A \in I$ implies that there is no clause with head A in $\mathsf{ground}(P)$, hence $\neg A \in \mathrm{CW}_P \uparrow 1$. If $A \in I$ then consider the set S of all atoms B with $l(B) = 0$ and $B \in I$. We show that S is a self-founded set of P with respect to $\mathrm{CW}_P\!\uparrow\!0 = \emptyset$, and this suffices since it implies $A \in \mathrm{CW}_P\!\uparrow\!1$ by the fact that $A \in S$. So let $C \in S$. Then $C \in I$ and C satisfies condition (Ci) with respect to I and l, and since $l(C) = 0$, we have that there is a definite clause with head C whose body atoms (if it has any) are all of level 0 and contained in I. Hence condition (Sii) (or (Si)) is satisfied for this clause and S is a self-founded set of P with respect to I. So assume now that

the induction hypothesis holds for all $B \in B_P$ with $l(B) < \alpha$, and let A be such that $l(A) = \alpha$. We consider two cases.

(Case i) If $A \in I$, consider the set S of all atoms B with $l(B) = \alpha$ and $B \in I$. We show that S is a self-founded set of P with respect to $\mathrm{CW}_P \uparrow \alpha$, and this suffices since it implies $A \in \mathrm{CW}_P \uparrow (\alpha + 1)$ by the fact that $A \in S$. First note that $S \subseteq I$, so $S \cup I \in I_P$. Now let $C \in S$. Then $C \in I$ and C satisfies condition (Ci) with respect to I and l, so there is a clause $A \leftarrow A_1, \ldots, A_n, \neg B_1, \ldots, \neg B_m$ in $\mathsf{ground}(P)$ such that $A_i, \neg B_j \in I$, $l(A) \geq l(A_i)$, and $l(A) > l(B_j)$ for all i and j. By induction hypothesis we obtain $\neg B_j \in \mathrm{CW}_P \uparrow \alpha$. If $l(A_i) < l(A)$ for some A_i then we have $A_i \in \mathrm{CW}_P \uparrow \alpha$, also by induction hypothesis. If there is no A_i with $l(A_i) = l(A)$, then (Si) holds, while $l(A_i) = l(A)$ implies $A_i \in S$, so (Sii) holds.

(Case ii) If $\neg A \in I$, then A satisfies (Fii) with respect to I and l. Hence for all clauses $A \leftarrow \mathsf{body}$ in $\mathsf{ground}(P)$ we have that there is $L \in \mathsf{body}$ with $\neg L \in I$ and $l(L) < \alpha$. Hence for all these L we have $\neg L \in \mathrm{CW}_P \uparrow \alpha$ by induction hypothesis, and consequently for all clauses $A \leftarrow \mathsf{body}$ in $\mathsf{ground}(P)$ we obtain that body is false in $\mathrm{CW}_P \uparrow \alpha$ which yields $\neg A \in \mathrm{CW}_P \uparrow (\alpha + 1)$. This establishes fact (2) and concludes the proof of (i).

(ii) We first introduce some notation. Let

$$\begin{aligned}
L_0 &= \emptyset, & G_0 &= B_P, \\
L_{\alpha+1} &= \mathrm{CGL}_P(G_\alpha), & G_{\alpha+1} &= \mathrm{CGL}_P(L_\alpha) \qquad \text{for any ordinal } \alpha, \\
L_\alpha &= \bigcup_{\beta<\alpha} L_\beta, & G_\alpha &= \bigcap_{\beta<\alpha} G_\beta \qquad \text{for limit ordinal } \alpha, \\
L_P &= \mathsf{lfp}(\mathrm{CGL}_P^2), & G_P &= \mathsf{gfp}(\mathrm{CGL}_P^2).
\end{aligned}$$

By transfinite induction, it is easily checked that $L_\alpha \subseteq L_\beta \subseteq G_\beta \subseteq G_\alpha$ whenever $\alpha \leq \beta$. So $L_P = \bigcup L_\alpha$ and $G_P = \bigcap G_\alpha$.

Let $M = L_P \cup \neg(B_P \setminus G_P)$. We intend to apply (i) and first define an M-partial level mapping l. We will take as image set of l, pairs (α, γ) of ordinals, with the lexicographic ordering. This can be done without loss of generality since any set of such pairs, under the lexicographic ordering, is well-ordered, and therefore order-isomorphic to an ordinal. For $A \in L_P$, let $l(A)$ be the pair $(\alpha, 0)$, where α is the least ordinal such that $A \in L_{\alpha+1}$. For $B \notin G_P$, let $l(B)$ be the pair (β, γ), where β is the least ordinal such that $B \notin G_{\beta+1}$, and γ is least such that $B \notin T_{P/L_\beta} \downarrow \gamma$. It is easily shown that l is well-defined, and we show next by transfinite induction that P satisfies (CW) with respect to M and l.

Let $A \in L_1 = \mathsf{gfp}\left(T^+_{P/B_P}\right)$. Since P/B_P contains exactly all clauses from $\mathsf{ground}(P)$ which contain no negation, we have that A is contained in the greatest two-valued model of a definite subprogram of P, namely P/B_P. So there must be a definite clause in $\mathsf{ground}(P)$ with head A whose corresponding body atoms are also true in L_1, which, by definition of l, must have the same level as A, hence (Ci) is satisfied. Now let $\neg B \in \neg(B_P \setminus G_P)$ such that $B \in (B_P \setminus G_1) = B_P \setminus \mathsf{gfp}\left(T^+_{P/\emptyset}\right)$. Since $P/\emptyset$ contains all clauses from $\mathsf{ground}(P)$ with all negative literals removed, we obtain that B is not contained in the greatest two-valued model of the definite program $P/\emptyset$, and (Fii) is satisfied by Theorem 5 using a simple induction argument.

Assume now that, for some ordinal α, we have shown that A satisfies (CW) with respect to M and l for all $A \in B_P$ with $l(A) < (\alpha, 0)$.

Let $A \in L_{\alpha+1} \setminus L_\alpha = \mathsf{gfp}\left(T^+_{P/G_\alpha}\right) \setminus L_\alpha$. Then $A \in \left(T^+_{P/G_\alpha}{\downarrow}\gamma\right) \setminus L_\alpha$ for some γ; note that all (negative) literals which were removed by the Gelfond-Lifschitz transformation from clauses with head A have level less than $(\alpha, 0)$. Then A satisfies (Ci) with respect to M and l by definition of l.

Let $A \in (B_P \setminus G_{\alpha+1}) \cap G_\alpha$. Then $A \notin \mathsf{gfp}\left(T^+_{P/L_\alpha}\right)$ and we conclude again from Theorem 5, using a simple induction argument, that A satisfies (CW) with respect to M and l.

This finishes the proof that P satisfies (CW) with respect to M and l. It remains to show that M is greatest with this property.

So assume that $M_1 \supset M$ is the greatest model such that P satisfies (CW) with respect to M_1 and some M_1-partial level mapping l_1. Assume $L \in M_1 \setminus M$ and, without loss of generality, let the literal L be chosen such that $l_1(L)$ is minimal. We consider two cases.

(Case i) If $L = \neg A \in M_1 \setminus M$ is a negated atom, then by (Fii) for each clause $A \leftarrow L_1, \ldots, L_n$ in $\mathsf{ground}(P)$ there exists i with $\neg L_i \in M_1$ and $l_1(A) > l_1(L_i)$. Hence, $\neg L_i \in M$ and consequently for each clause $A \leftarrow \mathsf{body}$ in P/L_P we have that some atom in body is false in $M = L_P \cup \neg(B_P \setminus G_P)$. But then $A \notin \mathrm{CGL}_P(L_P) = G_P$, hence $\neg A \in M$, contradicting $\neg A \in M_1 \setminus M$.

(Case ii) If $L = A \in M_1 \setminus M$ is an atom, then $A \notin M = L_P \cup \neg(B_P \setminus G_P)$ and in particular $A \notin L_P = \mathsf{gfp}\left(T^+_{P/G_P}\right)$. Hence $A \notin T^+_{P/G_P}{\downarrow}\gamma$ for some γ, which can be chosen to be least with this property. We show by induction on γ that this leads to a contradiction, to finish the proof.

If $\gamma = 1$, then there is no clause with head A in P/G_P, i.e. for all clauses $A \leftarrow \mathsf{body}$ in $\mathsf{ground}(P)$ we have that body is false in M, hence in M_1, which contradicts $A \in M_1$.

Now assume that there is no $B \in M_1 \setminus M$ with $B \notin T^+_{P/G_P} \downarrow \delta$ for any $\delta < \gamma$, and let $A \in M_1 \setminus M$ with $A \notin T^+_{P/G_P} \downarrow \gamma$, which implies that γ is a successor ordinal. By $A \in M_1$ and (Ci) there must be a clause $A \leftarrow A_1, \ldots, A_n \neg B_1, \ldots, \neg B_m$ in $\mathsf{ground}(P)$ with $A_i, \neg B_j \in M_1$ for all i and j. However, since $A \notin T^+_{P/G_P} \downarrow \gamma$ we obtain that for each $A \leftarrow A_1, \ldots, A_n$ in P/G_P, hence for each $A \leftarrow A_1, \ldots, A_n, \neg B_1, \ldots, \neg B_m$ in $\mathsf{ground}(P)$ with $\neg B_1, \ldots, \neg B_m \in \neg(B_P \setminus G_P) \subseteq M \subseteq M_1$ there is A_i with $A_i \notin T^+_{P/G_P} \downarrow (\gamma - 1) \subseteq M$, and by induction hypothesis we obtain $A_i \notin M_1$. So $A_i \in M_1$ and $A_i \notin M_1$, which is a contradiction and concludes the proof.

Definition 6. *For a normal program P, we call* $\mathsf{lfp}(\mathrm{CW}_P)$ *the* maximally circular well-founded model *(*maxwf model*) of P.*

5 Conclusions and Further Work

We have displayed a coherent picture of different semantics for normal logic programs. We have added to well-known results new ones which complete the formerly incomplete picture of relationships. The richness of theory and relationships turns out to be very appealing and satisfactory. From a mathematical perspective one expects major notions in a field to be strongly and cleanly interconnected, and it is fair to say that this is the case for declarative semantics for normal logic programs.

The situation becomes much more difficult when discussing extensions of the logic programming paradigm like disjunctive [22], quantitative [23], or dynamic [24] logic programming. For many of these extensions it is as yet to be determined what the best ways of providing declarative semantics for these frameworks are, and the lack of interconnections between the different proposals in the literature provides an argument for the case that no satisfactory answers have yet been found.

We believe that successful proposals for extensions will have to exhibit similar interrelationships as observed for normal programs. How, and if, this can be achieved, however, is as yet rather uncertain. Formal studies like the one in this paper may help in designing satisfactory semantics, but a discussion of this is outside the scope of our exhibition, and will be pursued elsewhere.

References

1. Hitzler, P., Wendt, M.: The well-founded semantics is a stratified Fitting semantics. In Jarke, M., Koehler, J., Lakemeyer, G., eds.: Proceedings of the 25th Annual German Conference on Artificial Intelligence, KI2002, Aachen, Germany, September 2002. Volume 2479 of Lecture Notes in Artificial Intelligence., Springer, Berlin (2002) 205–221
2. Hitzler, P., Wendt, M.: A uniform approach to logic rogramming semantics. Technical Report WV–02–14, Knowledge Representation and Reasoning Group, Artificial Intelligence Institute, Department of Computer Science, Dresden University of Technology, Dresden, Germany (2002) Submitted.
3. Reiter, R.: A logic for default reasoning. Artificial Intelligence **13** (1980) 81–132
4. Lloyd, J.W.: Foundations of Logic Programming. Springer, Berlin (1988)
5. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In Kowalski, R.A., Bowen, K.A., eds.: Logic Programming. Proceedings of the 5th International Conference and Symposium on Logic Programming, MIT Press (1988) 1070–1080
6. van Gelder, A., Ross, K.A., Schlipf, J.S.: The well-founded semantics for general logic programs. Journal of the ACM **38** (1991) 620–650
7. Fitting, M.: A Kripke-Kleene-semantics for general logic programs. The Journal of Logic Programming **2** (1985) 295–312
8. Apt, K.R., Blair, H.A., Walker, A.: Towards a theory of declarative knowledge. In Minker, J., ed.: Foundations of Deductive Databases and Logic Programming. Morgan Kaufmann, Los Altos, CA (1988) 89–148

9. Przymusinski, T.C.: On the declarative semantics of deductive databases and logic programs. In Minker, J., ed.: Foundations of Deductive Databases and Logic Programming. Morgan Kaufmann, Los Altos, CA (1988) 193–216
10. Fitting, M.: Fixpoint semantics for logic programming — A survey. Theoretical Computer Science **278** (2002) 25–51
11. Denecker, M., Marek, V.W., Truszczynski, M.: Approximating operators, stable operators, well-founded fixpoints and applications in non-monotonic reasoning. In Minker, J., ed.: Logic-based Artificial Intelligence. Kluwer Academic Publishers, Boston (2000) 127–144
12. Przymusinski, T.C.: Well-founded semantics coincides with three-valued stable semantics. Fundamenta Informaticae **13** (1989) 445–464
13. van Gelder, A.: The alternating fixpoint of logic programs with negation. In: Proceedings of the Eighth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Philadelphia, Pennsylvania, ACM Press (1989) 1–10
14. Dung, P.M., Kanchanasut, K.: A fixpoint approach to declarative semantics of logic programs. In Lusk, E.L., Overbeek, R.A., eds.: Logic Programming, Proceedings of the North American Conference 1989, NACLP'89, Cleveland, Ohio, MIT Press (1989) 604–625
15. Fages, F.: Consistency of Clark's completion and existence of stable models. Journal of Methods of Logic in Computer Science **1** (1994) 51–60
16. Wendt, M.: Unfolding the well-founded semantics. Journal of Electrical Engineering, Slovak Academy of Sciences **53** (2002) 56–59 (Proceedings of the 4th Slovakian Student Conference in Applied Mathematics, Bratislava, April 2002).
17. Hitzler, P.: Circular belief in logic programming semantics. Technical Report WV–02–13, Knowledge Representation and Reasoning Group, Artificial Intelligence Institute, Department of Computer Science, Dresden University of Technology, Dresden, Germany (2002)
18. Jachymski, J.: Order-theoretic aspects of metric fixed-point theory. In Kirk, W.A., Sims, B., eds.: Handbook of Metric Fixed Point Theory. Kluwer Academic Publishers, Dordrecht, The Netherlands (2001) 613–641
19. Fitting, M.: Bilattices and the semantics of logic programming. The Journal of Logic Programming **11** (1991) 91–116
20. Abramsky, S., Jung, A.: Domain theory. In Abramsky, S., Gabbay, D., Maibaum, T.S., eds.: Handbook of Logic in Computer Science. Volume 3. Clarendon, Oxford (1994)
21. Przymusinska, H., Przymusinski, T.C.: Weakly stratified logic programs. Fundamenta Informaticae **13** (1990) 51–65
22. Wang, K.: A comparative study of well-founded semantics for disjunctive logic programs. In Eiter, T., Faber, W., Truszczynski, M., eds.: Logic Programming and Nonmonotonic Reasoning, 6th International Conference, LPNMR 2001, Vienna, Austria, September 17–19, 2001, Proceedings. Volume 2173 of Lecture Notes in Artificial Intelligence., Springer (2001) 133–146
23. Mateis, C.: Quantitative disjunctive logic programming: Semantics and computation. AI communications **13** (2000) 225–248
24. Leite, J.A.: Evolving Knowledge Bases. Volume 81 of Frontiers of Artificial Intelligence and Applications. IOS Press (2003)

Eingereicht beim 19. Workshop on (Constraint) Logic Programming, Ulm, Februar 2005.

Level Mapping Characterizations of Selector Generated Models for Logic Programs

Pascal Hitzler[1]* and Sibylle Schwarz[2]

[1] Fakultät für Informatik, Technische Universität Dresden
email: phitzler@inf.tu-dresden.de
[2] Institut für Informatik, Martin-Luther-Universität Halle-Wittenberg
email: schwarzs@informatik.uni-halle.de

Technical Report WV–04–04
Wissensverarbeitung, Fakultät für Informatik, TU Dresden

Abstract. Assigning semantics to logic programs via selector generated models (Schwarz 2002/2003) extends several semantics, like the stable, the inflationary, and the stable generated semantics, to programs with arbitrary formulae in rule heads and bodies. We study this approach by means of a unifying framework for characterizing different logic programming semantics using level mappings (Hitzler and Wendt 200x, Hitzler 2003), thereby supporting the claim that this framework is very flexible and applicable to very diversely defined semantics.

1 Introduction

Hitzler and Wendt [8–10] have recently proposed a unifying framework for different logic programming semantics. This approach is very flexible and allows to cast semantics of very different origin and style into uniform characterizations using level mappings, i.e. mappings from atoms to ordinals, in the spirit of the definition of acceptable programs [2], the use of stratification [1, 13] and a characterization of stable models by Fages [3]. These characterizations display syntactic and semantic dependencies between language elements by means of the preorders on ground atoms induced by the level mappings, and thus allow inspection of and comparison between different semantics, as exhibited in [8–10].

For the syntactically restricted class of normal logic programs, the most important semantics — and some others — have already been characterized and compared, and this was spelled out in [8–10]. Due to the inherent flexibility of the framework, it is clear that studies of extended syntax are also possible, but have so far not been carried out. In this paper, we will present a non-trivial technical result which provides a first step towards a comprehensive comparative study of different semantics for logic programs under extended syntax.

* This author acknowledges the hospitality of the Graduiertenkolleg *Wissensverarbeitung* at the University of Leipzig, Germany, for their hospitality while working on a first draft of this paper.

Table 1. Notions of specific types of rules.

rule is called	*set*	*condition*
definite	LP	$\mathsf{body}(r) \in \mathsf{Lg}(\{\wedge, \mathbf{t}\}, A)$ and $\mathsf{head}(r) \in A$
normal	NLP	$\mathsf{body}(r) \in \mathsf{Lg}(\{\wedge, \mathbf{t}\}, \mathsf{Lit}(A))$ and $\mathsf{head}(r) \in A$
head-atomic	HALP	$\mathsf{body}(r) \in \mathsf{Lg}\left(\Sigma^{cl} \setminus \{\mathbf{f}\}, A\right)$ and $\mathsf{head}(r) \in A$
pos. head disj.	DLP^+	$\mathsf{body}(r) \in \mathsf{Lg}(\{\wedge, \mathbf{t}\}, \mathsf{Lit}(A))$ and $\mathsf{head}(r) \in \mathsf{Lg}(\{\vee\}, A)$
disjunctive	DLP	$\mathsf{body}(r) \in \mathsf{Lg}(\{\wedge, \mathbf{t}\}, \mathsf{Lit}(A))$, $\mathsf{head}(r) \in \mathsf{Lg}(\{\vee, \mathbf{f}\}, \mathsf{Lit}(A))$
head-disjunctive	HDLP	$\mathsf{body}(r) \in \mathsf{Lg}\left(\Sigma^{cl} \setminus \{\mathbf{f}\}, A\right)$, $\mathsf{head}(r) \in \mathsf{Lg}(\{\vee, \mathbf{f}\}, \mathsf{Lit}(A))$
generalized	GLP	no condition

More precisely, among the many proposals for semantics for logic programs under extended syntax we will study a very general approach due to Schwarz [14–16]. In this framework, arbitrary formulae are allowed in rule heads and bodies, and it encompasses the inflationary semantics [11], the stable semantics [5], the stable semantics for disjunctive programs [12], and the stable generated semantics [7]. It can itself be understood as a unifying framework for different semantics.

In this paper, we will provide a single theorem — and some corollaries thereof — which gives a characterization of general selector generated models by means of level mappings. It thus provides a link between these two frameworks, and implicitly yields level mapping characterizations of the semantics encompassed by the selector generated approach.

The plan of the paper is as follows. In Section 2 we will fix preliminaries and notation. In Section 3 we will review selector generated models as introduced in [14–16]. In Section 4, we will prove our main result, Theorem 4, which gives a level-mapping characterization of general selector generated models in the style of [8–10]. In Section 5 we study corollaries from Theorem 4 concerning specific cases of interest encompassed by the result. We eventually conclude and discuss further work in Section 6.

2 Preliminaries

Throughout the paper, we will consider a language $\mathcal{L}$ of propositional logic over some set of propositional variables, or *atoms*, A, and connectives $\Sigma^{cl} = \{\neg, \vee, \wedge, \mathbf{t}, \mathbf{f}\}$, as usual. A *rule* r is a pair of formulae from $\mathcal{L}$ denoted by $\varphi \Rightarrow \psi$. φ is called the *body* of the rule, denoted by $\mathsf{body}(r)$, and ψ is called the *head* of the rule, denoted by $\mathsf{head}(r)$. A *program* is a set of rules. A *literal* is an atom or a negated atom, and $\mathsf{Lit}(A)$ denotes the set of all literals in $\mathcal{L}$. For a set of connectives $C \subseteq \Sigma^{cl}$ we denote by $\mathsf{Lg}(C, A)$ the set of all formulae over $\mathcal{L}$ in which only connectives from C occur.

Further terminology is introduced in Table 1. The abbreviations in the second column denote the sets of all rules with the corresponding property. A program containing only definite (normal, etc.) rules is called *definite* (*normal*, etc.).

Programs not containing the negation symbol $\neg$ are called *positive*. *Facts* are rules r where $\mathsf{body}(r) = \mathbf{t}$, denoted by $\Rightarrow \mathsf{head}(r)$. Any set B of atoms defines the set of facts $\mathsf{fact}(B) = \{\Rightarrow a \mid a \in B\}$.

The *base* B_P is the set of all atoms occurring in a program P. A two-valued *interpretation* of a program P is represented by a subset of B_P, as usual. By $\mathbf{I}_P$ we denote the set of all interpretations of P. It is a complete lattice with respect to the subset ordering $\subseteq$. For an interpretation $I \in \mathbf{I}_P$, we define $\uparrow I = \{J \in \mathbf{I}_P \mid I \subseteq J\}$ and $\downarrow I = \{J \in \mathbf{I}_P \mid J \subseteq I\}$. $[I, J] = \uparrow I \cap \downarrow J$ is called an *interval* of interpretations.

The model relation $M \models \varphi$ for an interpretation M and a propositional formula φ is defined as usual in propositional logic, and $\mathrm{Mod}(\varphi)$ denotes the set of all models of φ. Two formulae φ and ψ are *logically equivalent*, written $\varphi \equiv \psi$, iff $\mathrm{Mod}(\varphi) = \mathrm{Mod}(\psi)$.

A formula φ is *satisfied* by a set $\mathbf{J} \subseteq \mathbf{I}_P$ of interpretations if each interpretation $J \in \mathbf{J}$ is a model of φ. For a program P, a set $\mathbf{J} \subseteq \mathbf{I}_P$ of interpretations determines the set of all rules which *fire* under $\mathbf{J}$, formally $\mathsf{fire}(P, \mathbf{J}) = \{r \in P \mid \forall J \in \mathbf{J} : J \models \mathsf{body}(r)\}$. An interpretation M is called a *model* of a rule r (or *satisfies* r) if M is a model of the formula $\neg\mathsf{body}(r) \vee \mathsf{head}(r)$. An interpretation M is a *model* of a program P if it satisfies each rule in P.

For conjunctions or disjunctions φ of literals, φ^+ denotes the set of all atoms occurring positively in φ, and φ^- contains all atoms that occur negated in φ. For instance, for the formula $\varphi = (a \wedge \neg b \wedge \neg a)$ we have $\varphi^+ = \{a\}$ and $\varphi^- = \{a, b\}$. In heads φ consisting only of disjunctions of literals, we always assume without loss of generality that $\varphi^+ \cap \varphi^- = \emptyset$.

If φ is a *conjunction* of literals, we abbreviate $M \models \bigwedge_{a \in \varphi^+} a$ (i.e. $\varphi^+ \subseteq M$) by $M \models \varphi^+$ and $M \models \bigwedge_{a \in \varphi^-} \neg a$ (i.e. $\varphi^- \cap M = \emptyset$) by $M \models \varphi^-$, abusing notation. If φ is a *disjunction* of literals, we write $M \models \varphi^+$ for $M \models \bigvee_{a \in \varphi^+} a$ (i.e. $M \cap \varphi^+ \neq \emptyset$) and $M \models \varphi^-$ for $M \models \bigvee_{a \in \varphi^-} \neg a$ (i.e. $\varphi^- \nsubseteq M$).

By iterative application of rules from a program $P \subseteq \mathsf{GLP}$ starting in the least interpretation $\emptyset \in \mathbf{I}_P$, we can create monotonically increasing (transfinite) sequences of interpretations of the program P, as follows.

Definition 1. *A (transfinite) sequence C of length α of interpretations of a program $P \subseteq \mathsf{GLP}$ is called a P-*chain *iff*

(**C0**) $C_0 = \emptyset$,
(**C**β) $C_{\beta+1} \in \mathrm{Min}(\uparrow C_\beta \cap \mathrm{Mod}(\mathsf{head}(Q_\beta)))$ *for some set of rules $Q_\beta \subseteq P$ and for all β with $\beta + 1 < \alpha$, and*
(**C**λ) $C_\lambda = \bigcup\{C_\beta \mid \beta < \lambda\}$ *for all limit ordinals $\lambda < \alpha$.*

$\mathbf{C}_P$ *denotes the collection of all P-chains.*

Note that all P-chains increase monotonically with respect to $\subseteq$.

In the proof of Theorem 4, we will make use of the following straightforward lemma from [16].

Lemma 1. *For any set of interpretations $\mathbf{J} \subseteq \mathbf{I}_P$ and any interpretation $K \in \mathbf{I}_P$ we have* $\mathrm{Min}(\mathbf{J} \cap \downarrow K) = \mathrm{Min}(\mathbf{J}) \cap \downarrow K$. $\square$

3 Selector generated models

In [14–16], a framework for defining declarative semantics of generalized logic programs was introduced, which encompasses several other semantics, as already mentioned in the introduction. Parametrization within this framework is done via so-called *selector functions*, defined as follows.

Definition 2. *A* selector *is a function* $\mathrm{Sel} : \mathbf{C}_P \times \mathbf{I}_P \to 2^{\mathbf{I}_P}$*, satisfying* $\emptyset \neq \mathrm{Sel}(C, I) \subseteq [I, \sup(C)]$ *for all P-chains C and each interpretation* $I \in {\downarrow} \sup(C)$*.*

We use selectors Sel to define nondeterministic successor functions Ω_P on $\mathbf{I}_P$, as follows.

Definition 3. *Given a selector* $\mathrm{Sel} : \mathbf{C}_P \times \mathbf{I}_P \to 2^{\mathbf{I}_P}$ *and a program P, the function Ω_P is defined by*

$$\Omega_P : (\mathbf{C}_P \times \mathbf{I}_P \to 2^{\mathbf{I}_P}) \times \mathbf{C}_P \times \mathbf{I}_P \to 2^{\mathbf{I}_P}$$
$$\Omega_P(\mathrm{Sel}, C, I) = \mathrm{Min}\left([I, \sup(C)] \cap \mathrm{Mod}\left(\mathsf{head}\left(\mathsf{fire}\left(P, \mathrm{Sel}\left(C, I\right)\right)\right)\right)\right).$$

Example 1. In this paper, we will have a closer look at the following selectors.

lower bound selector	$\mathrm{Sel}_{\mathsf{l}}(C, I) = \{I\}$
lower and upper bound selector	$\mathrm{Sel}_{\mathsf{lu}}(C, I) = \{I, \sup(C)\}$
interval selector	$\mathrm{Sel}_{\mathsf{i}}(C, I) = [I, \sup(C)]$
chain selector	$\mathrm{Sel}_{\mathsf{c}}(C, I) = [I, \sup(C)] \cap C$

With the first two arguments (the selector Sel and the chain C) fixed, the function $\Omega_P(\mathrm{Sel}, C, I)$ can be understood as a nondeterministic consequence operator. Iteration of the function $\Omega_P(\mathrm{Sel}, C, \cdot)$ from the least interpretation $\emptyset$ creates monotonic sequences of interpretations. This leads to the following definition of (P, M, Sel)-chains.

Definition 4. *A* (P, M, Sel)-chain *is a P-chain satisfying*

($\mathbf{C}\sup$) $M = \sup(C)$ *and*
($\mathbf{C}\beta_{\mathrm{Sel}}$) $C_{\beta+1} \in \Omega_P(\mathrm{Sel}, C, C_\beta)$ *for all β, where $\beta + 1 < \kappa$ and κ is the length of the transfinite sequence C.*

Thus, (P, M, Sel)-chains are monotonic sequences C of interpretations, that reproduce themselves by iterating Ω_P.

Definition 5. *A model M of a program* $P \subseteq \mathsf{GLP}$ *is* Sel-generated *if and only if there exists a* (P, M, Sel)*-chain C. The* Sel*-semantics of the program P is the set* $\mathrm{Mod}_{\mathrm{Sel}}(P)$ *of all* Sel*-generated models of P.*

Example 2. Let P be the program consisting of the following rules.

$$\Rightarrow a \qquad (1)$$
$$a \Rightarrow b \qquad (2)$$
$$(a \vee \neg c) \wedge (c \vee \neg a) \Rightarrow c \qquad (3)$$

Then $\{a, b, c\}$ is the only $\mathrm{Sel}_{\mathsf{l}}$-generated model for P, namely via the chain $C_1 = (\emptyset \overset{(1),(3)}{\rightarrow} \{a, c\} \overset{(2)}{\rightarrow} \{a, b, c\}))$. $\{a, b\}$ and $\{a, b, c\}$ are $\mathrm{Sel}_{\mathsf{lu}}$-generated (and $\mathrm{Sel}_{\mathsf{c}}$-generated) models, namlely via the chains $C_2 = (\emptyset \overset{(1)}{\rightarrow} \{a\} \overset{(2)}{\rightarrow} \{a, b\})$ and C_1). $\{a, b\}$ is the only $\mathrm{Sel}_{\mathsf{i}}$-generated model of P, namely via C_2.

Some properties of semantics generated by the selectors in Example 1 were studied in [14]. In Section 5, we will make use of the following results from [14].

Theorem 1. *1. For definite programs $P \subseteq \mathsf{DLP}$, the unique element contained in $\mathrm{Mod}_{\mathsf{l}}(P) = \mathrm{Mod}_{\mathsf{lu}}(P) = \mathrm{Mod}_{\mathsf{c}}(P) = \mathrm{Mod}_{\mathsf{i}}(P)$ is the* least *model of P.*
2. For normal programs $P \subseteq \mathsf{NLP}$, the unique element of $\mathrm{Mod}_{\mathsf{l}}(P)$ is the inflationary *model of P (as introduced in [11]).*
3. For normal programs $P \subseteq \mathsf{NLP}$, the set $\mathrm{Mod}_{\mathsf{lu}}(P) = \mathrm{Mod}_{\mathsf{c}}(P) = \mathrm{Mod}_{\mathsf{i}}(P)$ contains exactly all stable *models of P (as defined in [5]).*
4. For disjunctive programs $P \subseteq \mathsf{DLP}^+$, the minimal elements in $\mathrm{Mod}_{\mathsf{lu}}(P) = \mathrm{Mod}_{\mathsf{c}}(P) = \mathrm{Mod}_{\mathsf{i}}(P)$ are exactly all stable models of P (as defined in [12]), but for generalized programs $P \subseteq \mathsf{GLP}$, the sets $\mathrm{Mod}_{\mathsf{lu}}(P)$, $\mathrm{Mod}_{\mathsf{c}}(P)$, and $\mathrm{Mod}_{\mathsf{i}}(P)$ may differ.
5. For generalized programs $P \subseteq \mathsf{GLP}$, $\mathrm{Mod}_{\mathsf{i}}(P)$ is the set of stable generated models *of P (as defined in [7]).* $\square$

This shows that the framework of selector semantics covers some of the most important declarative semantics for normal logic programs. Selector generated models provide a natural extension of these semantics to generalized logic programs and allow systematic comparisons of many new and well-known semantics.

For all selectors Sel, it was shown in [14] that the Sel-semantics of programs in GLP is invariant with respect to the following transformations. ($\rightarrow_{\mathsf{eq}}$) The replacement of the body and the head of a rule by logically equivalent formulas. ($\rightarrow_{\mathsf{hs}}$) The splitting of conjunctive heads, more precisely the replacement $P \cup \{\varphi \Rightarrow \psi \wedge \psi'\} \rightarrow_{\mathsf{hs}} P \cup \{\varphi \Rightarrow \psi, \varphi \Rightarrow \psi'\}$. As every formula $\mathsf{head}(r)$ is logically equivalent to a formula in conjunctive normal form, it suffices to study head disjunctive programs.

4 Selector generated models via level mappings

In [8–10], a uniform approach to different semantics for logic programs was given, using the notion of *level mapping*, as follows.

Definition 6. *A* level mapping *for a logic program $P \subseteq \mathsf{GLP}$ is a function $l : \mathsf{B}_P \rightarrow \alpha$, where α is an ordinal.*

In order to display the style of level-mapping characterizations for semantics, we give two examples from [10] which we will further discuss later on.

Theorem 2. *Every definite program $P \subseteq \mathsf{LP}$ has exactly one model M, such that there exists a level mapping $l : \mathsf{B}_P \rightarrow \alpha$ satisfying*

(Fd) *for every atom $a \in M$ there exists a rule $\bigwedge_{b\in B} b \Rightarrow a \in P$ such that $B \subseteq M$ and $\max\{l(b) \mid b \in B\} < l(a)$.*

Furthermore, M coincides with the least model of P. □

The following is actually due to Fages [4].

Theorem 3. *Let P be a normal program and M be an interpretation for P. Then M is a stable model of P iff there exists a level mapping $l : \mathsf{B}_P \to \alpha$ satisfying*

(Fs) *for each atom $a \in M$ there exists a rule $r \in P$ with $\mathsf{head}(r) = a$, $\mathsf{body}(r)^+ \subseteq M$, $\mathsf{body}(r)^- \cap M = \emptyset$, and $\max\{l(b) \mid b \in \mathsf{body}(r)^+\} < l(a)$.* □

It is evident, that among the level mappings satisfying the respective conditions in Theorems 2 and 3, there exist pointwise minimal ones.

We set out to prove a general theorem which characterizes selector generated models by means of level mappings, in the style of the results displayed above. The following notion will ease notation considerably.

Definition 7. *For a level mapping $l : \mathsf{B}_P \to \alpha$ for a program $P \subseteq \mathsf{GLP}$ and an interpretation $M \subseteq \mathsf{B}_P$, the (transfinite) sequence $\mathsf{C}^{l,M}$ consisting of interpretations of P is defined by*

$$\mathsf{C}^{l,M}_\beta = \{a \in M \mid l(a) < \beta\} = M \cap \bigcup_{\gamma<\beta} l^{-1}(\gamma)$$

for all $\beta < \alpha$.

Remark 1. Definition 7 implies the following properties of the (transfinite) sequence $\mathsf{C}^{l,M}$. (1) $\mathsf{C}^{l,M}$ is monotonically increasing, (2) $\mathsf{C}^{l,M}_0 = \emptyset$, and (3) $M = \bigcup_{\beta<\alpha} \mathsf{C}^{l,M}_\beta = \sup \mathsf{C}^{l,M}$.

The following is our main result.

Theorem 4. *Let $P \subseteq \mathsf{HDLP}$ be a head disjunctive program and $M \in \mathbf{I}_P$. Then M is a* Sel*-generated model of P iff there exists a level mapping $l : \mathsf{B}_P \to \alpha$ satisfying the following properties.*

(L1) $M = \sup\left(\mathsf{C}^{l,M}\right) \in \mathrm{Mod}\,(P)$.

(L2) *For all β with $\beta + 1 < \alpha$ we have*

$$\mathsf{C}^{l,M}_{\beta+1} \setminus \mathsf{C}^{l,M}_\beta \in \mathrm{Min}\left\{ J \in \mathbf{I}_P \,\middle|\, J \models \mathsf{head}\left(R\left(\mathsf{C}^{l,M}_\beta, J\right)\right)^+ \right\}, \quad \textit{where}$$
$$R\left(\mathsf{C}^{l,M}_\beta, J\right) = \left\{ r \in \mathsf{fire}\left(P, \mathrm{Sel}\left(\mathsf{C}^{l,M}, \mathsf{C}^{l,M}_\beta\right)\right) \,\middle|\, \begin{array}{l} \mathsf{C}^{l,M}_\beta \not\models \mathsf{head}\,(r)^+ \ \textit{and} \\ J \cup \mathsf{C}^{l,M}_\beta \not\models \mathsf{head}\,(r)^- \end{array} \right\}.$$

(L3) *For all limit ordinals $\lambda < \alpha$ we have $\mathsf{C}^{l,M}_\lambda = \bigcup_{\beta<\lambda} \mathsf{C}^{l,M}_\beta$.*

Remark 2. As P is a head disjunctive program, we have $\mathsf{C}_{\beta}^{l,M} \not\models \mathsf{head}\,(r)^{+}$ iff $\mathsf{head}\,(r)^{+} \cap \mathsf{C}_{\beta}^{l,M} = \emptyset$, and $J \cup \mathsf{C}_{\beta}^{l,M} \not\models \mathsf{head}\,(r)^{-}$ iff $\mathsf{head}\,(r)^{-} \subseteq J \cup \mathsf{C}_{\beta}^{l,M}$, thus

$$R\left(\mathsf{C}_{\beta}^{l,M}, J\right) = \left\{ r \in \mathsf{fire}\left(P, \mathrm{Sel}\left(\mathsf{C}^{l,M}, \mathsf{C}_{\beta}^{l,M}\right)\right) \,\middle|\, \begin{array}{l} \mathsf{head}\,(r)^{+} \cap \mathsf{C}_{\beta}^{l,M} = \emptyset \text{ and} \\ \mathsf{head}\,(r)^{-} \subseteq J \cup \mathsf{C}_{\beta}^{l,M} \end{array} \right\}.$$

Also note that for every rule $r \in \mathsf{fire}\left(P, \mathrm{Sel}\left(\mathsf{C}^{l,M}, \mathsf{C}_{\beta}^{l,M}\right)\right) \setminus R\left(\mathsf{C}_{\beta}^{l,M}, J\right)$, we have $\downarrow\left(\mathsf{C}_{\beta}^{l,M} \cup J\right) \subseteq \mathrm{Mod}\left(\mathsf{head}\,(r)^{-}\right)$ or $\uparrow \mathsf{C}_{\beta}^{l,M} \subseteq \mathrm{Mod}\left(\mathsf{head}\,(r)^{+}\right)$. Thus all of these rules are satisfied in the interval $[\mathsf{C}_{\beta}^{l,M}, \mathsf{C}_{\beta}^{l,M} \cup J]$.

Proof. (of Theorem 4)
($\Longrightarrow$) By Definition 5, an interpretation M is a Sel-generated model of P iff there exists a (P, M, Sel)-chain. Let M be a model of P and M be Sel-generated by the (P, M, Sel)-chain C of length α.

Define the level mapping $l : \mathsf{B}_P \to \alpha$ by $l(a) = \min\{\beta \mid a \in C_{\beta}\} - 1$ for all $a \in \mathsf{B}_P$. We show that this function l satisfies **(L1)**,**(L2)** and **(L3)**.

(1) We first show $\mathsf{C}^{l,M} = C$ for the sequence $\mathsf{C}^{l,M}$ determined by l and M according to Definition 7. From Remark 1, we know $\mathsf{C}_{0}^{l,M} = \emptyset$ and $\sup\left(\mathsf{C}^{l,M}\right) = M$. Moreover, for each $\beta < \alpha$, we have

$$\begin{aligned} \mathsf{C}_{\beta}^{l,M} &= \{a \in M \mid l\,(a) < \beta\} \\ &= \{a \in M \mid \min\{\gamma \mid a \in C_{\gamma}\} - 1 < \beta\} \quad \text{(by Definition 7)} \\ &= \{a \in M \mid a \in C_{\gamma}, \gamma - 1 < \beta\} \qquad\quad \text{(by definition of } l\text{)} \\ &= C_{\beta}. \end{aligned}$$

Therefore, we have $\mathsf{C}_{\lambda}^{l,M} = \bigcup_{\beta<\lambda} \mathsf{C}_{\beta}^{l,M} = \bigcup_{\beta<\lambda} C_{\beta} = C_{\lambda}$ for all limit ordinals $\lambda < \alpha$. This proves $C = \mathsf{C}^{l,M}$.

(2) C is a (P, M, Sel)-chain, so it satisfies **(L1)** and **(L3)**. It remains to show that C satisfies **(L2)**. For all β with $\beta + 1 < \alpha$, we show

(a) $C_{\beta+1} \setminus C_{\beta} \models \mathsf{head}\,(R\,(C_{\beta}, C_{\beta+1} \setminus C_{\beta}))^{+}$ for

$$\begin{aligned} &R\,(C_{\beta}, C_{\beta+1} \setminus C_{\beta}) \\ &= \left\{ r \in \mathsf{fire}\,(P, \mathrm{Sel}\,(C, C_{\beta})) \,\middle|\, \begin{array}{l} C_{\beta} \not\models \mathsf{head}\,(r)^{+} \text{ and} \\ C_{\beta} \cup C_{\beta+1} \setminus C_{\beta} \not\models \mathsf{head}\,(r)^{-} \end{array} \right\} \\ &= \left\{ r \in \mathsf{fire}\,(P, \mathrm{Sel}\,(C, C_{\beta})) \mid C_{\beta} \cap \mathsf{head}\,(r)^{+} = \emptyset \text{ and } \mathsf{head}\,(r)^{-} \subseteq C_{\beta+1} \right\} \end{aligned}$$

and

(b) for all interpretations $J \subseteq C_{\beta+1} \setminus C_{\beta}$ where $J \models \mathsf{head}\,(R\,(C_{\beta}, J))^{+}$, we have $J = C_{\beta+1} \setminus C_{\beta}$.

(a) C is a (P, M, Sel)-chain, hence we have

$$C_{\beta+1} \in \mathit{\Omega}_P\,(\mathrm{Sel}, C, C_{\beta}) = \mathrm{Min}\,([C_{\beta}, M] \cap \mathrm{Mod}\,(\mathsf{head}\,(\mathsf{fire}\,(P, \mathrm{Sel}\,(C, C_{\beta}))))),$$

and we obtain $C_{\beta+1} \models \mathsf{head}\left(\mathsf{fire}\left(P, \mathrm{Sel}\left(C, C_\beta\right)\right)\right)$.

For each rule $r \in R\left(C_\beta, C_{\beta+1} \setminus C_\beta\right)$, we know

$$R\left(C_\beta, C_{\beta+1} \setminus C_\beta\right) \subseteq \mathsf{fire}\left(P, \mathrm{Sel}\left(C, C_\beta\right)\right)$$

and hence $C_{\beta+1} \models \mathsf{head}\left(r\right)$.

By the definition of this set and Remark 2, the set $R\left(C_\beta, C_{\beta+1} \setminus C_\beta\right)$ does not contain any rule $r \in \mathsf{fire}\left(P, \mathrm{Sel}\left(C, C_\beta\right)\right)$, where $C_{\beta+1} \models \mathsf{head}\left(r\right)$ is satisfied by $C_{\beta+1} \models \mathsf{head}\left(r\right)^-$ or $C_\beta \models \mathsf{head}\left(r\right)^+$, i.e. $\mathsf{head}\left(r\right)^+ \cap C_\beta \neq \emptyset$. Hence all rules r from $R\left(C_\beta, C_{\beta+1} \setminus C_\beta\right) \subseteq \mathsf{fire}\left(P, \mathrm{Sel}\left(C, C_\beta\right)\right)$ satisfy $C_\beta \models \mathsf{head}\left(r\right)$ by $C_{\beta+1} \setminus C_\beta \cap \mathsf{head}\left(r\right)^+ \neq \emptyset$, i.e. $C_{\beta+1} \setminus C_\beta \models \mathsf{head}\left(r\right)^+$. This shows (a).

(b) Assume $J \subseteq C_{\beta+1} \setminus C_\beta$ and $J \models \mathsf{head}\left(R\left(C_\beta, J\right)\right)^+$. We show $J \cup C_\beta \supseteq C_{\beta+1}$ which implies $J \supseteq C_{\beta+1} \setminus C_\beta$.

First note that $J \cup C_\beta \subseteq [C_\beta, M] \cap \mathrm{Mod}\left(\mathsf{head}\left(\mathsf{fire}\left(P, \mathrm{Sel}\left(C, C_\beta\right)\right)\right)\right)$. Indeed $J \cup C_\beta \in \uparrow C_\beta$ is obvious and $J \cup C_\beta \in \downarrow M$ is implied by $J \subseteq C_{\beta+1} \setminus C_\beta$, i.e. $J \cup C_\beta \subseteq C_{\beta+1}$, and $C_{\beta+1} \subseteq M$ by monotonicity of the chain C.

Now we show $J \cup C_\beta \models \mathsf{head}\left(\mathsf{fire}\left(P, \mathrm{Sel}\left(C, C_\beta\right)\right)\right)$. Note first that all rules r in the set $\mathsf{fire}\left(P, \mathrm{Sel}\left(C, C_\beta\right)\right)$ satisfy one of the following conditions.

1. $C_\beta \models \mathsf{head}\left(r\right)^+$ (i.e. $C_\beta \cap \mathsf{head}\left(r\right)^+ \neq \emptyset$) and therefore $J \cup C_\beta \models \mathsf{head}\left(r\right)$ by $C_\beta \subseteq J \cup C_\beta$ or
2. $J \cup C_\beta \models \mathsf{head}\left(r\right)^-$ and therefore $J \cup C_\beta \models \mathsf{head}\left(r\right)$ or
3. none of 1. or 2. Then we have $r \in R\left(C_\beta, J\right)$ and due to the assumption $J \in \mathrm{Mod}\left(\mathsf{head}\left(R\left(C_\beta, J\right)\right)^+\right)$ we have $J \cup C_\beta \models \mathsf{head}\left(r\right)^+$ and therefore $J \cup C_\beta \models \mathsf{head}\left(r\right)$.

We can now conclude $J \cup C_\beta \supseteq C_{\beta+1}$ because $C_{\beta+1}$ is a minimal element of $[C_\beta, M] \cap \mathrm{Mod}\left(\mathsf{head}\left(\mathsf{fire}\left(P, \mathrm{Sel}\left(C, C_\beta\right)\right)\right)\right)$, which proves (b).

Together, we have shown that $C_{\beta+1} \setminus C_\beta$ is a minimal element in

$$\left\{J \in \mathbf{I}_P \mid J \models \mathsf{head}\left(R\left(C_\beta, J\right)\right)^+\right\},$$

which shows that the level mapping l satisfies **(L2)**. This finishes the first part of the proof.

($\Longleftarrow$) For the converse, we show that for every level mapping l for a program P and an interpretation M satisfying **(L1)**,**(L2)** and **(L3)** the sequence $\mathsf{C}^{l,M}$ is a (P, M, Sel)-chain.

Let $l : \mathsf{B}_P \to \alpha$ be a level mapping and M an interpretation for a program P. According to Definition 4, we have to show the following properties of the sequence $\mathsf{C}^{l,M}$:

(**C0**) $\mathsf{C}^{l,M}_0 = \emptyset$,
(**C**λ) $\mathsf{C}^{l,M}_\lambda = \bigcup\left\{\mathsf{C}^{l,M}_\beta \mid \beta < \lambda\right\}$ for all limit ordinals $\lambda < \alpha$,
(**C** sup) $M = \bigcup\left\{\mathsf{C}^{l,M}_\beta \mid \beta < \alpha\right\} = \sup \mathsf{C}^{l,M}$ and
(**C**β_{Sel}) $\mathsf{C}^{l,M}_{\beta+1} \in \mathit{\Omega}_P\left(\mathrm{Sel}, \mathsf{C}^{l,M}, \mathsf{C}^{l,M}_\beta\right)$ for all β with $\beta + 1 < \alpha$.

By Remark 1 we know that $\mathsf{C}^{l,M}$ increases monotonically and $\mathsf{C}^{l,M}_0 = \emptyset$, i.e. (**C0**), is satisfied. By condition **(L1)** we have $M = \sup\left(\mathsf{C}^{l,M}\right) \in \mathrm{Mod}\,(P)$, i.e. ($\mathbf{C}\sup$), and condition **(L3)** implies ($\mathbf{C}\lambda$).

For ($\mathbf{C}\beta_{\mathrm{Sel}}$) we have to show that for all β with $\beta + 1 < \alpha$ the equation

$$\begin{aligned}\mathsf{C}^{l,M}_{\beta+1} \in \mathit{\Omega}_P\left(\mathrm{Sel}, \mathsf{C}^{l,M}, \mathsf{C}^{l,M}_\beta\right)\\ = \mathrm{Min}\left(\left[\mathsf{C}^{l,M}_\beta, M\right] \cap \mathrm{Mod}\left(\mathsf{head}\left(\mathsf{fire}\left(P, \mathrm{Sel}\left(\mathsf{C}^{l,M}, \mathsf{C}^{l,M}_\beta\right)\right)\right)\right)\right)\end{aligned}$$

holds. For this, by Lemma 1, it suffices to show that

$$\mathsf{C}^{l,M}_{\beta+1} \in \mathrm{Min}\left(\uparrow \mathsf{C}^{l,M}_\beta \cap \mathrm{Mod}\left(\mathsf{head}\left(\mathsf{fire}\left(P, \mathrm{Sel}\left(\mathsf{C}^{l,M}, \mathsf{C}^{l,M}_\beta\right)\right)\right)\right)\right) \cap \downarrow M.$$

Now by monotonicity of $\mathsf{C}^{l,M}$ we know $\mathsf{C}^{l,M}_{\beta+1} \in \downarrow M$. For

$$\mathsf{C}^{l,M}_{\beta+1} \in \mathrm{Min}\left(\uparrow \mathsf{C}^{l,M}_\beta \cap \mathrm{Mod}\left(\mathsf{head}\left(\mathsf{fire}\left(P, \mathrm{Sel}\left(\mathsf{C}^{l,M}, \mathsf{C}^{l,M}_\beta\right)\right)\right)\right)\right)$$

we will proceed by proving the steps (a) and (b), as follows.

(a) $\mathsf{C}^{l,M}_{\beta+1} \in \uparrow \mathsf{C}^{l,M}_\beta \cap \mathrm{Mod}\left(\mathsf{head}\left(\mathsf{fire}\left(P, \mathrm{Sel}\left(\mathsf{C}^{l,M}, \mathsf{C}^{l,M}_\beta\right)\right)\right)\right)$.

(b) For all interpretations $J \in \uparrow \mathsf{C}^{l,M}_\beta \cap \mathrm{Mod}\left(\mathsf{head}\left(\mathsf{fire}\left(P, \mathrm{Sel}\left(\mathsf{C}^{l,M}, \mathsf{C}^{l,M}_\beta\right)\right)\right)\right)$, where $J \subseteq \mathsf{C}^{l,M}_{\beta+1}$, we have $J = \mathsf{C}^{l,M}_{\beta+1}$.

(a) By monotonicity of $\mathsf{C}^{l,M}$ it suffices to show that

$$\mathsf{C}^{l,M}_{\beta+1} \in \mathrm{Mod}\left(\mathsf{head}\left(\mathsf{fire}\left(P, \mathrm{Sel}\left(\mathsf{C}^{l,M}, \mathsf{C}^{l,M}_\beta\right)\right)\right)\right).$$

First note that for every rule $r \in \mathsf{fire}\left(P, \mathrm{Sel}\left(\mathsf{C}^{l,M}, \mathsf{C}^{l,M}_\beta\right)\right)$ one of the following holds:

1. $\mathsf{C}^{l,M}_\beta \models \mathsf{head}\,(r)^+$, i.e. $\mathsf{head}\,(r)^+ \cap \mathsf{C}^{l,M}_\beta \neq \emptyset$, and therefore $\mathsf{C}^{l,M}_{\beta+1} \models \mathsf{head}\,(r)$ by $\mathsf{C}^{l,M}_\beta \subseteq \mathsf{C}^{l,M}_{\beta+1}$ or
2. $\mathsf{C}^{l,M}_{\beta+1} \models \mathsf{head}\,(r)^-$ and therefore $\mathsf{C}^{l,M}_{\beta+1} \models \mathsf{head}\,(r)$ or
3. none of 1. or 2. Then $r \in R\left(\mathsf{C}^{l,M}_\beta, \mathsf{C}^{l,M}_{\beta+1} \setminus \mathsf{C}^{l,M}_\beta\right)$ and thus $\mathsf{C}^{l,M}_{\beta+1} \models \mathsf{head}\,(r)$ by $\mathsf{C}^{l,M}_{\beta+1} \setminus \mathsf{C}^{l,M}_\beta \models \mathsf{head}\,(r)^+$ and condition **(L2)**.

Hence for each rule $r \in \mathsf{fire}\left(P, \mathrm{Sel}\left(\mathsf{C}^{l,M}, \mathsf{C}^{l,M}_\beta\right)\right)$ we have $\mathsf{C}^{l,M}_{\beta+1} \models \mathsf{head}\,(r)$ and thus $\mathsf{C}^{l,M}_{\beta+1} \in \uparrow \mathsf{C}^{l,M}_\beta \cap \mathrm{Mod}\left(\mathsf{head}\left(\mathsf{fire}\left(P, \mathrm{Sel}\left(\mathsf{C}^{l,M}, \mathsf{C}^{l,M}_\beta\right)\right)\right)\right)$, which shows (a).

(b) Let $J \in \uparrow \mathsf{C}^{l,M}_\beta \cap \mathrm{Mod}\left(\mathsf{head}\left(\mathsf{fire}\left(P, \mathrm{Sel}\left(\mathsf{C}^{l,M}, \mathsf{C}^{l,M}_\beta\right)\right)\right)\right)$ for some $J \subseteq \mathsf{C}^{l,M}_{\beta+1}$. Since $J \in \uparrow \mathsf{C}^{l,M}_\beta$ we obtain $J \in \uparrow \mathsf{C}^{l,M}_{\beta+1}$ by showing $J \setminus \mathsf{C}^{l,M}_\beta \supseteq \mathsf{C}^{l,M}_{\beta+1} \setminus \mathsf{C}^{l,M}_\beta$. Indeed $J \setminus \mathsf{C}^{l,M}_\beta \in \left\{K \in \mathbf{I}_P \mid K \models \mathsf{head}\left(R\left(\mathsf{C}^{l,M}_\beta, K\right)\right)^+\right\}$ and therefore $J \setminus$

$\mathsf{C}_{\beta}^{l,M} \models \mathsf{head}\left(R\left(\mathsf{C}_{\beta}^{l,M}, J \setminus \mathsf{C}_{\beta}^{l,M}\right)\right)^{+}$. Condition **(L2)**, i.e. minimality of $C_{\beta+1} \setminus \mathsf{C}_{\beta}^{l,M}$ in this set, implies $J \setminus \mathsf{C}_{\beta}^{l,M} \supseteq \mathsf{C}_{\beta+1}^{l,M} \setminus \mathsf{C}_{\beta}^{l,M}$ as desired.

By $J \in \mathrm{Mod}\left(\mathsf{head}\left(\mathsf{fire}\left(P, \mathrm{Sel}\left(\mathsf{C}^{l,M}, \mathsf{C}_{\beta}^{l,M}\right)\right)\right)\right)$ we have $J \models \mathsf{head}(r)$ for all rules $r \in \mathsf{fire}\left(P, \mathrm{Sel}\left(\mathsf{C}^{l,M}, \mathsf{C}_{\beta}^{l,M}\right)\right)$. For each of these rules r, $J \models \mathsf{head}(r)$ is satisfied by $J \models \mathsf{head}(r)^{-}$ or by $\mathsf{C}_{\beta}^{l,M} \models \mathsf{head}(r)^{+}$ and in both cases we have $r \notin R\left(\mathsf{C}_{\beta}^{l,M}, J\right)$. For all remaining rules, we know that $J \models \mathsf{head}(r)$ is satisfied by $J \setminus \mathsf{C}_{\beta}^{l,M} \cap \mathsf{head}(r)^{+} \neq \emptyset$, i.e. $J \setminus \mathsf{C}_{\beta}^{l,M} \models \mathsf{head}(r)^{+}$, and therefore we know $J \setminus \mathsf{C}_{\beta}^{l,M} \in \left\{K \in \mathbf{I}_P \mid K \models \mathsf{head}\left(R\left(\mathsf{C}_{\beta}^{l,M}, K\right)\right)^{+}\right\}$.

By $J \setminus \mathsf{C}_{\beta}^{l,M} \subseteq \mathsf{C}_{\beta+1}^{l,M} \setminus \mathsf{C}_{\beta}^{l,M}$ and minimality of $\mathsf{C}_{\beta+1}^{l,M} \setminus \mathsf{C}_{\beta}^{l,M}$ in the set

$$\left\{K \in \mathbf{I}_P \mid K \models \mathsf{head}\left(R\left(\mathsf{C}_{\beta}^{l,M}, K\right)\right)^{+}\right\}$$

we have $J \setminus \mathsf{C}_{\beta}^{l,M} = \mathsf{C}_{\beta+1}^{l,M} \setminus \mathsf{C}_{\beta}^{l,M}$ and therefore $J = \mathsf{C}_{\beta+1}^{l,M}$, which shows (b).

This proves the minimality of $\mathsf{C}_{\beta+1}^{l,M}$ in the set

$$[\mathsf{C}_{\beta}^{l,M}, M] \cap \mathrm{Mod}\left(\mathsf{head}\left(\mathsf{fire}\left(P, \mathrm{Sel}\left(\mathsf{C}^{l,M}, \mathsf{C}_{\beta}^{l,M}\right)\right)\right)\right).$$

Thus, $\mathsf{C}_{\beta+1}^{l,M} \in \Omega_P\left(\mathrm{Sel}, \mathsf{C}^{l,M}, \mathsf{C}_{\beta}^{l,M}\right)$.

Hence $\mathsf{C}^{l,M}$ is a (P, M, Sel)-chain. This proves $M \in \mathrm{Mod}_{\mathrm{Sel}}(P)$ and concludes the proof. □

By the remarks made at the end of Section 3, we obtain the following immediate corollary.

Corollary 1. *Let P be a generalized program and M an interpretation of P. Then M is a* Sel*-generated model of P iff for a head disjunctive program Q with $P \rightarrow^{*}_{\mathsf{eq,hs}} Q$ there exists a level mapping $l : \mathsf{B}_Q \rightarrow \alpha$ satisfying **(L1)**, **(L2)** and **(L3)** of Theorem 4.* □

5 Corollaries

We can now apply Theorem 4 in order to obtain level mapping characterizations for every semantics generated by a selector, in particular for those semantics generated by the selectors defined in Example 1 and listed in Theorem 1. For syntactically restricted programs, we can furthermore simplify the properties **(L1)**,**(L2)** and **(L3)** in Theorem 4. Alternative level mapping characterizations for some of these semantics were already obtained directly in [10].

Programs with positive disjunctions in all heads

For rules $r \in \mathsf{HDLP}$, where $\mathsf{head}(r)$ is a disjunction of atoms, we have $\mathsf{head}(r)^- = \emptyset$. Hence we have $\mathsf{head}(r)^- \subseteq I$, i.e. $I \not\models \mathsf{head}(r)^-$, for all interpretations $I \in \mathbf{I}_P$. Thus the set $R\left(\mathsf{C}_\beta^{l,M}, J\right)$ from **(L2)** in Theorem 4 can be specified by

$$R\left(\mathsf{C}_\beta^{l,M}, J\right) = \left\{r \in \mathsf{fire}\left(P, \mathrm{Sel}\left(\mathsf{C}^{l,M}, \mathsf{C}_\beta^{l,M}\right)\right) \mid \mathsf{C}_\beta^{l,M} \not\models \mathsf{head}(r)^+\right\}.$$

We furthermore observe that the set $R\left(\mathsf{C}_\beta^{l,M}, J\right)$ does not depend on the interpretation J, so we obtain

$$R'\left(\mathsf{C}_\beta^{l,M}\right) = \left\{r \in \mathsf{fire}\left(P, \mathrm{Sel}\left(\mathsf{C}^{l,M}, \mathsf{C}_\beta^{l,M}\right)\right) \mid \mathsf{C}_\beta^{l,M} \cap \mathsf{head}(r)^+ = \emptyset\right\}$$

and hence

$$\mathrm{Min}\left\{J \in \mathbf{I}_P \,\middle|\, J \models \mathsf{head}\left(R\left(\mathsf{C}_\beta^{l,M}, J\right)\right)^+\right\} = \mathrm{Min}\left(\mathrm{Mod}\left(\mathsf{head}\left(R'\left(\mathsf{C}_\beta^{l,M}\right)\right)\right)\right).$$

Thus for programs containing only rules whose heads are disjunctions of atoms we can rewrite condition **(L2)** in Theorem 4, as follows.

(L2d) For every β with $\beta + 1 < \alpha$ we have

$$\mathsf{C}_{\beta+1}^{l,M} \setminus \mathsf{C}_\beta^{l,M} \in \mathrm{Min}\left(\mathrm{Mod}\left(\mathsf{head}\left(R'\left(\mathsf{C}_\beta^{l,M}\right)\right)\right)\right), \text{ where}$$
$$R'\left(\mathsf{C}_\beta^{l,M}\right) = \left\{r \in \mathsf{fire}\left(P, \mathrm{Sel}\left(\mathsf{C}^{l,M}, \mathsf{C}_\beta^{l,M}\right)\right) \,\middle|\, \mathsf{C}_\beta^{l,M} \cap \mathsf{head}(r)^+ = \emptyset\right\}.$$

Programs with atomic heads

Single atoms are a specific kind of disjunctions of atoms. Hence for programs with atomic heads we can replace condition **(L2)** in Theorem 4 by **(L2d)**, and further simplify it as follows.

For rules with atomic heads we have $\mathsf{head}\left(\{r \in P \mid \mathsf{head}(r) \notin I\}\right) = \mathsf{head}(P) \setminus I$ and therefore

$$\begin{aligned}
&\mathsf{head}\left(R'\left(\mathsf{C}_\beta^{l,M}\right)\right)\\
&= \mathsf{head}\left(\left\{r \in \mathsf{fire}\left(P, \mathrm{Sel}\left(\mathsf{C}^{l,M}, \mathsf{C}_\beta^{l,M}\right)\right) \mid \mathsf{head}(r) \cap \mathsf{C}_\beta^{l,M} = \emptyset\right\}\right)\\
&= \mathsf{head}\left(\left\{r \in \mathsf{fire}\left(P, \mathrm{Sel}\left(\mathsf{C}^{l,M}, \mathsf{C}_\beta^{l,M}\right)\right) \mid \mathsf{head}(r) \notin \mathsf{C}_\beta^{l,M}\right\}\right)\\
&= \mathsf{head}\left(\mathsf{fire}\left(P, \mathrm{Sel}\left(\mathsf{C}^{l,M}, \mathsf{C}_\beta^{l,M}\right)\right)\right) \setminus \mathsf{C}_\beta^{l,M}.
\end{aligned}$$

Because all formulae in $\mathsf{head}(P)$ are atoms we obtain

$$\begin{aligned}
\mathrm{Min}\left(\mathrm{Mod}\left(\mathsf{head}\left(R'\left(\mathsf{C}_\beta^{l,M}\right)\right)\right)\right) &= \mathrm{Min}\left(\uparrow\left(\mathsf{head}\left(R'\left(\mathsf{C}_\beta^{l,M}\right)\right)\right)\right)\\
&= \left\{\mathsf{head}\left(R'\left(\mathsf{C}_\beta^{l,M}\right)\right)\right\}
\end{aligned}$$

and this allows us to simplify **(L2)** in Theorem 4 to the following.

(L2a) For each β with $\beta + 1 < \alpha$ we have

$$\mathsf{C}^{l,M}_{\beta+1} \setminus \mathsf{C}^{l,M}_{\beta} = \mathsf{head}\left(\mathsf{fire}\left(P, \mathrm{Sel}\left(\mathsf{C}^{l,M}, \mathsf{C}^{l,M}_{\beta}\right)\right)\right) \setminus \mathsf{C}^{l,M}_{\beta}.$$

Inflationary models From Section 3 we know that for normal programs P the selector $\mathrm{Sel}_{\mathsf{I}}$ generates exactly the inflationary model of P as defined in [11]. The generalizations of the definition of inflationary models and this result to head atomic programs are immediate. From [16] we also know that every $\mathrm{Sel}_{\mathsf{I}}$-generated model is generated by a $(P, M, \mathrm{Sel}_{\mathsf{I}})$-chain of length ω. Thus level mappings $l : \mathsf{B}_P \to \omega$ are sufficient to characterize inflationary models of head atomic programs. In this case, condition **(L3)** applies only to the limit ordinal $0 < \omega$. But by remark 1, all level mappings satisfy this property. Therefore we do not need condition **(L3)** in the characterization of inflationary models.

Using Theorem 4 and the considerations above, we obtain the following characterization of inflationary models.

Corollary 2. *Let $P \subseteq \mathsf{HDLP}$ be a head atomic program and M an interpretation for P. Then M is the inflationary model of P iff there exists a level mapping $l : \mathsf{B}_P \to \omega$ with the following properties.*

(L1) $M = \sup\left(\mathsf{C}^{l,M}\right) \in \mathrm{Mod}(P)$.

(L2i)

$$\mathsf{C}^{l,M}_{n+1} \setminus \mathsf{C}^{l,M}_{n} = \mathsf{head}\left(\mathsf{fire}\left(P, \mathsf{C}^{l,M}_{n}\right)\right) \setminus \mathsf{C}^{l,M}_{n}$$

for all $n < \omega$. □

Normal programs

For normal programs, the heads of all rules are single atoms. Hence the simplification **(L2a)** of condition **(L2)** in Theorem 4 applies for all selector generated semantics for normal programs.

The special structure of the bodies of all rules in normal programs allows an alternative formulation of **(L2a)**. In every normal rule, the body is a conjunction of literals. Thus for any set of interpretations $\mathbf{J}$ we have $\mathbf{J} \models \mathsf{body}(r)$ iff $\mathsf{body}(r)^+ \subseteq J$ and $\mathsf{body}(r)^- \cap J = \emptyset$ for all interpretations $J \in \mathbf{J}$.

Stable models We develop next a characterization for stable models of normal programs, as introduced in [5]. The selector $\mathrm{Sel}_{\mathsf{lu}}$ generates exactly all stable models for normal programs. In [16], it was also shown that all $\mathrm{Sel}_{\mathsf{lu}}$-generated models M of a program P are generated by a (P, M, Sel)-chain of length $\leq \omega$. So for the same reasons as discussed for inflationary models, level mappings with range ω are sufficient to characterize stable models and condition **(L3)** can be neglected.

For a normal rule r and two interpretations $I, M \in \mathbf{I}_P$ with $I \leq M$ we have $\{I, M\} \models \mathsf{body}(r)$, i.e. $I \models \mathsf{body}(r)$ and $M \models \mathsf{body}(r)$, iff $\mathsf{body}(r)^+ \subseteq I$ and $\mathsf{body}(r)^- \cap M = \emptyset$. Combining this with **(L2a)** we obtain the following characterization of stable models for normal programs.

Corollary 3. *Let $P \subseteq \mathsf{NLP}$ be a normal program and M an interpretation for P. Then M is a stable model of P iff there exists a level mapping $l : \mathsf{B}_P \to \omega$ satisfying the following properties:*

(L1) $M = \sup\left(\mathsf{C}^{l,M}\right) \in \mathrm{Mod}(P)$.
(L2s)

$$\mathsf{C}^{l,M}_{n+1} \setminus \mathsf{C}^{l,M}_n = \mathsf{head}\left(\left\{r \in P \mid \mathsf{body}(r)^+ \subseteq \mathsf{C}^{l,M}_n, \mathsf{body}(r)^- \cap M = \emptyset\right\}\right) \setminus \mathsf{C}^{l,M}_n$$

for all $n < \omega$. □

Comparing this with Theorem 3, we note that both theorems characterize the same set of models. Thus for a model M of P there exists a level mapping $l : \mathsf{B}_P \to \alpha$ satisfying **(L1)** and **(L2s)** iff there exists a level mapping $l : \mathsf{B}_P \to \omega$ satisfying **(Fs)**. The condition imposed on the level mapping in Theorem 3, however, is weaker than the condition in Corollary 3, because levelmappings defined by (P, M, Sel)-chains are always pointwise minimal.

Definite programs

In order to characterize the least model of definite programs, we can further simplify condition **(L2)** in Theorem 4. Definite programs are a particular kind of head atomic programs. Thus we can replace condition **(L2)** in Theorem 4 by **(L2i)**. Since the body of every definite rule is a conjunction of atoms we obtain

$$\mathsf{fire}(P, I) = \left\{r \in P \mid \mathsf{body}(r)^+ \subseteq I\right\}$$

for every interpretation $I \in \mathbf{I}_P$. Thus we get the following result.

Corollary 4. *Let $P \subseteq \mathsf{LP}$ be a definite program and M an interpretation for P. Then M is the least model of P iff there exists a level mapping $l : \mathsf{B}_P \to \omega$ satisfying the following conditions.*

(L1) $M = \sup\left(\mathsf{C}^{l,M}\right) \in \mathrm{Mod}(P)$.
(L2l)

$$\mathsf{C}^{l,M}_{n+1} \setminus \mathsf{C}^{l,M}_n = \mathsf{head}\left(\left\{r \in P \mid \mathsf{body}(r)^+ \subseteq \mathsf{C}^{l,M}_n\right\}\right) \setminus \mathsf{C}^{l,M}_n$$

for every $n < \omega$. □

Comparing this to Theorem 2, we note that the relation between the conditions **(L2l)** and **(Fd)** are similar to those of the conditions **(Fs)** und **(L2s)**.

6 Conclusions and Further Work

Our main result, Theorem 4, provides a characterization of selector generated models — in general form — by means of level mappings in accordance with the uniform approach proposed in [8–10]. As corollaries from this theorem, we have also achieved level mapping characterizations of several semantics encompassed by the selector generated approach due to [14–16].

Our contribution is technical, and provides a first step towards a comprehensive comparative study of different semantics of logic programs under extended syntax by means of level mapping characterizations. Indeed, a very large number of syntactic extensions for logic programs are currently being investigated in the community, and even for some of the less fancy proposals there is often no agreement on the preferable way of assigning semantics to these constructs.

A particularly interesting case in point is provided by disjunctive and extended disjunctive programs, as studied in [6]. While there is more or less general agreement on an appropriate notion of stable model, as given by the notion of *answer set* in [6], there exist various different proposals for a corresponding well-founded semantics, see e.g. [17]. We expect that recasting them by means of level-mappings will provide a clearer picture on the specific ways of modelling knowledge underlying these semantics.

Eventually, we expect that the study of level mapping characterizations of different semantics will lead to methods for extracting other, e.g. procedural, semantic properties from the characterizations, like complexity or decidability results.

References

1. Krzysztof R. Apt, Howard A. Blair, and Adrian Walker. Towards a theory of declarative knowledge. In Jack Minker, editor, *Foundations of deductive databases and logic programs*. Morgan Kaufmann, Los Altos, US, 1988.
2. Krzysztof R. Apt and Dino Pedreschi. Reasoning about termination of pure Prolog programs. *Information and Computation*, 106(1), September 1993.
3. François Fages. Consistency of Clark's completion and existence of stable models. *Journal of Methods of Logic in Computer Science*, 1:51–60, 1994.
4. François Fages. A new fixpoint semantics for general logic programs compared with the well-founded and the stable model semantics. In Peter Szeredi and David H.D. Warren, editors, *Proceedings of the 7th International Conference on Logic Programming (ICLP '90)*, Jerusalem, June 1990. MIT Press.
5. Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert A. Kowalski and Kenneth Bowen, editors, *Proceedings of the Fifth International Conference on Logic Programming*, Cambridge, Massachusetts, 1988. The MIT Press.
6. Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3/4), 1991.
7. Heinrich Herre and Gerd Wagner. Stable models are generated by a stable chain. *Journal of Logic Programming*, 30(2), February 1997.
8. Pascal Hitzler. Towards a systematic account of different logic programming semantics. In Andreas Günter, Rudolf Kruse, and Bernd Neumann, editors, *KI2003: Advances in Artificial Intelligence. Proceedings of the 26th Annual German Conference on Artificial Intelligence, KI2003, Hamburg, Germany, September 2003*, volume 2821 of *Lecture Notes in Artificial Intelligence*, pages 355–369. Springer, Berlin, 2003.

9. Pascal Hitzler and Matthias Wendt. The well-founded semantics is a stratified Fitting semantics. In Matthias Jarke, Jana Koehler, and Gerhard Lakemeyer, editors, *Proceedings of the 25th Annual German Conference on Artificial Intelligence, KI2002, Aachen, Germany, September 2002*, volume 2479 of *Lecture Notes in Artificial Intelligence*, pages 205–221. Springer, Berlin, 2002.
10. Pascal Hitzler and Matthias Wendt. A uniform approach to logic programming semantics. *Theory and Practice of Logic Programming*, 200x. To appear.
11. Phokion G. Kolaitis and Christos H. Papadimitriou. Why not negation by fixpoint? In *PODS '88. Proceedings of the Seventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems: March 21–23, 1988, Austin, Texas*, New York, NY 10036, USA, 1988. ACM Press.
12. Teodor Przymusinski. Stable Semantics for Disjunctive Programs. *New Generation Computing Journal*, 9, 1991.
13. Teodor C. Przymusinski. On the declarative semantics of deductive databases and logic programs. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 193–216. Morgan Kaufmann, Los Altos, CA, 1988.
14. Sibylle Schwarz. Answer sets generated by selector functions. In *Proceedings of the Workshop on Nonmonotonic Reasoning'2002*, pages 247–253, Toulouse, 2002. http://www.tcs.hut.fi/~ini/nmr2002/schwarz.ps.
15. Sibylle Schwarz. Answer sets generated by selector functions. In Bertram Fronhöfer and Steffen Hölldobler, editors, *17. WLP: Workshop Logische Programmierung, TU Dresden, December 11–13, 2002*, number TUD–FI03–03 in Technische Berichte der Fakultät Informatik, pages 5–13. TU Dresden, 01062 Dresden, April 2002.
16. Sibylle Schwarz. *Selektor-erzeugte Modelle verallgemeinerter logischer Programme*. PhD thesis, Universität Leipzig, 2004. http://www.informatik.uni-leipzig.de/~schwarz/ps/thes.ps.gz.
17. Kewen Wang. A comparative study of well-founded semantics for disjunctive logic programs. In Thomas Eiter, Wolfgang Faber, and Miroslaw Truszczynski, editors, *Logic Programming and Nonmonotonic Reasoning, 6th International Conference, LPNMR 2001, Vienna, Austria, September 17-19, 2001, Proceedings*, volume 2173 of *Lecture Notes in Artificial Intelligence*, pages 133–146. Springer, 2001.

Available online at www.sciencedirect.com
SCIENCE DIRECT®
Journal of Applied Logic 2 (2004) 245–272
JOURNAL OF APPLIED LOGIC
www.elsevier.com/locate/jal

Logic programs and connectionist networks ☆

Pascal Hitzler [a], Steffen Hölldobler [a,*], Anthony Karel Seda [b]

[a] *Technische Universität Dresden, International Center for Computational Logic, 01062 Dresden, Germany*
[b] *Department of Mathematics, University College Cork, Cork, Ireland*

Available online 10 June 2004

Abstract

One facet of the question of integration of Logic and Connectionist Systems, and how these can complement each other, concerns the points of contact, in terms of semantics, between neural networks and logic programs. In this paper, we show that certain semantic operators for propositional logic programs can be computed by feedforward connectionist networks, and that the same semantic operators for first-order normal logic programs can be approximated by feedforward connectionist networks. Turning the networks into recurrent ones allows one also to approximate the models associated with the semantic operators. Our methods depend on a well-known theorem of Funahashi, and necessitate the study of when Funahashi's theorem can be applied, and also the study of what means of approximation are appropriate and significant.
© 2004 Elsevier B.V. All rights reserved.

Keywords: Logic programming; Metric spaces; Connectionist networks

1. Introduction

It is widely recognized that Logic and Neural Networks are two rather distinct yet major areas within Computing Science, and that each of them has proved to be especially important in relation to Artificial Intelligence, both in the context of its implementation and in the context of providing it with theoretical foundations. However, in many ways Logic, manifested through Computational Logic or Logic Programming, and Neural Net-

☆ This is a revised and extended treatment of results which to date have appeared only in [Proc. ECAI94 Workshop on Combining Symbolic and Connectionist Processing, ECCAI, 1994, pp. 68–77] and [Proc. Fourth Irish Workshop on Formal Methods, IWFM'00, Electronic Workshops in Computing (eWiC), British Computer Society, 2000; Proc. 26th German Conference on Artificial Intelligence, KI2003, Lecture Notes in Artificial Intelligence, vol. 2821, Springer, Berlin, 2003, pp. 105–119].

* Corresponding author.
E-mail address: sh@iccl.tu-dresden.de (S. Hölldobler).

1570-8683/$ – see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.jal.2004.03.002

works are quite complementary. For example, there is a widespread belief that the ability to represent and reason about structured objects and structure-sensitive processes is crucial for rational agents (see, for example, [20,42]), and Computational Logic is well-suited to doing this. On the other hand, rational agents should have additional properties which are not easily found in logic based systems such as, for example, the ability to learn, the ability to adapt to new environments, and the ability to degrade gracefully; these latter properties are typically met by Connectionist Systems or Neural Networks.

For such reasons, there is considerable interest in integrating the Logic based and Neural Network based approaches to Artificial Intelligence with a view to bringing together the advantages to be gained from connectionism and from symbolic AI. However, in attempting to do this, there are considerable obstacles to be overcome. For example, from the computational point of view, most connectionist systems developed so far are propositional in nature. John McCarthy called this a propositional fixation [39] in 1988, and not much has changed since then. Although it is known that connectionist systems are Turing-equivalent, we are unaware of any connectionist reasoning system which fully incorporates the power of symbolic computation. Systems like SHRUTI [47] or the BUR-calculus [27] allow n-place predicate symbols and a finite set of constants and, thus, are propositional in nature. Systems like CHCL [28] allow a fixed number of first-order clauses, but cannot copy clauses on demand and, thus, the entailment relation is decidable. Connectionist mechanisms for representing terms like holographic reduced representations [43] or recursive auto-associative memories [44] and variations thereof can handle some recursive structures, but as soon as the depth of the represented terms increases, the performance of these methods degrades quickly [40]. Furthermore, whilst logic programs have a rather well-developed theory of their semantics, it is not so clear how Neural Networks can be assigned any well-defined meaning which plays an important role comparable with that played by the supported models, the stable model or the well-founded model typically assigned to a logic program to capture its meaning.

It is an important fact that the models just mentioned are fixed points of various operators determined by programs. In particular, the supported models, or Clark completion semantics [9], of a normal logic program P coincide with the fixed points of the immediate consequence operator T_P. Furthermore, the fixed points themselves are frequently found by iterating the corresponding operators.

The previous observation establishes a clear semantical connection between logic programs and neural networks which is the main focus of study in this paper, and it arises because neural networks can be used to compute semantic operators such as T_P. Specifically, in this paper we develop this link between propositional (as well as first-order) logic programs and recursive networks. Our first main observation is that for any given propositional logic program P, one can construct a feedforward connectionist network which can compute the immediate consequence operator T_P. Unfortunately, the methods used in the propositional case do not extend immediately to the first-order case, and our second main observation is that approximation techniques can be used instead to approximate, arbitrarily well, both the semantic operators themselves and also their fixed points, at least if the feedforward networks are turned into recurrent ones. Our methods here are based on a well-known theorem of Funahashi [21] which shows that every continuous function on the reals can be uniformly approximated by a 3-layer feedforward neural network. However, application of Funahashi's theorem depends on T_P itself being continuous in a precise sense to be defined later. This in turn leads us to study conditions under which T_P meets this criterion, and in doing this we find it convenient to work with quite general semantic operators employing many valued logics. Furthermore, it also raises rather technical questions concerning what are the appropriate approximations to use.

Thus, the overall structure of the paper is as follows. In Section 2, we collect together the basic notions we need concerning logic programs, neural networks, and metric spaces. In Section 3, we establish our claim above that T_P can be computed, for propositional programs P, by feedforward connectionist networks. In Section 4, we take up the issue of extending the results of Section 3 to the first-order case by means of approximation. This involves a fairly detailed study of the (topological) continuity of semantic operators, extending results to be found in [49], before we can ultimately take up the question of applying results such as Funahashi's theorem and discussing measures of approximation appropriate to the study of neural networks. Finally, in Section 5, we present our conclusions and discuss future work. In essence, our techniques and thinking are somewhat in the spirit of dynamical systems, and provide a link between the areas of logic programming, topology and connectionist systems.

2. Basic notions

In this section, we collect together the basic concepts and notation we need from logic programming, metric spaces and connectionist networks, as can be found, for example, in [25,38,54]. A reader familiar with these notions may skip this section.

2.1. Logic programs

A (*normal*) *logic program* is a finite set of *clauses* of the form

$$\forall(A \leftarrow L_1 \wedge \cdots \wedge L_n),$$

where $n \in \mathbb{N}$ may differ for each clause, A is an atom in some first-order language $\mathcal{L}$ and $L_1, \ldots, L_n$ are literals, that is, atoms or negated atoms in $\mathcal{L}$. As is customary in logic programming, we will write such a clause in the form

$$A \leftarrow L_1 \wedge \cdots \wedge L_n,$$

in which the universal quantifier is understood. Then A is called the *head* of the clause, each L_i is called a *body literal* of the clause and their conjunction $L_1 \wedge \cdots \wedge L_n$ is called the *body* of the clause. We allow $n = 0$, by an abuse of notation, which indicates that the body is empty; in this case, the clause is called a *unit clause* or a *fact*. We will occasionally use the notation $A \leftarrow \texttt{body}$ for clauses, so that $\texttt{body}$ stands for the conjunction of the body literals of the clause. If no negation symbol occurs in a logic program, the program is called a *definite* logic program.

The Herbrand base underlying a given program P will be denoted by B_P, and the set of all Herbrand interpretations by I_P, and we note that the latter can be identified simultaneously with the power set of B_P and with the set $\mathbf{2}^{B_P}$ of all functions mapping B_P into the set **2** consisting of two distinct elements. The set **2** is usually considered to be the set $\{\mathbf{t}, \mathbf{f}\}$ of truth values. Any interpretation can be extended to literals, clauses and programs in the usual way. A *model* for P is an interpretation which maps P to $\mathbf{t}$. The *immediate consequence operator* (or *single-step operator*) T_P, mapping interpretations to interpretations, is defined as follows. Let I be an interpretation and let A be an atom. Then $T_P(I)(A) = \mathbf{t}$ if and only if there exists a ground instance $A \leftarrow L_1 \wedge \cdots \wedge L_n$ of a clause in P such that $I(L_1 \wedge \cdots \wedge L_n) = \mathbf{t}$. By ground$(P)$, we will denote the set of all ground instances of clauses in P.

The immediate consequence operator is a convenient tool for capturing the logical meaning, or semantics, of logic programs: an interpretation I is a model for a program P if and only if $T_P(I) \leqslant I$, that is, if and only if I is a pre-fixed point of T_P, where $\mathbf{2}^{B_P}$ is endowed with the pointwise ordering induced by the unique partial order defined on **2** in which $\mathbf{f} < \mathbf{t}$. Fixed points of T_P are called *supported models* for P. They coincide with the models for the so-called *Clark completion* of a program [9] and are considered to be particularly well-suited to capturing the intended meaning of logic programs.

2.2. Metric spaces and contraction mappings

Let X be a non-empty set. A function $d : X \times X \to R$ is called a *metric* (*on* X), and the pair (X, d) is called a *metric space*, if the following properties are satisfied.

1. For all $x, y \in X$, we have $d(x, y) \geqslant 0$ and $d(x, y) = 0$ iff $x = y$.
2. For all $x, y \in X$, we have $d(x, y) = d(y, x)$.
3. For all $x, y, z \in X$, we have $d(x, z) \leqslant d(x, y) + d(y, z)$.

Let d be a metric defined on a set X. Then a sequence (x_n) in X is said to *converge to* $x \in X$, and x is called the *limit* of (x_n), if, for each $\varepsilon > 0$, there is a natural number n_0 such that for all $n \geqslant n_0$ we have $d(x_n, x) < \varepsilon$. Note that the limit of any sequence is unique if it exists. Furthermore, a sequence (x_n) is said to be a *Cauchy sequence* if, for each $\varepsilon > 0$, there is a natural number n_0 such that whenever $m, n \geqslant n_0$ we have $d(x_m, x_n) < \varepsilon$. It is clear that any sequence which converges is a Cauchy sequence. On the other hand, a metric space (X, d) is called *complete* if every Cauchy sequence in X converges.

Let (X, d) be a metric space. Then a function $f : X \to X$ is called a *contraction mapping* or simply a *contraction* if there exists a real number $\lambda \in [0, 1)$ satisfying $d(f(x), f(y)) \leqslant \lambda\, d(x, y)$ for all $x, y \in X$. Finally, an element x_0 (of a set X) is called a *fixed point* of a function $f : X \to X$ if, as usual, we have $f(x_0) = x_0$.

One of the main results concerning contraction mappings defined on complete metric spaces is the following well-known theorem.

Theorem 2.1 (Banach Contraction Mapping Theorem [54]). *Let f be a contraction mapping defined on a complete metric space (X, d). Then f has a unique fixed point $x_0 \in X$. Furthermore, the sequence $x, f(x), f(f(x)), \ldots$ converges to x_0 for any $x \in X$.*

If a program P is such that there exists a metric which renders T_P a contraction, then Theorem 2.1 shows that P has a unique supported model. Semantic analysis of logic programs along these general lines was initiated in [18], and has subsequently been studied and generalized by a number of authors. The recent publication [34] contains both a state-of-the-art treatment using this approach and a comprehensive list of references on this topic.

The following definition will be very convenient for our purposes.

Definition 2.2. A normal logic program P is called *strongly determined* if there exists a complete metric d on I_P such that T_P is a contraction with respect to d.

It follows from Theorem 2.1 that every strongly determined program has a unique supported model, that is, is *uniquely determined*. Certain well-known classes of programs turn out to contain only strongly determined programs, amongst these are the classes of acyclic and acceptable programs [3,5,8,18], which are fundamental in termination analysis under Prolog. More generally, all programs called Φ_ω-accessible in [34] are strongly determined. Indeed, we will take the trouble to define acyclic programs next since we will need this notion in subsequent discussions. To do this, we need first to recall the notion of level mapping, familiar in the context of studies of termination, see [3] for example.

A *level mapping* for a program P is a mapping $l : B_P \to \alpha$ for some ordinal α. As usual, we always assume that l has been extended to all literals by setting $l(\neg A) = l(A)$ for each $A \in B_P$. An *ω-level mapping* for P is a level mapping $l : B_P \to \mathbb{N}$.

Definition 2.3. A logic program P is called *acyclic* if there exists an ω-level mapping for P such that for each clause $A \leftarrow L_1 \wedge \cdots \wedge L_n$ in $\mathrm{ground}(P)$ we have $l(A) > l(L_i)$ for all $i = 1, \ldots, n$.

2.3. Connectionist networks

A *connectionist network* is a directed graph. A *unit* k in this graph is characterized, at time t, by its *input vector* $(i_{k1}(t), \ldots, i_{kn_k}(t))$, its *potential* $p_k(t) \in \mathbb{R}$, its *threshold* $\theta_k \in \mathbb{R}$, and its *value* $v_k(t)$. Units are connected via a set of directed and weighted connections. If there is a connection from unit j to unit k, then $w_{kj} \in \mathbb{R}$ denotes the *weight* associated with this connection, and $i_{kj}(t) = w_{kj} v_j(t)$ is the *input* received by k from j at time t. Fig. 1 shows a typical unit. The units are updated synchronously. In each update, the potential and value of a unit are computed with respect to an *activation* and an *output function* respectively. All units considered in this paper compute their potential as the weighted sum of their inputs minus their threshold:

$$p_k(t) = \left(\sum_{j=1}^{n_k} w_{kj} v_j(t) \right) - \theta_k.$$

Having fixed the activation function, we consider three types of units mainly distinguished by their output function. A unit is said to be a *binary threshold unit* if its output function is a threshold function:

$$v_k(t + \Delta t) = \begin{cases} 1 & \text{if } p_k(t) \geqslant 0, \\ 0 & \text{otherwise.} \end{cases}$$

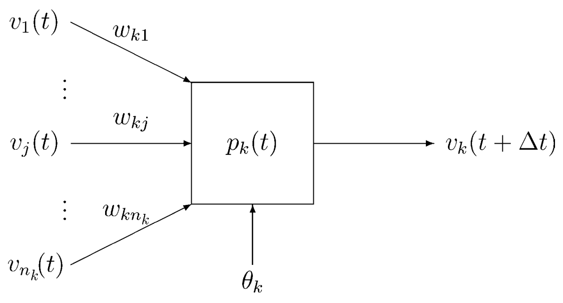


Fig. 1. Unit k in a connectionist network.

A unit is said to be a *linear unit* if its output function is the identity and its threshold θ is 0. A unit is said to be a *sigmoidal* or *squashing unit* if its output function ϕ is non-decreasing and is such that $\lim_{t\to\infty}(\phi(p_k(t))) = 1$ and $\lim_{t\to-\infty}(\phi(p_k(t))) = 0$. Such functions are called *squashing functions*.

In this paper, we will only consider connectionist networks where the units can be organized in layers. A *layer* is a vector of units. An *n-layer feedforward network $\mathcal{F}$* consists of the *input* layer, $n-2$ *hidden* layers, and the *output* layer, where $n \geqslant 2$. Each unit occurring in the ith layer is connected to each unit occurring in the $(i+1)$st layer, $1 \leqslant i < n$. Let r and s be the number of units occurring in the input and output layers, respectively. A connectionist network $\mathcal{F}$ is called a *multilayer feedforward network* if it is an n-layer feedforward network for some n. A multilayer feedforward network $\mathcal{F}$ computes a function $f_{\mathcal{F}}: \mathbb{R}^r \to \mathbb{R}^s$ as follows. The input vector (the argument of $f_{\mathcal{F}}$) is presented to the input layer at time t_0 and propagated through the hidden layers to the output layer. At each time point, all units update their potential and value. At time $t_0 + (n-1)\Delta t$, the output vector (the image under $f_{\mathcal{F}}$ of the input vector) is read off the output layer.

For a 3-layer network with r linear units in the input layer, squashing units in the hidden layer, and a single linear unit in the output layer, the input-output function of the network as described above can thus be obtained as a mapping $f: \mathbb{R}^r \to \mathbb{R}$ with

$$f(x_1,\dots,x_r) = \sum_j c_j \phi\left(\sum_i w_{ji}x_i - \theta_j\right),$$

where c_j is the weight associated with the connection from the jth unit of the hidden layer to the single unit in the output layer, ϕ is the squashing output function of the units in the hidden layer, w_{ji} is the weight associated with the connection from the ith unit of the input layer to the jth unit of the hidden layer and θ_j is the threshold of the jth unit of the hidden layer.

It is our aim to obtain results on the representation or approximation of consequence operators by input-output functions of 3-layer feedforward networks. Some of our results rest on the following theorem, which is due to Funahashi, see [21].

Theorem 2.4. *Suppose that $\phi: \mathbb{R} \to \mathbb{R}$ is a non-constant, bounded, monotone increasing and continuous function. Let $K \subseteq \mathbb{R}^n$ be compact, let $f: K \to \mathbb{R}$ be a continuous mapping and let $\varepsilon > 0$. Then there exists a* 3*-layer feedforward network with squashing function ϕ*

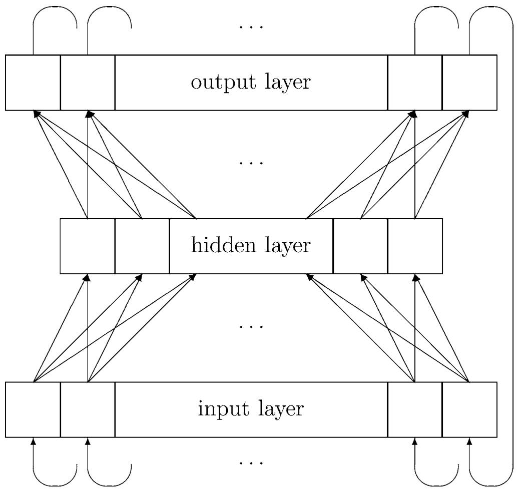


Fig. 2. Sketch of a 3-layered recurrent network.

whose input-output mapping $\bar{f}: K \to \mathbb{R}$ *satisfies* $\max_{x\in K} d(f(x), \bar{f}(x)) < \varepsilon$, *where* d *is a metric which induces the natural topology*[1] *on* $\mathbb{R}$.

In other words, each continuous function $f: K \to \mathbb{R}$ can be uniformly approximated by input-output functions of 3-layer networks. For our purposes, it will suffice to assume that K is a compact subset of the set of real numbers, so that $n = 1$ in the statement of the theorem.

An *n-layer recurrent network* $\mathcal{N}$ consists of an n-layer feedforward network such that the number of units in the input and output layer are identical. Furthermore, each unit in the kth position of the output layer is connected with weight 1 to the unit in the kth position of the input layer, where $1 \leqslant k \leqslant N$ and N is the number of units in the output (or input) layer. Fig. 2 shows a 3-layer recurrent network. The subnetwork consisting of the three layers and the connections between the input and the hidden as well as between the hidden and the output layer is a 3-layer feedforward network called the *kernel* of $\mathcal{N}$.

3. Propositional logic programs

In this section, we consider the propositional case following [24] and show that for each logic program P we can construct a 3-layer feedforward network of binary threshold units computing T_P. Turning such a network into a recurrent one allows one to compute the unique fixed point of T_P provided that P is strongly determined.

The main question addressed in this section is: can we specify a connectionist network of binary threshold units for a propositional logic program P such that it computes T_P

[1] For example, $d(x, y) = |x - y|$.

and, if it exists, the least fixed point of T_P? It is well-known that 3-layer feedforward connectionist networks with sigmoidal hidden layer are universal approximators [21,35]. Hence, we expect that recurrent networks with a 3-layer feedforward kernel will do, where the kernel computes T_P and, by the recurrent connections, T_P is iterated.

The question addressed in the following subsection is whether or not even simpler networks, viz. recurrent networks with a 2-layer feedforward kernel of binary threshold units will do. Such networks are called *perceptrons* [46]. It is well-known that their computing capabilities are limited to computing solutions for linearly separable problems [41].

3.1. Hidden layers are needed

Usually, the need for a hidden layer is shown by demonstrating that the exclusive-or cannot be modelled by a feedforward network without hidden layers (see [41], for example). A straightforward program to compute the exclusive-or of two propositional atoms A and B such as the program

$$P_1 = \{C \leftarrow A \wedge \neg B,\ C \leftarrow \neg A \wedge B\}$$

is not definite and from this we can only conclude that 2-layer feedforward networks cannot compute T_P for normal P. An even stronger result is the following.

Proposition 3.1. 2-*layer connectionist networks of binary threshold units cannot compute T_P for definite P.*

Proof. Consider the following program

$$P_2 = \{A \leftarrow B,\ A \leftarrow C \wedge D,\ A \leftarrow E \wedge F\}.$$

Let $\mathcal{F}$ be the 2-layer feedforward network of binary threshold units shown in Fig. 3 and assume that the weights in $\mathcal{F}$ are selected in such a way that it computes T_{P_2}. Let $w_{ij} = 0$

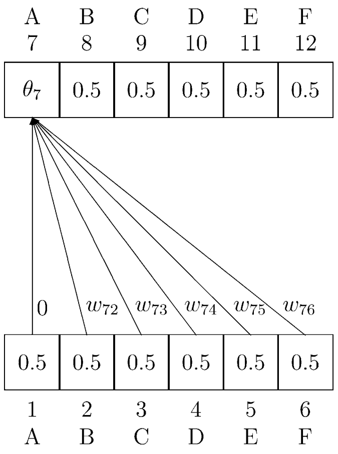


Fig. 3. A 2-layer feedforward network of binary threshold units for P_2. The numbers occurring within the units are thresholds. Connections which are not shown have weight 0.

and $\theta_i = 0.5$ if $i \in [8, 12]$, so that no unit encoding the atoms B to F in the output layer will ever become active and this property is, moreover, independent of the activation pattern of the input layer. Thus, as far as these units are concerned, the network behaves correctly as no atom B to F is evaluated to **t** by $T_{P_2}(I)$ for any interpretation I. For unit 7 to behave correctly, we have to find a threshold θ_7 and weights w_{7j}, $1 \leqslant j \leqslant 6$, such that

$$\begin{aligned} &T_{P_2}(I)(A) = \mathbf{t} \quad \text{iff} \\ &w_{71}v_1 + w_{72}v_2 + w_{73}v_3 + w_{74}v_4 + w_{75}v_5 + w_{76}v_6 - \theta_7 \geqslant 0, \end{aligned} \tag{1}$$

where $I = (v_1, \dots, v_6)$ is the current interpretation, that is, the activation or output pattern of the input layer. Obviously, the output of unit 1 should not influence the potential of unit 7 and hence $w_{71} = 0$. Thus, (1) reduces to

$$T_{P_2}(I)(A) = \mathbf{t} \quad \text{iff} \quad w_{72}v_2 + w_{73}v_3 + w_{74}v_4 + w_{75}v_5 + w_{76}v_6 - \theta_7 \geqslant 0. \tag{2}$$

As the conjunction in the conditions of clauses is commutative, (2) can be transformed to

$$T_{P_2}(I)(A) = \mathbf{t} \quad \text{iff} \quad w_{72}v_2 + w_{74}v_3 + w_{73}v_4 + w_{75}v_5 + w_{76}v_6 - \theta_7 \geqslant 0$$

and

$$T_{P_2}(I)(A) = \mathbf{t} \quad \text{iff} \quad w_{72}v_2 + w_{73}v_3 + w_{74}v_4 + w_{76}v_5 + w_{75}v_6 - \theta_7 \geqslant 0.$$

Hence, with $w_1 = \frac{1}{2}(w_{73} + w_{74})$ and $w_2 = \frac{1}{2}(w_{75} + w_{76})$ Eq. (2) becomes

$$T_{P_2}(I)(A) = \mathbf{t} \quad \text{iff} \quad w_{72}v_2 + w_1(v_3 + v_4) + w_2(v_5 + v_6) - \theta_7 \geqslant 0. \tag{3}$$

As the disjunction between clauses is commutative, using an argument similar to that used before we find $w = \frac{1}{3}(w_{72} + w_1 + w_2)$ such that (3) becomes

$$T_{P_2}(I)(A) = \mathbf{t} \quad \text{iff} \quad w(v_2 + v_3 + v_4 + v_5 + v_6) - \theta_7 \geqslant 0. \tag{4}$$

Thus, with $x = \sum_{j=2}^{6} v_j$ we obtain the polynomial $wx - \theta_7$. Now, for $\mathcal{F}$ to compute T_{P_2} the following must hold.

$$\begin{array}{llll} wx - \theta_7 < 0 & \text{if} & x = 0 & (v_2 = \cdots = v_6 = 0). \\ wx - \theta_7 \geqslant 0 & \text{if} & x = 1 & (v_2 = 1,\ v_3 = \cdots = v_6 = 0). \\ wx - \theta_7 < 0 & \text{if} & x = 2 & (v_2 = v_4 = v_6 = 0,\ v_3 = v_5 = 1). \end{array}$$

However, the first derivative of the polynomial $wx - \theta_7$ cannot change its sign and, consequently, there cannot be weights and thresholds such that the 2-layer feedforward network computes T_{P_2}. □

This result shows the need for hidden layers and it is easy to verify that the 3-layer feedforward network of binary threshold units shown in Fig. 4 computes T_{P_2} for the program P_2.

One should observe that each rule R in P_2 is mapped from the input to the output layer through exactly one unit in the hidden layer. The potential of this unit is greater than 0 at $t_0 + \Delta t$ and, thus, the unit becomes active at $t_0 + \Delta t$ if and only if each unit in the input layer representing a condition of R is active at t_0, that is, if and only if each condition of R

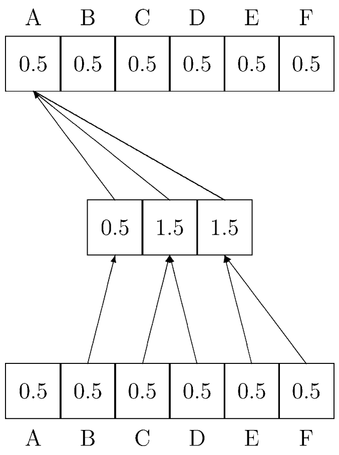


Fig. 4. A 3-layer feedforward network of binary threshold units computing T_{P_2}. Only connections with non-zero weights are shown, and these connections have weight 1. The numbers occurring within units denote thresholds.

is assigned **t**. The potential of the output unit representing A is greater than 0 at $t_0 + 2\Delta t$ and, thus, the unit becomes active at $t_0 + 2\Delta t$ if and only if at least one hidden unit that is connected to A is active at $t_0 + \Delta t$.

Consequently, the number of units in the hidden layer as well as the number of connections between the hidden and the output layer with non-zero weight is equal to the number of clauses in P. Furthermore, the number of connections between the input and the hidden layer with non-zero weight is equal to the number of literals occurring in the conditions of program clauses, and the number of units in the input and output layers is equal to the number of propositional variables occurring in the program. Hence, the size of the network is bounded by the size of the program, and the operator T_P is computed in constant time, viz. in 2 steps.

These construction principles are extended to normal programs in the following subsection.

3.2. Relating propositional programs to networks

Theorem 3.2. *For each program P, there exists a* 3*-layer feedforward network computing T_P.*

Proof. Let m and n be the number of propositional variables and the number of clauses occurring in P, respectively. Without loss of generality, we may assume that the variables are ordered. The network associated with P can now be constructed by the following *translation algorithm*:

1. The input and output layer is a vector of binary threshold units of length m, where the ith unit in the input and output layer represents the ith variable, $1 \leqslant i \leqslant m$. The threshold of each unit occurring in the input or output layer is set to 0.5.
2. For each clause of the form $A \leftarrow L_1 \wedge \cdots \wedge L_k$, $k \geqslant 0$, occurring in P, do the following.

2.1. Add a binary threshold unit c to the hidden layer.
2.2. Connect c to the unit representing A in the output layer with weight 1.
2.3. For each literal L_j, $1 \leqslant j \leqslant k$, connect the unit representing L_j in the input layer to c and, if L_j is an atom, then set the weight to 1; otherwise set the weight to -1.
2.4. Set the threshold θ_c of c to $l - 0.5$, where l is the number of positive literals occurring in $L_1 \wedge \cdots \wedge L_k$.

Each interpretation I for P can be represented by a binary vector $(v_1, \ldots, v_m)$. Such an interpretation is given as input to the network by externally activating corresponding units of the input layer at time t_0. It remains to show that $T_P(I)(A) = \mathbf{t}$ if and only if the unit representing A in the output layer becomes active at time $t_0 + 2\Delta t$.

If $T_P(I)(A) = \mathbf{t}$, then there is a clause $A \leftarrow L_1 \wedge \cdots \wedge L_k$ in P such that for all $1 \leqslant j \leqslant k$ we have $I(L_j) = \mathbf{t}$. Let c be the unit in the hidden layer associated with this clause according to item 2.1 of the construction. From 2.3 and 2.4 we conclude that c becomes active at time $t_0 + \Delta t$. Consequently, 2.2 and the fact that units occurring in the output layer have a threshold of 0.5 (see item 1) ensure that the unit representing A in the output layer becomes active at time $t_0 + 2\Delta t$.

Conversely, suppose that the unit representing the atom A in the output layer becomes active at time $t_0 + 2\Delta t$. From the construction of the network, we find a unit c in the hidden layer which must have become active at time $t_0 + \Delta t$. This unit is associated with a clause $A \leftarrow L_1 \wedge \cdots \wedge L_k$. If $k = 0$, that is, if the body of the clause is empty, then, according to item 2.4, c has a threshold of -0.5. Furthermore, according to item 2.3, c does not receive any input, that is, $p_c = 0 + 0.5$ and consequently c will always be active. Otherwise, if $k \geqslant 1$, then c becomes active only if each unit in the input layer representing a positive literal and no unit representing a negative literal in the body of the clause is active at time t_0 (see items 2.3 and 2.4). Hence, we have found a clause $A \leftarrow L_1 \wedge \cdots \wedge L_k$ such that for all $1 \leqslant j \leqslant k$ we have $I(L_j) = \mathbf{t}$ and consequently $T_P(I)(A) = \mathbf{t}$. □

As an example, reconsider

$$P_1 = \{C \leftarrow A \wedge \neg B,\ C \leftarrow \neg A \wedge B\}$$

and extend it to

$$P_3 = \{A,\ C \leftarrow A \wedge \neg B,\ C \leftarrow \neg A \wedge B\}.$$

Their corresponding connectionist networks are shown in Fig. 5. One should observe that P_3 exemplifies the representation of unit clauses in 3-layer feedforward networks.[2]

As already mentioned at the end of Section 3.1, the number of units and the number of connections in a network $\mathcal{F}$ corresponding to a program P are bounded by $\mathrm{O}(m + n)$ and $\mathrm{O}(m \times n)$, respectively, where n is the number of clauses and m is the number of propositional variables occurring in P. Furthermore, $T_P(I)$ is computed in 2 steps. As

[2] We can save the unit in the hidden layer corresponding to the unit clause, if we change the threshold of the unit representing A in the output layer to -0.5.

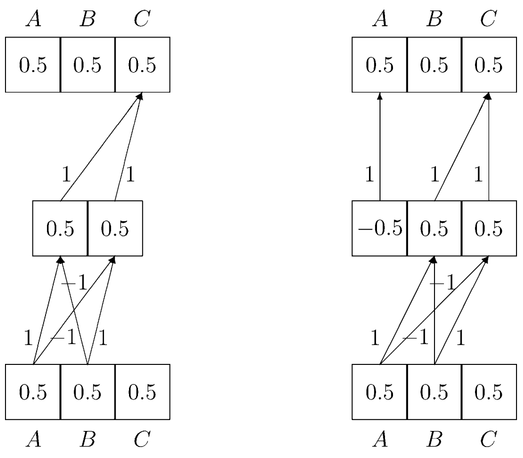


Fig. 5. Two 3-layer feedforward networks of binary threshold units computing T_{P_1} and T_{P_3}, respectively. Only connections with non-zero weight are shown. The number occurring within units denote thresholds.

the sequential time to compute $T_P(I)$ is bounded by $\mathrm{O}(n \times m)$ (assuming that no literal occurs more than once in the conditions of a clause), the parallel computational model is optimal.[3]

We can now apply the Banach contraction mapping theorem, Theorem 2.1, to obtain the following result.

Corollary 3.3. *Let P be a strongly determined* (*propositional*) *program. Then there exists a* 3-*layer recurrent network such that each computation starting with an arbitrary initial input converges and yields the unique fixed point of T_P, that is, the unique supported model for P.*

Let us mention in passing that a kind of converse of Corollary 3.3 also holds, as follows. Let P be a (propositional) program such that the corresponding network has the property that each computation starting with an arbitrary initial input converges, and in all cases converges to the same state. Then this means that iteration of the T_P-operator exhibits the same behaviour, that is, for each initial interpretation it yields one and the same constant value after a finite number of iterations. By [31, Theorem 2], this suffices to guarantee the existence of a complete metric which renders T_P a contraction. A direct proof of this observation is given in [24].

Returning to the programs P_1 and P_3 again, we observe that both programs are strongly determined.[4] Hence, Fig. 5 shows the kernels of corresponding recurrent networks which

[3] A parallel computational model requiring $p(n)$ processors and $t(n)$ time to solve a problem of size n is *optimal* if $p(n) \times t(n) = \mathrm{O}(T(n))$, where $T(n)$ is the sequential time to solve this problem (see for example [37]).

[4] They are even acceptable, as can be seen by mapping C to 2, and A as well as B to 1 and considering the model $I(A) = I(C) = \mathbf{t}$ and $I(B) = \mathbf{f}$.

compute the least fixed point of T_{P_1} (the interpretation represented by the vector $(0,0,0)$) and of T_{P_3} (the interpretation represented by the vector $(1,0,1)$).

The time needed by the network to settle down into the unique stable state is equal to the time needed by a sequential machine to compute the least fixed point of T_P in the worst case. As an example, consider the definite program

$$P_4 = \{A_1\} \cup \{A_{i+1} \leftarrow A_i \mid 1 \leqslant i < n\}.$$

The least fixed point of T_P is the interpretation which evaluates each A_i, $1 \leqslant i \leqslant n$, to **t**. Using the technique described in [10] and [48], it can be computed in $\mathrm{O}(n)$ steps.[5] Obviously, the parallel computational model needs as many steps. More generally, let P be a definite program containing n clauses. The time needed by the network to settle down into the unique stable state is $3n$ in the worst case and, thus, the time is linear with respect to the number of clauses occurring in the program. This comes as no surprise as it follows from [36] that satisfiability of propositional Horn formulae is P-complete and, thus, is unlikely to be in the class NC (see for example [37]). On the other hand, consider the program

$$P_5 = \{A_i \mid 1 \leqslant i \leqslant n \text{ and } i \text{ even}\} \cup \{A_{i+1} \leftarrow A_i \mid 1 \leqslant i \leqslant n \text{ and } i \text{ even}\}.$$

The least model mapping each atom to **t** is computed in five steps by the recurrent network corresponding to P_5.

3.3. Extensions

In this subsection, various extensions of the basic model developed in Section 3.2 are briefly discussed. In particular, we focus on learning, rule extraction and propositional modal logics.

Learning. The networks corresponding to logic programs and constructed by the translation algorithm presented in the proof of Theorem 3.2 cannot be trained by the usual learning methods applied to connectionist systems. It was observed in [15] (see also [12, 14]) that results similar to Theorem 3.2 and Corollary 3.3 can be achieved if the binary threshold units occurring in the hidden layer of the feedforward kernels are replaced by sigmoidal units. We omit the technical details here and refer to the abovementioned literature. Such a move renders the kernels accessible to the backpropagation algorithm, a standard technique for training feedforward networks [45].

Rule extraction. After training a feedforward network with sigmoidal units in the hidden layer, the knowledge encoded in the network is mostly inaccessible to a human without postprocessing. Numerous techniques have been proposed to extract rules from trained feedforward networks (see for example [1] and [11]). We can now envision a cycle in which a given (preliminary) logic program is translated into a feedforward network, this

[5] To be precise, the algorithm described in [10] needs $\mathrm{O}(n)$ time, where n denotes the total number of occurrences of propositional variables in the formula.

network is trained by examples using backpropagation, and a new (refined) logic program is extracted from the network after training (see [51]). The reference [12] contains several examples of such cyclic knowledge processing.

Propositional modal logics. The approach discussed so far has been extended to (*propositional*) *modal programs*, where literals occurring in a clause may be prefixed by the modalities $\Box$ and $\Diamond$, clauses are labelled by the world in which they hold, and a finite set of relations between worlds is given [13]. It was shown that Theorem 3.2 can be extended to such modal programs in that for each such program there exists a 3-layer connectionist network computing the modal fixed point operator of the given program. The main idea is to construct for each world a 3-layer feedforward network using a variation of the translation algorithm specified in the proof of Theorem 3.2 and then to connect the worlds with respect to the given set of relations between worlds and the usual Kripke semantics of the modalities. It is an interesting open problem to show how to model the temporal aspects of reasoning with respect to modal programs within a connectionist setting other than by just copying the complete network from one point in time to the next one.

4. First-order logic programs

In this section, we extend the approach presented in Section 3 to the first-order case. In particular, we consider conditions under which semantic operators for first-order logic programs as well as their fixed points can be approximated by connectionist networks.

In the first-order case, (Herbrand) interpretations usually consist of countably many ground atoms. Hence, the simple solution for the propositional case, where each ground atom is represented by a binary threshold unit in the input and the output layer, is no longer feasible. To extend the representational capability of the networks used, binary threshold units are replaced by sigmoidal ones. The values generated by sigmoidal units are real numbers, and we will use real numbers to represent interpretations. In Fig. 6, the recurrent nets considered in this section are sketched. This section extends results published in [26] and therefore we review the previous work in the following subsection.

4.1. Previous work

The reference [26] was concerned with the following problem. Suppose we are given a first-order logic program P together with a continuous consequence operator $T_P : 2^{B_P} \to 2^{B_P}$, where B_P is the Herbrand base of P. We want to know whether or not there exists a class of logic programs such that for each program in this class we can find an invertible mapping $\iota : 2^{B_P} \to \mathbb{R}$ and a function $f_P : \mathbb{R} \to \mathbb{R}$ satisfying the following conditions:

1. $T_P(I) = I'$ implies $f_P(\iota(I)) = \iota(I')$ and $f_P(r) = r'$ implies $T_P(\iota^{-1}(r)) = \iota^{-1}(r')$,
2. T_P is a contraction on 2^{B_P} iff f_P is a contraction on $\mathbb{R}$, and
3. f_P is continuous on $\mathbb{R}$.

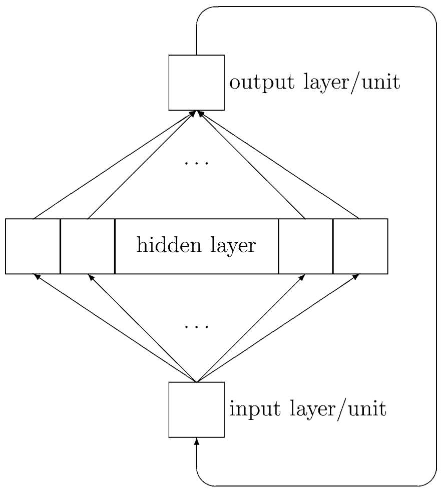


Fig. 6. Sketch of a recurrent network for a first-order logic program.

The first condition ensures that f_P is a sound and complete encoding of T_P. The second condition ensures that the contraction property, and thus fixed points, are preserved. The third condition ensures that we can apply Theorem 2.4 which then yields a 3-layer feedforward network with sigmoidal units in the hidden layer approximating f_P arbitrarily well. Moreover, the corresponding recurrent network approximates the least fixed point of T_P arbitrarily well also.

It was shown in [26] that this problem can be solved for the class of acyclic logic programs with injective level mapping. In the following, we will lift some of these observations to a much more general level, see [30,33]. In particular, we will show that acyclic programs with injective level mappings represent only a small fraction of the programs for which f_P can be approximated satisfactorily. We will also abstract from the single-step operator and generalize the approach to more general types of semantic operators.

Throughout the rest of the paper, we will make substantial use of elementary notions and results from topology, and our standard background reference to this subject is [54]. Indeed, the results presented subsequently are based on the observation that acyclicity with respect to an injective level mapping is a sufficient, but not necessary, condition for continuity of the single-step operator with respect to a topology which is homeomorphic to the Cantor topology on the real line, namely, the *query* or *atomic topology* studied in [7,49] and elsewhere in logic programming. We will therefore start by studying the basic topological facts relevant to our task before turning to the applications we ultimately want to make of these ideas and methods.

4.2. Continuity of semantic operators

From now on, we will impose the standing condition on the language $\mathcal{L}$ that it contains at least one constant symbol and at least one function symbol with arity greater than 0. If this is not done, then ground(P) may be a finite set of ground instances of clauses, and can be treated essentially as a propositional program, for which appropriate methods were laid out in Section 3.

In logic programming semantics, it has turned out to be both useful and convenient to use many-valued logics. Our investigations will therefore begin by studying suitable topologies on spaces of many-valued interpretations. We assume we have given a finite set $\mathcal{T} = \{t_1, \dots, t_n\}$ of truth values containing at least the two distinguished values t_1 and t_n, which are interpreted as being the truth values for "false", and "true", respectively. We also assume that we have truth tables for the usual connectives $\vee$, $\wedge$, $\leftarrow$, and $\neg$. Given a logic program P, we denote the set of all (Herbrand) *interpretations* or *valuations* in this logic by $I_{P,n}$; thus $I_{P,n}$ is the set $\mathcal{T}^{B_P}$ of all functions $I : B_P \to \mathcal{T}$. If n is clear from the context, we will use the notation I_P instead of $I_{P,n}$ and we note that this usage is consistent with the one given above for $n = 2$. As usual, any interpretation I can be extended, using the truth tables, to give a truth value in $\mathcal{T}$ to any variable-free formula in $\mathcal{L}$.

Definition 4.1. Given any logic program P, the *generalized atomic topology* $\mathcal{Q}$ on $I_P = I_{P,n}$ is defined to be the product topology on $\mathcal{T}^{B_P}$, where $\mathcal{T} = \{t_1, \dots, t_n\}$ is endowed with the discrete topology.

We note that this topology can be defined analogously for the non-Herbrand case. For $n = 2$, the generalized atomic topology $\mathcal{Q}$ specializes to the query topology of [7] (in the Herbrand case) and to the atomic topology Q of [49] (in the non-Herbrand case). The following results follow immediately since $\mathcal{Q}$ is a product of the discrete topology on a finite set, and hence is a topology of pointwise convergence.

Proposition 4.2. *For $A \in B_P$ and t_i a truth value, let $\mathcal{G}(A, t_i) = \{I \in I_{P,n} \mid I(A) = t_i\}$. Then the following hold.*

(a) *$\mathcal{Q}$ is the topology generated by the subbase $\mathcal{G} = \{\mathcal{G}(A, t_i) \mid A \in B_P,\ i \in \{1, \dots, n\}\}$.*
(b) *A net (I_λ) in I_P converges in $\mathcal{Q}$ to I in I_P if and only if for every $A \in B_P$ there exists some λ_0 such that $I_\lambda(A)$ is constant and equal to $I(A)$ for all $\lambda \geqslant \lambda_0$.*
(c) *$\mathcal{Q}$ is a second countable totally disconnected compact Hausdorff topology which is dense in itself. Hence, $\mathcal{Q}$ is metrizable and homeomorphic to the Cantor topology on the unit interval in the real line.*

We note that the second countability of $\mathcal{Q}$ rests on the fact that B_P is countable, so that this property does not in general carry over to the non-Herbrand case.

The study of topologies such as $\mathcal{Q}$ comes from our desire to be able to control the iterative behaviour of semantic operators. Topologies which are closely related to order structures, as common in denotational semantics [2], are of limited applicability since non-monotonic operators frequently arise naturally in the logic programming context. See also [23] for a study of these issues.

We proceed next with studying a rather general notion of semantic operator, akin to Fitting's approach in [19], which generalizes standard notions occurring in the literature.

Definition 4.3. An operator T on I_P is called a *consequence operator* for P if for every $I \in I_P$ the following condition holds: for every ground clause $A \leftarrow \mathtt{body}$ in P, where $T(I)(A) = t_i$, say, and $I(\mathtt{body}) = t_j$, say, we have that the truth table for $t_i \leftarrow t_j$ yields the truth value t_n, that is, "true".

It turns out that this notion of consequence operator relates nicely to $\mathcal{Q}$, yielding the following result which was reported in [23,32]. If T is a consequence operator for P and if for any $I \in I_P$ we have that the sequence of iterates $T^m(I)$ converges in $\mathcal{Q}$ to some $M \in I_P$, then M is a model, in a natural sense, for P. Furthermore, continuity of T yields the desirable property that M is a fixed point of T.

Intuitively, consequence operators should propagate "truth" along the implication symbols occurring in the program. From this point of view, we would like the outcome of the truth value of such a propagation to be dependent only on the relevant clause bodies. The next definition captures this intuition.

Definition 4.4. Let $A \in B_P$ and denote by $\mathcal{B}_A$ the set of all body atoms of clauses with head A that occur in ground(P). A consequence operator T is called $(P$-$)$*local* if for every $A \in B_P$ and any two interpretations $I, K \in I_P$ which agree on all atoms in $\mathcal{B}_A$, we have $T(I)(A) = T(K)(A)$.

It is our desire to study continuity in $\mathcal{Q}$ of local consequence operators. Since $\mathcal{Q}$ is a product topology, it is reasonable to expect that finiteness conditions will be involved, and indeed conditions which ensure finiteness in the sense of Definition 4.5 below, due to [49], have made their appearance in this context.

Definition 4.5. Let C be a clause in P and let $A \in B_P$ be such that A coincides with the head of C. The clause C is said to be of *finite type relative to* A if C has only finitely many different ground instances with head A. The program P will be said to be of *finite type relative to* A if each clause in P is of finite type relative to A, that is, if the set of all clauses in ground(P) with head A is finite. Finally, P will be said to be of *finite type* if P is of finite type relative to A for every $A \in B_P$.

A *local variable* is a variable which appears in a clause body but not in the corresponding head. Local variables appear naturally in practical logic programs, but their occurrence is awkward from the point of view of denotational semantics, especially if they occur in negated body literals since this leads to the so-called floundering problem, see [38].

It is easy to see that, in the context of Herbrand-interpretations, and if function symbols are present, then the absence of local variables is equivalent to a program being of finite type.

Proposition 4.6. *Let P be a logic program of finite type and let T be a local consequence operator for P. Then T is continuous in $\mathcal{Q}$.*

Proof. Let $I \in I_P$ be an interpretation and let $G_2 = \mathcal{G}(A, t_i)$ be a subbasic neighbourhood of $T(I)$ in $\mathcal{Q}$, and note that G_2 is the set of all $K \in I_P$ such that $K(A) = t_i$. We need to find a neighbourhood G_1 of I such that $T(G_1) \subseteq G_2$. Since P is of finite type, the set $\mathcal{B}_A$ is finite. Hence, the set $G_1 = \bigcap_{B \in \mathcal{B}_A} \mathcal{G}(B, I(B))$ is a finite intersection of open sets and is therefore open. Since each $K \in G_1$ agrees with I on $\mathcal{B}_A$, we obtain $T(K)(A) = T(I)(A) = t_i$ for each $K \in G_1$ by locality of T. Hence, $T(G_1) \subseteq G_2$. □

Now, if P is not of finite type, but we can ensure by some other property of P that the possibly infinite intersection $\bigcap_{B \in \mathcal{B}_A} \mathcal{G}(B, I(B))$ is open, then the above proof will carry over to programs which are not of finite type. Alternatively, we would like to be able to disregard the infinite intersection entirely under conditions which ensure that we have to consider finite intersections only, as in the case of a program of finite type. The following definition is, therefore, quite a natural one to make.

Definition 4.7. Let P be a logic program and let T be a consequence operator on I_P. We say that T is (P-)*locally finite for* $A \in B_P$ *and* $I \in I_P$ if there exists a finite subset $S = S(A, I) \subseteq \mathcal{B}_A$ such that we have $T(J)(A) = T(I)(A)$ for all $J \in I_P$ which agree with I on S. We say that T is (P-)*locally finite* if it is locally finite for all $A \in B_P$ and all $I \in I_P$.

It is easy to see that a locally finite consequence operator is local. Conversely, a local consequence operator for a program of finite type is locally finite. This follows from the observation that, for a program of finite type, the sets $\mathcal{B}_A$, for any $A \in B_P$, are finite. But a much stronger result holds.

Theorem 4.8. *A local consequence operator is locally finite if and only if it is continuous in $\mathcal{Q}$.*

Proof. Let T be a locally finite consequence operator, let $I \in I_P$, let $A \in B_P$, and let $G_2 = \mathcal{G}(A, T(I)(A))$ be a subbasic neighbourhood of $T(I)$ in $\mathcal{Q}$. Since T is locally finite, there is a finite set $S \subseteq \mathcal{B}_A$ such that $T(J)(A) = T(I)(A)$ for all $J \in \bigcap_{B \in S} \mathcal{G}(B, I(B))$. By finiteness of S, the set $\bigcap_{B \in S} \mathcal{G}(B, I(B))$ is open, and this suffices for continuity of T.

For the converse, assume that T is continuous in $\mathcal{Q}$ and let $A \in B_P$ and $I \in I_P$ be chosen arbitrarily. Then $G_2 = \mathcal{G}(A, T(I)(A))$ is a subbasic open set, so that, by continuity of T, there exists a basic open set $G_1 = \mathcal{G}(B_1, I(B_1)) \cap \cdots \cap \mathcal{G}(B_k, I(B_k))$ with $T(G_1) \subseteq G_2$. In other words, we have $T(J)(A) = T(I)(A)$ for each $J \in \bigcap_{B \in S'} \mathcal{G}(B, I(B))$, where $S' = \{B_1, \ldots, B_k\}$ is a finite set. Since T is local, the value of $T(J)(A)$ depends only on the values $J(A)$ of atoms $A \in \mathcal{B}_A$. So, if we set $S = S' \cap \mathcal{B}_A$, then $T(J)(A) = T(I)(A)$ for all $J \in \bigcap_{B \in S} \mathcal{G}(B, I(B))$ which is to say that T is locally finite for A and I. Since A and I were chosen arbitrarily, we obtain that T is locally finite. □

The following corollary was communicated to us by Howard A. Blair in the two-valued case.

Corollary 4.9. *Let P be a program, let T be a local consequence operator and let l be an injective ω-level mapping for P with the following property: for each $A \in B_P$ there exists an $n_A \in \mathbb{N}$ such that $l(B) < n_A$ for all $B \in \mathcal{B}_A$. Then T is continuous in $\mathcal{Q}$.*

Proof. It follows easily from the given conditions that $\mathcal{B}_A$ is finite for all $A \in B_P$, which implies that T is locally finite. $\square$

We next take a short detour from our discussion of continuity to study the weaker notion of measurability [4] for consequence operators. For a collection M of subsets of a set X, we denote by $\sigma(M)$ the smallest σ-algebra containing M, called the σ-algebra *generated by* M. Recall that a function $f : X \to X$ is measurable with respect to $\sigma(M)$ if and only if $f^{-1}(A) \in \sigma(M)$ for each $A \in M$. If β is the subbase of a topology τ and β is countable, then $\sigma(\beta) = \sigma(\tau)$. It turns out that local consequence operators are always measurable with respect to the σ-algebra generated by a generalized atomic topology.

Theorem 4.10. *Local consequence operators are measurable with respect to* $\sigma(\mathcal{G}) = \sigma(\mathcal{Q})$.

Proof. Let T be a local consequence operator. We need to show that, for each subbasic set $\mathcal{G}(A, t_i)$, we have $T^{-1}(\mathcal{G}(A, t_i)) \in \sigma(\mathcal{G})$.

Let $A \in B_P$ and let $t \in \mathcal{T}$ both be chosen arbitrarily. Let F be the set of all functions from $\mathcal{B}_A$ to $\mathcal{T}$, and note that F is countable since $\mathcal{B}_A$ is countable and $\mathcal{T}$ is finite. Let F' be the subset of F which contains all functions f with the following property: whenever an interpretation I agrees with f on $\mathcal{B}_A$, then $T(I)(A) = t$. Then, $\bigcap_{B \in \mathcal{B}_A} \mathcal{G}(B, f(B)) \in T^{-1}(\mathcal{G}(A, t))$ for each $f \in F'$.

We obtain by locality of T that, whenever I is an interpretation for which $T(I)(A) = t$, there exists a function $f_I \in F'$ such that f_I and I agree on $\mathcal{B}_A$, and this yields $T^{-1}(\mathcal{G}(A, t)) = \bigcup_{f_I \in F'} \bigcap_{B \in \mathcal{B}_A} \mathcal{G}(B, I(B))$. Since F' and $\mathcal{B}_A$ are countable, the set on the right hand side of this last equality is measurable, as required. $\square$

We turn now to the study of the continuity of a particular operator introduced by Fitting [19] to logic programming semantics. To this end, we associate a set P^* with each logic program P by the following construction. Let $A \in B_P$. If A occurs as the head of some unit clause $A \leftarrow$ in ground(P), then replace it by the clause $A \leftarrow t_n$, where by a slight abuse of notation we interpret t_n to be an additional atom which we adjoin to the language $\mathcal{L}$ and always evaluate to $t_n \in \mathcal{T}$, that is, it evaluates to “true”. If A does not occur in the head of any clause in ground(P), then add the clause $A \leftarrow t_0$, where t_0 is interpreted as an additional atom which again we adjoin to $\mathcal{L}$ and always evaluate to $t_0 \in \mathcal{T}$, that is, it evaluates to “false”. The resulting (ground) program, which results from ground(P) by the changes just given with respect to every $A \in B_P$, will be denoted by P'. Now let P^* be the set of all *pseudo clauses* determined by P', that is, the set of all formulae of the form $A \leftarrow C_1 \vee C_2 \vee \cdots$, where the C_i are exactly the bodies of the clauses in P' with head A. We call A the *head* and $B_A = C_1 \vee C_2 \vee \cdots$ the *body* of such a pseudo clause, and we note that each $A \in B_P$ occurs in the head of exactly one pseudo clause in P^*. Bodies of pseudo clauses are possibly infinite disjunctions, but this will not pose any particular difficulty with respect to the logics which we are going to discuss. We note that a program P is of finite type if and only if all bodies of all pseudo clauses in P^* are finite.

Now, if we are given (suitable) truth tables for negation, conjunction and disjunction, we are able to evaluate the truth values of bodies of pseudo clauses relative to given interpretations.

Definition 4.11. Let P be a logic program. Define the mapping $F_P : I_{P,n} \to I_{P,n}$ relative to a given (suitable) logic with n truth values by $F_P(I) = J$, where J assigns to each $A \in B_P$ the truth value $I(B_A)$.

We call operators which satisfy Definition 4.11 *Fitting operators*. If we impose the mild assumption that $t_j \leftarrow t_j$ evaluates to "true" for every j with respect to the underlying logic, then we easily obtain that every Fitting operator is a local consequence operator. This will always be the case in what follows in this paper.

The virtue of Definition 4.11, due to Fitting [19], lies in the fact that several operators known from the theory of logic programming can be derived from it in a very concise way, and we refer to [16,19] for a discussion of these matters, see also [32]. We will now investigate some of these operators in the light of Theorem 4.8. In the following, we will denote the "true" truth value by **t** and the "false" truth value by **f**.

If the chosen logic is classical two-valued logic, then the corresponding Fitting operator is the *single-step* or *immediate consequence operator* T_P (for a given program P). Now, if $T_P(I)(A) = \mathbf{t}$, then there exists a clause $A \leftarrow \mathtt{body}$ in ground(P) such that $I(\mathtt{body})$ is true, and we obtain $T_P(J)(A) = \mathbf{t}$ whenever $J(\mathtt{body}) = \mathbf{t}$. The observation that bodies of clauses are finite conjunctions leads us to conclude the following lemma.

Lemma 4.12. *If $T_P(I)(A)$ is true, then T_P is locally finite for A and I. Furthermore, T_P is continuous if and only if it is locally finite for all A and I with $T_P(I)(A) = \mathbf{f}$.*

A body $\bigvee C_i$ of a pseudo clause is false if and only if all C_i are false. Since T_P is a Fitting operator, we obtain $T_P(I)(A) = \mathbf{f}$ if and only if all C_i are false. If we require T_P to be locally finite for A and I, then there must be a finite set $S \subseteq \mathcal{B}_A$ such that any $J \in I_P$ which agrees with I on S renders all C_i false. These observations now easily yield the following theorem from [49].

Theorem 4.13. *Let P be a normal logic program. Then T_P is continuous if and only if, for each $I \in I_P$ and for each $A \in B_P$ with $T_P(I)(A) = \mathbf{f}$, either there is no clause in P with head A or there exists a finite set $S(I, A) = \{A_1, \dots, A_k, B_1, \dots, B_{k'}\} \subseteq \mathcal{B}_A$ with the following properties*:

(i) *$A_1, \dots, A_k$ are true in I and $B_1, \dots, B_{k'}$ are false in I.*
(ii) *Given any clause C with head A, at least one $\neg A_i$ or at least one B_j occurs in the body of C.*

In the case of Kleene's strong three-valued logic, with set of truth values $\mathcal{T} = \{t, u, f\}$ and logical connectives as in Table 1, the associated Fitting operator was introduced in [17] and is denoted by Φ_P, for a given program P. As in the case of classical two-valued logic, we obtain the following lemma.

Table 1
Connectives for Kleene's strong three-valued logic

p	q	$p \wedge q$	$p \vee q$	$\neg p$
t	t	t	t	f
t	u	u	t	f
t	f	f	t	f
u	t	u	t	u
u	u	u	u	u
u	f	f	u	u
f	t	f	t	t
f	u	f	u	t
f	f	f	f	t

Lemma 4.14. *If $\Phi_P(I)(A) = t$, then Φ_P is locally finite for A and I. Furthermore, Φ_P is continuous if and only if it is locally finite for all A and I with $\Phi_P(I)(A) \in \{u, f\}$.*

Obtaining a theorem analogous to Theorem 4.13 is now straightforward, but tedious, and we omit the details. Similar considerations apply to the operator Ψ on Belnap's four-valued logic [19] and to the operators from [29].

We mention in passing the non-monotonic Gelfond–Lifschitz operator [22] in classical two-valued logic, whose fixed points yield the stable models of the program in question. It turns out that this operator is not a consequence operator in the sense discussed in this paper, and attempts to characterize continuity of it will involve different methods (by means of the results from [53], for example).

4.3. Approximation by artificial neural networks

We have now finished our general preparations and continue next with our main task, namely, the study of the representability of logic programs by means of connectionist networks. We recall that the Cantor set $\mathcal{C}$ is a compact subset of the real line, and the topology which $\mathcal{C}$ inherits as a subspace of $\mathbb{R}$ coincides with the Cantor topology on $\mathcal{C}$. Also, the Cantor space $\mathcal{C}$ is homeomorphic to $I_{P,n}$ when the latter is endowed with a generalized atomic topology $\mathcal{Q}$. Hence, if a consequence operator T is continuous in $\mathcal{Q}$, we can identify it with a mapping $\iota(T): x \mapsto \iota(T(\iota^{-1}(x)))$ on $\mathcal{C}$ which is continuous in the subspace topology of $\mathcal{C}$ in $\mathbb{R}$, as follows.

Theorem 4.15. *Let P be a program, let T be a consequence operator which is locally finite and let ι be a homeomorphism from $(I_{P,n}, \mathcal{Q})$ to $\mathcal{C}$. Then T (more precisely $\iota(T)$) can be uniformly approximated by input-output mappings of 3-layer feedforward networks.*

Proof. Under the conditions stated in the theorem, the operator T is continuous in $\mathcal{Q}$. Using the homeomorphism ι, the resulting function $\iota(T)$ is continuous on the Cantor set $\mathcal{C}$, which is a compact subset of $\mathbb{R}$. Applying Theorem 2.4, $\iota(T)$ can be uniformly approximated by input-output functions of 3-layer feedforward networks. □

The restriction to programs with continuous consequence operator is not entirely satisfactory. There is another approximation theorem, due to [35], which requires only measurability of the functions in question.

Theorem 4.16. *Suppose that ϕ is a monotone increasing function from $\mathbb{R}$ onto $(0,1)$. Let $f:\mathbb{R}^r \to \mathbb{R}$ be a Borel-measurable function and let μ be a probability Borel-measure on $\mathbb{R}^r$. Then, given any $\varepsilon > 0$, there exists a 3-layer feedforward network with squashing function ϕ whose input-output function $\bar{f}:\mathbb{R}^r \to \mathbb{R}$ satisfies*

$$\varrho_\mu(f,\bar{f}) = \inf\bigl\{\delta > 0\colon\ \mu\bigl\{x\colon\ \bigl|f(x) - \bar{f}(x)\bigr| > \delta\bigr\} < \delta\bigr\} < \varepsilon.$$

In other words, the class of functions computed by 3-layer feedforward neural nets is dense in the set of all Borel measurable functions $f:\mathbb{R}^r \to \mathbb{R}$ relative to the metric ϱ_μ defined in Theorem 4.16.

By means of Theorem 4.10, we can now view a local consequence operator T as a measurable function $\iota(T)$ on $\mathcal{C}$ by identifying $I_{P,n}$ with $\mathcal{C}$ via a homeomorphism ι. Since $\mathcal{C}$ is measurable as a subset of the real line, this operator can be extended[6] to a measurable function on $\mathbb{R}$ and we obtain the following result.

Theorem 4.17. *Given any program P with local consequence operator T, the operator T (more precisely $\iota(T)$) can be approximated in the manner of Theorem 4.16 by input-output mappings of 3-layer feedforward networks.*

This result is somewhat unsatisfactory since the approximation stated in Theorem 4.16 is only *almost everywhere*, that is, pointwise with the exception of a set of measure zero. The Cantor set is, however, a set of measure zero. We can strengthen the result a bit by giving an explicit construction for the two-valued case. We define a sequence (T_n) of measurable functions on $\mathbb{R}$ as follows, where $l(x) = \max\{y \in \mathcal{C}\colon\ y \leqslant x\}$ and $u(x) = \min\{y \in \mathcal{C}\colon\ y \geqslant x\}$ for each $x \in [0,1] \setminus \mathcal{C}$:

$$T_0(x) = \begin{cases} \iota(T_P)(x) & \text{if } x \in \mathcal{C},\\ \iota(T_P)(0) & \text{if } x < 0,\\ \iota(T_P)(1) & \text{if } x > 1,\\ 0 & \text{otherwise,} \end{cases}$$

$$T_1(x) = \begin{cases} \iota(T_P)(l(x)) + \frac{\iota(T_P)(u(x)) - \iota(T_P)(l(x))}{u(x)-l(x)} & \text{if } x \in [3^{-1}, 2\cdot 3^{-1}],\\ 0 & \text{otherwise,} \end{cases}$$

$$T_i(x) = \begin{cases} \iota(T_P)(l(x)) \\ \quad + \frac{\iota(T_P)(u(x)) - \iota(T_P)(l(x))}{u(x)-l(x)}(x - l(x)) & \text{if } x \in \bigcup_{k=1}^{2\cdot 3^{i-2}} [(2k-1)3^{-i}, \\ & \qquad\qquad 2k\cdot 3^{-i}],\\ 0 & \text{otherwise} \end{cases}$$

for $i \geqslant 2$.

[6] For example, as a function $T:\mathbb{R} \to \mathbb{R}$ with $T(x) = \iota(T_P(\iota^{-1}(x)))$ if $x \in \mathcal{C}$ and $T(x) = 0$ otherwise.

We define the function $T : \mathbb{R} \to \mathbb{R}$ by $T(x) = \sup_i T_i(x)$ and obtain $T(x) = \iota(T_P(x))$ for all $x \in \mathcal{C}$ and $T(\iota(I)) = \iota(T_P(I))$ for all $I \in I_P$. Since all the functions T_i, for $i \geqslant 1$, are piecewise linear and therefore measurable, the function T is also measurable. Intuitively, T is obtained by a kind of linear interpolation.

If $i : B_P \to \mathbb{N}$ is a bijective mapping, then we can obtain a homeomorphism $\iota : I_P \to \mathcal{C}$ from i as follows: we identify $I \in I_P$ with $x \in \mathcal{C}$ where x written in ternary form has 2 as its $i(A)$th digit (after the decimal point) if $A \in I$, and 0 as its $i(A)$th digit if $A \notin I$. If $I \in I_P$ is finite or cofinite[7], then the sequence of digits of $\iota(I)$ in ternary form is eventually constant 0 (if I finite) or eventually constant 2 (if I cofinite). Thus, each such interpretation is the endpoint of a linear piece of one of the functions T_i, and therefore of T.

Corollary 4.18. *Given any normal logic program P, its single-step operator T_P (more precisely $\iota(T_P)$) can be approximated by input-output mappings of 3-layer feedforward networks in the following sense: for every $\varepsilon > 0$ and for every $I \in I_P$ which is either finite or cofinite, there exist a 3-layer feedforward network with input-output function f and $x \in [0, 1]$ with $|x - \iota(I)| < \varepsilon$ such that $|\iota(T_P(I)) - f(x)| < \varepsilon$.*

Proof. We use a homeomorphism ι which is obtained from a bijective mapping $i : B_P \to \mathbb{N}$ as in the paragraph preceding the corollary. We can assume that the measure μ from Theorem 4.16 has the property that $\mu\{[x, x + \varepsilon]\} \leqslant \varepsilon$ for each $x \in \mathbb{R}$. Let $\varepsilon > 0$ and $I \in I_P$ be finite or cofinite. Then by construction of T, there exists an interval $[\iota(I), \iota(I) + \delta]$ with $\delta < \frac{\varepsilon}{2}$ (or analogously $[\iota(I) - \delta, \iota(I)]$) such that T is linear on $[\iota(I), \iota(I) + \delta]$ and $|T(\iota(I)) - T(x)| < \frac{\varepsilon}{2}$ for all $x \in [\iota(I), \iota(I) + \delta]$. By Theorem 4.16 and the previous paragraph, there exists a 3-layer feedforward network with input-output function f such that $\varrho_\mu(T, f) < \delta$, that is, $\mu\{x: |T(x) - f(x)| > \delta\} < \delta$. By our condition on μ, there is $x \in [\iota(I), \iota(I) + \delta]$ with $|T(x) - f(x)| \leqslant \delta < \frac{\varepsilon}{2}$. We can conclude that

$$\left|\iota\big(T_P(I)\big) - f(x)\right| = \left|T\big(\iota(I)\big) - f(x)\right| \leqslant \left|T\big(\iota(I)\big) - T(x)\right| + \left|T(x) - f(x)\right| < \varepsilon,$$

as required. □

It would be of interest to strengthen this approximation for sets other than the finite and cofinite elements of I_P, although it is interesting to note that the finite interpretations correspond to compact elements in the sense of domain theory, see [2].

We want to return now to the case discussed earlier in Theorem 4.15. In Section 3, and also in [26], the following recurrent neural network architecture was considered: we assume that the number of output and input units is equal and that, after each propagation through the network, the output values are fed back without changes into input values. For the case which we consider, it will again be sufficient to suppose that the input layer consists of one unit only, so that the architecture can be depicted as in Fig. 6.

We will show in the following that iterates of locally finite local consequence operators can be approximated arbitrarily closely by iterates of suitably chosen networks. This is in fact a consequence of the uniform approximation obtained from Theorem 2.4 and the compactness of the unit interval.

[7] $I \in I_P$ is cofinite if $B_P \setminus I$ is finite.

Let P be a logic program, let T be a locally finite local consequence operator for P and let $\iota : I_P \to \mathcal{C}$ be a homeomorphism. Let F be a continuous extension of $\iota(T)$ onto the unit interval $[0, 1]$ in the reals, let d be the natural metric on $\mathbb{R}$, and let $\varepsilon > 0$. By Theorem 4.15, there exists a 3-layer feedforward network with input-output mapping f such that $\max_{x\in[0,1]} d(f(x), F(x)) < \varepsilon$. Let us further assume that F is Lipschitz-continuous, that is, there exists $\lambda \geqslant 0$ such that for all $x, y \in [0, 1]$ we have $d(F(x), F(y)) \leqslant \lambda d(x, y)$. For $x, y \in [0, 1]$ we therefore obtain

$$d\bigl(f(x), F(y)\bigr) \leqslant d\bigl(f(x), F(x)\bigr) + d\bigl(F(x), F(y)\bigr) \leqslant \varepsilon + \lambda d(x, y). \tag{5}$$

Now let $x \in [0, 1]$ be arbitrarily chosen. By Eq. (5) we obtain

$$d\bigl(f^2(x), F^2(x)\bigr) \leqslant \varepsilon + \lambda d\bigl(f(x), F(x)\bigr) \leqslant \varepsilon + \lambda\varepsilon. \tag{6}$$

Inductively, we can prove that for all $n \in \mathbb{N}$ we have

$$d\bigl(f^n(x), F^n(x)\bigr) \leqslant \varepsilon + \lambda\varepsilon + \cdots + \lambda^{n-1}\varepsilon = \varepsilon\left(\sum_{i=0}^{n-1} \lambda^i\right) = \varepsilon \frac{1-\lambda^n}{1-\lambda}. \tag{7}$$

Thus, we obtain the following bound on the error produced by the recurrent network after n iterations.

Theorem 4.19. *With the notation and hypotheses above, for any $I \in I_P$ and any $n \in \mathbb{N}$ we have*

$$\bigl|f^n\bigl(\iota(I)\bigr) - \iota\bigl(T^n(I)\bigr)\bigr| \leqslant \varepsilon \frac{1-\lambda^n}{1-\lambda}.$$

Proof. Note that $\iota(T^n(I)) = F^n(\iota(I))$, and the assertion follows from Eq. (7) since d is the natural metric on $\mathbb{R}$. □

We derive a few corollaries from this result.

Corollary 4.20. *If F is a contraction on $[0, 1]$, so that $\lambda < 1$, then $(F^k(\iota(I)))$ converges for every I to the unique fixed point x of F and there exists $m \in \mathbb{N}$ such that for all $n \geqslant m$ we have*

$$\bigl|f^n\bigl(\iota(I)\bigr) - x\bigr| \leqslant \varepsilon \frac{1}{1-\lambda}.$$

Proof. The convergence follows from the Banach contraction mapping theorem. The inequality follows immediately from Theorem 4.19 using the well-known expression for limits of geometric series. □

If F is a contraction on $[0, 1]$, then T is a contraction on the complete subspace $\mathcal{C}$, and also has a fixed point M with $\iota(M) = x$. However, it seems difficult to guarantee the hypothesis of Corollary 4.20, although in [26] a similar result for acyclic programs with injective level mappings in classical logic was achieved. The following result may be more promising.

Corollary 4.21. *If, for some $I \in I_P$, $T^n(I)$ converges in $\mathcal{Q}$ to a fixed point M of T, and $\iota(T)$ is Lipschitz-continuous, then, for every $\delta > 0$, there exists a network with input-output function f and some $n \in \mathbb{N}$ such that $|f^n(\iota(I)) - \iota(M)| < \delta$.*

Proof. The hypothesis implies that $F^n(\iota(I))$ converges to $\iota(M)$ in the natural metric on $\mathbb{R}$. Given $\delta > 0$, there exists $n \in \mathbb{N}$ such that $|F^m(\iota(I)) - \iota(M)| < \frac{\delta}{2}$ for all $m \geqslant n$. Since F is fixed, we know the value of λ. Now, by the approximation results above, we choose a network with input-output function f such that $\varepsilon\frac{1-\lambda^n}{1-\lambda} < \frac{\delta}{2}$. Then using Theorem 4.19 and the triangle inequality we obtain

$$\begin{aligned}\left|f^n(\iota(I)) - \iota(M)\right| &\leqslant \left|f^n\big(\iota(I)\big) - F^n\big(\iota(I)\big)\right| + \left|F^n\big(\iota(I)\big) - \iota(M)\right| \\ &< 2 \cdot \frac{\delta}{2} \leqslant \delta. \qquad \Box\end{aligned}$$

We close by describing a class of programs for which the additional hypothesis from Corollary 4.21 is satisfied. The result is well-known for the case of classical two-valued logic and the immediate consequence operator. So, let P be acyclic with level mapping l, and let T be a local consequence operator for P. We define a mapping $d : I_P \times I_P \to \mathbb{R}$ by $d(I, J) = 2^{-n}$, where n is least such that I and J differ on some atom A with $l(A) = n$. It is easily verified that d is a complete metric on I_P, see [18].

Proposition 4.22. *With the stated hypotheses, T is a contraction with respect to d.*

Proof. Suppose $d(I, J) = 2^{-n}$. Then I and J coincide on all atoms of level less than n. Now let $A \in B_P$ with $l(A) = n$. Then by acyclicity of P we have that all atoms in $\mathcal{B}_A$ are of level less than n, and by locality of T we have that $T(I)(A) = T(J)(A)$. So $d(T(I), T(J)) \leqslant 2^{-(n+1)}$. $\Box$

We obtain finally the following theorem.

Theorem 4.23. *Let P be an acyclic program and let T be a local consequence operator for P. Then, for any $I \in I_P$, we have that $T^n(I)$ converges in $\mathcal{Q}$ to the unique fixed point M of T.*

Proof. By Proposition 4.22 and the fact that d is a complete metric, we can apply the Banach contraction mapping theorem to obtain the convergence of $T^n(I)$ in d to a unique fixed point M of T. By definition of d, the convergence of the sequence of interpretations $T^n(I)$ to M must be pointwise, hence is also convergence in $\mathcal{Q}$. $\Box$

Theorem 4.23 is remarkable since the existence of a fixed point of the semantic operator can be guaranteed without any particular knowledge about the underlying multi-valued logic.

5. Conclusions and further work

In considering the integration of Logic and Connectionist Systems, we have taken the natural point of contact between them provided by the immediate consequence operator T_P, associated with a normal logic program P, and the issue of its computation by means of neural networks. In so far as one may identify two logic programs with the same immediate consequence operator (subsumption equivalence), this provides a sort of semantics for a neural network which computes T_P, namely, the supported model semantics of P.

A number of questions arise out of these considerations, and we close by briefly mentioning a few of them, as follows. First, there is the question of giving explicit constructions of networks for approximating T_P in case that T_P is continuous, and this point is considered in [6]. A question which is also related to the results given in [6] is that of providing good bounds on Lipschitz constants for f_P, and this issue appears to be central to actually giving constructions of approximating networks. Another natural question concerns carrying over the programme given here for the supported model semantics of a normal logic program to the stable model semantics [22] and the well-founded semantics [52], and one possible means of doing this is provided by the results of [53]. From the connectionist point of view, the main open question is how to build a connectionist network given a first-order logic program. Ideally, assuming that this is done, we would then like to apply known connectionist learning techniques, in particular backpropagation, to such networks and, after training, extract a refined set of first-order clauses from the network. Finally, there is the purely mathematical question of what mathematical notions of approximation are useful and appropriate. Here we have discussed two well-known ones: uniform approximation on compacta, and a notion of approximation closely related to convergence in measure. However, others may prove to be significant, and this is a problem still to be investigated.

Acknowledgements

The authors wish to thank two anonymous referees for comments which helped to improve the presentation of this paper. The last named author wishes to thank the following people and institutions for their support of work related to the results contained in this paper: (i) the members of Professor Steffen Hölldobler's Knowledge Representation and Reasoning Group at TU Dresden, (ii) Deutscher Akademischer Austausch Dienst (DAAD), and (iii) the Boole Centre for Research in Informatics at University College Cork.

References

[1] R. Andrews, J. Diederich, A.B. Tickle, A survey and critique of techniques for extracting rules from trained artificial neural networks, Knowledge-Based Systems 8 (6) (1995) 373–389.
[2] S. Abramsky, A. Jung, Domain theory, in: S. Abramsky, D. Gabbay, T.S.E. Maibaum (Eds.), Handbook of Logic in Computer Science, vol. 3, Clarendon, Oxford, 1994.
[3] K.R. Apt, D. Pedreschi, Reasoning about termination of pure Prolog programs, Inform. and Comput. 106 (1993) 109–157.
[4] R.G. Bartle, The Elements of Integration, Wiley, New York, 1966.
[5] M. Bezem, Characterizing termination of logic programs with level mappings, in: E.L. Lusk, R.A. Overbeek (Eds.), Proceedings of the North American Conference on Logic Programming, MIT Press, Cambridge, MA, 1989, pp. 69–80.

[6] S. Bader, P. Hitzler, Logic programs, iterated function systems, and recurrent radial basis function networks, in this issue.
[7] A. Batarekh, V.S. Subrahmanian, Topological model set deformations in logic programming, Fund. Inform. 12 (1989) 357–400.
[8] L. Cavedon, Acyclic programs and the completeness of SLDNF-resolution, Theoret. Comput. Sci. 86 (1991) 81–92.
[9] K.L. Clark, Negation as failure, in: H. Gallaire, J. Minker (Eds.), Logic and Data Bases, Plenum Press, New York, 1978, pp. 293–322.
[10] W.F. Dowling, J.H. Gallier, Linear-time algorithms for testing the satisfiability of propositional Horn formulae, J. Logic Programming 1 (3) (1984) 267–284.
[11] A.S. d'Avila Garcez, K. Broda, D.M. Gabbay, Symbolic knowledge extraction from trained neural networks: A sound approach, Artificial Intelligence 125 (2001) 155–207.
[12] A.S. d'Avila Garcez, K.B. Broda, D.M. Gabbay, Neural-Symbolic Learning Systems—Foundations and Applications, in: Perspectives in Neural Computing, Springer, Berlin, 2002.
[13] A.S. d'Avila Garcez, L.C. Lamb, D.M. Gabbay, A connectionist inductive learning system for modal logic programming, in: Proceedings of the IEEE International Conference on Neural Information Processing ICONIP'02, Singapore, 2002.
[14] A.S. d'Avila Garcez, G. Zaverucha, The connectionist inductive learning and logic programming system, Appl. Intelligence (Special Issue on Neural Networks and Structured Knowledge) 11 (1) (1999) 59–77.
[15] A.S. d'Avila Garcez, G. Zaverucha, L.A.V. de Carvalho, Logical inference and inductive learning in artificial neural networks, in: C. Hermann, F. Reine, A. Strohmaier (Eds.), Knowledge Representation in Neural Networks, Logos Verlag, Berlin, 1997, pp. 33–46.
[16] M. Denecker, V. Wiktor Marek, M. Truszczynski, Approximating operators, stable operators, well-founded fixpoints and applications in non-monotonic reasoning, in: J. Minker (Ed.), Logic-based Artificial Intelligence, Kluwer Academic, Boston, 2000, pp. 127–144.
[17] M. Fitting, A Kripke–Kleene semantics for general logic programs, J. Logic Programming 2 (1985) 295–312.
[18] M. Fitting, Metric methods: Three examples and a theorem, J. Logic Programming 21 (3) (1994) 113–127.
[19] M. Fitting, Fixpoint semantics for logic programming—A survey, Theoret. Comput. Sci. 278 (1–2) (2002) 25–51.
[20] J.A. Fodor, Z.W. Pylyshyn, Connectionism and cognitive architecture: A critical analysis, in: S. Pinker, J. Mehler (Eds.), Connections and Symbols, MIT Press, Cambridge, MA, 1988, pp. 3–71.
[21] K.-I. Funahashi, On the approximate realization of continuous mappings by neural networks, Neural Networks 2 (1989) 183–192.
[22] M. Gelfond, V. Lifschitz, The stable model semantics for logic programming, in: R.A. Kowalski, K.A. Bowen (Eds.), Logic Programming. Proceedings of the 5th International Conference and Symposium on Logic Programming, MIT Press, Cambridge, MA, 1988, pp. 1070–1080.
[23] P. Hitzler, Generalized metrics and topology in logic programming semantics, PhD Thesis, Department of Mathematics, National University of Ireland, University College Cork, 2001.
[24] S. Hölldobler, Y. Kalinke, Towards a massively parallel computational model for logic programming, in: Proceedings ECAI94 Workshop on Combining Symbolic and Connectionist Processing, ECCAI, 1994, pp. 68–77.
[25] J. Hertz, A. Krogh, R.G. Palmer, Introduction to the Theory of Neural Computation, Addison-Wesley, Reading, MA, 1991.
[26] S. Hölldobler, Y. Kalinke, H.-P. Störr, Approximating the semantics of logic programs by recurrent neural networks, Appl. Intelligence 11 (1999) 45–58.
[27] S. Hölldobler, Y. Kalinke, J. Wunderlich, A recursive neural network for reflexive reasoning, in: S. Wermter, R. Sun (Eds.), Hybrid Neural Symbolic Integration, in: Lecture Notes in Artificial Intelligence, vol. 1778, Springer, Berlin, 2000, pp. 46–62.
[28] S. Hölldobler, Automated inferencing and connectionist models, Technical Report AIDA-93-06, Intellektik, Informatik, TH Darmstadt, 1993 (Postdoctoral Thesis).
[29] P. Hitzler, A.K. Seda, Characterizations of classes of programs by three-valued operators, in: M. Gelfond, N. Leone, G. Pfeifer (Eds.), Logic Programming and Non-Monotonic Reasoning, Proceedings of the 5th

International Conference on Logic Programming and Non-Monotonic Reasoning, LPNMR'99, El Paso, TX, in: Lecture Notes in Artificial Intelligence, vol. 1730, Springer, Berlin, 1999, pp. 357–371.

[30] P. Hitzler, A.K. Seda, A note on relationships between logic programs and neural networks, in: P. Gibson, D. Sinclair (Eds.), Proceedings of the Fourth Irish Workshop on Formal Methods, IWFM'00, Electronic Workshops in Computing (eWiC), British Computer Society, 2000.

[31] P. Hitzler, A.K. Seda, A "converse" of the Banach contraction mapping theorem, J. Electrical Engrg. 52 (10/s) (2001) 3–6. Proceedings of the 3rd Slovakian Student Conference in Applied Mathematics, SCAM2001, Bratislava, Slovak Academy of Sciences.

[32] P. Hitzler, A.K. Seda, Semantic operators and fixed-point theory in logic programming, in: Proceedings of the Joint IIIS & IEEE Meeting of the 5th World Multiconference on Systemics, Cybernetics and Informatics, SCI2001 and the 7th International Conference on Information Systems Analysis and Synthesis, ISAS2001, Orlando, FL, International Institute of Informatics and Systemics: IIIS, 2001.

[33] P. Hitzler, A.K. Seda, Continuity of semantic operators in logic programming and their approximation by artificial neural networks, in: A. Günter, R. Krause, B. Neumann (Eds.), Proceedings of the 26th German Conference on Artificial Intelligence, KI2003, in: Lecture Notes in Artificial Intelligence, vol. 2821, Springer, Berlin, 2003, pp. 105–119.

[34] P. Hitzler, A.K. Seda, Generalized metrics and uniquely determined logic programs, Theoret. Comput. Sci. 305 (1–3) (2003) 187–219.

[35] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural Networks 2 (1989) 359–366.

[36] N.D. Jones, W.T. Laaser, Complete problems for deterministic sequential time, Theoret. Comput. Sci. 3 (1977) 105–117.

[37] R.M. Karp, V. Ramachandran, Parallel algorithms for shared-memory machines, in: J. van Leeuwen (Ed.), Handbook of Theoretical Computer Science, Elsevier Science, New York, 1990, pp. 869–941.

[38] J.W. Lloyd, Foundations of Logic Programming, Springer, Berlin, 1988.

[39] J. McCarthy, Epistomological challenges for connectionism, Behavioral and Brain Sciences 11 (1988) 44. (Commentary on [50]).

[40] Y. McIntyre, Modellgenerierung mit konnektionistischen Systemen, PhD Thesis, TU Dresden, Fakultät Informatik, 2000.

[41] M.L. Minsky, S. Papert, Perceptrons, MIT Press, Cambridge, MA, 1972.

[42] A. Newell, Physical symbol systems, Cognitive Sci. 4 (1980) 135–183.

[43] T.A. Plate, Holographic reduced representations, in: Proceedings of the International Joint Conference on Artificial Intelligence, 1991, pp. 30–35.

[44] J.B. Pollack, Recursive auto-associative memory: Devising compositional distributed representations, in: Proceedings of the 10th Annual Conference of the Cognitive Science Society, 1988, pp. 33–39.

[45] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning internal representations by error propagation, in: Parallel Distributed Processing, MIT Press, Cambridge, MA, 1986.

[46] F. Rosenblatt, Principles of Neurodynamics: Perceptrons and the Theory of Brain Machines, Spartan Books, Washington, 1962.

[47] L. Shastri, V. Ajjanagadde, From associations to systematic reasoning: A connectionist representation of rules, variables and dynamic bindings using temporal synchrony, Behavioral and Brain Sciences 16 (3) (1993) 417–494.

[48] M.G. Scutellà, A note on Dowling and Gallier's top-down algorithm for propositional Horn satisfiability, J. Logic Programming 8 (1990) 265–273.

[49] A.K. Seda, Topology and the semantics of logic programs, Fund. Inform. 24 (4) (1995) 359–386.

[50] P. Smolensky, On the proper treatment of connectionism, Behavioral and Brain Sciences 11 (1988) 1–74.

[51] G.G. Towell, J.W. Shavlik, Knowledge-based artificial neural networks, Artificial Intelligence 70 (1–2) (1994) 119–165.

[52] A. van Gelder, K.A. Ross, J.S. Schlipf, The well-founded semantics for general logic programs, J. ACM 38 (3) (1991) 620–650.

[53] M. Wendt, Unfolding the well-founded semantics, J. Electrical Engineering, Slovak Academy of Sciences 53 (12/s) (2002) 56–59, Proceedings of the 4th Slovakian Student Conference in Applied Mathematics, Bratislava, April 2002.

[54] S. Willard, General Topology, Addison-Wesley, Reading, MA, 1970.

Available online at www.sciencedirect.com
SCIENCE DIRECT®
Journal of Applied Logic 2 (2004) 273–300
JOURNAL OF APPLIED LOGIC
www.elsevier.com/locate/jal

Logic programs, iterated function systems, and recurrent radial basis function networks

Sebastian Bader, Pascal Hitzler [*]

Artificial Intelligence Institute, Department of Computer Science, Dresden University of Technology, Germany

Available online 23 April 2004

Abstract

Graphs of the single-step operator for first-order logic programs—displayed in the real plane—exhibit self-similar structures known from topological dynamics, i.e., they appear to be *fractals*, or more precisely, attractors of iterated function systems. We show that this observation can be made mathematically precise. In particular, we give conditions which ensure that those graphs coincide with attractors of suitably chosen iterated function systems, and conditions which allow the approximation of such graphs by iterated function systems or by fractal interpolation. Since iterated function systems can easily be encoded using recurrent radial basis function networks, we eventually obtain connectionist systems which approximate logic programs in the presence of function symbols.
© 2004 Elsevier B.V. All rights reserved.

Keywords: Neural-symbolic integration; Logic program; Iterated function system; Radial basis function network; Recurrent neural network

1. Introduction

Intelligent systems based on logic programming on the one hand, and on artificial neural networks (sometimes called connectionist systems) on the other, differ substantially. Logic programs are highly recursive and well understood from the perspective of knowledge representation: The underlying language is that of first-order logic, which is symbolic in nature and makes it easy to encode problem specifications directly as programs. The success of artificial neural networks lies in the fact that they can be trained using raw data, and in some problem domains the generalization from the raw data made during the learn-

[*] Corresponding author.
E-mail addresses: s.bader@gmx.net (S. Bader), phitzler@inf.tu-dresden.de (P. Hitzler).
URLs: http://www.wv.inf.tu-dresden.de/~borstel (S. Bader), http://www.wv.inf.tu-dresden.de/~pascal (P. Hitzler).

1570-8683/$ – see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.jal.2004.03.003

ing process turns out to be highly adequate for the problem at hand, even if the training data contains some noise. Successful architectures, however, often do not use recursive (or recurrent) structures. Furthermore, the knowledge encoded by a trained neural network is only very implicitly represented, and no satisfactory methods for extracting this knowledge in symbolic form are currently known.

It would be very desirable to combine the robust neural networking machinery with symbolic knowledge representation and reasoning paradigms like logic programming in such a way that the strengths of either paradigm will be retained. Current state-of-the-art research, however, fails by far to achieve this ultimate goal. As one of the main obstacles to be overcome we perceive the question how symbolic knowledge can be encoded by artificial neural networks: Satisfactory answers to this will naturally lead the way to knowledge extraction algorithms and to hybrid neural-symbolic systems.

Earlier attempts to integrate logic and connectionist systems have mainly been restricted to propositional logic, or to first-order logic without function symbols. They go back to the pioneering work by McCulloch and Pitts [34], and have led to a number of systems developed in the 80s and 90s, including Towell and Shavlik's KBANN [44], Shastri's SHRUTI [43], the work by Pinkas [36], Hölldobler [26], and d'Avila Garcez et al. [12,14], to mention a few, and we refer to [10,13,17] for comprehensive literature overviews.

Without the restriction to the finite case (including propositional logic and first-order logic without function symbols), the task becomes much harder due to the fact that the underlying language is infinite but shall be encoded using networks with a finite number of nodes. The sole approach known to us for overcoming this problem (apart from work on recursive auto-associative memory, RAAM, initiated by Pollack [37], which concerns the learning of recursive terms over a first-order language) is based on a proposal by Hölldobler et al. [29], spelled out first for the propositional case in [28], and reported also in [20]. It is based on the idea that logic programs can be represented—at least up to subsumption equivalence [32]—by their associated single-step or immediate consequence operators. Such an operator can then be mapped to a function on the real numbers, which can under certain conditions in turn be encoded or approximated, e.g., by feedforward networks with sigmoidal activation functions using an approximation theorem due to Funahashi [16].

While contemplating this approach, we plotted graphs of resulting real-valued functions and found that in *all* cases these plots showed self-similar structures as known from topological dynamics. To be more precise, they looked like *fractals* in the sense of attractors of iterated function systems [3], see Figs. 1 and 7 for examples. While the general observation that logic programming is linked to topological dynamics and chaos theory is not new (see the work by Blair et al. [8,9]), the strikingly self-similar representation in the Euclidean plane offers a setting for developing real-valued iterated function systems for representing logic programs, with the concrete goal of in turn converting these into recurrent neural networks, thus obtaining connectionist representations of logic programs.

In this paper we substantiate formally the fact that these plots can indeed be obtained as attractors of iterated function systems, and give concrete representations of such systems. More generally, we give necessary and sufficient conditions under which graphs of single-step operators in the Euclidean plane arise as attractors of certain iterated function systems. We will give algorithms for constructing iterated function systems and fractal interpolation systems for approximating graphs of single-step operators. We will finally use

```
n(0).
n(s(X)) :- n(X).
```

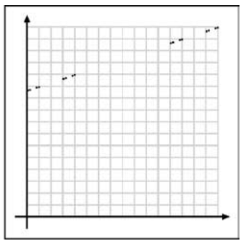


Fig. 1. The graph of a real-valued single-step operator.

our results for constructing recurrent radial basis function networks which approximate graphs of single-step operators.

The paper is structured as follows. In Section 2, we will introduce basic notions concerning logic programs and iterated function systems which we will need throughout the paper.

In Section 3, we show that graphs of logic programs can be obtained as attractors of iterated function systems. In particular, in Theorem 3.2 we will give necessary and sufficient conditions under which this is possible. Building on this, in Theorem 3.4 we will show that these conditions are satisfied whenever the embedded single-step operator is Lipschitz continuous with respect to the natural metric on the real numbers. The section closes with a concrete construction of an iterated function system and two detailed examples.

In Section 4 we shift our attention to the task of approximating logic programs—via their single-step operators—by means of fractal interpolation. More precisely, in Theorem 4.6 we show that programs with Lipschitz continuous single-step operator can be approximated uniformly by this method.

In Section 5 we will use our insights in order to show how logic programs can be represented or approximated by recurrent radial basis function networks.

The paper closes with a discussion of related and further work.

Most of the new results in this paper are discussed in more detail in [2].

2. Preliminaries

We will now shortly review and introduce terminology and notation from logic programming and iterated function systems, which we will use throughout. It will be helpful if the reader is familiar with these areas, but we will make an attempt to keep the paper self-contained in this respect, with terminology essentially following [31] respectively [3]. In some places we will have to assume basic knowledge of set-theoretic topology, our general reference being [45]. For Section 5, some familiarity with radial basis function networks (e.g., [7, Chapter 5]) will be helpful.

2.1. Logic programs

A (*normal*) *logic program* is a finite set of universally quantified *clauses* of the form

$$\forall(A \leftarrow L_1 \wedge \cdots \wedge L_n),$$

$\mathcal{P}$:	$B_{\mathcal{P}}$:	$\lvert\cdot\rvert$:

`n(0).`
 `n(s(X)) :- n(X).`
	$n(0), n(s(0)), n(s(s(0))),$
	 $n(s(s(s(0)))), \ldots$
		$\lvert n(s^x(0))\rvert = x + 1$

Fig. 2. A logic program, the corresponding Herbrand base, and a level mapping.

where $n \in \mathbb{N}$ may differ for each clause, A is an atom in a first-order language $\mathcal{L}$ and $L_1, \ldots, L_n$ are literals, that is, atoms or negated atoms, in $\mathcal{L}$. As is customary in logic programming, we will write such a clause in the form

$$A \leftarrow L_1, \ldots, L_n,$$

in which the universal quantifier is understood, or even as

$$A{:} - L_1, \ldots, L_n$$

following Prolog notation. Then A is called the *head* of the clause, each L_i is called a *body literal* of the clause and their conjunction $L_1, \ldots, L_n$ is called the *body* of the clause. We allow $n = 0$, by an abuse of notation, which indicates that the body is empty; in this case the clause is called a *unit clause* or a *fact*. The *Herbrand base* underlying a given program $\mathcal{P}$ is defined as the set of all ground instances of atoms over $\mathcal{L}$ and will be denoted by $B_{\mathcal{P}}$. Fig. 2 shows an example of a logic program and the corresponding Herbrand base. Subsets of the Herbrand base are called (*Herbrand*) *interpretations* of $\mathcal{P}$, and we can think of such a set as containing those atoms which are "true" under the interpretation. The set $I_{\mathcal{P}}$ of all interpretations of a program $\mathcal{P}$ can be identified with the power set of $B_{\mathcal{P}}$.

In this paper, we will not make use of any procedural aspects concerning logic programs. Indeed, logic programs are being used for many different purposes in computer science, e.g., as the language underlying Prolog [31], as languages for non-monotonic reasoning [30,33], for machine learning [35], etc., and the respective computational mechanisms differ substantially. Common to all these paradigms, however, is that logic programs are accepted as a convenient tool for knowledge representation in logical form. The knowledge represented by a logic program $\mathcal{P}$ can essentially be captured by the *immediate consequence* or *single-step operator* $T_{\mathcal{P}}$, which is defined as a mapping on $I_{\mathcal{P}}$ where for any $I \in I_{\mathcal{P}}$ we have that $T_{\mathcal{P}}(I)$ is the set of all $A \in B_{\mathcal{P}}$ for which there exists a ground instance $A \leftarrow A_1, \ldots, A_m, \neg B_1, \ldots, \neg B_n$ of a clause in $\mathcal{P}$ such that for all i we have $A_i \in I$ and for all j we have $B_j \notin I$.

A *level mapping* for a program $\mathcal{P}$ is a mapping $\lvert\cdot\rvert : B_{\mathcal{P}} \to \mathbb{N}$, and with a slight abuse of notation we set $\lvert\neg A\rvert = \lvert A\rvert$ for each $A \in B_{\mathcal{P}}$. Fig. 2 shows a simple logic program, the corresponding Herbrand base $B_{\mathcal{P}}$, and a possible level mapping. Level mappings can be used for describing dependencies between atoms in a program, and they have been studied in logic programming for many different purposes, e.g., for termination analysis under Prolog [1,6], or for giving uniform descriptions of different non-monotonic semantics [19,24,25]. For our investigations, we can restrict our attention to *injective* level mappings, which can simply be understood as enumerations of the Herbrand base. The latter perspective was employed, e.g., in [8]. It makes no essential difference, and we choose to stick with the more general notion of level mapping, and will explicitly require injectivity when needed.

Fitting [15] has used level mappings in order to define metrics on spaces of interpretations, an approach which was further extended in [18,23]. Recall that a *metric* over a set X is a mapping $d : X \times X \to \mathbb{R}$ satisfying (i) $d(x, y) = 0$ iff $x = y$, (ii) $d(x, y) = d(y, x)$, and (iii) $d(x, z) \leqslant d(x, y) + d(y, z)$ for all $x, y, z \in X$. The pair (X, d) is then called a *metric space*. A metric is called an *ultrametric* if it satisfies the stronger requirement (iii′) $d(x, z) \leqslant \max\{d(x, y), d(y, z)\}$ for all $x, y, z \in X$. On the real numbers, the function $d(x, y) = |x - y|$ is a metric and is called the *natural metric* on $\mathbb{R}$. A sequence $(x_n)_{n\in\mathbb{N}}$ in some metric space (X, d) *converges to* (or *has limit*) x, written $\lim x_n = x$, if for all $\varepsilon > 0$ there is some $n_0 \in \mathbb{N}$ such that $d(x_n, x) < \varepsilon$ for all $n \geqslant n_0$. A *Cauchy sequence* in a metric space (X, d) is a sequence (x_n) such that for each $\varepsilon > 0$ there exists $n_0 \in \mathbb{N}$ such that for all $m, n \geqslant n_0$ we have $d(x_m, x_n) < \varepsilon$. Converging sequences are always Cauchy sequences. A metric space in which every Cauchy sequence converges is called *complete*.

The following definition is a slight generalization of one given in [15].

Definition 2.1. Let $\mathcal{P}$ be a logic program, $2 \leqslant B \in \mathbb{N}$, and let $|\cdot|$ be a level mapping for $\mathcal{P}$. For $I, J \in I_{\mathcal{P}}$ define

$$d_B(I, J) = \begin{cases} 0 & \text{if } I = J, \\ B^{-n} & \text{if } I \text{ and } J \text{ differ on some atom } A \text{ with } |A| = n, \\ & \text{but agree on all atoms with a level smaller than } n. \end{cases}$$

It is easily verified that $(I_{\mathcal{P}}, d_B)$ is a complete metric space, indeed an ultrametric space. If $|\cdot|$ is injective—or more generally, if for each $n \in \mathbb{N}$ the set of all atoms with level n is finite—then the metric d_B, for any B, induces a topology on $I_{\mathcal{P}}$ which is known as the *query* [5] or *atomic* [40] topology Q. If furthermore the language underlying $\mathcal{P}$ contains at least one function symbol of arity at least 1, then $(I_{\mathcal{P}}, Q)$ is homeomorphic, i.e., topologically equivalent, to the Cantor space in the unit interval on the real line [40], which we will discuss further in Example 2.5.

A logic program $\mathcal{P}$ is *acyclic* [6,11] if there exists a level mapping $|\cdot|$ for $\mathcal{P}$ such that for each ground instance $A \leftarrow L_1, \dots, L_n$ of a clause in $\mathcal{P}$ we have that $|A| > |L_i|$ for all $i = 1, \dots, n$. In this case the operator $T_{\mathcal{P}}$ is a contraction on $(I_{\mathcal{P}}, d_B)$ with contractivity factor B^{-1}, i.e., it satisfies $d_B(T_{\mathcal{P}}(I), T_{\mathcal{P}}(J)) \leqslant B^{-1} d_B(I, J)$ for all $I, J \in I_{\mathcal{P}}$ [15,23].

2.2. *Iterated function systems*

Iterated function systems originate from the study of chaos theory and self-similar structures and they have found applications, e.g., in image compression. An excellent introduction to the field is [3], and we follow its notation, as already mentioned. We will later make use of the fact that real-valued iterated function systems can easily be encoded using recurrent neural networks, a point to which we will return in Section 5.

Recall that a function $f : X \to X$ on a metric space (X, d) is *continuous* if for all $\varepsilon > 0$ there exists $\delta > 0$ such that $d(f(x), f(y)) < \varepsilon$ whenever $d(x, y) < \delta$. A *Lipschitz continuous function* is a mapping $f : X \to X$ for which there exists a real number $\lambda \geqslant 0$, called a *Lipschitz constant* for f, such that $d(f(x), f(y)) \leqslant \lambda d(x, y)$ for all $x, y \in X$. *Contraction mappings* are exactly those Lipschitz continuous functions which have a Lipschitz constant (called *contractivity factor*) less than 1. Every contraction is Lipschitz continuous, and every Lipschitz continuous function is continuous. The importance of contractions lies in the fact that every contraction f on a complete metric space (X, d) has a unique fixed point x, which can be obtained as $\lim f^n(y)$, for all $y \in X$, where $f^n(y)$ denotes the nth iteration of the function f on the point y. This fact is well known as the *Banach contraction mapping theorem*.

Definition 2.2. A (hyperbolic) *iterated function system* (*IFS*) $((X, d), \Omega)$ is a pair consisting of a complete metric space (X, d) and a finite set $\Omega = \{\omega_1, \ldots, \omega_n\}$ of contraction mappings $\omega_i : X \to X$.

The idea behind iterated function systems is to lift the set Ω to be a contraction mapping on a space of certain subsets of X. More precisely, we consider *compact* subsets of X, which can be characterized as follows: $A \subseteq X$ is *compact* if for every (possibly infinite) collection of sets $B_{\varepsilon_i}(x_i) = \{y \mid d(x_i, y) < \varepsilon_i\}$ with $A \subseteq \bigcup_{i \in I} B_{\varepsilon_i}(x_i)$ there exists a finite selection $\{i_1, \ldots, i_n\} \subseteq I$ with $A \subseteq \bigcup_{k=1}^{n} B_{\varepsilon_{i_k}}(x_{i_k})$.

Given (X, d), we define $\mathcal{H}(X)$ to be the set of all non-empty compact subsets of X, and define the *Hausdorff distance* on $\mathcal{H}(X)$ as follows.

Definition 2.3. Let (X, d) be a complete metric space, $x \in X$ and $A, B \in \mathcal{H}(X)$. Then $d(x, B) = \min\{d(x, y) \mid y \in B\}$ is called the distance between the point x and the set B. The distance from A to B is then defined as $d(A, B) = \max\{d(a, B) \mid a \in A\}$. Finally, the *Hausdorff distance* h_d between A and B is defined as $h_d(A, B) = \max\{d(A, B), d(B, A)\}$.

The resulting *Hausdorff space* $(\mathcal{H}(X), h_d)$ is a complete metric space. A continuous mapping $f : X \to X$ can be extended to a function on $\mathcal{H}(X)$ in the usual way, i.e., by setting $f(A) = \{f(a) \mid a \in A\}$ (recalling that any continuous image of a compact set is compact). Given an IFS consisting of a metric space (X, d) and a set Ω of contractions, we identify Ω with a function on $\mathcal{H}(X)$ defined by $\Omega(A) = \bigcup_i w_i(A)$. The function Ω thus defined is a contractive mapping on $\mathcal{H}(X)$, and by the Banach contraction mapping theorem we can conclude that Ω has a unique fixed point $A \in \mathcal{H}(X)$, which hence obeys $A = \Omega(A)$ and can be obtained from any $B \in \mathcal{H}(X)$ as $A = \lim_{n\to\infty} \Omega^n(B)$, the limit being taken with respect to h_d. The fixed point $A \in \mathcal{H}(X)$ is called the *attractor* of the IFS $((X, d), \Omega)$.

Example 2.4. Fig. 3 depicts part of the iterative process leading to an attractor (starting from a square), in this case the so-called *Sierpinski triangle*. It is produced by an IFS consisting of the following three mappings on the space $(\mathbb{R}^2, d_2)$, where d_2 denotes the

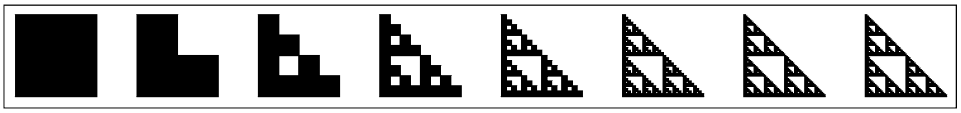


Fig. 3. The first iterations for the production of the Sierpinski triangle.

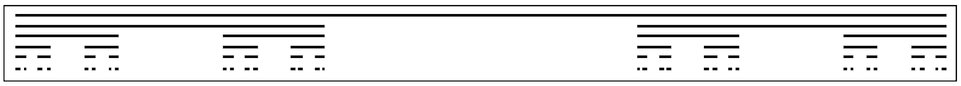


Fig. 4. The first iterations for the production of the Cantor set.

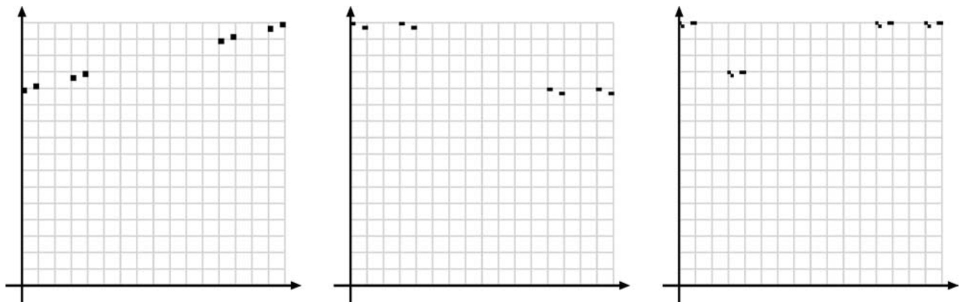


Fig. 5. Some attractors of iterated function systems.

Euclidean metric on $\mathbb{R}^2$ and $\omega = \{\omega_1, \omega_2, \omega_3\}$ with:

$$\omega_1\begin{pmatrix}x\\y\end{pmatrix} = \begin{pmatrix}0.5 & 0\\0 & 0.5\end{pmatrix}\begin{pmatrix}x\\y\end{pmatrix} + \begin{pmatrix}0\\0\end{pmatrix},$$

$$\omega_2\begin{pmatrix}x\\y\end{pmatrix} = \begin{pmatrix}0.5 & 0\\0 & 0.5\end{pmatrix}\begin{pmatrix}x\\y\end{pmatrix} + \begin{pmatrix}0.5\\0\end{pmatrix},$$

$$\omega_3\begin{pmatrix}x\\y\end{pmatrix} = \begin{pmatrix}0.5 & 0\\0 & 0.5\end{pmatrix}\begin{pmatrix}x\\y\end{pmatrix} + \begin{pmatrix}0\\0.5\end{pmatrix}.$$

Example 2.5. As a second example we give representations of Cantor space as compact subsets of the real numbers. The underlying space thus consists of the real numbers with the natural metric. As contractions, we choose

$$\omega_1 : \mathbb{R} \to \mathbb{R} : x \mapsto \frac{1}{B}x,$$

$$\omega_2 : \mathbb{R} \to \mathbb{R} : x \mapsto \frac{1}{B}x + a,$$

where $B > 2$ is a positive integer and a is chosen such that the images of the unit interval under ω_1 and ω_2 do not have more than a single point in common, but are both contained in the unit interval. The corresponding iterates of the unit interval are depicted in Fig. 4 for the values $B = 3$ and $a = \frac{2}{3}$. The subsets of the unit interval which can occur as attractors for different parameters are all homeomorphic, i.e., topologically equivalent, and also homeomorphic to the Cantor space and to $(I_{\mathcal{P}}, Q)$, if the Herbrand base $B_{\mathcal{P}}$ of the program $\mathcal{P}$ is countably infinite.

Some further examples of attractors of iterated function systems are depicted in Fig. 5, defined on the real plane. The projections of the attractors to the x-coordinate are homeomorphic to the Cantor space.

3. Logic programs as iterated function systems

In this section we show how logic programs can be represented by iterated function systems. We will review an embedding introduced by Hölldobler et al. in [29], which can be used to embed the graph of the single-step operator into the real plane. Plots of these graphs exhibit self-similar structures, i.e., they look like attractors of iterated function systems. We will provide a way to transform logic programs into iterated function systems such that the graph of the program coincides with the attractor of the IFS, or can at least be approximated by it.

Definition 3.1. Let $\mathcal{P}$ be a logic program, $|\cdot| : B_{\mathcal{P}} \to \mathbb{N}$ be an injective level mapping and let $B \in \mathbb{N}$, with $B > 2$. Then the mapping R assigns a unique real number $R(I)$ to every interpretation $I \in I_{\mathcal{P}}$ by

$$R : I_{\mathcal{P}} \to \mathbb{R} : I \mapsto \sum_{A \in I} B^{-|A|}.$$

The range $\{R(I) \mid I \in B_{\mathcal{P}}\} \subseteq \mathbb{R}$ of the mapping R will be denoted by D_{f} and the maximal value, which always exists, by $R_{\mathsf{m}} = R(B_{\mathcal{P}}) = \lim_{n\to\infty} \sum_{i=1}^{n} (B^{-i}) = \frac{1}{B-1}$. Without loss of generality we will treat R as a (bijective) function from $I_{\mathcal{P}}$ to D_{f}.

The probably most obvious base $B = 2$ does not create a valid embedding: Let $B = 2$, and $\mathcal{P}$ and $|\cdot|$ be defined as in Fig. 2. Let $I = \{n(0)\}$ and $J = B_{\mathcal{P}} \setminus \{n(0)\}$. It follows that $R(I) = B^{-1} = \frac{1}{2}$ and $R(J) = R_{\mathsf{m}} - B^{-1} = \frac{1}{B-1} - B^{-1} = \frac{1}{2}$, so the resulting function R is not injective. This is due to the fact that the numbers $0.111\ldots$ and $0.0111\ldots$ coincide in the number system with base 2. But for all $B > 2$ the mapping R is injective, if the level mapping is injective. Furthermore, it can be shown that R is a homeomorphism (a bijective mapping which preserves topological structure in both directions) from $(I_{\mathcal{P}}, Q)$ to D_{f} and that D_{f} is compact.

By means of the mapping R we can now embed $T_{\mathcal{P}}$ into $\mathbb{R}$ as shown in Fig. 6, i.e., for a given logic program $\mathcal{P}$ the function $R(T_{\mathcal{P}}) = f_{\mathcal{P}}$ is defined by

$$f_{\mathcal{P}} : D_{\mathsf{f}} \to D_{\mathsf{f}} : r \mapsto R\bigl(T_{\mathcal{P}}\bigl(R^{-1}(r)\bigr)\bigr),$$

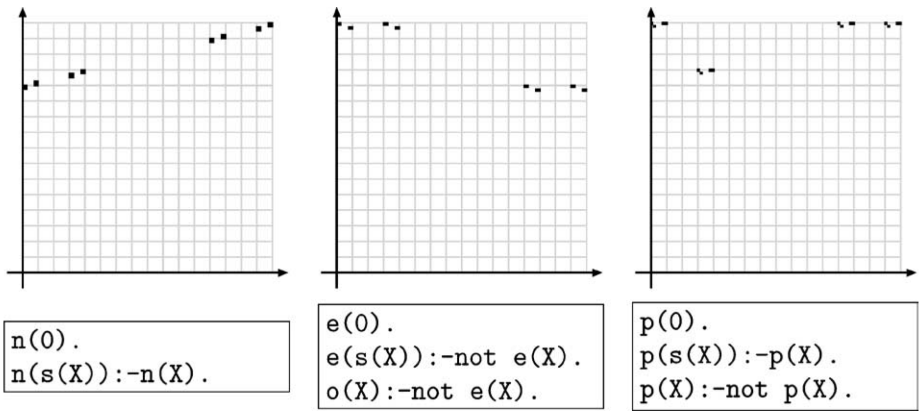
and its graph $F_{\mathcal{P}}$ is

$$F_{\mathcal{P}} = \bigl\{\bigl(R(I), R\bigl(T_{\mathcal{P}}(I)\bigr)\bigr) \mid I \in I_{\mathcal{P}}\bigr\} = \bigl\{\bigl(x, f_{\mathcal{P}}(x)\bigr) \mid x \in D_{\mathsf{f}}\bigr\}.$$

Fig. 7 shows some (embedded) graphs of logic programs. Note the similarity to the plots shown in Fig. 5. Indeed we have noticed that all plots of graphs obtained by the method described above showed self-similar structures, thus appearing to be attractors of iterated function systems on the real plane.

$$\begin{tikzcd}
I \in I_{\mathcal{P}} \arrow[r, "T_{\mathcal{P}}"] \arrow[d, shift right, "R"'] & I' \in I_{\mathcal{P}} \arrow[d, shift right, "R"'] \\
i \in D_{\mathsf{f}} \arrow[r, "f_{\mathcal{P}}"] \arrow[u, shift right, "R^{-1}"'] & i' \in D_{\mathsf{f}} \arrow[u, shift right, "R^{-1}"']
\end{tikzcd}$$

Fig. 6. The relation between $T_{\mathcal{P}}$ and $f_{\mathcal{P}}$.



```
n(0).
n(s(X)):-n(X).
```

```
e(0).
e(s(X)):-not e(X).
o(X):-not e(X).
```

```
p(0).
p(s(X)):-p(X).
p(X):-not p(X).
```

Fig. 7. Some graphs of logic programs.

3.1. Representation of logic programs by iterated function systems

We have just discussed that logic programs and iterated function systems create similar graphs. In this section we will link both by giving necessary and sufficient conditions under which the graph of a logic program is the attractor of a hyperbolic iterated function system. Since the iterated function systems shall approximate graphs in $\mathbb{R}^2$, they must be defined on that space. Therefore, we will focus on the space $(\mathbb{R}^2, d_2)$, where d_2 denotes the usual 2-dimensional Euclidean metric, i.e., $d_2((x_1, y_1), (x_2, y_2)) = \sqrt{|x_1 - x_2|^2 + |y_1 - y_2|^2}$, which is complete. For any function f on $\mathbb{R}^2$ we denote its coordinate functions by f^{x} and f^{y}, i.e., we have $f(a) = (f^{\mathsf{x}}(a), f^{\mathsf{y}}(a))$ for all $a \in \mathbb{R}^2$. Furthermore, let $\pi_{\mathsf{x}}(\cdot)$ denote the projection to the x-axis. The natural metric on $\mathbb{R}$ is denoted by d_1, i.e., $d_1(x, y) = |x - y|$ for all $x, y \in \mathbb{R}$.

The following theorem gives necessary and sufficient conditions for exact representability by an iterated function system.

Theorem 3.2 (First Representation Theorem). *Let $\mathcal{P}$ be a logic program, let $f_{\mathcal{P}}$ be the embedded $T_{\mathcal{P}}$-operator with graph $F_{\mathcal{P}}$, and let D_{f} be the range of the mapping R, as introduced earlier. Let $((\mathbb{R}^2, d_2), \Omega)$ be a (hyperbolic) iterated function system and let A be its uniquely determined attractor. Then the graph $F_{\mathcal{P}}$ coincides with the attractor A, i.e., $F_{\mathcal{P}} = A$, if and only if $\pi_{\mathsf{x}}(A) = D_{\mathsf{f}}$ and $f_{\mathcal{P}}(\omega_i^{\mathsf{x}}(a)) = \omega_i^{\mathsf{y}}(a)$ hold for all $a \in F_{\mathcal{P}}$ and all $\omega_i \in \Omega$.*

Proof. The proof is divided into two parts. First, we will show that $F_{\mathcal{P}} = A$ if $f_{\mathcal{P}}(\omega_i^{\mathsf{x}}(a)) = \omega_i^{\mathsf{y}}(a)$ and $\pi_{\mathsf{x}}(A) = D_{\mathsf{f}}$, and then the converse.

(i) To show the equivalence of $F_{\mathcal{P}}$ and A, we need to show that $F_{\mathcal{P}} \subseteq A$ and $A \subseteq F_{\mathcal{P}}$.

(i.a) From $\Omega(A) = A$ and $\pi_{\mathsf{x}}(A) = \pi_{\mathsf{x}}(F_{\mathcal{P}}) = D_{\mathsf{f}}$ we know that for each $a \in F_{\mathcal{P}}$ there must be an $a' \in A$ and an $\omega_i \in \Omega$ such that $\pi_{\mathsf{x}}(a) = \omega_i^{\mathsf{x}}(a')$. Using $f_{\mathcal{P}}(\omega_i^{\mathsf{x}}(a')) = \omega_i^{\mathsf{y}}(a')$ and the definition of $f_{\mathcal{P}}$ we know that $\omega_i^{\mathsf{y}}(a') = f_{\mathcal{P}}(\pi_{\mathsf{x}}(a)) = \pi_{\mathsf{y}}(a)$. So we can conclude that $(\pi_{\mathsf{x}}(a), \pi_{\mathsf{y}}(a)) = (\omega_i^{\mathsf{x}}(a'), \omega_i^{\mathsf{y}}(a'))$, i.e., $a = \omega_i(a')$. Since $a' \in A$, hence $\omega_i(a') \in A$, it follows that $a \in A$ and finally $F_{\mathcal{P}} \subseteq A$.

(i.b) From $f_\mathcal{P}(\omega_i^\mathsf{x}(a)) = \omega_i^\mathsf{y}(a)$ we can conclude that $(\omega_i^\mathsf{x}(a), \omega_i^\mathsf{y}(a)) \in F_\mathcal{P}$, i.e., $\omega_i(a) \in F_\mathcal{P}$. Knowing that this equation holds for all $a \in F_\mathcal{P}$ and all $\omega_i \in \Omega$ we obtain $\omega_i(F_\mathcal{P}) \subseteq F_\mathcal{P}$ and finally $\Omega(F_\mathcal{P}) \subseteq F_\mathcal{P}$. Hence $F_\mathcal{P} = A$.

(ii) Since $F_\mathcal{P} = A$ and $\pi_\mathsf{x}(F_\mathcal{P}) = D_\mathsf{f}$ we immediately obtain $\pi_\mathsf{x}(A) = D_\mathsf{f}$. Furthermore, we know that $F_\mathcal{P} = \Omega(F_\mathcal{P})$ and hence that $\Omega(a) \subseteq F_\mathcal{P}$ holds for all $a \in F_\mathcal{P}$. So we can conclude that for all $a \in F_\mathcal{P}$ and all $\omega_i \in \Omega$ there is an $a' \in F_\mathcal{P}$ such that $(\omega_i^\mathsf{x}(a), \omega_i^\mathsf{y}(a)) = a'$ holds. By the definition of $F_\mathcal{P}$ we know that $a' = (x', f_\mathcal{P}(x'))$, hence $\omega_i^\mathsf{x}(a) = x'$ and $\omega_i^\mathsf{y}(a) = f_\mathcal{P}(x')$. If follows that $f_\mathcal{P}(\omega_i^\mathsf{x}(a)) = \omega_i^\mathsf{y}(a)$ holds for all $a \in F_\mathcal{P}$ and all $\omega_i \in \Omega$ if $F_\mathcal{P} = A$. □

The proof of Theorem 3.2 does not make use of the fact that the function $f_\mathcal{P}$ (the graph of which is represented by an IFS) comes from the single-step operator of a logic program. Indeed it holds for all functions defined on D_f and is easily generalized to functions on other compact subsets of the reals. In particular, we note that it does not restrict the class of programs covered. We will use Theorem 3.2 for establishing a stronger result for logic programs whose embedded single-step operator $R(T_P)$ is Lipschitz continuous. Before we do so, however, we need to have a closer look at the set D_f. So assume that a base B is fixed, thus the mapping R is determined and in turn also D_f as the range of R. It is our desire to characterize D_f as the attractor of an IFS. Now define

$$\Omega_1^\mathsf{x} = \left\{x \mapsto \frac{1}{B}x + 0;\ x \mapsto \frac{1}{B}x + \frac{1}{B}\right\}.$$

For all $n > 1$ we define recursively

$$\begin{aligned}\Omega_n^\mathsf{x} &= \left\{f \circ g \mid f \in \Omega_1^\mathsf{x} \text{ and } g \in \Omega_{n-1}^\mathsf{x}\right\}\\ &= \left\{x \mapsto \frac{1}{B}\omega(x) + 0 \mid \omega \in \Omega_{n-1}^\mathsf{x}\right\} \cup \left\{x \mapsto \frac{1}{B}\omega(x) + \frac{1}{B} \mid \omega \in \Omega_{n-1}^\mathsf{x}\right\}.\end{aligned}$$

Note that every mapping $\omega_i^\mathsf{x} \in \Omega_P^\mathsf{x}$ is of the form $\omega_i^\mathsf{x} = \frac{1}{B^P}x + d_i^\mathsf{x}$, where d_i^x depends on the application of either the first or second mapping from Ω_1^x during the construction, i.e., d_i^x can be written as $\sum_{j=1}^{P} a_j \cdot B^{-j}$, where $a_j \in \{0, 1\}$. In particular we have that for each d_i^x there exists an interpretation I_i with $R(I_i) = d_i^\mathsf{x}$. More precisely, I_i consists of all those atoms A with $|A| \leqslant P$ such that $d_i^\mathsf{x} = \sum_{A \in I_i} B^{-|A|}$, and by injectivity of $|\cdot|$ the interpretation I_i is indeed uniquely determined by this equation.

The proof of the following lemma is straightforward.

Lemma 3.3. *For any $P \geqslant 1$ we have that D_f is the attractor of the IFS $((\mathbb{R}, d), \Omega_P^\mathsf{x})$.*

We are now ready to establish the promised second representation result. Even though it does not define a convenient way to construct an IFS, it explains why the plotted graphs of the programs are self-similar.

Theorem 3.4 (Second Representation Theorem). *Let $\mathcal{P}$ be a logic program. Let $f_\mathcal{P}$ be the embedded $T_\mathcal{P}$-operator using base $B > 2$, and let $F_\mathcal{P}$ be its graph. Furthermore assume that $f_\mathcal{P}$ is Lipschitz continuous. Then there exists an IFS on $(\mathbb{R}^2, d_2)$ with attractor $F_\mathcal{P}$.*

Proof. We prove this theorem by applying Theorem 3.2, i.e., we will show that under the stated hypotheses there is a hyperbolic IFS $((\mathbb{R}^2, d_2), \Omega)$ such that $\pi_{\mathsf{x}}(A) = D_{\mathsf{f}}$ and $f_{\mathcal{P}}(\omega_i^{\mathsf{x}}(a)) = \omega_i^{\mathsf{y}}(a)$ hold for all $a \in F_{\mathcal{P}}$ and all $\omega_i \in \Omega$.

By Lemma 3.3 we know that for each $P \geqslant 1$ there is a set Ω_P^{x} consisting of contractive mappings $\omega_i^{\mathsf{x}} : \mathbb{R} \to \mathbb{R}$ with contractivity factor $\frac{1}{B^P}$ and such that D_{f} is the attractor of the IFS $((\mathbb{R}, d_1), \Omega_P^{\mathsf{x}})$. For every $\omega_i^{\mathsf{x}} \in \Omega_P^{\mathsf{x}}$ we now define a mapping $\omega_i^{\mathsf{y}} : \mathbb{R} \to \mathbb{R}$ by $\omega_i^{\mathsf{y}}(x) = f_{\mathcal{P}}(\omega_i^{\mathsf{x}}(x))$. It remains to show that $((\mathbb{R}^2, d_2), \Omega)$ with $\Omega = \{(\omega_i^{\mathsf{x}} \circ \pi_{\mathsf{x}}, \omega_i^{\mathsf{y}} \circ \pi_{\mathsf{x}}) \mid \omega_i^{\mathsf{x}} \in \Omega_P^{\mathsf{x}}\}$ is a hyperbolic IFS for some suitably chosen $P \geqslant 1$, and for this it suffices to show that every $\omega_i = (\omega_i^{\mathsf{x}}, \omega_i^{\mathsf{y}}) \in \Omega$ is a contraction on $(\mathbb{R}^2, d_2)$.

Since $f_{\mathcal{P}}$ is Lipschitz continuous, there is a constant L with $d_1(f_{\mathcal{P}}(x), f_{\mathcal{P}}(y)) \leqslant L d_1(x, y)$ for all $x, y \in D_{\mathsf{f}}$. Taking this and the contractivity of ω_i^{x} into account we obtain for all $a, b \in \mathbb{R}^2$

$$\begin{aligned} d_2\big(\omega_i(a), \omega_i(b)\big)^2 &= d_1\big(\omega_i^{\mathsf{x}}\big(\pi_{\mathsf{x}}(a)\big), \omega_i^{\mathsf{x}}\big(\pi_{\mathsf{x}}(b)\big)\big)^2 + d_1\big(\omega_i^{\mathsf{y}}\big(\pi_{\mathsf{x}}(a)\big), \omega_i^{\mathsf{y}}\big(\pi_{\mathsf{x}}(b)\big)\big)^2 \\ &\leqslant B^{-2P} \big|\pi_{\mathsf{x}}(a) - \pi_{\mathsf{x}}(b)\big|^2 + L^2 B^{-2P} \big|\pi_{\mathsf{x}}(a) - \pi_{\mathsf{x}}(b)\big|^2 \\ &\leqslant \frac{L^2 + 1}{B^{2P}} \big|\pi_{\mathsf{x}}(a) - \pi_{\mathsf{x}}(b)\big|^2. \end{aligned}$$

Since π_{x} is continuous with Lipschitz constant 1 we obtain

$$d_2\big(\omega_i(a), \omega_i(b)\big) \leqslant \sqrt{\frac{L^2 + 1}{B^{2P}}}\, d_2(a, b).$$

We see now that it is possible to choose P such that ω_i is a contraction, and Theorem 3.2 is applicable. □

Before we move on, let us dwell a bit on the implications of Theorem 3.4 and also on some questions it raises. We require $f_{\mathcal{P}}$ to be Lipschitz continuous, which implies that $f_{\mathcal{P}}$ is continuous on the Cantor set as a subspace of $\mathbb{R}$, and hence that $T_{\mathcal{P}}$ is continuous with respect to the Cantor topology Q on $I_{\mathcal{P}}$. The latter notion is well-understood (see [20,40]). For example, it turns out that programs without local variables (called *covered programs* in [8]) have continuous single-step operators, where a local variable is a variable which occurs in some body literal of a program clause but not in its corresponding head.

The exact relationships between covered programs, continuity of the single-step operator in Q, Lipschitz continuity with respect to a metric generating Q, and Lipschitz continuity of the embedded single-step operator with respect to the natural metric on $\mathbb{R}$ remain to be studied, and these matters appear to be not straightforward. What we can say at this stage is that if $T_{\mathcal{P}}$ is continuous in Q then $f_{\mathcal{P}}$ is continuous on the Cantor space (because the latter is homeomorphic to $(I_{\mathcal{P}}, Q)$), and since the Cantor space is compact, we obtain that $f_{\mathcal{P}}$ must be uniformly continuous, which is stronger than continuity, but strictly weaker than Lipschitz continuity. The interested reader will also be able to verify that the single-step operator of the covered program

```
    p(X) :- p(f(X,X))
```

is not Lipschitz continuous with respect to any metric based on an injective level mapping as in Definition 2.1. We owe this example to Howard Blair.

Programs which are acyclic with respect to an injective level mapping also have continuous single-step operators, which is easily seen by observing that such programs cannot contain any local variables—or by considering the remark made earlier that for such programs the single-step operator is a contraction with respect to a metric which generates Q. Furthermore, it was shown in [29] that for base $B = 4$ (and hence for all larger bases) for the embedding R, the resulting embedded function $f_{\mathcal{P}} = R(T_{\mathcal{P}})$ is a contraction on a subset of $\mathbb{R}$, hence is Lipschitz continuous. If $\mathcal{P}$ is a program for which ground($\mathcal{P}$) is finite (and hence $B_{\mathcal{P}}$ is finite), then D_{f} is a finite subset of $\mathbb{R}$, and hence $f_{\mathcal{P}}$ is trivially Lipschitz continuous as a function on a subset of $\mathbb{R}$. We can thus state the following corollary.

Corollary 3.5. *For programs which are acyclic with respect to an injective level mapping, and for programs for which* ground($\mathcal{P}$) *is finite, there exists an IFS in the form given in the proof of Theorem* 3.4 *with attractor* $F_{\mathcal{P}}$.

Corollary 3.5 gives a formal, albeit not satisfactory, explanation for the observation which started our investigations: In order to obtain approximate plots of the graph of some $f_{\mathcal{P}}$, we restricted ourselves to plotting the graph corresponding to a *finite*, though large, subprogram of ground($\mathcal{P}$).

As yet, we know of no general method for obtaining Lipschitz constants of $f_{\mathcal{P}}$, or even for showing whether it is Lipschitz continuous at all. In the light of Theorem 3.4 and other results which we will discuss in the sequel, and also by considering our remarks made earlier on the unclear relations between different notions of continuity for single-step operators, we feel that investigations into these matter will have to be made in order to obtain satisfactory constructions of iterated function systems—or of connectionist systems—for representing logic programs in our approach.

3.2. Worked examples

Although Theorem 3.4 covers a wide range of programs, it is unsatisfactory since it does not provide a convenient way of constructing the iterated function system. Indeed, the IFS obtained in the proof of the theorem does involve the single-step operator for the calculation of the functions ω_i^{y}. In this section, we provide a simple but reasonable form of iterated function system which avoids this drawback, and show in detail that it covers some example programs. The same form of IFS will also be used in later parts of the paper.

Definition 3.6. Let the natural numbers $B > 2$ (hence also the mapping R) and $P \geqslant 1$ be fixed, and let $\mathcal{P}$ be a program. Then we associate with $\mathcal{P}$ the IFS $((\mathbb{R}^2, d_2), \Omega)$, where the $\omega_i \in \Omega$ are defined as $\omega_i : \mathbb{R}^2 \to \mathbb{R}^2 : (x, y) \mapsto (\omega_i^{\mathsf{x}}(x), \omega_i^{\mathsf{y}}(y))$ with ω_i^{x} and ω_i^{y} being

$$\omega_i^{\mathsf{x}} : \mathbb{R} \to \mathbb{R} : x \mapsto \frac{1}{B^P} x + d_i^{\mathsf{x}}, \quad \text{and}$$

$$\omega_i^{\mathsf{y}} : \mathbb{R} \to \mathbb{R} : y \mapsto \frac{1}{B^P} y + f_{\mathcal{P}}(d_i^{\mathsf{x}}) - \frac{f_{\mathcal{P}}(0)}{B^P}.$$

Algorithm 3.7 (Construction of $\mathrm{IFS}^l_{\mathcal{P}}$ for a given program $\mathcal{P}$). Let $\mathcal{P}$ be a logic program and $f_{\mathcal{P}}$ be its embedded $T_{\mathcal{P}}$-operator.

1. Choose a natural number (the *periodicity*) $P > 0$.
2. Compute Ω^{x}_P as explained in Section 3.1.
3. Construct for each $\omega^{\mathsf{x}}_i \in \Omega^{\mathsf{x}}_P$ the corresponding $\omega^{\mathsf{y}}_i : y \mapsto \frac{1}{B^P} y + f_{\mathcal{P}}(d^{\mathsf{x}}_i) - \frac{f_{\mathcal{P}}(0)}{B^P}$.
4. Return the set $\Omega = \{\omega_i = (\omega^{\mathsf{x}}_i, \omega^{\mathsf{y}}_i)\}$ as mappings for the $\mathrm{IFS}^l_{\mathcal{P}} = \{(\mathbb{R}^2, d_2), \Omega\}$.

Fig. 8. Constructing linear iterated function systems.

The parameter i, in this case, ranges from 1 to 2^P, and the ω^{x}_i and $d^{\mathsf{x}}_i \in D_{\mathsf{f}}$ are exactly as in the IFS $((\mathbb{R}, d), \Omega^{\mathsf{x}}_P)$ from Lemma 3.3. For convenience, we call such a resulting IFS *linear* and use the notation $\mathrm{IFS}^l_{\mathcal{P}}$ when we are referring to it. Note that whenever B, P, and $\mathcal{P}$ are fixed, then the corresponding $\mathrm{IFS}^l_{\mathcal{P}}$ is uniquely determined, so that our notation is sound.

We consider the base B fixed in the sequel. The parameter P, which we call *periodicity*, will usually depend on the program $\mathcal{P}$. How to construct an $\mathrm{IFS}^l_{\mathcal{P}}$ from a fixed B is also depicted in Fig. 8.

Before we explain the intuition behind Definition 3.6 we need to introduce a new operator denoted $\overset{\cdot}{\underset{\cdot}{\rightarrow}}$, which takes as arguments an interpretation and a natural number, and returns an interpretation. This operator defines a kind of shift operation on interpretations.

Definition 3.8. Let $\mathcal{P}$ be a logic program, $I \in I_{\mathcal{P}}$, $P \in \mathbb{N}$, and $|\cdot|$ be an injective level mapping for $\mathcal{P}$. Then define

$$\overset{I}{\underset{P}{\rightarrow}} = \bigl\{A \mid \text{there exists } A' \in I \text{ with } |A'| + P = |A|\bigr\}.$$

We call $\overset{\cdot}{\underset{\cdot}{\rightarrow}}$ the *right-shift operator*.

Proposition 3.9. $\overset{I}{\underset{P}{\rightarrow}} = R^{-1}(\frac{R(I)}{B^P})$ *holds for all $I \in I_{\mathcal{P}}$ and all $P \in \mathbb{N}$.*

Proof. The equation follows immediately from the definition of R since

$$\overset{I}{\underset{P}{\rightarrow}} = R^{-1}\Biggl(\sum_{A \in I} B^{-(|A|+P)}\Biggr) = R^{-1}\Biggl(\frac{\sum_{A \in I} B^{-|A|}}{B^P}\Biggr) = R^{-1}\Biggl(\frac{R(I)}{B^P}\Biggr). \qquad \square$$

We have already observed in Section 3.1 that for each d^{x}_i occurring in Definition 3.6 there exists some $I_i \in I_{\mathcal{P}}$ with $R(I_i) = d^{\mathsf{x}}_i$. Using Proposition 3.9 we can therefore carry over the functions ω^{x}_i to $I_{\mathcal{P}}$, as follows:

$$\omega^{\mathsf{x}}_i : \mathbb{R} \to \mathbb{R} : x \mapsto \frac{x}{B^P} + d^{\mathsf{x}}_i,$$

$$R^{-1}(\omega_i^{\mathsf{x}}) = \mathsf{w}_i^{\mathsf{x}} : I_{\mathcal{P}} \to I_{\mathcal{P}} : I \mapsto \overset{I}{\underset{P}{\rightarrow}} \cup I_i.$$

For the mappings ω_i^{y} the resulting function is a bit more involved, and can be represented as

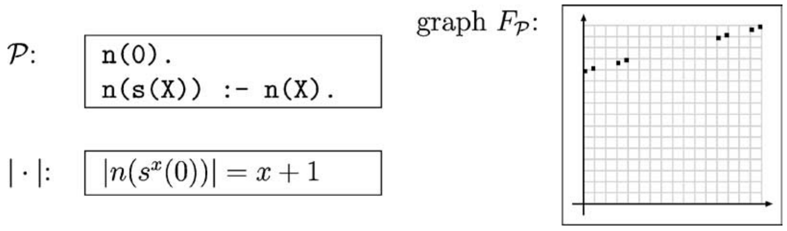
$$\omega_i^{\mathsf{y}} : \mathbb{R} \to \mathbb{R} : y \mapsto \frac{y}{B^P} - \frac{f_{\mathcal{P}}(0)}{B^P} + f_{\mathcal{P}}(d_i^{\mathsf{x}}),$$
$$R^{-1}(\omega_i^{\mathsf{y}}) = \mathsf{w}_i^{\mathsf{y}} : I_{\mathcal{P}} \to I_{\mathcal{P}} : I \mapsto \left(\overset{I}{\underset{P}{\rightarrow}} \setminus I_i^-\right) \cup I_i^+,$$

where $I_i^+ = R^{-1}(f_{\mathcal{P}}(d_i^{\mathsf{x}})) = T_{\mathcal{P}}(I_i)$ and $I_i^- = R^{-1}(\frac{f_{\mathcal{P}}(0)}{B^P}) = \overset{T_{\mathcal{P}}(\emptyset)}{\underset{P}{\rightarrow}}$.

Let us now explain the intuition behind the definitions of the mappings ω_i^{x} and ω_i^{y}. The choice of the ω_i^{x} is obvious for the same reasons as in Section 3.1 and in the proof of Theorem 3.4: It appears to be the most natural way to obtain D_{f} as projection of the resulting attractor to the x-axis, as required by Theorem 3.2. The corresponding mapping $\mathsf{w}_i^{\mathsf{x}}$ unmasks this as a right-shift with addition of a base point I_i. A first approximate candidate for $\mathsf{w}_i^{\mathsf{y}}(I)$ would therefore be $\overset{I}{\underset{P}{\rightarrow}} \cup T_{\mathcal{P}}(I_i)$—note that I in this case should be understood as being some image under $T_{\mathcal{P}}$. The occurrence of I_i^- is necessary as a correction in case of an overlap (i.e., a non-empty intersection) between $\overset{I}{\underset{P}{\rightarrow}}$ and $T_{\mathcal{P}}(I_i)$. This would not be necessary, strictly speaking, for $\mathsf{w}_i^{\mathsf{y}}$, where such an overlap would have no effect since it is ignored by the set-union operation. When carried over to the reals, however, this correction becomes necessary in order to avoid the situation that the resulting number would not correspond to an interpretation.

Linear iterated function systems are constructed such that $\pi_{\mathsf{x}}(A) = D_{\mathsf{f}}$, which is one of the conditions imposed by Theorem 3.2. The other condition, $f_{\mathcal{P}}(\omega_i^{\mathsf{x}}(a)) = \omega_i^{\mathsf{y}}(a)$ for all a, will be shown on a case base in the following examples. We fix $B = 4$ for the examples, in order to have a concrete setting. This choice was also made in [29], and the reason for this was to guarantee that $f_{\mathcal{P}}$ is a contraction for acyclic programs with injective level mappings, as already mentioned.

Consider first the program from Fig. 1. Fig. 9 shows an associated graph with corresponding level mapping—we use the notation $s^x(0)$ to denote the term $s(\ldots(0)\ldots)$ in which the symbol s occurs x times. We now use Algorithm 3.7 for constructing an $\mathrm{IFS}_{\mathcal{P}}^l$



$\mathcal{P}$:
```
n(0).
n(s(X)) :- n(X).
```

$|\cdot|$: $|n(s^x(0))| = x + 1$

Fig. 9. The natural numbers program and their embedded $T_{\mathcal{P}}$-operator.

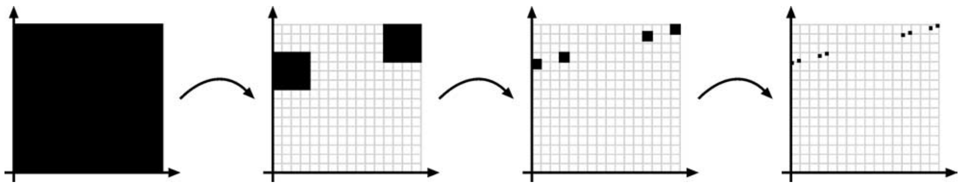


Fig. 10. The first three iterations of the mappings.

Table 1
$T_{\mathcal{P}}(\mathsf{w}_i^{\mathsf{x}}(I)) = \mathsf{w}_i^{\mathsf{y}}(T_{\mathcal{P}}(I))$ holds for the natural numbers program

$\omega_1^{\mathsf{x}}(x) = \frac{x}{4} + 0$	$\omega_1^{\mathsf{y}}(x) = \frac{x}{4} + \frac{3}{16}$
$\mathsf{w}_1^{\mathsf{x}}(I) = \xrightarrow[1]{I}$	$\mathsf{w}_1^{\mathsf{y}}(I) = \big(\xrightarrow[1]{I} \setminus \{n(s(0))\}\big) \cup \{n(0)\}$
$T_{\mathcal{P}}(\mathsf{w}_1^{x}(I)) = T_{\mathcal{P}}\big(\xrightarrow[1]{I}\big) = \big(\xrightarrow[1]{T_{\mathcal{P}}(I)} \setminus \{n(s(0))\}\big) \cup \{n(0)\} = \mathsf{w}_1^{\mathsf{y}}(T_{\mathcal{P}}(I))$	
$\omega_2^{\mathsf{x}}(x) = \frac{x}{4} + \frac{1}{4}$	$\omega_2^{\mathsf{y}}(x) = \frac{x}{4} + \frac{1}{4}$
$\mathsf{w}_2^{\mathsf{x}}(I) = \xrightarrow[1]{I} \cup \{n(0)\}$	$\mathsf{w}_2^{\mathsf{y}}(I) = \xrightarrow[1]{I} \cup \{n(0)\}$
$T_{\mathcal{P}}(\mathsf{w}_2^{x}(I)) = T_{\mathcal{P}}\big(\xrightarrow[1]{I} \cup \{n(0)\}\big) = \xrightarrow[1]{T_{\mathcal{P}}(I)} \cup \{n(0)\} = \mathsf{w}_2^{\mathsf{y}}(T_{\mathcal{P}}(I))$	

for the program. We choose periodicity $P = 1$ and obtain

$$\Omega = \left\{ \begin{pmatrix} \frac{1}{4} & 0 \\ 0 & \frac{1}{4} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0 \\ \frac{3}{16} \end{pmatrix}, \begin{pmatrix} \frac{1}{4} & 0 \\ 0 & \frac{1}{4} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \frac{1}{4} \\ \frac{1}{4} \end{pmatrix} \right\}.$$

The first three iterations of this $\mathrm{IFS}_{\mathcal{P}}^{l}$ are depicted in Fig. 10.

In order to show that the resulting attractor coincides with $F_{\mathcal{P}}$, we need to verify the hypotheses of Theorem 3.2, i.e., in particular, we need to show that $f_{\mathcal{P}}(\omega_i^{\mathsf{x}}(a)) = \omega_i^{\mathsf{y}}(a)$ for all $a \in F_{\mathcal{P}}$. By the discussion following Proposition 3.9 it therefore suffices to show that $T_{\mathcal{P}}(\mathsf{w}_i^{\mathsf{x}}(I)) = \mathsf{w}_i^{\mathsf{y}}(T_{\mathcal{P}}(I))$ holds for all $I \in I_{\mathcal{P}}$. The necessary calculations are shown in Table 1, some details are straightforward and have been omitted.

As another example we discuss the program from Fig. 11. We work with periodicity $P = 2$ and obtain the following $\mathrm{IFS}_{\mathcal{P}}^{l}$ by Algorithm 3.7:

$$\Omega = \left\{ \begin{pmatrix} \frac{1}{16} & 0 \\ 0 & \frac{1}{16} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0 \\ \frac{5}{16} \end{pmatrix}, \begin{pmatrix} \frac{1}{16} & 0 \\ 0 & \frac{1}{16} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \frac{1}{16} \\ \frac{5}{16} \end{pmatrix}, \right.$$
$$\left. \begin{pmatrix} \frac{1}{16} & 0 \\ 0 & \frac{1}{16} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \frac{1}{4} \\ \frac{15}{64} \end{pmatrix}, \begin{pmatrix} \frac{1}{16} & 0 \\ 0 & \frac{1}{16} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \frac{5}{16} \\ \frac{15}{64} \end{pmatrix} \right\}.$$

The first few iterations of the resulting $\mathrm{IFS}_{\mathcal{P}}^{l}$ are depicted in Fig. 12. Verification of correctness is performed similarly as for the natural numbers program and details are given in Table 2.

$\mathcal{P}$:

```
e(0).
e(s(X)) :- not e(X).
o(X) :- not e(X).
```

$|\cdot|$:

$$|e(s^x(0))| = 2\cdot x + 1$$
$$|o(s^x(0))| = 2\cdot x + 2$$

graph $F_{\mathcal{P}}$:

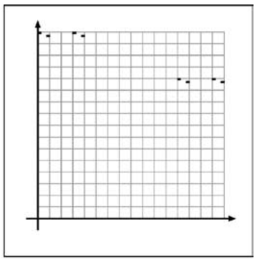


Fig. 11. The even and odd numbers program.

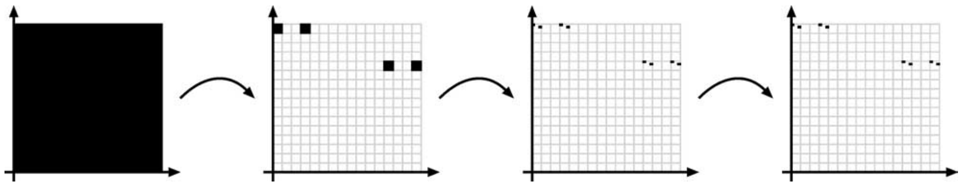


Fig. 12. The first three iterations of the mappings.

Table 2
$T_{\mathcal{P}}(\mathsf{w}_i^{\mathsf{x}}(I)) = \mathsf{w}_i^{\mathsf{y}}(T_{\mathcal{P}}(I))$ holds for the even and odd numbers program

$\omega_1^{\mathsf{x}}(x) = \frac{x}{16} + 0$	$\omega_1^{\mathsf{y}}(x) = \frac{x}{16} + \frac{5}{16}$
$\mathsf{w}_1^{\mathsf{x}}(I) = \xrightarrow[2]{I}$	$\mathsf{w}_1^{\mathsf{y}}(I) = \xrightarrow[2]{I} \cup \{e(0), o(0)\}$
$T_{\mathcal{P}}(\mathsf{w}_1^{x}(I)) = T_{\mathcal{P}}\big(\xrightarrow[2]{I}\ \big) = \xrightarrow[2]{T_{\mathcal{P}}(I)} \cup \{e(0), o(0)\} = \mathsf{w}_1^{\mathsf{y}}(T_{\mathcal{P}}(I))$	
$\omega_2^{\mathsf{x}}(x) = \frac{x}{16} + \frac{1}{16}$	$\omega_2^{\mathsf{y}}(x) = \frac{x}{16} + \frac{5}{16}$
$\mathsf{w}_2^{\mathsf{x}}(I) = \xrightarrow[2]{I} \cup \{o(0)\}$	$\mathsf{w}_2^{\mathsf{y}}(I) = \xrightarrow[2]{I} \cup \{e(0), o(0)\}$
$T_{\mathcal{P}}(\mathsf{w}_2^{x}(I)) = T_{\mathcal{P}}\big(\xrightarrow[2]{I} \cup \{o(0)\}\big) = \xrightarrow[2]{T_{\mathcal{P}}(I)} \cup \{e(0), o(0)\} = \mathsf{w}_2^{\mathsf{y}}(T_{\mathcal{P}}(I))$	
$\omega_3^{\mathsf{x}}(x) = \frac{x}{16} + \frac{1}{4}$	$\omega_3^{\mathsf{y}}(x) = \frac{x}{16} + \frac{15}{64}$
$\mathsf{w}_3^{\mathsf{x}}(I) = \xrightarrow[2]{I} \cup \{e(0)\}$	$\mathsf{w}_3^{\mathsf{y}}(I) = \big(\xrightarrow[2]{I} \setminus \{e(s(0))\}\big) \cup \{e(0)\}$
$T_{\mathcal{P}}(\mathsf{w}_3^{x}(I)) = T_{\mathcal{P}}\big(\xrightarrow[2]{I} \cup \{e(0)\}\big) = \big(\xrightarrow[2]{T_{\mathcal{P}}(I)} \setminus \{e(s(0))\}\big) \cup \{e(0)\} = \mathsf{w}_3^{\mathsf{y}}(T_{\mathcal{P}}(I))$	
$\omega_4^{\mathsf{x}}(x) = \frac{x}{16} + \frac{5}{16}$	$\omega_4^{\mathsf{y}}(x) = \frac{x}{16} + \frac{15}{64}$
$\mathsf{w}_4^{\mathsf{x}}(I) = \xrightarrow[2]{I} \cup \{e(0), o(0)\}$	$\mathsf{w}_4^{\mathsf{y}}(I) = \big(\xrightarrow[2]{I} \setminus \{e(s(0))\}\big) \cup \{e(0)\}$
$T_{\mathcal{P}}(\mathsf{w}_4^{x}(I)) = T_{\mathcal{P}}\big(\xrightarrow[2]{I} \cup \{e(0), o(0)\}\big) = \big(\xrightarrow[2]{T_{\mathcal{P}}(I)} \setminus \{e(s(0))\}\big) \cup \{e(0)\} = \mathsf{w}_4^{\mathsf{y}}(T_{\mathcal{P}}(I))$	

4. Logic programs by fractal interpolation

In Section 3 we have focused on the problem of exact representation of logic programs by iterated function systems. In this section we will provide a result for approximating logic programs by iterated function systems. Our approach is motivated by fractal interpolation

Algorithm 4.1 (Interpolation Data). This algorithm computes a set of interpolation data for a given program $\mathcal{P}$.

1. Choose a natural number (the *accuracy*) $P > 0$.
2. Compute the set $D = \{A \mid |A| \leqslant P \text{ and } A \in B_{\mathcal{P}}\}$ and its powerset $\mathcal{D} = \mathcal{P}(D)$.
3. For any set $X_i \in \mathcal{D}$ compute $Y_i = T_{\mathcal{P}}(X_i)$ with respect to the program $\mathcal{P}$.
4. Return the sequence of pairs $(R(X_i), R(Y_i))$, with $R(X_i) < R(X_j)$ for all $i < j$.

Fig. 13. Construction of interpolation data.

as described in [3, Chapter 6], but our setting differs in that we reuse the linear iterated function systems introduced in Definition 3.6.

We will again assume the parameter $B > 2$ and some injective level mapping to be fixed. The parameter P is going to be reinterpreted as *accuracy*. Given a logic program $\mathcal{P}$, for which $f_{\mathcal{P}}$ is Lipschitz continuous, and given some accuracy P, we consider the associated iterated function system as given by Definition 3.6. It will be shown that the attractor of each of these systems is the graph of a continuous function defined on D_{f}, and that the sequence of attractors associated with an increasing sequence of accuracies converges to the graph of $f_{\mathcal{P}}$, with respect to the maximum metric on the space of continuous functions.

We begin by describing in detail the fractal interpolation systems which we will be using. Given a program $\mathcal{P}$ we need to extract a set of interpolation data which we can use for the interpolation process. This procedure—for each accuracy P—is described in Fig. 13. Note that the data pairs $(R(X_i), R(Y_i))$ obtained in this way coincide with the values $(d_i^{\mathsf{x}}, f_{\mathcal{P}}(d_i^{\mathsf{x}}))$ used in Section 3.1.

Definition 4.2 (*IIFSP*). Let $\{(d_i^{\mathsf{x}}, d_i^{\mathsf{y}}) \mid 1 \leqslant i \leqslant 2^P\}$ be a sequence of interpolation data constructed via Algorithm 4.1 from the program $\mathcal{P}$ using accuracy P. Let $f_{\mathcal{P}}$ be the embedded $T_{\mathcal{P}}$-operator associated with the program $\mathcal{P}$ using the mapping R with base B. Then $((\mathbb{R}^2, d_2), \Omega)$ is called an *interpolating iterated function system* (*IIFSP*), with $\Omega = \{\omega_i \mid 1 \leqslant i \leqslant 2^P\}$ and $\omega_i : \mathbb{R}^2 \to \mathbb{R}^2 : (x, y) \mapsto (\omega_i^{\mathsf{x}}(x), \omega_i^{\mathsf{y}}(y))$, where ω_i^{x} and ω_i^{y} are defined by

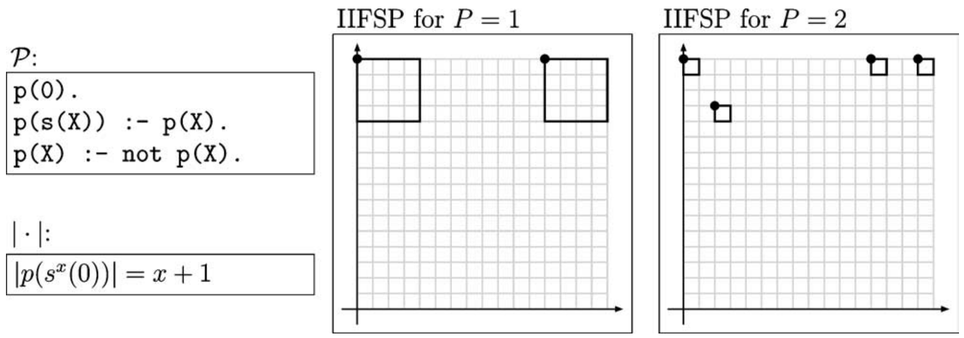
$$\omega_i^{\mathsf{x}}(x) = \frac{1}{B^P} x + d_i^{\mathsf{x}}, \quad \text{and}$$

$$\omega_i^{\mathsf{y}}(y) = \frac{1}{B^P} y + f_{\mathcal{P}}(d_i^{\mathsf{x}}) - \frac{f_{\mathcal{P}}(0)}{B^P}.$$

Fig. 14 shows a logic program and schematically two corresponding interpolating iterated function systems for $B = 4$.

Each IIFSP constructed for the program $\mathcal{P}$ and accuracy P corresponds to a linear iterated function system constructed for the periodicity P as in Algorithm 3.7. Therefore it is obvious that the resulting mappings indeed constitute hyperbolic iterated function systems which satisfy $\pi_{\mathsf{x}}(A) = D_{\mathsf{f}}$ for their attractors A.

For the remainder of this section we denote by $\mathcal{F}$ the set of all continuous functions from D_{f} to $\mathbb{R}$, and by d_{f} the maximum metric $d_{\mathsf{f}}(f, g) = \max_{x \in D_{\mathsf{f}}}\{|f(x) - g(x)|\}$ on this set. Thus $(\mathcal{F}, d_{\mathsf{f}})$ is a complete metric space, and convergence with respect to it is uniform convergence.



```
p(0).
p(s(X)) :- p(X).
p(X) :- not p(X).
```

Fig. 14. A logic program and two corresponding IIFSPs.

Lemma 4.3. *The function $T_P : \mathcal{F} \to \mathcal{F}$ defined by $T_P(f)(x) = \omega_i^{\mathsf{y}} \circ f \circ \omega_i^{\mathsf{x}^{-1}}(x)$, where i is chosen appropriately depending on x, is a contraction on $(\mathcal{F}, d_{\mathsf{f}})$ with contractivity factor $\frac{1}{B^P}$.*

Proof. The function $T_P f$ can be characterized by cases depending on the input x, by setting

$$T_P f(x) = T_{P,i} f(x) \quad \text{for } x \in \left[d_i^{\mathsf{x}}, d_i^{\mathsf{x}} + \frac{R_{\mathsf{m}}}{B^P}\right] \cap D_{\mathsf{f}}$$

with each $T_{P,i} f$ defined as

$$(T_{P,i} f)(x) = \omega_i^{\mathsf{y}}\left(f\left(\omega_i^{\mathsf{x}^{-1}}(x)\right)\right) = \frac{1}{B^P} \cdot f\left((x - d_i^{\mathsf{x}}) \cdot B^P\right) + f_{\mathcal{P}}(d_i^{\mathsf{x}}) - \frac{f_{\mathcal{P}}(0)}{B^P}.$$

In the sequel we will simply write $T_P f : D_{\mathsf{f}} \to \mathbb{R} : x \mapsto \omega_i^{\mathsf{y}}(f(\omega_i^{\mathsf{x}^{-1}}(x)))$ since $T_P f$ is a well-defined function from D_{f} to $\mathbb{R}$.

To show that T_P maps $\mathcal{F}$ to itself, we need to show that $T_P f : D_{\mathsf{f}} \to \mathbb{R}$ is a continuous function for all $f \in \mathcal{F}$. The continuity of each $T_{P,i} f$ is obvious, since it is a composition of continuous functions. Since $d_i^{\mathsf{x}} + \frac{R_{\mathsf{m}}}{B^P} < d_{i+1}^{\mathsf{x}}$ for each $i < 2^P$ this observation suffices.

Contractivity of T_P follows immediately from the definition since

$$\begin{aligned} d_{\mathsf{f}}(T_P f, T_P g) &= \max\left\{\left|T_P f(x) - T_P g(x)\right| \mid x \in D_{\mathsf{f}}\right\} \\ &= \frac{1}{B^P} \cdot \max\left\{\left|f\left((x - d_i^{\mathsf{x}}) \cdot B^P\right) - g\left((x - d_i^{\mathsf{x}}) \cdot B^P\right)\right| \mid x \in D_{\mathsf{f}}\right\} \\ &\leqslant \frac{1}{B^P} \cdot d_{\mathsf{f}}(f, g), \end{aligned}$$

and we can conclude that T_P is a contraction with contractivity factor $\frac{1}{B^P}$. □

Lemma 4.4. *Let $D = \{(d_i^{\mathsf{x}}, d_i^{\mathsf{y}})\}$ be a sequence of interpolation data and $((\mathbb{R}^2, d_2), \Omega)$ be an interpolating iterated function system with attractor A, as constructed in Definition 4.2 from the program $\mathcal{P}$ using accuracy P. Let $f_{\mathcal{P}}$ be the embedded $T_{\mathcal{P}}$-operator associated with the program $\mathcal{P}$ using the mapping R. Then there is a unique continuous function*

$f : D_{\mathsf{f}} \to \mathbb{R}$ with $T_P f = f$. Furthermore, f interpolates the data and its graph coincides with the attractor A.

Proof. The proof is divided in two steps. First, we will show that the function f is uniquely determined and interpolates the data. Afterwards, we will show that the graph of this function coincides with the attractor A.

(i) From Lemma 4.3 we know that the contraction T_P maps $\mathcal{F}$ to itself. By the Banach contraction mapping theorem we can conclude that there is exactly one function f with $T_P f = f$. This function is continuous since it is an element of $\mathcal{F}$. To show that f interpolates the data we need to show that $f(d_i^{\mathsf{x}}) = d_i^{\mathsf{y}} = f_{\mathcal{P}}(d_i^{\mathsf{x}})$ for all $(d_i^{\mathsf{x}}, d_i^{\mathsf{y}}) \in D$. Since we know that $T_P f = f$ we obtain

$$\begin{aligned} f(d_i^{\mathsf{x}}) &= \frac{1}{B^P} \cdot f\big((d_i^{\mathsf{x}} - d_i^{\mathsf{x}}) \cdot B^P\big) + f_{\mathcal{P}}(d_i^{\mathsf{x}}) - \frac{f_{\mathcal{P}}(0)}{B^P} \\ &= f_{\mathcal{P}}(d_i^{\mathsf{x}}) + \frac{f(0)}{B^P} - \frac{f_{\mathcal{P}}(0)}{B^P}. \end{aligned}$$

As there is a d_i^{x} which is equal to 0 we get

$$f(0) - \frac{f(0)}{B^P} = f_{\mathcal{P}}(0) - \frac{f_{\mathcal{P}}(0)}{B^P},$$

which gives us the equality $f(0) = f_{\mathcal{P}}(0)$ and hence $f(d_i^{\mathsf{x}}) = f_{\mathcal{P}}(d_i^{\mathsf{x}})$ holds for all d_i^{x}.

(ii) In order to show that the graph $F = \{(x, f(x)) \mid x \in D_{\mathsf{f}}\}$ of the function f coincides with the attractor, it suffices to show that $F = \Omega(F)$—since there is only one fixed point of Ω, it then follows that $F = A$. So it suffices to show that (ii.a) $\Omega(F) \subseteq F$ and (ii.b) $F \subseteq \Omega(F)$. In order to prove (ii.a) we show that $\omega_i((x, f(x))) \in F$ for all $(x, f(x)) \in F$ and all $\omega_i \in \Omega$, i.e., $(\omega_i^{\mathsf{x}}(x), \omega_i^{\mathsf{y}}(f(x))) \in F$. This follows immediately from $f = T_P f = \omega_i^{\mathsf{y}} \circ f \circ \omega_i^{\mathsf{x}^{-1}}$, since this implies $f \circ \omega_i^{\mathsf{x}} = \omega_i^{\mathsf{y}} \circ f$ and hence $(\omega_i^{\mathsf{x}}(x), \omega_i^{\mathsf{y}}(f(x))) \in F$. Consequently, $\Omega(F) \subseteq F$. Since $\pi_{\mathsf{x}}(A) = D_{\mathsf{f}}$ it follows that for all $x \in D_{\mathsf{f}}$ there is an $x' \in D_{\mathsf{f}}$ and an $\omega_i \in \Omega$ such that $x = \omega_i^{\mathsf{x}}(x')$. From $(x', f(x')) \in F$ and $f \circ \omega_i^{\mathsf{x}} = \omega_i^{\mathsf{y}} \circ f$ we can conclude that $f(x) = \omega_i^{\mathsf{y}}(f(x'))$ and hence $(x, f(x)) = (\omega_i^{\mathsf{x}}(x'), \omega_i^{\mathsf{y}}(f(x')))$. So (ii.b) holds which completes the proof. □

We call the function f from Lemma 4.4 a *fractal interpolation function for the program $\mathcal{P}$* with respect to accuracy P: It is an interpolation function for a set of points which belong to the graph of the embedded $T_{\mathcal{P}}$-operator $f_{\mathcal{P}}$. Both $f_{\mathcal{P}}$ and the fractal interpolation function coincide at least on the given data points, the number of which depends on the chosen accuracy P. In the remainder of this section we will study the sequence of fractal interpolation functions obtained by increasing the accuracy. We show first that this sequence is a Cauchy sequence, and then that its limit converges to $f_{\mathcal{P}}$ for programs with Lipschitz continuous $f_{\mathcal{P}}$.

We next need to obtain upper and lower bounds on the values of fractal interpolation functions. Fixing an accuracy P, recall that the corresponding fractal interpolation function f is the unique fixed point of the function T_P, i.e., $f = T_P(f) = \omega_i^{\mathsf{y}} \circ f \circ {\omega_i}^{\mathsf{x}^{-1}}$. Since

$\omega_i^{\mathsf{y}}(y) = \frac{y}{B^P} + f_{\mathcal{P}}(d_i^{\mathsf{x}}) - \frac{f_{\mathcal{P}}(0)}{B^P}$ it is easily verified that a lower bound for f is given by

$$f_{\mathsf{min}} = -\sum_{i=1}^{\infty} \frac{R_{\mathsf{m}}}{(B^P)^i} = -\frac{R_{\mathsf{m}}}{B^P - 1}.$$

Analogously, an upper bound f_{max} can be obtained as

$$f_{\mathsf{max}} = R_{\mathsf{m}} + \frac{R_{\mathsf{m}}}{B^P - 1}.$$

Lemma 4.5. *Let $\mathcal{P}$ be a program with Lipschitz continuous $f_{\mathcal{P}}$. For each accuracy i let f_i be the corresponding fractal interpolation function. Then the sequence $(f_i)_{i\in\mathbb{N}}$ is a Cauchy sequence in $(\mathcal{F}, d_{\mathsf{f}})$.*

Proof. The proof is divided into two steps. We first compute the distance between f_i and f_{i+1}, and then use this to show that the sequence is Cauchy.

(i) Let i be fixed. We compute the distance between the two fractal interpolation functions f_i and f_{i+1}. For convenience we use f for f_i and $\hat{f}$ for f_{i+1}. For both functions we know that $Tf = f$ and $\hat{T}\hat{f} = \hat{f}$ hold, where T and $\hat{T}$ denote the operators introduced in Lemma 4.3, constructed for the accuracies $P = i$ and $\hat{P} = i + 1$ respectively. We use the notation d_j^{x} for the interpolation values for f and $\hat{d}_k^{\mathsf{x}}$ for the interpolation values for $\hat{f}$. From $Tf = f$ and $\hat{T}\hat{f} = \hat{f}$ we can conclude that

$$f(x) = \frac{f((x - d_j^{\mathsf{x}}) \cdot B^i)}{B^i} + f_{\mathcal{P}}(d_j^{\mathsf{x}}) - \frac{f_{\mathcal{P}}(0)}{B^i} \quad \text{for } x \in \left[d_j^{\mathsf{x}}, d_j^{\mathsf{x}} + \frac{R_{\mathsf{m}}}{B^i}\right] \cap D_{\mathsf{f}},$$
$$\hat{f}(x) = \frac{\hat{f}((x - \hat{d}_k^{\mathsf{x}}) \cdot B^{i+1})}{B^{i+1}} + f_{\mathcal{P}}(\hat{d}_k^{\mathsf{x}}) - \frac{f_{\mathcal{P}}(0)}{B^{i+1}} \quad \text{for } x \in \left[\hat{d}_k^{\mathsf{x}}, \hat{d}_k^{\mathsf{x}} + \frac{R_{\mathsf{m}}}{B^{i+1}}\right] \cap D_{\mathsf{f}}.$$

Therefore, we get for the distance $d_{\mathsf{f}}(f, \hat{f})$:

$$\begin{aligned}
d_{\mathsf{f}}(f, \hat{f}) = \max_x \Bigg\{ \Bigg| &\left(\frac{f((x - d_j^{\mathsf{x}}) \cdot B^i)}{B^i} + f_{\mathcal{P}}(d_j^{\mathsf{x}}) - \frac{f_{\mathcal{P}}(0)}{B^i}\right) \\
&- \left(\frac{\hat{f}((x - \hat{d}_k^{\mathsf{x}}) \cdot B^{i+1})}{B^{i+1}} + f_{\mathcal{P}}(\hat{d}_k^{\mathsf{x}}) - \frac{f_{\mathcal{P}}(0)}{B^{i+1}}\right) \Bigg| \Bigg\} \\
\leqslant \max_x \Bigg\{ &\left| \frac{B \cdot f((x - d_j^{\mathsf{x}}) \cdot B^i) - \hat{f}((x - \hat{d}_k^{\mathsf{x}}) \cdot B^{i+1})}{B^{i+1}} \right| \\
&+ \left| f_{\mathcal{P}}(d_j^{\mathsf{x}}) - f_{\mathcal{P}}(\hat{d}_k^{\mathsf{x}}) \right| + \left| \frac{-B \cdot f_{\mathcal{P}}(0) + f_{\mathcal{P}}(0)}{B^{i+1}} \right| \Bigg\} \\
\leqslant \max_x \Bigg\{ &\frac{B \cdot (R_{\mathsf{m}} + \frac{R_{\mathsf{m}}}{B^i - 1}) + \frac{R_{\mathsf{m}}}{B^{i+1} - 1}}{B^{i+1}} + L \cdot |d_j^{\mathsf{x}} - \hat{d}_k^{\mathsf{x}}| + \frac{(B-1) \cdot R_{\mathsf{m}}}{B^{i+1}} \Bigg\}.
\end{aligned}$$

The last step uses the fact that $f_{\mathcal{P}}$ is continuous on (D_{f}, d_1) with some Lipschitz constant L, and the results concerning minima and maxima of f_i.

Since d_j^{x} and $\hat{d}_k^{\mathsf{x}}$ are chosen with respect to the same x we know that the distance between d_j^{x} and $\hat{d}_k^{\mathsf{x}}$ is bounded by $\frac{R_{\mathsf{m}}}{B^i}$. Hence

$$\begin{aligned} d_{\mathsf{f}}(f, \hat{f}) &\leqslant \frac{B \cdot (R_{\mathsf{m}} + \frac{R_{\mathsf{m}}}{B^i - 1}) + \frac{R_{\mathsf{m}}}{B^{i+1} - 1}}{B^{i+1}} + L \cdot \frac{R_{\mathsf{m}}}{B^i} + \frac{(B-1) \cdot R_{\mathsf{m}}}{B^{i+1}} \\ &\leqslant \frac{B \cdot (R_{\mathsf{m}} + \frac{R_{\mathsf{m}}}{B^i - 1}) + \frac{R_{\mathsf{m}}}{B^{i+1} - 1} + L \cdot R_{\mathsf{m}} + (B-1) \cdot R_{\mathsf{m}}}{B^{i+1}} \\ &\leqslant R_{\mathsf{m}} \cdot \frac{B \cdot (1 + \frac{1}{B^i - 1}) + \frac{1}{B^{i+1} - 1} + L + (B-1)}{B^{i+1}} \\ &< \frac{R_{\mathsf{m}}}{B^{i+1}} \cdot (4B + L) \leqslant \frac{R_{\mathsf{m}}(4 + L)}{B^i}. \end{aligned}$$

(ii) From part (i) we can conclude that for $j \leqslant k$ we have

$$d_{\mathsf{f}}(f_j, f_k) < \sum_{i=j}^{k} \frac{R_{\mathsf{m}}(4 + L)}{B^i} = \frac{R_{\mathsf{m}}(4 + L)}{B - 1} \left(\frac{1}{B^j} - \frac{1}{B^k} \right).$$

So for fixed j the value of $d_{\mathsf{f}}(f_j, f_k)$ is bounded by $d_{\mathsf{f}}(f_j, f_k) \leqslant \frac{R_{\mathsf{m}}(4+L)}{B-1} \cdot \frac{1}{B^j}$, and we obtain that $(f_i)_{i \in \mathbb{N}}$ is a Cauchy sequence, since for any $\varepsilon > 0$ there is some n such that for all $j, k > n$ we have $d_{\mathsf{f}}(f_j, f_k) < \varepsilon$. □

Theorem 4.6 (Approximation Theorem). *Let P be a program with Lipschitz continuous $f_{\mathcal{P}}$. Then the sequence $(f_i)_{i \in \mathbb{N}}$ of fractal interpolation functions with accuracies i converges uniformly to $f_{\mathcal{P}}$ in the complete metric space $(\mathcal{F}, d_{\mathsf{f}})$.*

Proof. First note that for each $x \in D_{\mathsf{f}}$ there is a sequence of interpolation data points $(d_i^{\mathsf{x}}, d_i^{\mathsf{y}})$ such that each $(d_i^{\mathsf{x}}, d_i^{\mathsf{y}})$ belongs to the interpolation data for accuracy i, each d_i^{x} is the offset of the appropriately chosen mapping ω_i^{x} for x, and $\lim_{i \to \infty} d_i^{\mathsf{x}} = x$.

From the continuity of f_i and the uniform convergence of f_i to some f by Lemma 4.5 we can conclude that the sequence $(f_i(d_i^{\mathsf{x}}))_{i \in \mathbb{N}}$ converges to $f(x)$. Knowing that the f_i are interpolation functions, hence $f_i(d_i^{\mathsf{x}}) = f_{\mathcal{P}}(d_i^{\mathsf{x}})$, we obtain that the sequence $(f_{\mathcal{P}}(d_i^{\mathsf{x}}))_{i \in \mathbb{N}}$ converges to $f(x)$. But $f_{\mathcal{P}}$ is continuous by assumption, so $\lim_i f_{\mathcal{P}}(d_i^{\mathsf{x}}) = f_{\mathcal{P}}(\lim_i d_i^{\mathsf{x}}) = f_{\mathcal{P}}(x)$ and hence $f(x) = f_{\mathcal{P}}(x)$ for all $x \in D_{\mathsf{f}}$. □

Theorem 4.6 shows that we can approximate the graph of any logic program for which $f_{\mathcal{P}}$ is Lipschitz continuous arbitrarily well. Unfortunately, the necessary number of mappings grows exponentially with the accuracy. From $d_{\mathsf{f}}(f_j, f_k) < \frac{R_{\mathsf{m}}(4+L)}{B-1}(\frac{1}{B^j} - \frac{1}{B^k})$ it follows that $d_{\mathsf{f}}(f_j, f_{\mathcal{P}}) \leqslant \frac{R_{\mathsf{m}}(4+L)}{B^j(B-1)}$, i.e., for any given $\varepsilon > 0$ we can construct an IIFSP, such that the corresponding fractal interpolation function f_P lies within an ε-neighbourhood of $f_{\mathcal{P}}$. This IFS needs to be constructed using accuracy P such that $\frac{R_{\mathsf{m}}(4+L)}{B^P(B-1)} < \varepsilon$, i.e.,

$$P > \ln_B \frac{R_{\mathsf{m}}(4 + L)}{\varepsilon(B - 1)}.$$

5. Logic programs as recurrent RBF-networks

We will now proceed to the task which motivated our investigations, namely the approximation of logic programs by artificial neural networks. Such networks consist of a number of simple computational units, which are connected in the sense that they can propagate simple information—usually in the form of real numbers—along these connections. We want to construct a network which computes an approximation of $f_{\mathcal{P}}(x)$ for a given x. To this end, we will employ the results of the previous sections. More precisely, we will show how the fractal interpolation systems from Section 4 can be encoded.

The basic idea underlying our encoding is to exploit the self-similarity of the fractal interpolation functions f, and the "recursive" nature of the corresponding iterated function systems. In order to obtain the function value $f(x)$ for some given $x \in D_{\mathsf{f}}$, we first need to find the correct mapping $\omega_i = (\omega_i^{\mathsf{x}}, \omega_i^{\mathsf{y}})$, i.e., the one for which $x \in \omega_i^{\mathsf{x}}(D_{\mathsf{f}})$, and compute $\omega_i^{\mathsf{y}}(y)$ (where initially $y = 0$). Then we zoom in on the image of $R_{\mathsf{m}} \times R_{\mathsf{m}}$ under ω_i and repeat the process.

For our implementation of this idea we use radial basis function networks (RBF-networks). These consist of simple units which perform "radial basis functions" as input-output-mappings. These are functions f for which the values $f(x)$ are distributed symmetrically around a center. Two examples and a very simple schematic RBF-network are shown in Fig. 15.

RBF-networks are known to be universal approximators, i.e., with networks as shown in Fig. 15 it is possible to approximate any continuous function to any given accuracy, provided sufficiently many units are being used in the middle layer.

To simplify our exhibition and the construction of the network we introduce a new type of unit, which we call an $\mathrm{RBF}_{s,x}^{x,y}$-unit. It computes two distinct output-values x' and y'. Furthermore, it computes a parametrised radial basis function, where an additional scaling s is applied to y'. These units can be understood as abbreviations, since they can be converted into a network consisting of simple units, i.e., although we are using $\mathrm{RBF}_{s,x}^{x,y}$-units it is possible to encode the entire resulting network using standard RBF-units. Fig. 16 shows the dynamics and a schematic plot of an $\mathrm{RBF}_{s,x}^{x,y}$-unit. The static parameters of each $\mathrm{RBF}_{s,x}^{x,y}$-unit are the center μ, the width σ and the height h.

Using $\mathrm{RBF}_{s,x}^{x,y}$-units we can construct the network as shown in Fig. 18 using the algorithm shown in Fig. 17. The example program for the construction is taken from Sec-

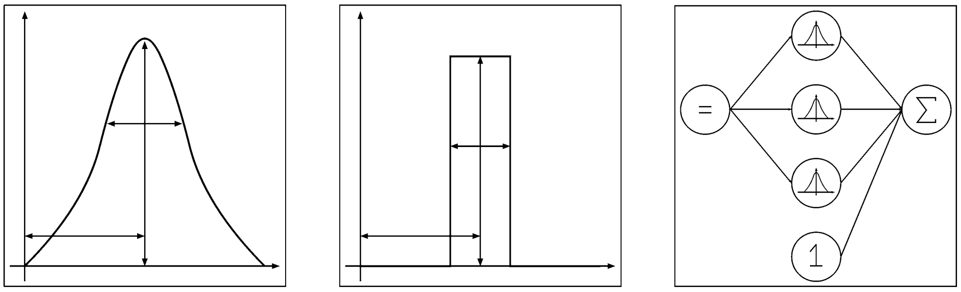


Fig. 15. Two examples of radial basis functions and a simple RBF-network.

$$x' = \begin{cases} (x - \mu + \sigma) \cdot B^P & \text{if } |\mu - x| \le \sigma \\ 0 & \text{otherwise} \end{cases}$$

$$y' = \begin{cases} s \cdot h & \text{if } |\mu - x| \le \sigma \\ 0 & \text{otherwise} \end{cases}$$

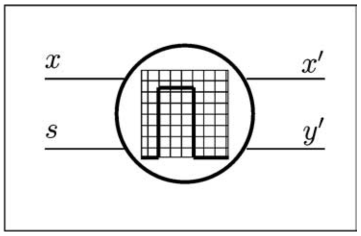


Fig. 16. Dynamics and scheme of an $\mathrm{RBF}^{x,y}_{s,x}$-unit.

Algorithm 5.1 (Construction of recurrent RBFN$_{\mathcal{P}}$). Let $\mathcal{P}$ be a logic program and B the base of the embedding R.

1. Choose a periodicity $P \geqslant 1$.
2. Create an empty 3-layered RBF-network. Add three input units (s, x, y) to the first layer and three output units (s', x', y') to the third. The input units compute the identity function and the output units return a weighted sum of their inputs.
3. The hidden layer consists of 2^P $\mathrm{RBF}^{x,y}_{s,x}$-units initialised as follows:
   a) Compute the $\mathrm{IFS}^l_{\mathcal{P}}$ $((\mathbb{R}^2, d_2), \Omega)$ for $\mathcal{P}$ using the periodicity P, with $\Omega = \{(\omega^{\mathsf{x}}_i, \omega^{\mathsf{y}}_i) \mid 1 \leqslant i \leqslant 2^P\}$, $\omega^{\mathsf{x}}_i(x) = \frac{1}{B^P} \cdot x + d^{\mathsf{x}}_i$ and $\omega^{\mathsf{y}}_i(y) = \frac{1}{B^P} \cdot y + d^{\mathsf{y}}_i$, as described in Algorithm 3.7.
   b) For all i the $\mathrm{RBF}^{x,y}_{s,x}$, i-unit is initialised with $\sigma_i = \frac{1}{2 \cdot B^P}$, $\mu_i = d^{\mathsf{x}}_i + \sigma_i$ and $h_i = d^{\mathsf{y}}_i$.
4. Connect the units as shown in Fig. 18, where all weights are set to 1, but the connection from s to s' is set to B^{-P}.

Fig. 17. Algorithm constructing RBF-network.

```
e(0).
e(s(X)) :- not e(X).
o(X) :- not e(X).
```

$$|e(s^x(0))| = 2 \cdot x + 1$$
$$|o(s^x(0))| = 2 \cdot x + 2$$

$$\Omega = \left\{ \begin{pmatrix} \frac{1}{16} & 0 \\ 0 & \frac{1}{16} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0 \\ \frac{5}{16} \end{pmatrix}, \begin{pmatrix} \frac{1}{16} & 0 \\ 0 & \frac{1}{16} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \frac{1}{16} \\ \frac{5}{16} \end{pmatrix}, \begin{pmatrix} \frac{1}{16} & 0 \\ 0 & \frac{1}{16} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \frac{1}{4} \\ \frac{15}{64} \end{pmatrix}, \begin{pmatrix} \frac{1}{16} & 0 \\ 0 & \frac{1}{16} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \frac{5}{16} \\ \frac{15}{64} \end{pmatrix} \right\}$$

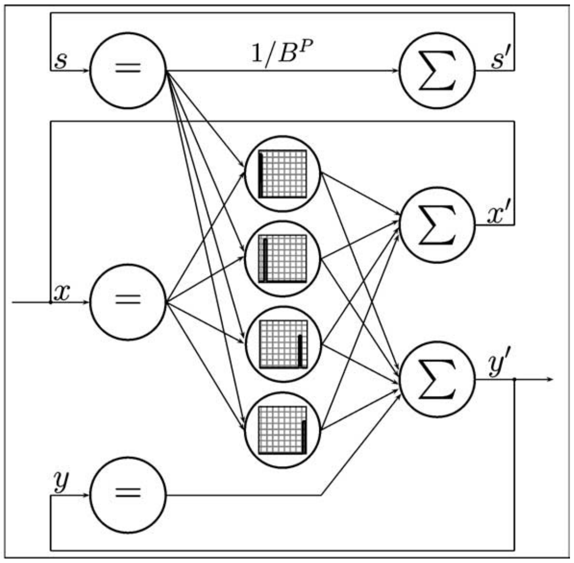


Fig. 18. A logic program, level mapping, interpolating IFS for $P = 2$ and corresponding recurrent $\mathrm{RBF}^{x,y}_{s,x}$-network. The example program and its $\mathrm{IFS}^l_{\mathcal{P}}$ are taken from the end of Section 3.2.

tion 3.2, where the corresponding $\mathrm{IFS}^l_{\mathcal{P}}$ was already computed. The three initial inputs s_0, y_0 and x_0 need to be initialised with $s_0 = 1$, $y_0 = 0$, and $x_0 = x$. The network computes an approximation of $f_{\mathcal{P}}$ for a given input x. Each iteration through the network performs the following computations:

- The scaling factor s is multiplied with $\frac{1}{B^P}$.
- Each $\mathrm{RBF}^{x,y}_{s,x}$-unit computes the corresponding outputs x' and y', where for exactly one unit $x', y' \neq 0$. Since x_0 was initialised with $R(I)$, the output of the "active" $\mathrm{RBF}^{x,y}_{s,x}$-units after the first iteration is $x' = (x - \mu + \sigma) \cdot B^P$, i.e., we have $x' = (x - d^{\mathrm{x}}_i) \cdot B^P = R(R^{-1}(x) \setminus R^{-1}(d^{\mathrm{x}}_i)) \cdot B^P$. This is the "zooming into the interpretation" mentioned earlier, i.e., x' is a left-shifted version of x.
- The current y'-output of the "active" unit is added to the previous y.

The output y of the network converges to the value of the fractal interpolation function f defined by the IFS, which was used to construct the network. More precisely, we have $d_1(y, f(x)) = (\frac{1}{B^P})^i$, where i denotes the number of iterations performed and P is the accuracy used for the construction. Furthermore, we know that $d_{\mathrm{f}}(f, f_{\mathcal{P}}) \leqslant \frac{R_{\mathrm{m}}(4+L)}{B^P(B-1)}$, which yields $d_1(y, f_{\mathcal{P}}(x)) \leqslant \frac{R_{\mathrm{m}}(4+L)}{B^P(B-1)} + (\frac{1}{B^P})^i$. We conclude that we can approximate the single-step operator of any logic program for which the embedding is Lipschitz continuous up to any desired degree of accuracy.

6. Related work

One of the key ideas on which our work on neural-symbolic integration is based, is to represent logic programs by representing their associated immediate consequence operators. This approach was put forward by Hölldobler and Kalinke [28], and reported also in [20], in order to encode propositional logic programs by feedforward neural networks with threshold activation functions. They also observe that these networks can be cast into a recurrent architecture in order to mimic the iterative behaviour of the operator.

Two major lines of investigation were spawned by this work. D'Avila Garcez, Broda, Gabbay, and Zaverucha [12,14] extend the work by Hölldobler and Kalinke to cover networks with sigmoidal activation functions, and study machine learning and knowledge extraction aspects of the resulting frameworks.

The second line of investigation was initiated by Hölldobler, Kalinke, and Störr [29], who study first-order logic programs and how to approximate their single-step operators by feedforward neural networks. A general approximation theorem due to Funahashi [16] is of central importance for their approach, which is restricted to the study of acyclic programs with injective level mappings. They show that these programs can be approximated arbitrarily well by feedforward networks, but do not specify any means for actually constructing them.

Generalizations of this approach to programs with continuous single-step operators, and also to other semantic operators, are obtained by Hitzler and Seda [18,21,22], reported also in [20]. At this stage, topological and metric studies of declarative semantics, originally developed for entirely different purposes [4,5,15,23,38,39,41,42], come into play. From this perspective, our work is in the spirit of the general programme of research laid out by Blair et al. [9].

Work by Blair et al. on continualizations of discrete systems [8] relates very closely to the particular tool we have chosen for our approach, namely iterated function systems. In their paper, Blair et al. study covered programs and show, amongst other things, that their single-step operators can be obtained by means of attractors of affine hyperbolic finite automata, which in turn can be understood as iterated function systems. Their work also shows the intimate relationship between logic programming and dynamical systems related to self-similarity and chaos theory, which we have been able to put to use in this paper.

7. Conclusions and further work

We have presented results for exact and approximate representation of single-step operators associated with logic programs by iterated function systems, fractal interpolation systems, and recurrent radial basis function networks. Our results cover first-order logic programs with function symbols under the provision that the embedded associated single-step operator is Lipschitz continuous. We have given algorithms for constructing approximating iterated function systems and recurrent radial basis function networks for given logic programs.

As to the relation with the work by Blair et al. [8], we note that the exact relationship between the class of programs covered by their results, namely covered programs, and ours, namely those whose embedded single-step operator is Lipschitz continuous, remains to be determined and will require further research, as already mentioned. While the general observation in [8] that covered logic programs can be represented by iterated function systems breaks the ground for deep investigations into these matters, our results provide explicit approximations in the Euclidean plane, which can be converted to a standard neural network architecture in a straightforward way. The concrete results and constructions which we provide, however, come at the price of the stronger hypothesis of Lipschitz continuity required for our results. We believe that this requirement can be weakened, but different mathematical approaches than the one employed here may be needed in order to obtain satisfactory results.

There is also one caveat: If one would like to construct an approximating system or network which approximates a given logic program within some a priori given error bound, then we can only guarantee this if a Lipschitz constant L of the function $f_{\mathcal{P}}$—which is the embedding of the single-step operator $T_{\mathcal{P}}$ in the reals—is not only existent but also *known*. This can be seen from the calculations of upper error bounds at the ends of Sections 4 and 5. We do not know of any general method for obtaining Lipschitz constants, and ways of doing this will be subject to further research. For certain well-behaved programs, Lipschitz constants are easily calculated. For acyclic programs with injective level mappings as covered in [29], for example, a Lipschitz constant is $\frac{1}{B-2}$, where $B > 2$ is the base used for the embedding R. In these cases our results yield exact algorithms for obtaining approximating networks given an a priori error bound.

Our results surpass those of [29] in at least two ways. Firstly, for the programs covered in [29], namely acyclic ones with injective level mappings, we are now able to give an algorithm for constructing approximating networks. Secondly, we show that a larger class of programs than covered in [29] can be approximated in principle, namely those with Lipschitz continuous embedded single-step operator, and furthermore, we have shown that for these we can provide explicit parameters for approximating recurrent neural networks, provided a suitable Lipschitz constant can be determined. This latter point is related to the results in [20–22], where a larger class of programs—those with continuous single-step operator—were treated, but without providing explicit constructions of approximating networks. So our conclusions are stronger, but so are our assumptions.

Let us also note that we use a different network architecture than in [20–22,29], namely recurrent RBF-networks instead of three-layer feedforward networks with sigmoidal activation functions. Indeed, we believe that RBF-networks constitute a much more natural choice for representing logic programs at least under the general approach inspired by [28]. This is due to the intuition that points or interpretations which are "close" to each other (topologically or metrically speaking) are supposed to represent similar meaning. The specific shape of the activation functions in RBF-networks thus can be understood in such a way that a unit becomes active only for a cluster of values, i.e., interpretations, which have similar meaning. The binary nature of sigmoidal activation functions seems to be much more difficult to explain from an intuitive perspective. Certainly, our recurrent network can be unfolded to a feedforward architecture with several layers if this is desired, and on the mathematical level it should not make much of a difference which architecture is being used. The question of how to obtain algorithms for constructing approximating networks with sigmoidal activation functions, however, is probably rather hard, but may be solvable by first understanding Lipschitz constants of embedded single-step operators.

Investigating Lipschitz constants as mentioned provides a natural next step in our investigations. It has to be said, however, that it is not yet clear how our results can be used for designing useful hybrid systems. Nevertheless, certain questions are natural to be asked at this stage. Can we use our approach for extracting symbolic knowledge from trained neural networks? Can network learning then be understood from a symbolic perspective by observing changes in the (extracted) symbolic knowledge during the learning process? Even in the finite (propositional) case research has not yet led to satisfactory answers to these questions, and the case of first-order logic which we address here is naturally much more difficult to work with, but should be investigated. Entirely new methods may have to be developed for this purpose, as argued by Hölldobler in [27].

Acknowledgements

We benefited substantially from discussions with Steffen Hölldobler and his support of the project. The comments of three anonymous referees were highly appreciated and led to improvements of the presentation of our results. We are very grateful that Howard Blair spotted a mistake in an earlier version of this paper, which we were now able to remove.

References

[1] K.R. Apt, D. Pedreschi, Reasoning about termination of pure Prolog programs, Inform. and Comput. 106 (1993) 109–157.
[2] S. Bader, From logic programs to iterated function systems, Master's Thesis, Department of Computer Science, Dresden University of Technology, 2003.
[3] M. Barnsley, Fractals Everywhere, Academic Press, San Diego, CA, 1993.
[4] A. Batarekh, V.S. Subrahmanian, The query topology in logic programming, in: Proceedings of the 1989 Symposium on Theoretical Aspects of Computer Science, in: Lecture Notes in Computer Science, vol. 349, Springer, Berlin, 1989, pp. 375–387.
[5] A. Batarekh, V.S. Subrahmanian, Topological model set deformations in logic programming, Fund. Inform. 12 (1989) 357–400.
[6] M. Bezem, Characterizing termination of logic programs with level mappings, in: E.L. Lusk, R.A. Overbeek (Eds.), Proceedings of the North American Conference on Logic Programming, MIT Press, Cambridge, MA, 1989, pp. 69–80.
[7] C.M. Bishop, Neural Networks for Pattern Recognition, Oxford University Press, 1995.
[8] H.A. Blair, J. Chidella, F. Dushin, A. Ferry, P. Humenn, A continuum of discrete systems, Ann. Math. Artificial Intelligence 21 (2–4) (1997) 155–185.
[9] H.A. Blair, F. Dushin, D.W. Jakel, A.J. Rivera, M. Sezgin, Continuous models of computation for logic programs, in: K.R. Apt, V.W. Marek, M. Truszczyński, D.S. Warren (Eds.), The Logic Programming Paradigm: A 25-Year Perspective, Springer, Berlin, 1999, pp. 231–255.
[10] A. Browne, R. Sun, Connectionist inference models, Neural Networks 14 (10) (2001) 1331–1355.
[11] L. Cavedon, Acyclic programs and the completeness of SLDNF-resolution, Theoret. Comput. Sci. 86 (1991) 81–92.
[12] A.S. d'Avila Garcez, K. Broda, D.M. Gabbay, Symbolic knowledge extraction from trained neural networks: A sound approach, Artificial Intelligence 125 (2001) 155–207.
[13] A.S. d'Avila Garcez, K.B. Broda, D.M. Gabbay, Neural-Symbolic Learning Systems—Foundations and Applications, in: Perspectives in Neural Computing, Springer, Berlin, 2002.
[14] A.S. d'Avila Garcez, G. Zaverucha, The connectionist inductive learning and logic programming system, Applied Intelligence (Special Issue on Neural Networks and Structured Knowledge) 11 (1) (1999) 59–77.
[15] M. Fitting, Metric methods: Three examples and a theorem, J. Logic Programming 21 (3) (1994) 113–127.
[16] K.-I. Funahashi, On the approximate realization of continuous mappings by neural networks, Neural Networks 2 (1989) 183–192.
[17] H.W. Güsgen, S. Hölldobler, Connectionist inference systems, in: B. Fronhöfer, G. Wrightson (Eds.), Parallelization in Inference Systems, in: Lecture Notes in Artificial Intelligence, vol. 590, Springer, Berlin, 1992, pp. 82–120.
[18] P. Hitzler, Generalized metrics and topology in logic programming semantics, PhD Thesis, Department of Mathematics, National University of Ireland, University College Cork, 2001.
[19] P. Hitzler, Towards a systematic account of different logic programming semantics, in: A. Günter, R. Kruse, B. Neumann (Eds.), KI2003: Advances in Artificial Intelligence. Proceedings of the 26th Annual German Conference on Artificial Intelligence, KI2003, Hamburg, Germany, September 2003, in: Lecture Notes in Artificial Intelligence, vol. 2821, Springer, Berlin, 2003, pp. 355–369.
[20] P. Hitzler, S. Hölldobler, A.K. Seda, Logic programs and connectionist networks, J. Appl. Logic (2004). In this volume.
[21] P. Hitzler, A.K. Seda, A note on relationships between logic programs and neural networks, in: P. Gibson, D. Sinclair (Eds.), Proceedings of the Fourth Irish Workshop on Formal Methods, IWFM'00, Electronic Workshops in Computing (eWiC), British Computer Society, 2000.
[22] P. Hitzler, A.K. Seda, Continuity of semantic operators in logic programming and their approximation by artificial neural networks, in: A. Günter, R. Kruse, B. Neumann (Eds.), KI2003: Advances in Artificial Intelligence. Proceedings of the 26th Annual German Conference on Artificial Intelligence, KI2003, Hamburg, Germany, September 2003, in: Lecture Notes in Artificial Intelligence, vol. 2821, Springer, Berlin, 2003, pp. 105–119.
[23] P. Hitzler, A.K. Seda, Generalized metrics and uniquely determined logic programs, Theoret. Comput. Sci. 305 (1–3) (2003) 187–219.

[24] P. Hitzler, M. Wendt, The well-founded semantics is a stratified Fitting semantics, in: M. Jarke, J. Koehler, G. Lakemeyer (Eds.), Proceedings of the 25th Annual German Conference on Artificial Intelligence, KI2002, Aachen, Germany, September 2002, in: Lecture Notes in Artificial Intelligence, vol. 2479, Springer, Berlin, 2002, pp. 205–221.
[25] P. Hitzler, M. Wendt, A uniform approach to logic programming semantics, Theory and Practice of Logic Programming, in press.
[26] S. Hölldobler, Automated inferencing and connectionist models, Fakultät Informatik, Technische Hochschule Darmstadt, Habilitationsschrift, 1993.
[27] S. Hölldobler, Challenge problems for the integration of logic and connectionist systems, in: F. Bry, U. Geske, D. Seipel (Eds.), Proceedings 14. Workshop Logische Programmierung, in: GMD Report, vol. 90, GMD, 2000, pp. 161–171.
[28] S. Hölldobler, Y. Kalinke, Towards a massively parallel computational model for logic programming, in: Proceedings ECAI94 Workshop on Combining Symbolic and Connectionist Processing, ECCAI, 1994, pp. 68–77.
[29] S. Hölldobler, Y. Kalinke, H.-P. Störr, Approximating the semantics of logic programs by recurrent neural networks, Appl. Intelligence 11 (1999) 45–58.
[30] V. Lifschitz, Answer set planning, in: D. De Schreye (Ed.), Logic Programming. Proceedings of the 1999 International Conference on Logic Programming, MIT Press, Cambridge, MA, 1999, pp. 23–37.
[31] J.W. Lloyd, Foundations of Logic Programming, Springer, Berlin, 1988.
[32] M.J. Maher, Equivalences of logic programs, in: J. Minker (Ed.), Foundations of Deductive Databases and Logic Programming, Morgan Kaufmann, Los Altos, CA, 1988, pp. 627–658.
[33] V.W. Marek, M. Truszczyński, Stable models and an alternative logic programming paradigm, in: K.R. Apt, V.W. Marek, M. Truszczyński, D.S. Warren (Eds.), The Logic Programming Paradigm: A 25-Year Perspective, Springer, Berlin, 1999, pp. 375–398.
[34] W.S. McCulloch, W. Pitts, A logical calculus of the ideas immanent in nervous activity, Bull. Math. Biophys. 5 (1943) 115–133.
[35] S. Muggleton, L. de Raedt, Inductive logic programming: Theory and applications, J. Logic Programming 19–20 (1994) 629–679.
[36] G. Pinkas, Propositional non-monotonic reasoning and inconsistency in symmetric neural networks, in: J. Mylopoulos, R. Reiter (Eds.), Proceedings of the 12th International Joint Conference on Artificial Intelligence, Morgan Kaufmann, 1991, pp. 525–530.
[37] J.B. Pollack, Recursive distributed representations, Artificial Intelligence 46 (1) (1990) 77–105.
[38] S. Prieß-Crampe, P. Ribenboim, Logic programming and ultrametric spaces, Rendiconti di Mathematica VII (2000) 1–13.
[39] S. Prieß-Crampe, P. Ribenboim, Ultrametric spaces and logic programming, J. Logic Programming 42 (2000) 59–70.
[40] A.K. Seda, Topology and the semantics of logic programs, Fund. Inform. 24 (4) (1995) 359–386.
[41] A.K. Seda, R. Heinze, P. Hitzler, Convergence classes and spaces of partial functions, in: G.-Q. Zhang, J. Lawson, Y.M. Liu, M.K. Luo (Eds.), Domain Theory, Logic and Computation, in: Semantic Structures in Computation, vol. 3, Kluwer, 2003, pp. 75–115.
[42] A.K. Seda, M. Lane, On continuous models of computation: Towards computing the distance between (logic) programs, in: Proceedings of the Sixth International Workshop in Formal Methods (IWFM'03), Dublin City University, Dublin, Ireland, July 2003, Electronic Workshops in Computing (eWiC), British Computer Science, 2003.
[43] L. Shastri, Advances in Shruti—A neurally motivated model of relational knowledge representation and rapid inference using temporal synchrony, Appl. Intelligence 11 (1999) 78–108.
[44] G.G. Towell, J.W. Shavlik, Knowledge-based artificial neural networks, Artificial Intelligence 70 (1–2) (1994) 119–165.
[45] S. Willard, General Topology, Addison-Wesley, Reading, MA, 1970.

In: D. Seipel, M. Hanus, U. Geske, and O. Bartenstein, Proceedings of the 15th International Conference on Applications of Declarative Programming and Knowledge Management and the 18th Workshop on Logic Programming, Potsdam, Germany, March 4-6, 2004, 13-27. Technical Report 327, Bayerische Julius-Maximilians-Universität Würzburg, Institut für Informatik, March 2004.

Corollaries on the fixpoint completion: studying the stable semantics by means of the Clark completion

Pascal Hitzler[**]

Department of Computer Science, Dresden University of Technology
www.wv.inf.tu-dresden.de/~pascal/
phitzler@inf.tu-dresden.de

Abstract. The fixpoint completion $\mathsf{fix}(P)$ of a normal logic program P is a program transformation such that the stable models of P are exactly the models of the Clark completion of $\mathsf{fix}(P)$. This is well-known and was studied by Dung and Kanchanasut [15]. The correspondence, however, goes much further: The Gelfond-Lifschitz operator of P coincides with the immediate consequence operator of $\mathsf{fix}(P)$, as shown by Wendt [51], and even carries over to standard operators used for characterizing the well-founded and the Kripke-Kleene semantics. We will apply this knowledge to the study of the stable semantics, and this will allow us to almost effortlessly derive new results concerning fixed-point and metric-based semantics, and neural-symbolic integration.

1 Introduction

The fixpoint completion of normal logic programs was introduced in [15], and independently under the notion of residual program in [9]. In essence, the fixpoint completion $\mathsf{fix}(P)$ of a given program P is obtained by performing a complete unfolding through all positive body literals in the program, and by disregarding all clauses with remaining positive body literals. Its importance lies in the fact that the stable models [20] of P are exactly the supported models of $\mathsf{fix}(P)$, i.e. the models of the Clark completion [11] of $\mathsf{fix}(P)$. Also, the well-founded model [50] of P is exactly the Fitting or Kripke-Kleene model [16] of $\mathsf{fix}(P)$. These correspondences are well-known and have been employed by many authors for investigating the stable and the well-founded semantics, see e.g. [7].

The relation between a program and its fixpoint completion, however, is not exhausted by the correspondences between the different semantics just mentioned: It also concerns the semantic operators underlying these semantics, as shown in [51]. The virtue of this observation lies in the fact that it allows to carry over operator-based results on the supported, respectively, Fitting semantics, to

[**] This work was supported by a fellowship within the Postdoc-Programme of the German Academic Exchange Service (DAAD) and carried out while the author was visiting the Department of Electrical Engineering and Computer Science at Case Western Reserve University, Cleveland, Ohio.

the stable, respectively, well-founded semantics. To the best of our knowledge, this has not been noted before.

In this paper, we display the strength of the operator-based correspondence by drawing a number of corollaries on the stable semantics from it. While these results are of interest in their own right, they do not constitute the main point we want to make here. Some of them are not even new, although we give new proofs. The goal of this paper is to provide a new technical tool for studying the stable and the well-founded semantics, namely the correspondences via the fixpoint completion between the semantic operators mentioned. To display this, we draw several corollaries from results in the literature, which are all valid for logic programs over a first-order language.

The structure of the paper is as follows. In Section 2 we recall the fixpoint completion and the results due to [51] which provide the starting points for our report. In Section 3 we study continuity of the Gelfond-Lifschitz operator in the Cantor topology, thereby providing technical results which will be of use later. In Section 4 we study methods for obtaining stable models by means of limits of iterates of the Gelfond-Lifschitz operator, and in Section 5 we will discuss results on the representation of logic programs by artificial neural networks. We briefly conclude in Section 6.

Acknowledgement. Thanks go to Matthias Wendt for helpful discussions and comments.

2 The Fixpoint Completion

A (*normal*) *logic program* is a finite set of universally quantified *clauses* of the form

$$\forall(A \leftarrow L_1 \wedge \cdots \wedge L_n),$$

where $n \in \mathbb{N}$ may differ for each clause, A is an atom in a first order language $\mathcal{L}$ and $L_1, \ldots, L_n$ are literals, that is, atoms or negated atoms, in $\mathcal{L}$. As is customary in logic programming, we will write such a clause in the form

$$A \leftarrow L_1, \ldots, L_n,$$

in which the universal quantifier is understood, or even as

$$A\texttt{:-}L_1, \ldots, L_n$$

following Prolog notation. Then A is called the *head* of the clause, each L_i is called a *body literal* of the clause and their conjunction $L_1, \ldots, L_n$ is called the *body* of the clause. We allow $n = 0$, by an abuse of notation, which indicates that the body is empty; in this case the clause is called a *unit clause* or a *fact.* If no negation symbol occurs in a logic program, the program is called a *definite* logic program. The Herbrand base underlying a given program P, i.e. the set of all ground instances of atoms over $\mathcal{L}$, will be denoted by B_P, and the set of all

Herbrand interpretations by I_P, and we note that the latter can be identified simultaneously with the power set of B_P and with the set $\mathbf{2}^{B_P}$ of all functions mapping B_P into the set $\mathbf{2}$ consisting of two distinct elements. Since the set I_P is the power set of B_P, it carries set-inclusion as natural ordering, which makes it a complete lattice. By $\mathsf{ground}(P)$ we denote the (possibly infinite) set of all ground instances of clauses in P.

The *single-step* or *immediate consequence operator* [37] of P is defined as a function $T_P : I_P \to I_P$, where $T_I(I)$ is the set of all $A \in B_P$ for which there exists a clause $A \leftarrow L_1, \dots, L_n$ with $I \models L_i$ for all $i = 1, \dots, n$. A *supported model* of P is a fixed point of T_P. Supported models correspond to models of the Clark completion of P, as noted in [1]. The pre-fixed points of T_P, i.e. interpretations $I \in I_P$ with $I \subseteq T_P(I)$, are exactly the Herbrand models of P, in the sense of first-order logic. If P is definite, then T_P is in fact a Scott- (or order-) continuous operator on I_P [37], and its least fixed point $\mathsf{fix}(T_P)$ coincides with the least Herbrand model of P. The least fixed point, in this case, can be obtained as $\mathsf{fix}(T_P) = T_P \uparrow \omega := \sup_n (T_P \uparrow n) = \bigcup_n T_P \uparrow n$, where $T_P \uparrow 0 = \emptyset$ and recursively $T_P \uparrow (n+1) = T_P(T_P \uparrow n)$.

The *Gelfond-Lifschitz transformation* [20] of a program P with respect to an interpretation I is denoted by P/I, and consists of exactly those clauses $A \leftarrow A_1, \dots, A_n$, where $A_1, \dots, A_n \in B_P$, for which there exists a clause $A \leftarrow A_1, \dots, A_n, \neg B_1, \dots, \neg B_m$ in $\mathsf{ground}(P)$ with $B_1, \dots, B_m \in I$. Thus P/I is a definite program, and $\mathsf{fix}(T_{P/I})$ is well-defined. The *Gelfond-Lifschitz operator* [20] of P is now defined by $\mathrm{GL}_P : I_P \to I_P : I \mapsto \mathsf{fix}(T_{P/I})$. We call $I \in I_P$ a *stable model* of P if it is a fixed point of GL_P.

Definition 1. *A* quasi-interpretation[1] *is a set of clauses of the form $A \leftarrow \neg B_1, \dots, \neg B_m$, where A and B_i are ground atoms for all $i = 1, \dots, m$. Given a normal logic program P and a quasi-interpretation Q, we define $T'_P(Q)$ to be the quasi-interpretation consisting of the set of all clauses*

$$A \leftarrow \mathtt{body}_1, \dots, \mathtt{body}_n, \neg B_1, \dots, \neg B_m$$

for which there exists a clause

$$A \leftarrow A_1, \dots, A_n, \neg B_1, \dots, \neg B_m$$

in $\mathsf{ground}(P)$ *and clauses $A_i \leftarrow \mathtt{body}_i$ in Q for all $i = 1, \dots, n$. We explicitly allow the cases $n = 0$ or $m = 0$ in this definition.*

Note that the set of all quasi-interpretations is a complete partial order (cpo) with respect to set-inclusion. It was shown in [15], that for normal programs P, the operator T'_P is Scott-continuous on the set of all quasi-interpretations. So we can define the *fixpoint completion* $\mathsf{fix}(P)$ of P by $\mathsf{fix}(P) = T'_P \uparrow \omega$, i.e. $\mathsf{fix}(P)$ is the least fixed point of the operator T'_P.

The following was reported in [51].

[1] This notion is due to [15]. We stick to the old terminology, although quasi-interpretations should really be thought of as, and indeed are, programs with negative body literals only.

Theorem 1. *For any normal program P and (two-valued) interpretation I, we have*

$$\mathrm{GL}_P(I) = T_{\mathsf{fix}(P)}(I).$$

Proof. We show first that for every $A \in \mathrm{GL}_P(I)$ there exists a clause in $\mathsf{fix}(P)$ with head A whose body is true in I, which implies $A \in T_{\mathsf{fix}(P)}(I)$. We show this by induction on the powers of $T_{P/I}$; recall that $\mathrm{GL}_P(I) = T_{P/I}\uparrow\omega$.

For the base case $T_{P/I}\uparrow 0 = \emptyset$ there is nothing to show.

So assume now that for all $A \in T_{P/I}\uparrow n$ there exists a clause in $\mathsf{fix}(P)$ with head A, whose body is true in I. For $A \in T_{P/I}\uparrow(n+1)$ there exists a clause $A \leftarrow A_1, \ldots, A_n$ in P/I such that $A_1, \ldots, A_n \in T_{P/I}\uparrow n$, hence by construction of P/I there is a clause $A \leftarrow A_1, \ldots, A_n, \neg B_1, \ldots, \neg B_m$ in $\mathsf{ground}(P)$ with $B_1, \ldots, B_m \notin I$. By induction hypothesis we obain that for each $i = 1, \ldots, n$ there exists a clause $A_i \leftarrow \mathtt{body}_i$ in $\mathsf{fix}(P)$ with $I \models \mathtt{body}_i$, hence $A_i \in T_{\mathsf{fix}(P)}(I)$. So by definition of T'_P the clause $A \leftarrow \mathtt{body}_1, \ldots \mathtt{body}_n, \neg B_1, \ldots, \neg B_m$ is contained in $\mathsf{fix}(P)$. From $I \models \mathtt{body}_i$ and $B_1, \ldots, B_m \notin I$ we obtain $A \in T_{\mathsf{fix}(P)}(I)$ as desired.

This closes the induction argument and we obtain $\mathrm{GL}_P(I) \subseteq T_{\mathsf{fix}(P)}(I)$.

Now conversly, assume that $A \in T_{\mathsf{fix}(P)}(I)$. We show that $A \in \mathrm{GL}_P(I)$ by proving inductively on k that $T_{T'_P\uparrow k}(I) \subseteq \mathrm{GL}_P(I)$ for all $k \in \mathbb{N}$.

For the base case, we have $T_{T'_P\uparrow 0}(I) = \emptyset$ so there is nothing to show.

So assume now that $T_{T'_P\uparrow k}(I) \subseteq \mathrm{GL}_P(I)$, and let $A \in T_{T'_P\uparrow(k+1)}(I) \setminus T_{T'_P\uparrow k}(I)$. Then there exists a clause $A \leftarrow \mathtt{body}_1, \ldots, \mathtt{body}_n, \neg B_1, \ldots, \neg B_m$ in $T'_P \uparrow (k+1)$ whose body is true in I. Thus $B_1, \ldots, B_m \notin I$ and for each $i = 1, \ldots, n$ there exists a clause $A_i \leftarrow \mathtt{body}_i$ in $T'_P \uparrow k$ with $\mathtt{body}_i$ true in I. So $A_i \in T_{T'_P\uparrow k}(I) \subseteq \mathrm{GL}_P(I)$. Furthermore, by definition of T'_P there exists a clause $A \leftarrow A_1, \ldots, A_n, \neg B_1, \ldots, \neg B_m$ in $\mathsf{ground}(P)$, and since $B_1, \ldots, B_m \notin I$ we obtain $A \leftarrow A_1, \ldots, A_n \in P/I$. Since we know that $A_1, \ldots, A_n \in \mathrm{GL}_P(I)$ we obtain $A \in \mathrm{GL}_P(I)$, and hence $T_{T'_P\uparrow(k+1)}(I) \subseteq \mathrm{GL}_P(I)$. This closes the induction argument and we obtain $T_{\mathsf{fix}(P)}(I) \subseteq \mathrm{GL}_P(I)$. $\square$

The proof of Theorem 1 is taken directly from [52], which appeared in compressed form as [51]. We have included it here for completeness of the exhibition and because the result is central for the rest of this paper. This correspondence can also be carried over to the Fitting/well-founded semantics. More precisely, the following was shown in [51], from which Theorem 1 is an easy Corollary.

Theorem 2. *For any normal program P and any three-valued interpretation I we have $\Psi_P(I) = \Phi_{\mathsf{fix}(P)}(I)$, where Ψ_P is the operator due to [6] used for characterizing three-valued stable models of P, and $\Phi_{\mathsf{fix}(P)}$ is the operator from [16] used for characterizing the Fitting or Kripke-Kleene semantics of* $\mathsf{fix}(P)$.

We do not include details on this result here since we will need it only in passing in the sequel. The interested reader should consult [51]. A corollary from the result just mentioned is that the well-founded model of some given program P coincides with the Fitting model of $\mathsf{fix}(P)$.

3 Continuity

Theorem 1 enables us to carry over results on the single-step operator, respectively on the supported-model semantics, to the Gelfond-Lifschitz operator respectively the stable-model semantics. The following observation is of technical importance.

Proposition 1. *Let P be a definite program, $A \in B_P$, and $n \in \mathbb{N}$. Then $A \in T_P \uparrow n$ if and only if $A \leftarrow$ is a clause in $T'_P \uparrow n$.*

Proof. Let $A \in T_P \uparrow n$ for some $n \in \mathbb{N}$. We proceed by induction on n. If $n = 1$, then there is nothing to show. So assume that $n > 1$. Then there is a clause $A \leftarrow \texttt{body}$ in $\mathsf{ground}(P)$ such that all atoms B_i in $\texttt{body}$ are contained in $T_P \uparrow (n-1)$, and by induction hypothesis there are claues $B_i \leftarrow$ in $T'_P \uparrow (n-1)$. Unfolding these clauses with $A \leftarrow \texttt{body}$ shows that $A \leftarrow$ is also contained in $T'_P \uparrow n$.

Conversely, assume there is a clause $A \leftarrow$ in $T'_P \uparrow n$. We proceed again by induction. If $n = 1$, there is nothing to show. So let $n > 1$. Then there exists a clause $A \leftarrow A_1, \ldots, A_k$ in $\mathsf{ground}(P)$ and clauses $A_i \leftarrow$ in $T'_P \uparrow (n-1)$. By induction hypothesis, we obtain $A_i \in T_P \uparrow (n-1)$ for all i, and hence $A \in T_P \uparrow n$. $\square$

Since the single-step operator is not monotonic in general, several authors have made use of metric-based [17, 18, 22, 25–27, 29, 46] or even topological [3, 4, 22, 24, 43, 45, 47] methods for obtaining fixed-points and hence supported models of the programs in question. Central to these investigations is the Cantor topology Q on I_P, which was studied as the *query topology* in [4] and in more general terms as the *atomic topology* in [45]. It is the product topology on $\{\mathbf{t}, \mathbf{f}\}^{B_P}$, where the set of truth values $\{\mathbf{t}, \mathbf{f}\}$ is endowed with the discrete topology, and we refer to [53] for basic notions of topology. A subbase of the Cantor topology can be given as

$$\{\{I \in I_P \mid I \models L\} \mid L \text{ is a ground literal}\},$$

which was noted in [45]. We can now employ Theorem 1 to carry over some of these results to the treatment of the Gelfond-Lifschitz operator and the stable semantics.

Given a program P, we know by Theorem 1 that GL_P is continuous at some $I \in I_P$ in Q if and only if $T_{\mathsf{fix}(P)}$ is continuous at I. This gives rise to the following theorem.

Theorem 3. *Let P be a normal logic program and let $I \in I_P$. Then GL_P is continuous at I in Q if and only if whenever $\mathrm{GL}_P(I)(A) = \mathbf{f}$, then either there is no clause with head A in $\mathsf{ground}(P)$ or there exists a finite set $S(I, A) = \{A_1, \ldots, A_k\} \subseteq B_P$ such that $I(A_i) = \mathbf{t}$ for all i and for every clause $A \leftarrow \texttt{body}$ in $\mathsf{ground}(P)$ at least one $\neg A_i$ or some B with $\mathrm{GL}_P(I)(B) = \mathbf{f}$ occurs in $\texttt{body}$.*

Proof. The proof is based on the characterization of continuity of the T_P-operator given in [45], in the formulation which can be found in [29, Theorem 45], which reads as follows.

> *The single-step operator T_P is continuous in Q if and only if, for each $I \in I_P$ and for each $A \in B_P$ with $A \notin T_P(I)$, either there is no clause in P with head A or there is a finite set $S(I, A) = \{A_1, \dots, A_k, B_1, \dots, B_{k'}\}$ of elements of B_P with the following properties:*
> *(i) $A_1, \dots, A_k \in I$ and $B_1, \dots, B_{k'} \notin I$.*
> *(ii) Given any clause C with head A, at least one $\neg A_i$ or at least one B_j occurs in the body of C.*

Using this and Theorem 1, and by observing that there are no positive body atoms occuring in $\mathsf{fix}(P)$, we obtain the following:

> GL_P *is continuous at I if and only if whenever* $\mathrm{GL}_P(I)(A) = \mathbf{f}$*, then either there exists no clause with head A in* $\mathsf{fix}(P)$ *or there exists a finite set $S(I, A) = \{A_1, \dots, A_k\} \subseteq B_P$ such that $I(A_i) = \mathbf{t}$ for all i and for every clause $A \leftarrow$* $\mathtt{body}$ *in* $\mathsf{fix}(P)$ *at least one $\neg A_i$ occurs in* $\mathtt{body}$.

So let P be such that GL_P is continuous at I. If there is no clause with head A in $\mathsf{ground}(P)$, then there is nothing to show. So assume that there is a clause with head A in $\mathsf{ground}(P)$. We already know that then there exists a finite set $S(I, A) = \{A_1, \dots, A_k\} \subseteq B_P$ such that $I(A_i) = \mathbf{t}$ for all i and for every clause $A \leftarrow \mathtt{body}$ in $\mathsf{fix}(P)$ at least one $\neg A_i$ occurs in $\mathtt{body}$. Now let $A \leftarrow B_1, \dots, B_k, \neg C_1, \dots, \neg C_m$ be a clause in $\mathsf{ground}(P)$ and assume that no $\neg A_i$ occurs in its body. We show that there is some B_i with $\mathrm{GL}_P(I)(B_i) = \mathbf{f}$. Assume the contrary, i.e. that $\mathrm{GL}_P(I)(B_i) = \mathbf{t}$ for all i. Then for each B_i we have $B_i \in \mathrm{GL}_P(I) = T_{P/I} \uparrow \omega$. As in the proof of Proposition 1 we derive that there is a clause $A \leftarrow \neg D_1, \dots, \neg D_n, \neg C_1, \dots, \neg C_m$ in $\mathsf{fix}(P)$ with $D_j \notin I$ for all $j = 1, \dots, n$. Since the clause $A \leftarrow \neg D_1, \dots, \neg D_n, \neg C_1, \dots, \neg C_m$ is contained in $\mathsf{fix}(P)$, we know that some atom from the set $S(I, A)$ must occur in its body. It cannot occur as any D_i because $I(D_j) = \mathbf{f}$ for all i. It also cannot occur as any C_i by assumption. So we obtain a contradiction, which finishes the argument.

Conversely, let P be such that the condition on GL_P in the statement of the theorem holds. We will again make use of the observation made at the beginning of this proof. So let $A \in B_P$ with $\mathrm{GL}_P(I)(A) = \mathbf{f}$. If there is no clause with head A in $\mathsf{fix}(P)$, then there is nothing to show. So assume there is a clause with head A in $\mathsf{fix}(P)$. Then there is a clause with head A in P, and by assumption we know that there exists a finite set $S(I, A) = \{A_1, \dots, A_k\} \subseteq B_P$ such that $I(A_i) = \mathbf{t}$ for all i and for every clause $A \leftarrow \mathtt{body}$ in $\mathsf{ground}(P)$ at least one $\neg A_i$ or some B with $\mathrm{GL}_P(I)(B) = \mathbf{f}$ occurs in $\mathtt{body}$. Now let $A \leftarrow \neg B_1, \dots, \neg B_n$ be a clause in $\mathsf{fix}(P) = T'_P \uparrow \omega$, i.e. there is $k \in \mathbb{N}$ with $A \leftarrow \neg B_1, \dots, \neg B_n$ contained in $T'_P \uparrow k$. Note that $n = 0$ is impossible since this would imply $\mathrm{GL}_P(I)(A) = \mathbf{t}$ contradicting the assumption on A. We proceed by induction on k. If $k = 1$, then $A \leftarrow \neg B_1, \ldots, \neg B_n$ is contained in $\mathsf{ground}(P)$, hence one of the B_j is contained in $S(I, A)$ which suffices. For $k > 1$, there is a clause $A \leftarrow C_1, \ldots, C_m, \neg D_1, \ldots, \neg D_{m'}$ in $\mathsf{ground}(P)$ and clauses $C_i \leftarrow \mathtt{body}_i$ in $T'_P \uparrow (k-1)$ which unfold to $A \leftarrow \neg B_1, \ldots, \neg B_n$. By assumption we either have $D_j \in S(I, A)$ for some j, in which case there remains nothing to show, or we have that $\mathrm{GL}_P(I)(C_i) = \mathbf{f}$ for some i. In the latter case we obtain that $\mathtt{body}_i$ is non-empty by an argument similar to that of the proof of Proposition 1, so by assumption there is a (negated) atom in $\mathtt{body}_i$, and hence in $\{B_1, \ldots, B_n\}$, which is also in $S(I, A)$, which finishes the proof. $\square$

We can also observe the following special instance. A *local variable* is a variable occuring in some clause body but not in the corresponding head.

Corollary 1. *Let P be a normal program without local variables. Then GL_P is continuous in Q.*

Proof. We employ Theorem 3. Let $I \in I_P$ and $A \in B_P$ with $\mathrm{GL}_P(I)(A) = \mathbf{f}$. Since P has no local variables, it is of finite type. So the set $\mathcal{B}$ of all negated body atoms in clauses with head A is finite. Let $S(I, A) = \{B \in \mathcal{B} \mid I(B) = \mathbf{f}\}$, which is also finite. If each clause with head A contains some negated atom from $S(I, A)$, then there is nothing to show. So assume there is a clause $A \leftarrow A_1, \ldots, A_n, \neg B_1, \ldots, \neg B_m$ in $\mathsf{ground}(P)$ with $B_j \notin S(I, A)$ for all j, i.e. $I(B_j) = \mathbf{t}$ for all j. But then $A \leftarrow A_1, \ldots, A_n$ is a clause in P/I and $A \notin T_{P/I} \uparrow \omega$, which implies that there is some i with $A_i \notin T_{P/I} \uparrow \omega = \mathrm{GL}_P(I)$, which finishes the argument by Theorem 3. $\square$

Measurability is much simpler to deal with.

Theorem 4. *Let P be a normal program. Then GL_P is measurable with respect to the σ-algebra $\sigma(Q)$ generated by Q.*

Proof. By [28, Theorem 2], which states that T_P is measurable with respect to $\sigma(Q)$ for all P, we obtain that $T_{\mathsf{fix}(P)}$ is measurable with respect to $\sigma(Q)$, and by Theorem 1 we know that $T_{\mathsf{fix}(P)} \equiv \mathrm{GL}_P$. $\square$

4 Obtaining models

As already mentioned above, topological methods in logic programming can for example be used for obtaining models of programs iteratively, although the underlying operator is not monotonic. The following variant of [29, Theorem 44] can be proven directly.

Theorem 5. *Let P be a normal program and let GL_P be continuous and such that the sequence of iterates $\mathrm{GL}_P^m(I)$ converges in Q to some $M \in I_P$. Then M is a stable model of P.*

Proof. By continuity we obtain

$$M = \lim \mathrm{GL}_P^m(I) = \mathrm{GL}_P(\lim \mathrm{GL}_P^m(I)) = \mathrm{GL}_P(M).$$

□

We can also employ knowledge about relationships between the single-step operator and the Fitting operator [16]. The latter is defined on three-valued interpretations, which consist of sets of ground *literals* (instead of ground *atoms*) which do not contain complementary literals. As such, they carry set-inclusion as an ordering, which renders the space $I_{P,3}$ of all three-valued interpretations a complete partial order (cpo). It is in fact exactly the Plotkin domain $\mathbb{T}^\omega$ due to [41]. Alternatively, we can understand three-valued interpretations as mappings from atoms to the set $\{\mathbf{f}, \mathbf{u}, \mathbf{t}\}$ of truth values, where $\mathbf{u}$ stands for *undefined* or *undetermined*. The Fitting operator Φ_P, for given program P, is now defined as a function $\Phi_P : I_{P,3} \to I_{P,3} : I \mapsto t_P(I) \cup f_P(I)$, where $t_P(I)$ contains all $A \in B_P$ for which there exists a clause $A \leftarrow L_1, \ldots, L_n$ in $\mathsf{ground}(P)$ with $L_1, \ldots, L_n \in I$, and $f_P(I)$ contains all $\neg A$ such that for all clauses $A \leftarrow L_1, \ldots, L_n$ in $\mathsf{ground}(P)$ there is at least one $L_i \notin I$. It was shown in [16] that Φ_P is a monotonic operator on $I_{P,3}$.

If I is a three-valued interpretation, then I^+ denotes the two-valued interpretation assigning truth value $\mathbf{t}$ to exactly those atoms which are true in I.

Proposition 2. *Let P be a normal program and assume that the well-founded model M of P is total (i.e. every atom is true or false in it). Then $\mathrm{GL}_P^n(\emptyset)$ converges in Q to M^+, and M^+ is the unique stable model of P.*

Proof. This follows immeditately from Theorem 1 and [24, Theorem 4.4], which shows the following.

> *If $M = \Phi_R \uparrow \omega$ is total, then $T_R^n(\emptyset)$ converges in Q to M^+, and M^+ is the unique supported model of R.*

□

Metric-based approaches also carry over. A *level mapping* is a mapping from B_P to some ordinal α. A program P is *locally stratified* [44] if there exists a level mapping $l : B_P \to \alpha$, where α is some ordinal, such that for each clause $A \leftarrow A_1, \ldots, A_m, \neg B_1, \ldots, \neg B_n$ in $\mathsf{ground}(P)$ we have $l(A) \geq l(A_i)$ and $l(A) > l(B_j)$ for all i and j. It is called *locally hierarchical* [10], if additionally $l(A) > l(A_i)$ for all i. Given a level mapping $l : B_P \to \alpha$, we denote by Γ_l the set of all symbols $2^{-\beta}$ for $\beta \leq \alpha$, ordered by $2^{-\beta} < 2^{-\gamma}$ iff $\gamma < \beta$. Γ_l can be understood as a subset of the reals if $\alpha = \omega$, i.e. if l maps into the natural numbers. For two (two-valued) interpretations I and J, we define $d_l(I, J) = 2^{-\beta}$, where β is the least ordinal such that there is an atom of level β on which I and J disagree. If $\alpha = \omega$, then d_l is an ultrametric on I_P, and this construction was put to use e.g. in [17]. In the general case, d_l is a generalized ultrametric on I_P, as used in logic programming e.g. in [25, 29, 43]. A mapping f is called *strictly contracting* with respect to a generalized ultrametric d if $d(f(x), f(y)) < d(x, y)$ for all x, y with

$x \neq y$. Strictly contracting mappings have unique fixed points if the underlying generalized ultrametric space satisfies a completeness condition called *spherical completeness* [43].

Theorem 6. *Let P be locally stratified with corresponding level mapping l. Then GL_P is strictly contracting with respect to d_l, which is spherically complete. If l maps to ω, then GL_P is a contraction with respect to d_l. Furthermore, in both cases, GL_P has a unique fixed point and P has a unique stable model.*

Proof. If P is locally stratified with respect to l, then $\mathsf{fix}(P)$ is locally hierarchical with respect to l. It thus suffices to apply Theorem 1 in conjunction with Theorem [47, Theorem 3.8], which shows the following.

> *Let R be a normal logic program which is locally hierarchical with respect to a level mapping $l : B_R \to \gamma$. Then T_R is strictly contracting with respect to the generalized ultrametric d_l induced by l. Therefore, T_R has a unique fixed point and hence R has a unique supported model.*

□

With the remarks already made on the fact that the well-founded model of some given program P coincides with the Fitting model of $\mathsf{fix}(P)$, for any normal program P, we can also derive the following result. *Dislocated generalized ultrametric spaces* are defined by relaxing one of the defining conditions on generalized ultrametrics, for details see [29]. Strictly contracting mappings can be defined analogously, and have similar properties.

Theorem 7. *Let P be a program with total well-founded model $I \cup \neg(B_P \setminus I)$, with $I \subseteq B_P$. Then GL_P is strictly contracting on the spherically complete dislocated generalized ultrametric space (I_P, ϱ), where we have $\varrho(J, K) = \max\{d_l(J, I), d_l(I, K)\}$ for all $J, K \in I_P$, and l is defined by $l(A)$ to be the minimal α such that $\Phi_{\mathsf{fix}(P)} \uparrow (\alpha + 1)(A) = I(A)$.*

Proof. The program P has a total well-founded model, which implies that $\mathsf{fix}(P)$ has a total Fitting model. So l as given by the statement is well-defined, and $\mathsf{fix}(P)$ is *Φ-accessible* in the sense of [29]. Now apply [29, Proposition 41], which shows that T_P is strictly contracting for every Φ-accessible program. □

5 Neural-symbolic integration

Intelligent systems based on logic programming on the one hand, and on artificial neural networks (sometimes called connectionist sytems) on the other, differ substantially. Logic programs are highly recursive and well understood from the perspective of knowledge representation: The underlying language is that of first-order logic, which is symbolic in nature and makes it easy to encode problem specifications directly as programs. The success of artificial neural networks lies in the fact that they can be trained using raw data, and in some problem domains the generalization from the raw data made during the learning process turns out to be highly adequate for the problem at hand, even if the training data contains some noise. Successful architectures, however, often do not use recursive (or recurrent) structures. Furthermore, the knowledge encoded by a trained neural network is only very implicitly represented, and no satisfactory methods for extracting this knowledge in symbolic form are currently known.

It would be very desirable to combine the robust neural networking machinery with symbolic knowledge representation and reasoning paradigms like logic programming in such a way that the strenghts of either paradigm will be retained. Current state-of-the-art research, however, fails by far to achieve this ultimate goal. As one of the main obstacles to be overcome we perceive the question how symbolic knowledge can be encoded by artificial neural networks: Satisfactory answers to this will naturally lead the way to knowledge extraction algorithms and to hybrid neural-symbolic systems.

Earlier attempts to integrate logic and connectionist systems have mainly been restricted to propositional logic, or to first-order logic without function symbols. They go back to the pioneering work by McCulloch and Pitts [39], and have led to a number of systems developed in the 80s and 90s, including Towell and Shavlik's KBANN [49], Shastri's SHRUTI [48], the work by Pinkas [40], Hölldobler [30], and d'Avila Garcez et al. [12, 14], to mention a few, and we refer to [8, 13, 21] for comprehensive literature overviews.

Without the restriction to the finite case (including propositional logic and first-order logic without function symbols), the task becomes much harder due to the fact that the underlying language is infinite but shall be encoded using networks with a finite number of nodes. The sole approach known to us for overcoming this problem (apart from work on recursive autoassociative memory, RAAM, initiated by Pollack [42], which concerns the learning of recursive terms over a first-order language) is based on a proposal by Hölldobler et al. [32], spelled out first for the propositional case in [31], and reported also in [23]. It is based on the idea that logic programs can be represented — at least up to subsumption equivalence [38] — by their associated single-step operators. Such an operator can then be mapped to a function on the real numbers, which can under certain conditions in turn be encoded or approximated e.g. by feedforward networks with sigmoidal activation functions using an approximation theorem due to Funahashi [19].

We will carry over this result to the Gelfond-Lifschitz operator and the stable model semantics. Since the topology Q introduced earlier is homeomorphic to the Cantor topology on the real line [45], there exists a homeomorphism $\iota : I_P \to \mathcal{C}$, where $\mathcal{C}$ is the Cantor set within the unit interval, endowed with the subspace topology inherited from the reals. We can thus embed any function $f : I_P \to I_P$ which is continuous in Q as a continuous function $\iota(f) : \mathcal{C} \to \mathcal{C} : \iota(f)(x) = \iota(f(\iota^{-1}(x)))$. By well-known results, e.g. [19] as mentioned earlier, such functions can be approximated uniformly by artificial neural networks in many different network architectures.

Theorem 8. *Let P be a normal logic program. Then GL_P can be approximated almost everywhere up to an arbitrarily chosen error bound by input-output functions of three-layer feedforward neural networks with sigmoidal activation functions. If GL_P is furthermore continuous in Q, then uniform approximation is possible on all of $\mathcal{C}$.*

Proof. We use Theorem 1. The first statement then follows from Theorem 4 together with a result from [33] saying that each measurable function can be approximated almost everywhere by three-layer feedforward networks in the indicated way — see also [28, Theorem 7]. The second statement follows from [19] or from [28, Theorem 5]. □

The references mentioned in the proof of Theorem 8 provide further results, in particular on error bounds, and they can also be carried over straightforwardly.

Another improvement on the basic results by Hölldobler et al [32] employed an alternative network architecture. In [2], results were provided for encoding and approximating $\iota(T_P)$ by iterated function systems, which in turn could be encoded using a recurrent neural networks structure. The advantage of this approach is that algorithms for constructing approximating networks can be given explicitly, in contrast to the results in [23, 28, 32]. These results also hinge on continuity or Lipschitz-continuity of $\iota(T_P)$ with respect to the Cantor topology only, and can be carried over to the Gelfond-Lifschitz operator in a straightforward way. The paper [5] provides related results using cellular automata, treating logic programs without local variables — a property which also carries over to the fixpoint completion. Hence these results carry over *mutatis mutandis* to the Gelfond-Lifschitz operator.

6 Conclusions

We have displayed the usefulness of the results reported in [51] to the operator-based analysis of knowledge representation under the stable semantics. We have shown that many results from the study of the supported-model semantics by means of the single-step operator can be carried over to the stable semantics almost without effort.

Our results are of a theoretical nature, and we do not propose to study them for implementation purposes. The idea to use the fixpoint completion for obtaining stable models (or similar constructions for obtaining answer sets or well-founded models etc.) of programs is already folklore knowledge in the community, and need not be further mentioned. The emphasis of our exhibition is on the observation that not only *models*, but also corresponding *semantic operators* are related by means of the fixpoint completion, and on the aspects which this new insight allows to study.

Our observations are valid for first-order languages including function symbols, a syntax whose study is often neglected in the non-monotonic reasoning community. It is not at all surprising, that for *finite* languages alternative methods of program transformation can be found, which allow for efficient computation of stable models [34–36].

References

1. Krzysztof R. Apt, Howard A. Blair, and Adrian Walker. Towards a theory of declarative knowledge. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 89–148. Morgan Kaufmann, Los Altos, CA, 1988.
2. Sebastian Bader and Pascal Hitzler. Logic programs, iterated function systems, and recurrent radial basis function networks. *Journal of Applied Logic*, 200x. To appear.
3. Aida Batarekh and V.S. Subrahmanian. The query topology in logic programming. In *Proceedings of the 1989 Symposium on Theoretical Aspects of Computer Science*, volume 349 of *Lecture Notes in Computer Science*, pages 375–387. Springer, Berlin, 1989.
4. Aida Batarekh and V.S. Subrahmanian. Topological model set deformations in logic programming. *Fundamenta Informaticae*, 12:357–400, 1989.
5. Howard A. Blair, Jagan Chidella, Fred Dushin, Audrey Ferry, and Polar Humenn. A continuum of discrete systems. *Annals of Mathematics and Artificial Intelligence*, pages 153–186, 1997.
6. Stefan Bonnier, Ulf Nilsson, and Torbjörn Näslund. A simple fixed point characterization of three-valued stable model semantics. *Information Processing Letters*, 40(2):73–78, 1991.
7. Stefan Brass, Jürgen Dix, Burkhardt Freitag, and Ulrich Zukowski. Transformation-based bottom-up computation of the well-founded model. *Theory and Practice of Logic Programming*, 1(5):497–538, 2001.
8. Anthony Browne and Ron Sun. Connectionist inference models. *Neural Networks*, 14(10):1331–1355, 2001.
9. François Bry. Negation in logic programming: A formalization in constructive logic. In Dimitris Karagiannis, editor, *Information Systems and Artificial Intelligence: Integration Aspects, First Workshop, Ulm, FRG, March 19-21, 1990, Proceedings*, volume 474 of *Lecture Notes in Computer Science*, pages 30–46. Springer, 1991.
10. Lawrence Cavedon. Acyclic programs and the completeness of SLDNF-resolution. *Theoretical Computer Science*, 86:81–92, 1991.
11. Keith L. Clark. Negation as failure. In Hervé Gallaire and Jack Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, New York, 1978.
12. Artur S. d'Avila Garcez, Krysia Broda, and Dov M. Gabbay. Symbolic knowledge extraction from trained neural networks: A sound approach. *Artificial Intelligence*, 125:155–207, 2001.
13. Artur S. d'Avila Garcez, Krysia B. Broda, and Dov M. Gabbay. *Neural-Symbolic Learning Systems — Foundations and Applications.* Perspectives in Neural Computing. Springer, Berlin, 2002.
14. Artur S. d'Avila Garcez and Gerson Zaverucha. The connectionist inductive lerarning and logic programming system. *Applied Intelligence, Special Issue on Neural networks and Structured Knowledge*, 11(1):59–77, 1999.
15. Phan Minh Dung and Kanchana Kanchanasut. A fixpoint approach to declarative semantics of logic programs. In Ewing L. Lusk and Ross A. Overbeek, editors, *Logic Programming, Proceedings of the North American Conference 1989, NACLP'89, Cleveland, Ohio*, pages 604–625. MIT Press, 1989.
16. Melvin Fitting. A Kripke-Kleene-semantics for general logic programs. *The Journal of Logic Programming*, 2:295–312, 1985.
17. Melvin Fitting. Metric methods: Three examples and a theorem. *The Journal of Logic Programming*, 21(3):113–127, 1994.

18. Melvin Fitting. Fixpoint semantics for logic programming — A survey. *Theoretical Computer Science*, 278(1–2):25–51, 2002.
19. Ken-Ichi Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2:183–192, 1989.
20. Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert A. Kowalski and Kenneth A. Bowen, editors, *Logic Programming. Proceedings of the 5th International Conference and Symposium on Logic Programming*, pages 1070–1080. MIT Press, 1988.
21. Hans W. Güsgen and Steffen Hölldobler. Connectionist inference systems. In Bertram Fronhöfer and Graham Wrightson, editors, *Parallelization in Inference Systems*, volume 590 of *Lecture Notes in Artificial Intelligence*, pages 82–120. Springer, Berlin, 1992.
22. Pascal Hitzler. *Generalized Metrics and Topology in Logic Programming Semantics.* PhD thesis, Department of Mathematics, National University of Ireland, University College Cork, 2001.
23. Pascal Hitzler, Steffen Hölldobler, and Anthony K. Seda. Logic programs and connectionist networks. *Journal of Applied Logic*, 200x. To appear.
24. Pascal Hitzler and Anthony K. Seda. Acceptable programs revisited. In *Proceedings of the Workshop on Verification in Logic Programming, 16th Int. Conf. on Logic Programming, ICLP'99, Las Cruces, New Mexico*, volume 30 of *Electronic Notes in Theoretical Computer Science*, pages 1–18. Elsevier, 1999.
25. Pascal Hitzler and Anthony K. Seda. The fixed-point theorems of Priess-Crampe and Ribenboim in logic programming. In *Valuation Theory and its Applications, Proceedings of the 1999 Valuation Theory Conference, University of Saskatchewan in Saskatoon, Canada*, volume 32 of *Fields Institute Communications Series*, pages 219–235. American Mathematical Society, 1999.
26. Pascal Hitzler and Anthony K. Seda. Some issues concerning fixed points in computational logic: Quasi-metrics, multivalued mappings and the Knaster-Tarski theorem. In *Proceedings of the 14th Summer Conference on Topology and its Applications: Special Session on Topology in Computer Science, New York*, volume 24 of *Topology Proceedings*, pages 223–250, 1999.
27. Pascal Hitzler and Anthony K. Seda. A new fixed-point theorem for logic programming semantics. In *Proceedings of the joint IIIS & IEEE meeting of the 4th World Multiconference on Systemics, Cybernetics and Informatics, SCI2000, and the 6th International Conference on Information Systems Analysis and Synthesis, ISAS2000, Orlando, Florida, USA*, volume VII, Computer Science and Engineering Part 1, pages 418–423. International Institute of Informatics and Systemics: IIIS, 2000.
28. Pascal Hitzler and Anthony K. Seda. Continuity of semantic operators in logic programming and their approximation by artificial neural networks. In Andreas Günter, Rudolf Krause, and Bernd Neumann, editors, *Proceedings of the 26th German Conference on Artificial Intelligence, KI2003*, volume 2821 of *Lecture Notes in Artificial Intelligence*, pages 105–119. Springer, 2003.
29. Pascal Hitzler and Anthony K. Seda. Generalized metrics and uniquely determined logic programs. *Theoretical Computer Science*, 305(1–3):187–219, 2003.
30. Steffen Hölldobler. *Automated Inferencing and Connectionist Models.* Fakultät Informatik, Technische Hochschule Darmstadt, 1993. Habilitationsschrift.
31. Steffen Hölldobler and Yvonne Kalinke. Towards a massively parallel computational model for logic programming. In *Proceedings ECAI94 Workshop on Combining Symbolic and Connectionist Processing*, pages 68–77. ECCAI, 1994.

32. Steffen Hölldobler, Yvonne Kalinke, and Hans-Peter Störr. Approximating the semantics of logic programs by recurrent neural networks. *Applied Intelligence*, 11:45–58, 1999.
33. Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.
34. Joohyung Lee and Vladimir Lifschitz. Loop formulas for disjunctive logic programs. In Catuscia Palamidessi, editor, *Logic Programming, 19th International Conference, ICLP 2003, Mumbai, India, December 2003, Proceedings*, volume 2916 of *Lecture Notes in Computer Science*, pages 451–465. Springer, 2003.
35. Fangzhen Lin and Jicheng Zhao. On tight logic programs and yet another translation from normal logic programs to propositional logic. In Georg Gottlob and Toby Walsh, editors, *Proceedings of the 18th International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 2003*, pages 853–858. Morgan Kaufmann Publishers, 2003.
36. Fangzhen Lin and Yiting Zhao. ASSAT: Computing answer sets of a logic program by SAT solvers. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence, July/August, 2002, Edmonton, Alberta, Canada*, pages 112–118. AAAI Press, 2002.
37. John W. Lloyd. *Foundations of Logic Programming.* Springer, Berlin, 1988.
38. Michael J. Maher. Equivalences of logic programs. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 627–658. Morgan Kaufmann, Los Altos, CA, 1988.
39. Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.
40. Gadi Pinkas. Propositional non-monotonic reasoning and inconsistency in symmetric neural networks. In John Mylopoulos and Raymond Reiter, editors, *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 525–530. Morgan Kaufmann, 1991.
41. Gordon Plotkin. T^ω as a universal domain. *Journal of Computer and System Sciences*, 17:209–236, 1978.
42. Jordan B. Pollack. Recursive distributed representations. *Artificial Intelligence*, 46(1):77–105, 1990.
43. Sibylla Prieß-Crampe and Paolo Ribenboim. Ultrametric spaces and logic programming. *The Journal of Logic Programming*, 42:59–70, 2000.
44. Teodor C. Przymusinski. On the declarative semantics of deductive databases and logic programs. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 193–216. Morgan Kaufmann, Los Altos, CA, 1988.
45. Anthony K. Seda. Topology and the semantics of logic programs. *Fundamenta Informaticae*, 24(4):359–386, 1995.
46. Anthony K. Seda. Quasi-metrics and the semantics of logic programs. *Fundamenta Informaticae*, 29(1):97–117, 1997.
47. Anthony K. Seda and Pascal Hitzler. Topology and iterates in computational logic. In *Proceedings of the 12th Summer Conference on Topology and its Applications: Special Session on Topology in Computer Science, Ontario, August 1997*, volume 22 of *Topology Proceedings*, pages 427–469, 1997.
48. Lokenda Shastri. Advances in Shruti — A neurally motivated model of relational knowledge representation and rapid inference using temporal synchrony. *Applied Intelligence*, 11:78–108, 1999.
49. Geoffrey G. Towell and Jude W. Shavlik. Knowledge-based artificial neural networks. *Artificial Intelligence*, 70(1–2):119–165, 1994.

50. Allen van Gelder, Kenneth A. Ross, and John S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.
51. Matthias Wendt. Unfolding the well-founded semantics. *Journal of Electrical Engineering, Slovak Academy of Sciences*, 53(12/s):56–59, 2002. (Proceedings of the 4th Slovakian Student Conference in Applied Mathematics, Bratislava, April 2002)[2].
52. Matthias Wendt. Unfolding the well-founded semantics. Technical Report WV–02–08, Knowledge Representation and Reasoning Group, Department of Computer Science, Dresden University of Technology, 2002. http://www.wv.inf.tu-dresden.de/Publications/2002/.
53. Stephen Willard. *General Topology.* Addison-Wesley, Reading, MA, 1970.

[2] Available online as [52].

In: L. Li and K.K. Yen, Proceedings of the Third International Conference on Information, Tokyo, Japan, November/December 2004. ISBN 4-901329-02-2, International Information Institute, pp. 22-33.

The Integration of Connectionism and First-Order Knowledge Representation and Reasoning as a Challenge for Artificial Intelligence

Sebastian Bader[1], Pascal Hitzler[2], Steffen Hölldobler[1]

[1]International Center for Computational Logic
Technische Universität Dresden, Germany
[2]AIFB, Universität Karlsruhe, Germany

Abstract

Intelligent systems based on first-order logic on the one hand, and on artificial neural networks (also called connectionist systems) on the other, differ substantially. It would be very desirable to combine the robust neural networking machinery with symbolic knowledge representation and reasoning paradigms like logic programming in such a way that the strengths of either paradigm will be retained. Current state-of-the-art research, however, fails by far to achieve this ultimate goal. As one of the main obstacles to be overcome we perceive the question how symbolic knowledge can be encoded by means of connectionist systems: Satisfactory answers to this will naturally lead the way to knowledge extraction algorithms and to integrated neural-symbolic systems.

1 Introduction

Artificial neural networks — also called *connectionist systems* — exhibit many desirable properties of intelligent systems like, for example, being massively parallel, context-sensitive, adaptable and robust (see eg. [14]). It is strongly believed that intelligent systems must also be able to represent and reason about structured objects and structure-sensitive processes (see eg. [16, 35]). Unfortunately, we are unaware of any connectionist system which can handle structured objects and structure-sensitive processes in a satisfying way. Logic systems were designed to cope with such objects and processes and, consequently, it is a long-standing research goal to combine the advantages of connectionist and logic systems in a single system.

Earlier attempts to integrate logic and connectionist systems have mainly been restricted to propositional logic, or to first-order logic without function symbols. They go back to the pioneering work by McCulloch and Pitts [34], and have led to a number of systems developed in the 80s and 90s, including Towell and Shavlik's KBANN [45], Shastri and Ajjanagadde's SHRUTI [43], Lange and Dryer's ROBIN [32] the work by Pinkas [37], Hölldobler [22], and d'Avila Garcez et al. [10, 13], to mention a few, and we refer to [9, 11] for comprehensive literature overviews.

Without the restriction to the finite case (including propositional logic and first-order logic without function symbols), the task becomes much harder due to the fact that the underlying language is infinite but shall be encoded using networks with a finite number of nodes. One of the few approaches for overcoming this problem (apart from work on recursive autoassociative memory, RAAM, initiated by Pollack [40], which concerns the learning of recursive terms over a first-order language) is based on a proposal by Hölldobler et al. [27], and reported also in [18]. It is based on the idea that logic programs can be represented by their associated single-step or immediate consequence operators. Such an operator can then be mapped to a function on the real numbers, which can under certain conditions in turn be encoded or approximated e.g. by feedforward networks with sigmoidal activation functions.

The purpose of this paper is twofold. First, we will give an overview of recent progress made in the representation of first-order logic programs by connectionist systems (Section 2). We will then discuss in detail some questions which we find of central importance in order to advance towards an integration of logic and connectionism (Section 3). Our selections are certainly very subjective, so we also provide ample references to related work and literature for further reading. Our discussions will be in very general terms, and we will make most of our general exhibition accessible to the general reader. Some familiarity with basic notions from logic and artificial neural networks, and also from set-theoretic topology and iterated function systems will however be helpful for understanding some of the details. As general references we recommend [33, 7, 47, 4].

Acknowledgements. The first author is supported by the GK334 of the German Research Foundation. The second author is supported by the German Federal Ministry of Education and Research under the SmartWeb project and the EU Network of Excellence KnowledgeWeb. The second and third author acknowledge substantial support by the Boole Centre for Research in Informatics at the National University of Ireland, Cork, for presenting this paper.

2 Recent Progress

Integrating first-order logical knowledge representation and connectionism necessitates to find a common framework in which both kinds of systems can be expressed and somehow unified.

Logical knowledge representation is symbolic in nature, i.e. the data structures under consideration basically consist of words over some language or of collections of finite trees, for example, depending on the perspective taken or on the problem at hand. Logic programs, more specifically, consist of sets of first-order formulae under a restricted syntax, more precisely, a logic program is a set of (universally quantified) disjunctions, called *clauses* or *rules*, which in turn consist of atoms and negated atoms only. Equivalently, one can say that logic programs are basically formulae in conjunctive normal form — although their meaning, i.e. the way they are evaluated, is not identical to their meaning in first-order logic. Input (queries) and output (answers) of a logic program essentially consist of certain logical formulae or of models of the program.

Successful connectionist architectures, however, can be understood as networks (essentially, directed graphs) of simple computational units, in which activation is propagated and combined in certain ways adhering to connectionist principles. In many cases like, for example, in multilayer perceptrons, the activation is encoded as a real number; input and output of such networks consist of tuples (vectors) of real numbers. So, while logic is *symbolic* and thus *discrete*, standard connectionist systems are *continuous*, i.e. they deal with real values in Euclidean space.

In order to integrate logic and connectionism we thus need to bridge the gap between the discrete, symbolic setting of logic, and the continuous, real-valued setting of artificial neural networks. The method of our choice — motivated by several reasons which will become clear below — is to employ *Cantor space* for this purpose.

Cantor space $\mathcal{C}$ is — up to homeomorphism — a subset of the unit interval of the real numbers endowed with the topological structure inherited from the reals. The set is best described as the set of all real numbers in the unit interval which can be expressed in the ternary number system using the digits 0 and 2 only. More precisely, $\mathcal{C}$ is the set of all real numbers of the form $\sum_{i=1}^{\infty} a_i 3^{-i}$, where $a_i \in \{0, 2\}$ for all i. We remark that topologically, we obtain homeomorphic subsets of the reals by considering all real numbers of the form $\sum_{i=1}^{\infty} a_i B^{-i}$, where $a_i \in \{0, 1\}$ and B is fixed to some natural number greater than or equal to 3. This lies in the fact that Cantor space can be uniquely described — up to homeomorphism — as the topological space which is totally disconnected, compact, Hausdorff, second countable, and dense in itself.

How do we relate Cantor space to first-order logic? The topological characterization of $\mathcal{C}$ just given already shows that it can be described independently of the real numbers. Now consider some first-order language $\mathcal{L}$. Interpretations (or valuations) over $\mathcal{L}$ can be understood as mappings from the countable set of ground atoms over $\mathcal{L}$ — which we call the *Herbrand base* $B_{\mathcal{L}}$ over $\mathcal{L}$ — to the set of truth values $\{\mathbf{t}, \mathbf{f}\}$. Identifying $\mathbf{t}$ with 2 (or 1) and $\mathbf{f}$ with 0, the set of all interpretations over $\mathcal{L}$ can be identified with the set of all mappings from $B_{\mathcal{L}}$ to $\{0, 2\}$. Since $B_{\mathcal{L}}$ is countable, we can also choose an enumeration of $B_{\mathcal{L}}$, which is essentially an identification of $B_{\mathcal{L}}$ with $\mathbb{N}$, the set of natural numbers excluding zero, or, in other words, a bijective mapping $l : B_{\mathcal{L}} \to \mathbb{N}$. We can thus identify the set of all interpretations over $\mathcal{L}$, which are of the form $I : B_{\mathcal{L}} \to \{\mathbf{t}, \mathbf{f}\}$, with the set of all mappings $f : \mathbb{N} \to \{0, 2\}$.

Now, formally, let $l : B_{\mathcal{L}} \to \mathbb{N}$ be an (arbitrarily chosen) bijection and let $I_{\mathcal{L}}$ be the set of all interpretations over $\mathcal{L}$, i.e. the set of all mappings from $B_{\mathcal{L}}$ to $\{0, 2\}$. We define a mapping ι from $I_{\mathcal{L}}$ to $\mathcal{C}$ by

$$\iota(I) = \sum_{i=1}^{\infty} I(l^{-1}(i)) 3^{-i}.$$

It is easily verified that ι is a bijection.

The mapping ι allows to understand the set of interpretations as the Cantor set in the real line. But does it also preserve meaningful structure, i.e. does it relate meaningful structure for logic programs on the one side with meaningful structure for connectionist sytems on the other side? We will see that it does, and in order to proceed, we reproduce next a theorem due to Funahashi [17].

Theorem 1 *Suppose that $\phi : \mathbb{R} \to \mathbb{R}$ is non-constant, bounded, monotone increasing and continuous. Let $K \subseteq \mathbb{R}^n$ be compact, let $f : K \to \mathbb{R}$ be a continuous mapping and let $\varepsilon > 0$. Then there exists a 3-layer feedforward network with squashing function ϕ whose input-output mapping $\bar{f} : K \to \mathbb{R}$ satisfies* $\max_{x \in K} d(f(x), \bar{f}(x)) < \varepsilon$*, where d is a metric which induces the natural topology on $\mathbb{R}$.*

For the reader who is not familiar with the terminology of the theorem, we state its intuitive meaning: Every continuous real-valued function defined on a compact subset of the reals can be uniformly approximated by input-output mappings of artificial neural networks of a certain architecture. The details of this architecture will not concern us for our general discussion.

Funahashi's theorem provides an existence result for approximating continous functions on the reals. So if we manage to interpret logic programs as such functions in a meaningful way, then we know that approximation of logic programs by neural networks is possible in a reasonable way. We need two more steps in order to realize this idea.

Firstly, we note that it is very common in logic programming to associate operators to logic programs in such a way that the behaviour of the operator reflects the meaning of the program. One of the most popular — and arguably the most natural — operator is the so-called *immediate consequence operator* T_P associated with a given program P. Details of the definition of T_P will be of no concern for our general discussion, so we will not spell them out. For the same reason, we also omit a formal definition of a logic program, and just remark that logic programs are certain sets of first-order logical formulae, as already mentioned. The operator T_P is an operator which acts on $I_{\mathcal{L}}$, i.e. on the space of all interpretations of the underlying first-order language. Since ι maps $I_{\mathcal{L}}$ bijectively onto $\mathcal{C}$, we can carry over the operator T_P via ι to the reals, by defining

$$\iota(T_P) : \mathcal{C} \to \mathcal{C} : x \mapsto \iota(T_P(\iota^{-1}(x))).$$

Hence, $\iota(T_P)$ is a mapping on Cantor set which carries the meaning of P.

Secondly, we need to ensure that the embedded mapping $\iota(T_P)$ just defined is continuous on Cantor space, such that Funahashi's theorem can be applied. This, for example, is the case if T_P is continuous with respect to the initial topology induced by ι on $I_{\mathcal{L}}$ — let us denote this topology by Q. Since ι is a bijection, it follows that it is a homeomorphism from $(I_{\mathcal{L}}, Q)$ to Cantor space $\mathcal{C}$ — i.e. $(I_{\mathcal{L}}, Q)$ *is* Cantor space up to homeomorphism, and $\iota(T_P)$ is continuous as a function on $\mathcal{C}$ if and only if T_P is continuous as a function on $(I_{\mathcal{L}}, Q)$. Together, we obtain the following result, which was reported in [18] in a more general form.

Theorem 2 *Let P be a logic program such that T_P is continuous in Q, and let ι be a homeomorphism from $(I_{\mathcal{L}}, Q)$ to $\mathcal{C}$. Then $\iota(T_P)$ can be approximated uniformly by input-output functions of artificial neural networks of the kind used in Theorem 1.*

The importance of Theorem 2 lies in the fact that the topology Q on $I_{\mathcal{L}}$ is well-known in logic programming. Indeed, it is the most important topology for the study of fixed-point semantics for programs with negation. It dates back to the work by Batarekh and Subrahmanian [5] where it was called the *query topology*. Seda [42] studies a generalization of it under the name *atomic topology*, and in the same paper it was also shown that continuity in Q can naturally be characterized without making reference to topological notions. It is also strongly related to the studies of fixed-point semantics of logic programs by means of generalized metrics, as e.g. undertaken by Fitting [15], Prieß-Crampe & Ribenboim [41] and Seda & Hitzler [20].

Due to its very general nature, Theorem 2 carries a lot of inherent flexibility. The particular instance of ι given earlier is only one very specific example of a homeomorphism which can be used. Indeed, the number of automorphisms of Cantor space is uncountable. The specific representations of Cantor space as a subspace of the reals given earlier are also just very particular examples of such representations. Results analogous to that by Funahashi furthermore hold for many popular neural network architectures, such that our investigations are not a priori restricted to certain types of connectionist systems.

But the flexibility gained by the general nature of Theorem 2 does not come for free. In particular, it provides no means of actually obtaining an approximating network from a concretely given program. At best, we would like to be able to read off parameters for an approximating network directly from a given program. To date, it is an open problem how to do this along the lines of Theorem 2.

A different approach towards obtaining concrete approximations was undertaken by us in [2]. It was based on the observation that graphs of embedded operators $\iota(T_P)$, displayed in the Euclidean plane, exhibit self-similar structures known from chaos theory. More precisely, the graphs appeared to be attractors of iterated function systems as studied, for example, in the well-known book by Barnsley [4]. This led to the following theorem, which is stated in slightly more general form in [2].

Theorem 3 *Let P be a logic program such that $\iota(T_P)$ is Lipschitz-continuous. Then there exists an iterated function system on the Euclidean plane whose attractor is the graph of $\iota(T_P)$.*

The importance of Theorem 3 lies in the fact that iterated function systems can be encoded very easily as some standard type of recurrent neural networks, and we have spelled this out in [2]. The very general Theorem 3 also leads to concrete instances of iterated function systems — and thus of corresponding networks — for approximating $\iota(T_P)$: Given a program P and an arbitrarily chosen accuracy of the approximation $i \in \mathbb{N}$, we need only determine a finite number of explicitly determined function values of $\iota(T_P)$, in order to arrive at an iterated function system $\mathcal{S}_i$ whose attractor f_i is the graph of a continuous function — details of the construction can be found in [2]. Our result now reads as follows.

Theorem 4 *Let P be a program with Lipschitz-continuous $\iota(T_P)$. Then the sequence $(f_i)_{i\in\mathbb{N}}$ of attractors, as mentioned above, converges uniformly to $\iota(T_P)$.*

A concrete open problem remaining with Theorem 4 is that the determination of a suitable iterated function system $\mathcal{S}_i$ from a given program P hinges on the explicit knowledge about an upper bound for the Lipschitz-constant for $\iota(T_P)$ — if it exists at all.

We close our brief survey with a number of further remarks.

(1) The idea to represent a logic program via its semantic operator traces back to Hölldobler & Kalinke [23], surveyed in [18], where this idea was employed for the propositional case. D'Avila Garcez et al. [10, 13] have molded this into an integrated learning system which uses backpropagation.

(2) A very restricted version of Theorem 2 was shown in [27] using different methods. There, and in [18], the network architecture was also extended in order

to mimic iterations of the immediate consequence operator, and corresponding results on the convergence behaviour and speed of these iterations were provided.

(3) It was shown in [18] that many semantic operators in logic programming, including the immediate consequence operator, are measurable. While there exist approximation results relating measurability to artificial neural networks, e.g. by Hornik et al. [29], it is an open issue whether this fact can be exploited for neural-symbolic integration.

(4) A recent result by Wendt [46] relates semantic operators used in answer set programming [44] to the immediate consequence operator, and thus allows to use our results for studying non-monotonic reasoning with logic programs in a connectionist setting. This remains to be spelled out. Some preliminary investigations can be found in [19].

(5) We are recently investigating the use of weighted automata and fibring neural networks for our purposes [3, 12].

After this survey on the current state of the art of relating logic programs and connectionist networks we will identify a number of open research problems in the following section.

3 Challenges

3.1 How can first-order terms be represented and manipulated in a connectionist system?

This is the main question that needs to be answered, and our recent results presented in the previous section are along this line. We consider this question to be of central importance because the development of a satisfactory and usable representation of first-order formulae is the first necessary step towards neural-symbolic integration. The proposals made so far do not give a satisfying answer to this question: Structured connectionist networks as used e.g. in [21] are completely local. The unification and matching operations can directly be implemented in these networks. However, the number of units is quadratic and the number of connections even cubic wrt the size of the terms. It is not obvious at all how such networks can be learned.

Vectors of fixed length are used to represent terms in the recursive auto-associative memory and its derivatives [39, 1]. Unfortunately, in extensive tests none of these proposals has led to satisfying results: The systems could not safely store and recall terms of depth larger than five [30].

In hybrid systems terms are represented and manipulated in a conventional way. But this is not a kind of integration we are hoping for because in this case results from connectionist systems cannot be applied to the conventional part.

The phase-coding mechanism suggested in SHRUTI [43] and used in the BUR system [25] restricts the first-order language to contain only constants and multi-place relation symbols.

We definitely need new ideas to solve this challenge problem! Connectionist encodings for conventional data structures like counters and stacks [24, 31] have been proposed and may be of use, and the study of relationships between logic programs, neural networks, and other paradigms in computing and mathematics like cellular and weighted automata, dynamical systems, and the like, may provide new ideas.

3.2 How can first-order rules be extracted from a connectionist network?

To the best of our knowledge all rule extraction techniques for connectionist networks are propositional in the sense that they only generate propositional rules. For example, the propositional networks in [23] were slightly modified in [13, 10] such that backpropagation could be applied and standard rule extraction techniques could be used to extract new revised — but propositional — rules.

The results from [18] guarantee the existence of recurrent networks with a feed forward kernel to approximate the meaning of a first-order program. Backpropagation can again in principle be used to train these kernels. But it is by no means obvious how the rule extraction techniques known so far can be generalized such that first-order rules are extracted from these kernels.

3.3 How can distributed knowledge representation in connectionist networks be understood from a symbolic perspective?

Although this question is being subsumed by the previous two, we want to emphasize the difficulties underlying distributed representations explicitly. The representation of propositional logic in connectionist networks most often is very local, while standard network training normally leads to distributed representation, which is very difficult to interpret in a symbolic manner.

The situation becomes worse for first-order logic, where due to the infinitary nature of the underlying language there seems to be no way at all to avoid distributed representation. We understand that this issue provides the main obstacle in developing constructive methods for the representation of first-order logic programs by means of Funahashi's theorem, and we also expect this to be a major issue in order to make advances in first-order rule extraction.

3.4 How can established learning algorithms like backpropagation be combined with symbolic knowledge representation?

For the propositional system developed by d'Avila Garcez et al. [10, 13], symbolic knowledge is being represented by a network, which is then trained using backpropagation. Afterwards, the learned knowledge may be extracted. A similiar approach underlies the Kbann system due to Towell and Shavlik [45].

While this is a good idea, we see the risk that the initial knowledge may be lost in the course of the training process, although it should rather influence it. We envision an integration via continuous interaction of standard learning techniques with background and dynamically acquired knowledge. How this can be achieved, however, is as yet entirely unclear.

Another problem is posed by the fact that the representation of first-order knowledge easily leads to non-standard network architectures, like in the Shruti system [43], which cannot be trained easily, or at least cannot be trained with established methods without loosing the specific logically meaningful architecture. The latter would be the case e.g. with the recurrent networks obtained from iterated function systems as mentioned in Section 2. Modified learning algorithms will have to be established and studied for these purposes.

3.5 How can multiple instances of first-order rules be represented in a connectionist system?

One of the properties of first-order reasoning is that it cannot be determined in advance how many copies of a rule are needed to answer a given query or, equivalently, to prove a theorem. In local connectionist systems like Chcl [26] this problem is defined away by simply assuming that each rule is used only once. A similar assumption is made in Shruti, where each relation may be instantiated only a fixed number of times in one reasoning episode. This is not a general solution since even for datalogic programs — which do not need function symbols in the underlying language — exponentially many copies may be needed. The Bur system from [25] does not provide multiple copies, which is the reason for the fact that the system may be unsound if multiplace relation symbols are involved.

The results from [18] suggest that the problem of generating new instances of a rule can be mapped on the problem of obtaining a better approximation of the least model of a logic problem in the following sense: The level assigned to ground atoms occurring in the n-th iteration of the meaning function for the first time should be higher than the level assigned to ground atoms which occur at earlier stages. If the accuracy of an approximation can then be correlated to the available hardware resources as in [24], we might obtain a solution for this challenge problem.

3.6 How can insights from neuroscience be used to design integrated systems which are biologically feasible?

Artificial neural networks are very coarse abstractions of biological networks. Connectionist networks used for the study of neural-symbolic integration, however, are often biologically much less feasible than standard architectures like multilayer perceptrons. While it is important to study the formal relationships between first-order logic and connectionist systems, we believe that it is also important to study biological networks from the perspective of symbolic knowledge processing. Can the accumulation of electric potential within a dendritic tree be understood from a logial perspective? Can we develop methods to understand the temporal aspects of different transmission times between different neurons? Can we assign logical meaning to firing patterns of collections of neurons? Interdisciplinary efforts are required to answer these questions!

3.7 What is the exact relationship between neural-symbolic integration and chaos theory? Can this be exploited?

This question is prompted not only by our results reported in Section 2, but also by work by Blair et al. on the relationship between cellular automata, topological dynamics, logic programming, and other paradigms related to chaos theory [8]. The structural coincidences are striking, but research in this direction is difficult due to the fact that the related paradigms all turn out to be equally hard to study, and advances will most likely necessitate entirely new ideas for approaching these issues.

3.8 What does a theory for the integration of logic and connectionist systems look like?

The results achieved so far on connectionist inference systems are more or less unrelated to each other. Different logics are mapped onto different connectionist systems and very often not much effort has been spent on (i) formally showing properties of the system, (ii) formally relating the logic to the system and (iii) formally relating the various systems to each other. There are some exceptions though, eg. in [21] it was proven that the presented connectionist system really solves the unification and matching problem or in [6] we have given a rigorous logical reconstruction of the backward reasoning version of SHRUTI.

We would like to see a theory where in various layers of increased expressiveness logics, their corresponding connectionist models, their time and space complexities, their properties concerning learning and rule extraction as well as learning and rule extraction algorithms are specified. Such a theory could be developed along the lines proposed in [23], the BUR system, [10, 13], and [2, 18]: In each layer the logic would be defined by a certain class of logic programs and the corresponding connectionist systems would be recurrent neural networks.

For example, if the logic programs are propositional, then interpretations are represented locally. The units in the corresponding recurrent neural network are logical threshold units. If learning shall be applied, then the threshold units in the hidden layer must be replaced by sigmoidal ones. If the programs are datalogic programs, then the corresponding recurrent neural network must be able to bind variables to constants which can be done by using phase coding as in SHRUTI and BUR. If the logic programs are full first order, then interpretations shall be represented by vectors of real numbers and models are only approximated, etc.

The logics in such a theory shall not only be the standard monotonic ones, but we should also consider nonmonotonic ones. By the way, nonmonotonic reasoning was originally proposed as a technique for "jumping to a conclusion". Nowadays conventional nonmonotonic reasoning systems have time and space complexities which are not at all in accordance with the original goal. It may well be that connectionist techniques may help to put the research in nonmonotonic reasoning techniques back on track.

A general theory for the integration of logic and connectionist systems could also be developed for symmetric networks [28]. Pinkas has shown that the problem of finding a model for a propositional logic formula is equivalent to finding a global minimum in an energy function [38]. He has extended his results to some nonmonotonic [37] and first-order logics [36]. Again, the picture is far from being complete.

3.9 Can such a theory be applied in a real domain outperforming conventional approaches?

All applications of connectionist inference systems that we have seen so far are toy examples. We have to come up with applications in real domains which outperform conventional approaches. This can only be done if we use hardware which exploits the massive parallelism of connectionist networks. If we are reasoning in a logic whose entailment problem is in $\mathcal{NC}$ and an efficient or optimal parallel algorithm for deciding this problem is known, then it does not suffice to simulate this algorithm on a computer with just a few processors.

Because a general theory for integrating logic and connectionist systems may be layered, applications we are looking for should have a similar structure. It may be worth while to look for such an application in the area of integrating the low-level control of a real robot with the high-level control developed in the area of cognitive robotics. Such an application would also be a good showcase for learning and rule extraction: Even if such a robot is initialized with some knowledge, it must learn to behave in its environment, and this learning never stops. Consequently, the knowledge is constantly updated.

4 Summary

In this paper we have given an overview on how first-order logic programs can be represented in a connectionist setting and outlined various challenges for developing a truly connectionist system capable of representing structured objects and performing structure-sensitive processes.

References

[1] M. J. Adamson and R. I. Damper. B-RAAM: A connectionist model which develops holistic internal representations of symbolic structures. *Connection Science*, 11(1):41–71, 1999.

[2] S. Bader and P. Hitzler. Logic programs, iterated function systems, and recurrent radial basis function networks. *Journal of Applied Logic*, 2(3):273–300, 2004.

[3] S. Bader, S. Hölldobler, and A. Scalzitti. Semiring artificial neural networks and weighted automata – and an application to digital image encoding –. In *Proceedings of the 27th German Conference on Artificial Intelligence, Ulm, Germany, September 2004, LNAI.* Springer, 2004. To appear.

[4] M. Barnsley. *Fractals Everywhere.* Academic Press, San Diego, 1993.

[5] A. Batarekh and V. S. Subrahmanian. Topological model set deformations in logic programming. *Fundamenta Informaticae*, 12:357–400, 1989.

[6] A. Beringer and S. Hölldobler. On the adequateness of the connection method. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 9–14, 1993.

[7] C. M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, 1995.

[8] H. A. Blair, F. Dushin, D. W. Jakel, A. J. Rivera, and M. Sezgin. Continuous models of computation for logic programs. In K. R. Apt, V. W. Marek, M. Truszczyński, and D. S. Warren, editors, *The Logic Programming Paradigm: A 25-Year Persepective*, pages 231–255. Springer, 1999.

[9] A. Browne and R. Sun. Connectionist inference models. *Neural Networks*, 14(10):1331–1355, 2001.

[10] A. S. d'Avila Garcez, K. Broda, and D. M. Gabbay. Symbolic knowledge extraction from trained neural networks: A sound approach. *Artificial Intelligence*, 125:155–207, 2001.

[11] A. S. d'Avila Garcez, K. Broda, and D. M. Gabbay. *Neural-Symbolic Learning Systems — Foundations and Applications.* Perspectives in Neural Computing. Springer, 2002.

[12] A. S. d'Avila Garcez and D. M. Gabbay. Fibring neural networks. In *Proceedings of the 19th National Conference on Artificial Intelligence. San Jose, California, USA, July 2004*, pages 342–347. AAAI Press, 2004.

[13] A. S. d'Avila Garcez and G. Zaverucha. The connectionist inductive lerarning and logic programming system. *Applied Intelligence, Special Issue on Neural networks and Structured Knowledge*, 11(1):59–77, 1999.

[14] J. A. Feldman and D. H. Ballard. Connectionist models and their properties. *Cognitive Science*, 6(3):205–254, 1982.

[15] M. Fitting. Fixpoint semantics for logic programming — A survey. *Theoretical Computer Science*, 278(1–2):25–51, 2002.

[16] J. A. Fodor and Z. W. Pylyshyn. Connectionism and cognitive architecture: A critical analysis. In Pinker and Mehler, editors, *Connections and Symbols*, pages 3–71. MIT Press, 1988.

[17] K.-I. Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2:183–192, 1989.

[18] P. Hitzler, S. Hölldobler, and A.K. Seda. Logic programs and connectionist networks. *Journal of Applied Logic*, 2(3):245–272, 2004.

[19] P. Hitzler. Corollaries on the fixpoint completion: studying the stable semantics by means of the clark completion. In D. Seipel, M. Hanus, U. Geske, and O. Bartenstein, editors, *Proceedings of the INAP'04 and WLP'04, Potsdam, Germany, March 2004*, volume 327 of *Technichal Report*, pages 13–27. Universität Würzburg, Institut für Informatik, 2004.

[20] P. Hitzler and A. K. Seda. Generalized metrics and uniquely determined logic programs. *Theoretical Computer Science*, 305(1–3):187–219, 2003.

[21] S. Hölldobler. A structured connectionist unification algorithm. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 587–593, 1990.

[22] S. Hölldobler. Automated inferencing and connectionist models. Technical Report AIDA–93–06, Intellektik, Informatik, TH Darmstadt, 1993. (Postdoctoral Thesis).

[23] S. Hölldobler and Y. Kalinke. Towards a massively parallel computational model for logic programming. In *Proc. of the ECAI94 Workshop on Combining Symbolic and Connectionist Processing*, pages 68–77. ECCAI, 1994.

[24] S. Hölldobler, Y. Kalinke, and H. Lehmann. Designing a counter: Another case study of dynamics and activation landscapes in recurrent networks. In *Proceedings of the KI97: Advances in Artificial Intelligence*, volume 1303 of *LNAI*, pages 313–324. Springer, 1997.

[25] S. Hölldobler, Y. Kalinke, and J. Wunderlich. A recursive neural network for reflexive reasoning. In S. Wermter and R. Sun, editors, *Hybrid Neural Symbolic Integration*, number 1778 in LNAI, pages 46–62. Springer, 2000.

[26] S. Hölldobler and F. Kurfess. CHCL – A connectionist inference system. In B. Fronhöfer and G. Wrightson, editors, *Parallelization in Inference Systems*, pages 318 – 342. Springer, LNAI *590*, 1992.

[27] S. Hölldobler, Y. Kalinke, and H.-P. Störr. Approximating the semantics of logic programs by recurrent neural networks. *Applied Intelligence*, 11:45–58, 1999.

[28] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. In *Proceedings of the National Academy of Sciences USA*, pages 2554 – 2558, 1982.

[29] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[30] Y. Kalinke. Using connectionist term representation for first–order deduction – a critical view. In F. Maire, R. Hayward, and J. Diederich, editors, *Connectionist Systems for Knowledge Representation Deduction*. Queensland University of Technology, 1997. Proc. CADE–14 Workshop.

[31] Y. Kalinke and H. Lehmann. Computations in recurrent neural networks: From counters to iterated function systems. In G. Antoniou and J. Slaney, editors, *Advanced Topics in Artificial Intelligence*, volume 1502 of *LNAI*, Springer, 1998. Proceedings of the 11th Australian Joint Conference on Artificial Intelligence (AI'98).

[32] T. E. Lange and M. G. Dyer. Frame selection in a connectionist model of high-level inferencing. In *Proceedings of the Annual Conference of the Cognitive Science Society*, pages 706–713, 1989.

[33] J. W. Lloyd. *Foundations of Logic Programming*. Springer, 1988.

[34] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[35] A. Newell. Physical symbol systems. *Cognitive Science*, 4:135–183, 1980.

[36] G. Pinkas. Expressing first-order logic in symmetric connectionist networks. In L. N. Kanal and C. B. Suttner, editors, *Informal Proceedings of the International Workshop on Parallel Processing for AI*, pages 155–160, Sydney, Australia, August 1991.

[37] G. Pinkas. Propositional non-monotonic reasoning and inconsistency in symmetrical neural networks. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 525–530, 1991.

[38] G. Pinkas. Symmetric neural networks and logic satisfiability. *Neural Computation*, 3:282–291, 1991.

[39] J. B. Pollack. Recursive auto-associative memory: Devising compositional distributed representations. In *Proceedings of the Annual Conference of the Cognitive Science Society*, pages 33–39, 1988.

[40] J. B. Pollack. Recursive distributed representations. *Artificial Intelligence*, 46:77–105, 1990.

[41] S. Prieß-Crampe and P. Ribenboim. Ultrametric spaces and logic programming. *The Journal of Logic Programming*, 42:59–70, 2000.

[42] A. K. Seda. Topology and the semantics of logic programs. *Fundamenta Informaticae*, 24(4):359–386, 1995.

[43] L. Shastri and V. Ajjanagadde. From associations to systematic reasoning: A connectionist representation of rules, variables and dynamic bindings using temporal synchrony. *Behavioural and Brain Sciences*, 16(3):417–494, 1993.

[44] P. Simons, I. Niemelä, and T. Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*. To appear.

[45] G. G. Towell and J. W. Shavlik. Knowledge-based artificial neural networks. *Artificial Intelligence*, 70(1–2):119–165, 1994.

[46] M. Wendt. Unfolding the well-founded semantics. *Journal of Electrical Engineering, Slovak Academy of Sciences*, 53(12/s):56–59, 2002.

[47] S. Willard. *General Topology*. Addison-Wesley, Reading, MA, 1970.

Eingereicht zur FLAIRS 2005, Clearwater Beach, Florida, Mai 2005,
Special Track on Integrated Intelligent Systems.

Computing First-Order Logic Programs by Fibring Artificial Neural Networks

Sebastian Bader[*]
Department of Computer Science
Technische Universität Dresden
Germany

Artur S. d'Avila Garcez[†]
Department of Computing
City University London
UK

Pascal Hitzler[‡]
Institute AIFB
University of Karlsruhe
Germany

Abstract

The integration of symbolic and neural-network-based artificial intelligence paradigms constitutes a very challenging area of research. The overall aim is to merge these two very different major approaches to intelligent systems engineering while retaining their respective strengths. For symbolic paradigms that use the syntax of some first-order language this appears to be particularly difficult. In this paper, we will extend on an idea proposed by Garcez and Gabbay (2004) and show how first-order logic programs can be represented by fibred neural networks. The idea is to use a neural network to iterate a global counter n. For each clause C_i in the logic program, this counter is combined (fibred) with another neural network, which determines whether C_i outputs an atom of level n for a given interpretation I. As a result, the fibred network computes the single-step operator $T_{\mathcal{P}}$ of the logic program, thus capturing the semantics of the program.

Introduction

Intelligent systems based on artificial neural networks differ substantially from those based on symbolic knowledge processing like logic programming. Neural networks are trainable from raw data and are robust, but practically impossible to read declaratively. Logic programs can be implemented from problem specifications and can be highly recursive, while lacking good training methods and robustness, particularly when data are noisy (Thrun & others 1991). It is obvious that an integration of both paradigms into single systems would be very beneficiary if the respective strengths could be retained.

There exists a notable body of work investigating the integration of neural networks with propositional — or similarly finitistic — logic. We refer to (Browne & Sun 2001; d'Avila Garcez, Broda, & Gabbay 2002) for overviews. For first-order logic, however, it is much less clear how a reasonable integration can be achieved, and there are systematic difficulties which slow down recent research efforts, as spelled out in (Bader, Hitzler, & Hölldobler 2004). Different techniques for overcoming these obstacles are currently under investigation, including the use of metric spaces and topology, and of iterated function systems (Hitzler, Hölldobler, & Seda 2004; Bader & Hitzler 2004).

At the heart of these integration efforts is the question of how first-order knowledge can be represented by neural network architectures. In this paper, we present a novel approach using *fibring neural networks* as proposed by (d'Avila Garcez & Gabbay 2004). For each clause C_i of a logic program, a neural network that iterates a counter n is combined (fibred) with another neural network, which determines whether C_i outputs an atom of level n for a given interpretation I. Fibring offers a modular way of performing complex functions by using relatively simple networks (modules) in an ensemble.

The paper is organized as follows. In the next section we briefly review fibring neural networks and logic programs. We then present the fundamental ideas underlying our representation results, before giving the details of our implementation and a worked example. We conclude with some discussions.

Preliminaries

We introduce standard terminology for artificial neural networks, fibring neural networks, and logic programs. We refer the reader to (Bishop 1995; d'Avila Garcez & Gabbay 2004; Lloyd 1988), respectively, for further background.

Artificial Neural Networks

Artificial neural networks consist of simple computational units (neurons), which receive real numbers as inputs via weighted connections and perform *simple* operations: the weighted inputs are added and simple functions like threshold, sigmoidal, identity or truncate are applied to the sum.

The neurons are usually organized in layers. Neurons which do not receive input from other neurons are called input neurons, and those without outgoing connections to other neurons are output neurons. So a network computes a function from $\mathbb{R}^n$ to $\mathbb{R}^m$, where n and m are the number of input, respectively, output units. A key to the success of

[*]Sebastian Bader is supported by the GK334 of the German Research Foundation.

[†]Artur Garcez is partly supported by The Nuffield Foundation.

[‡]Pascal Hitzler is supported by the German Federal Ministry of Education and Research under the SmartWeb project and by the European Union under the KnowledgeWeb Network of Excellence.

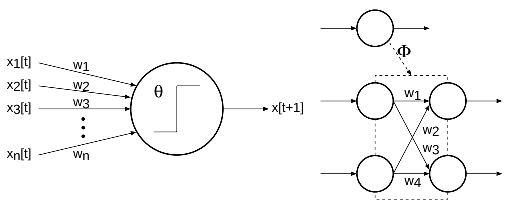


Figure 1: An artificial neuron (left) and a simple fibring network (right)

neural network architectures rests on the fact that they can be trained effectively using training samples in the form of input-output pairs.

For convenience, we make the following assumptions for the networks depicted in this paper: The layers are updated sequentially from left to right and within a layer the neurons are updated from top to bottom.

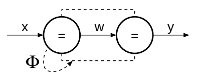
Recently, (d'Avila Garcez & Gabbay 2004) introduced a new model of neural networks, namely *fibring neural networks*. Briefly, the activation of a certain unit may influence the behaviour of other units by changing their weights. Our particular architecture is a slight variant of the original proposal, which appears to be more natural for our purposes.

Definition 1 *A* fibring function Φ_i *associated with neuron* i *maps some weights* w *of the network to new values, depending on* w *and the input* x *of neuron* i.

Fibring functions can be understood as modeling *presynaptic weights*, which play an important role in biological neural networks. Certainly, a necessary requirement for biological plausibility is that fibring functions compute either *simple* functions or tasks which can in turn be performed by neural networks. We will return to this point later.

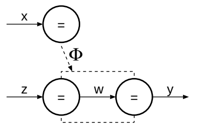
Throughout this paper we will use dashed lines, as in Figure 1, to indicate the weights which may be changed by some fibring function. As described above, we will use an update dynamics from left to right, and top to bottom. And, as soon as the activation of a fibring neuron is (re)calculated, the corresponding fibring function is applied and the respective weights are modified.

Example 2 A simple fibring network for squaring numbers. Each node computes the weighted sum of its inputs and performs the operation *identity* on it. The fibring function takes input x and multiplies it by W. If $W = 1$, the output will be $y = x^2$:



$$\Phi : (w, x) \mapsto x$$

Example 3 A simple fibring network implementing a gate-like behaviour. Nodes behave as in Example 2:



$$\Phi : (w, x) \mapsto \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

The question of plausible types of fibring functions, as well as the computational power of those networks, will be studied separately and are touched here only slightly. We will start with very general fibring functions, but later we restrict ourselves to simple ones only, e.g. the fibred weight is simply multiplied by the activation.

Sometimes we will use the output of a neuron instead of the activation, or apply linear transformations to it, and it is clear that such modifications could also be achieved by adding another neuron to the network and use this for the fibring. Therefore these modifications can be understood as abbreviations to keep the networks simple.

Logic Programs

A *logic program* is a finite set of *clauses* $H \leftarrow L_1 \wedge \cdots \wedge L_n$, where $n \in \mathbb{N}$ may differ for each clause, H is an atom in a first order language $\mathcal{L}$ and $L_1, \ldots, L_n$ are literals, that is, atoms or negated atoms, in $\mathcal{L}$. The clauses of a program are understood as being universally quantified. H is called the *head* of the clause, each L_i is called a *body literal* and their conjunction $L_1 \wedge \cdots \wedge L_n$ is called the *body* of the clause. We allow $n = 0$, by an abuse of notation, which indicates that the body is empty; in this case the clause is called a *unit clause* or a *fact*.

An atom is said to be *ground* if it does not contain variables, and the *Herbrand base* underlying a given program $\mathcal{P}$ is defined as the set of all ground instances of atoms, denoted $B_{\mathcal{P}}$. Example 4 shows a logic program and its corresponding Herbrand base. Subsets of the Herbrand base are called (*Herbrand*) *interpretations* of $\mathcal{P}$, and we can think of such a set as containing those atoms which are *true* under the interpretation. The set $I_{\mathcal{P}}$ of all interpretations of a program $\mathcal{P}$ can thus be identified with the power set of $B_{\mathcal{P}}$.

Example 4 The *natural numbers* program $\mathcal{P}$, the underlying language $\mathcal{L}$ and the corresponding Herbrand base $B_{\mathcal{P}}$. The intended meaning of s is the successor function:

$\mathcal{P}$:	$nat(0)$. $nat(s(X)) \leftarrow nat(X)$.
$\mathcal{L}$:	constants: $\mathcal{C} = \{0\}$ functions: $\mathcal{F} = \{s/1\}$ relations: $\mathcal{R} = \{nat/1\}$
$B_{\mathcal{P}}$:	$nat(0), nat(s(0)), nat(s(s(0))), \ldots$

Logic programs are accepted as a convenient tool for knowledge representation in logical form. Furthermore, the knowledge represented by a logic program $\mathcal{P}$ can essentially be captured by the *immediate consequence* or *single-step operator* $T_{\mathcal{P}}$, which is defined as a mapping on $I_{\mathcal{P}}$ where for any $I \in I_{\mathcal{P}}$ we have that $T_{\mathcal{P}}(I)$ is the set of all $H \in B_{\mathcal{P}}$ for which there exists a ground instance $H \leftarrow A_1 \wedge \cdots \wedge A_m \wedge \neg B_1 \wedge \cdots \wedge \neg B_n$ of a clause in $\mathcal{P}$ such that for all i we have $A_i \in I$ and for all j we have $B_j \notin I$. Fixed points of $T_{\mathcal{P}}$ are called *supported models* of P, which can be understood to represent the declarative semantics of P.

In the sequel of this paper we will often need to enumerate the Herbrand base, which is done via *level mappings*:

Definition 5 *Given a logic program P, a* level mapping *is a function $|\cdot| : B_{\mathcal{P}} \to \mathbb{N}^+$, where $\mathbb{N}^+$ denotes the set of positive integers excluding zero.*

Level mappings — in slightly more general form — are commonly used for controlling recursive dependencies between atoms, and the most prominent notion is probably the following.

Definition 6 *Let $\mathcal{P}$ be a logic program and $|\cdot|$ be a level mapping. If for all clauses $A \leftarrow L_1 \wedge L_2 \wedge \ldots \wedge L_n \in ground(\mathcal{P})$ and all $1 \le i \le n$ we have that $|A| > |L_i|$, then $\mathcal{P}$ is called* acyclic with respect to $|\cdot|$. *A program is called* acyclic, *if there exists such a level mapping.*

Acyclic programs are known to have unique supported models (Cavedon 1991). The programs from Examples 4 and 7 below are acyclic.

Example 7 The "even and odd numbers" program and a level mapping:

$$\mathcal{P}: \quad \boxed{\begin{array}{l} even(0). \\ even(s(X)) \leftarrow \neg even(X). \\ odd(s(X)) \leftarrow even(X). \end{array}}$$

$$|\cdot|: \quad \boxed{|A| = \begin{cases} 2 \cdot n + 1 & \text{if } A = even(s^n(0)) \\ 2 \cdot n + 2 & \text{if } A = odd(s^n(0)) \end{cases}}$$

Throughout this paper we will assume that level mappings are bijective, i.e. for each $n \in \mathbb{N}^+$ there is exactly one $A \in B_{\mathcal{P}}$, such that $|A| = n$. Thus, for the purposes of our paper, a level mapping is simply an enumeration of the Herbrand base. Since level mappings induce an order on the atoms, we can use them to define a *prefix*-function on interpretations, returning only the first n atoms:

Definition 8 *The* prefix *of length n of a given interpretation I is defined as*

$$\begin{aligned} \mathrm{pref} \ &: I_{\mathcal{P}} \times \mathbb{N}^+ \to I_{\mathcal{P}} \\ &(I, n) \mapsto \{A | A \in I \text{ and } |A| \le n\}. \end{aligned}$$

We will write $\mathrm{pref}_n(I)$ *for* $\mathrm{pref}(I, n)$.

For acyclic programs, it follows that in order to decide whether the atom with level $n+1$ must be included in $T_{\mathcal{P}}(I)$, it is sufficient to consider $\mathrm{pref}_n(I)$ only.

From Logic Programs to Fibring Networks

We will show how to represent acyclic logic programs by means of fibring neural networks. We follow up on the basic idea from (Hölldobler & Kalinke 1994; Hölldobler, Kalinke, & Störr 1999), and further developed in (Hitzler, Hölldobler, & Seda 2004; Bader & Hitzler 2004), to represent the single-step operator $T_{\mathcal{P}}$ by a network, instead of the program $\mathcal{P}$ itself. This is a reasonable thing to do since the single-step operator essentially captures the semantics of the program it is associated with, as mentioned before.

In order to represent $T_{\mathcal{P}}$ by the input-output mapping of a network, we also need an encoding of $I_{\mathcal{P}}$ as a suitable subset of the real numbers. We also use an idea from (Hölldobler, Kalinke, & Störr 1999) for this purpose. Let $B > 2$ be some integer, and let $|\cdot|$ be a bijective level mapping. Define

$$R : I_{\mathcal{P}} \to \mathbb{R} : I \mapsto \sum_{A \in I} B^{-|A|}.$$

We exclude $B = 2$, because in this case R would not be injective. It will be convenient to assume $B = 3$ throughout the paper, but our results do not depend on this. We denote the range of R by $\mathrm{range}(R)$.

There are systematic reasons why this way of embedding $I_{\mathcal{P}}$ into the reals is reasonable, and they can be found in (Hitzler, Hölldobler, & Seda 2004; Bader & Hitzler 2004), but will not concern us here. Using R, the prefix operation can be expressed naturally on the reals.

Proposition 9 *For $I \in I_{\mathcal{P}}$ and $x \in \mathrm{range}(R)$ we have*

$$\mathrm{pref}(I, n) = R^{-1}\left(\frac{\mathrm{trunc}(R(I) \cdot B^n)}{B^n}\right) \text{ and}$$

$$R(\mathrm{pref}(R^{-1}(x), n)) = \frac{\mathrm{trunc}(x \cdot B^n)}{B^n}.$$

For convenience, we overload pref and set $\mathrm{pref}(x, n) = R(\mathrm{pref}(R^{-1}(x), n))$ and $\mathrm{pref}_n(x) = \mathrm{pref}(x, n)$.

We will now turn to the construction of fibring networks which approximate given programs. We will first describe our approach in general terms, and spell it out in a more formal and detailed way later on. The goal is to construct a neural network, which will compute $R(T_{\mathcal{P}})(x) = R(T_{\mathcal{P}}(R^{-1}(x)))$ for a given $x \in \mathrm{range}(R)$. The network is designed in such a way that it successively approximates $R\,(T_P)\,(x)$ while running.

There will be a main loop iterating a global counter n. This counter fibres the kernel, which will evaluate whether the atom of level n is contained in $T_{\mathcal{P}}(I)$ or not, i.e. the kernel will output B^{-n} if the atom is contained, and 0 otherwise. Furthermore, there will be an input subnetwork providing $R(I)$ all the time, and the output subnetwork which will accumulate the outputs of the kernel, and hence converge to $R(T_{\mathcal{P}}(I))$.

For each clause C_i there is a subnetwork, which determines whether C_i outputs the atom of level n for the given interpretation I, or not. This is done by fibring the subnetwork such that it computes the corresponding ground instance $C_i^{(n)}$, with head of level n, if existent. If there is no such ground instance, this subnetwork will output 0, otherwise it will determine whether the body is $true$ under the interpretation I. A detailed description of these clause networks will be given in the next section. Note that this construction is only possible for programs which are *covered*. This means that they do not have any local variables, i.e. every variable occuring in some body also occurs in the corresponding head. Obviously, programs which are acyclic with respect to a bijective level mapping are always covered.

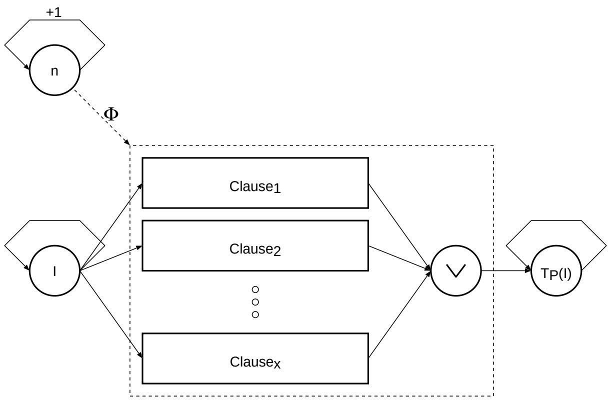


Figure 2: General architecture

If $\mathcal{P}$ is acyclic we can compute the unique supported model of the program directly, by connecting the output and the input region of the network as shown in Figure 3. This is simply due to the above mentioned fact: If we want to decide whether the atom of level n should be included in $T_{\mathcal{P}}(I)$, it is sufficient to look at the atoms $A \in I$ with level $< n$. We also have the following result.

Proposition 10 *Let $\mathcal{P}$ be a program which is acyclic with respect to a bijective level mapping $|\cdot|$, let $A \in B_{\mathcal{P}}$ with $|A| = n$. Then for each $I \in I_{\mathcal{P}}$ we have that $A \in T^n_{\mathcal{P}}(I)$ iff A is true with respect to the unique supported model of $\mathcal{P}$.*

Proof This is an immediate result from the application of the Banach contraction mapping principle to the semantic analysis of acyclic programs, see (Hitzler & Seda 2003). $\square$

So, for acyclic programs, we can start with the empty (or any other) interpretation and let the (recurrent) network run.

Implementing Clauses

In order to complete the construction from the previous section, we give an implementation of the clauses. For a clause C of the form $H \leftarrow L_1 \wedge L_2 \wedge \ldots \wedge L_k$, let $C^{(n)}$ denote the ground instance of C for which the head has level n, assuming it exists. The idea of the following construction is to create a network which implements C, and will be fibred by the counter n such that it implements $C^{(n)}$. In case that there is no ground instance of C with head of level n, the network will output 0, otherwise it will output 1 if the body is true with respect to the interpretation I, and 0 if it is not.

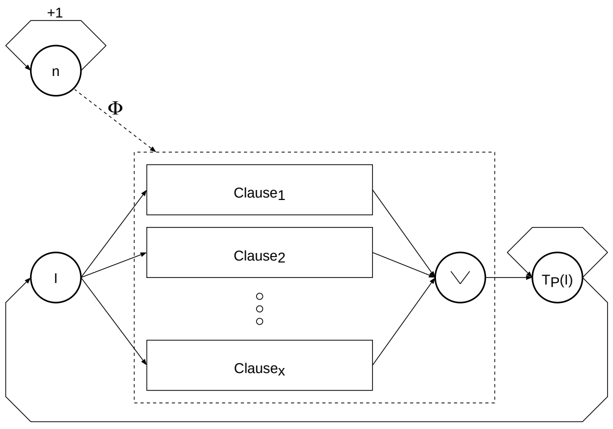


Figure 3: Recurrent architecture for acyclic programs

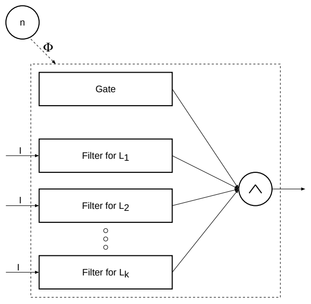


Figure 4: Implementing clauses

The idea, as shown in Figure 4, is that each subnetwork implementing a clause $C : H \leftarrow L_1 \wedge \ldots \wedge L_k$ with k body literals, consists of $k + 1$ parts — one *gate* and k *filters*. The gate will output 1, if the clause C has a ground instance $C^{(n)}$ where the level of the head is n. Furthermore there is a filter for each body literal L_i, which outputs 1, if the corresponding ground literal L_i is true under I. If all conditions are satisfied the final conjunction-neuron will become active, i.e. the subnetwork outputs 1.

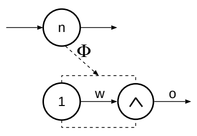
Note that this construction again is sufficient only for programs which are covered. If we allowed local variables, then more than one (in fact infinitely many) ground instances of C with a head of level n could exist.

Let us have a closer look at the type of fibring function needed for our construction. For the gate, it implicitly performs a very *simple pattern matching* operation, checking whether the atom with level n unifies with the head of the clause. For the filters, it checks whether corresponding instances of body literals are true in the given interpretation, i.e. it implicitly performs a *variable binding* and an *elementary check of set-inclusion*.

We argue that the operations performed by the fibring function are indeed biologically feasible. The perspective which we take in this paper is that they should be understood as functions performed by a separate network, which we do not give explicitly, although we will substantiate this point to a certain extent in the next section. And pattern matching is indeed a task that connectionist networks perform well. The variable binding task will also be addressed in the next section when we give examples for implementing the filters.

Neural Gates

As specified above, the gate for a clause $C : H \leftarrow L_1 \wedge \ldots \wedge L_k$ fires if there is a ground instance $C^{(n)}$ of C with head is of level n, as depicted in Figure 5. The decision based on



$$\Phi(w, n) = \begin{cases} 1 & \text{if ground instance with} \\ & \text{head of level } n \text{ exists} \\ 0 & \text{otherwise} \end{cases}$$

Figure 5: A neural gate

simple pattern matching is embedded into the fibring function. In what follows, we will discuss a number of different cases of how to unfold this fibring function into a network, in order to give plausible network topologies and yet simpler fibring functions. Other implementations are possible, and the cases presented here shall serve as examples only.

Ground-headed clauses. Let us first consider a clause for which the head does not contain variables, i.e. a ground clause, like for example the first clause given in Example 7 above. Since the level of the head in this case is fixed to some value, say m, the corresponding gate subnetwork should fire if and only if the general counter n is equal to m. This can be done using the network shown in Figure 6 (left): The neuron "1!" will always output 1 and the neuron "$= 0$" will output 1 if and only if the weighted inputs sum up to 0. This can easily be implemented using e.g. threshold units.

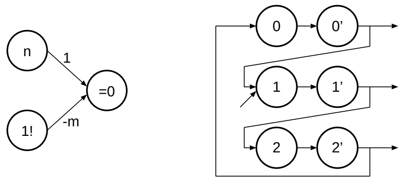
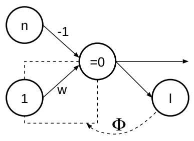


Figure 6: Simple gates for ground-headed clauses (left) and remainder classes (right)

Remainder classes. If the levels l_i of ground instantiated heads for a certain clause can be expressed as multiples of a certain fixed number m, i.e. $l_i = i \cdot m$ for all i (like clauses number 2 and 3 of Example 7), we can construct a simple subnetwork, as depicted in Figure 6 (right). The neurons symbolize the equivalence classes for the remainders of the devision by 3. The network will be initialized by activating "1". Every time it is reevaluated the activation simply proceeds to the next row.

Powers. If the level l_i of ground instantiated heads for a certain clause can be expressed as powers of a certain fixed number m, i.e. $l_i = m^i$ for all i, we can construct a simple subnetwork as shown in Figure 7.



$$w_{init} = -1$$
$$\Phi(w, x) = w \cdot m$$

Figure 7: A simple gate for powers

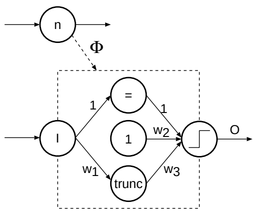
Filtering Interpretations

For a network to implement the ground instance $C^{(n)} : H_n \leftarrow L_1^{(n)} \wedge \ldots \wedge L_k^{(n)}$ of a clause C with head of level n, we need to know the *distance* between the head and the body literals — in terms of levels — as a function in n, i.e. we need a set of functions $\{b_i : \mathbb{N} \rightarrow \mathbb{N} \mid i = 1, \ldots, k\}$ — one for each body literal — where b_i computes the level of the literal L_i, taking as input the level of the head, as illustrated in Example 11.

Example 11 For the "even and odd numbers" program from Example 7, we can use the following b_i-functions:

$even(0).$	$\{\}$
$even(s(X)) \leftarrow \neg even(X).$	$\{b_1 : n \mapsto n - 2\}$
$odd(s(X)) \leftarrow even(X).$	$\{b_1 : n \mapsto n - 1\}$

For each body literal we will now construct a *filter* subnetwork, that fires if the corresponding ground body literal $L_i^{(n)}$ of $C^{(n)}$ is included in I. Given an interpretation I, we need to decide whether a certain atom A is included or not. The underlying idea is the following. In order to decide whether the atom A of level n is included in the interpretation I, we construct an interpretation J containing all atoms of I with level smaller than n, and the atom A, i.e. $J = \text{pref}_{n-1}(I) \cup \{A\}$, or, expressed on the reals, $R(J) = \text{pref}_{n-1}(R(I)) + B^{-n}$. If we evaluate $R(I) - R(J)$ the result will be non-negative if and only if A is included in I. This can be done using the network shown in Figure 8.



$$\Phi(w_1, n) = B^n$$
$$\Phi(w_2, n) = -B^{-n}$$
$$\Phi(w_3, n) = B^{-n}$$

Figure 8: Schematic plot and fibring function of a filter for the atom of level n

It is clear that we can construct networks to filter an atom of level $b_i(n)$, if the function b_i can itself be implemented in a neural network. Since fibring networks can implement any polynomial function, as shown in (d'Avila Garcez & Gabbay

2004) and indicated in Example 2, our approach is flexible and very general.

A Worked Example

Let us now give a complete example by extending on the logic program and the level mapping from Example 7 above. For the first clause we need a ground-headed gate only. To implement the second clause a remainder-class gate for the devision by 2 is needed, which returns 1 for all odd numbers. Furthermore, we need a filter which returns 1 if the atom of level $n-2$ is not included in I. For the last clause of the example, we need a gate returning 1 for all even numbers and a similar filter as for clause number 2. Combining all three parts and taking into account that $\mathcal{P}$ is acyclic, we get the network shown in Figure 9. If run on any initial value,

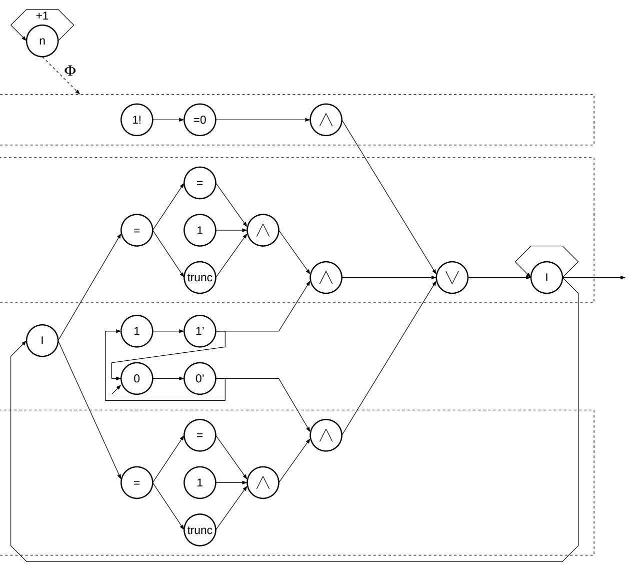


Figure 9: Neural implementation of the whole example

its outputs converge to the unique supported model of $\mathcal{P}$, i.e. the sequence of outputs of the right-most neuron is a sequence of real numbers which converges to $R(M)$, where M is the unique supported model of $\mathcal{P}$.

Conclusions

This paper contributes to advance the state of the art on neural-symbolic integration by showing how first-order logic programs can be implemented in fibring neural networks. Generic ways for representing the needed fibring functions in a biologically plausible fashion remain to be investigated in detail, as well as the task of extending our proposal towards a fully functional neural-symbolic learning and reasoning system.

Fibring offers a modular way of performing complex functions, such as logical reasoning, by combining relatively simple modules (networks) in an ensemble. If each module is kept simple enough, we should be able to apply standard neural learning algorithms to them. Ultimately, this may provide an integrated system with robust learning and expressive reasoning capability.

References

Bader, S., and Hitzler, P. 2004. Logic programs, iterated function systems, and recurrent radial basis function networks. *Journal of Applied Logic* 2(3):273–300.

Bader, S.; Hitzler, P.; and Hölldobler, S. 2004. The integration of connectionism and first-order knowledge representation and reasoning as a challenge for artificial intelligence. In *Proceedings of the Third International Conference on Information, Tokyo, Japan*. To appear.

Bishop, C. M. 1995. *Neural Networks for Pattern Recognition*. Oxford University Press.

Browne, A., and Sun, R. 2001. Connectionist inference models. *Neural Networks* 14(10):1331–1355.

Cavedon, L. 1991. Acyclic programs and the completeness of SLDNF-resolution. *Theoretical Computer Science* 86:81–92.

d'Avila Garcez, A. S., and Gabbay, D. M. 2004. Fibring neural networks. In McGuinness, D. L., and Ferguson, G., eds., *Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence, July 25-29, 2004, San Jose, California, USA*, 342–347. AAAI Press / The MIT Press.

d'Avila Garcez, A. S.; Broda, K. B.; and Gabbay, D. M. 2002. *Neural-Symbolic Learning Systems — Foundations and Applications*. Perspectives in Neural Computing. Springer, Berlin.

Hitzler, P., and Seda, A. K. 2003. Generalized metrics and uniquely determined logic programs. *Theoretical Computer Science* 305(1–3):187–219.

Hitzler, P.; Hölldobler, S.; and Seda, A. K. 2004. Logic programs and connectionist networks. *Journal of Applied Logic* 2(3):245–272.

Hölldobler, S., and Kalinke, Y. 1994. Towards a massively parallel computational model for logic programming. In *Proceedings ECAI94 Workshop on Combining Symbolic and Connectionist Processing*, 68–77. ECCAI.

Hölldobler, S.; Kalinke, Y.; and Störr, H.-P. 1999. Approximating the semantics of logic programs by recurrent neural networks. *Applied Intelligence* 11:45–58.

Lloyd, J. W. 1988. *Foundations of Logic Programming*. Springer, Berlin.

Thrun, S. B., et al. 1991. The MONK's problems: A performance comparison of different learning algorithms. Technical Report CMU-CS-91-197, Carnegie Mellon University.

In: S. Biundo, T. Frühwirth, and G. Palm (eds.), Proceedings of the 27th German conference on Artificial Intelligence, KI'2004, Ulm, Germany, September 2004, Lecture Notes in Artificial Intelligence 3238, Springer, Berlin, 2004, pp. 351-365. (mit zusätzlichem Appendix)

Default reasoning over domains and concept hierarchies

Pascal Hitzler

Department of Computer Science, Dresden University of Technology

Abstract. W.C. Rounds and G.-Q. Zhang have proposed to study a form of disjunctive logic programming generalized to algebraic domains [1]. This system allows reasoning with information which is hierarchically structured and forms a (suitable) domain. We extend this framework to include reasoning with *default negation*, giving rise to a new nonmonotonic reasoning framework on hierarchical knowledge which encompasses answer set programming with extended disjunctive logic programs. We also show that the hierarchically structured knowledge on which programming in this paradigm can be done, arises very naturally from formal concept analysis. Together, we obtain a default reasoning paradigm for conceptual knowledge which is in accordance with mainstream developments in nonmonotonic reasoning.

1 Introduction

In [1], Rounds and Zhang propose to study a form of clausal logic generalized to algebraic domains, in the sense of domain theory [2]. In essence, they propose to interpret finite sets of compact elements as clauses, and develop a theory which links corresponding logical notions to topological notions on the domain. Amongst other things, they establish a sound and complete resolution rule and a form of disjunctive logic programming over domains. A corresponding semantic operator turns out to be Scott-continuous.

We will utilize this proposal as a link between the formerly unrelated areas of formal concept analysis, on the one hand, and nonmonotonic reasoning, in the form of answer set programming, on the other. The relationships thus worked out serve a threefold purpose, namely (1) to obtain a sound domain-theoretic perspective on answer set programming, (2) to provide a formal link between domain logics and formal concept analysis for the purpose of cross-transfer of methods and results, and (3) to devise a reasoning paradigm which encompasses two formerly unrelated formalisms for commonsense reasoning, namely formal concept analysis, and answer set programming.

So in this paper, we will extend the logic programming paradigm due to Rounds and Zhang to include reasoning with default negation. We are motivated by the gain in expressiveness through the use of negation in artificial intelligence paradigms related to nonmonotonic reasoning. This approach, using ideas from default logic [3], treats negation under the intuition that the negation of an item shall be believed if there is no reason to believe the item itself. This perspective on negation has recently led to the development of applications in the form of nonmonotonic reasoning systems known as *answer set programming*, the two most popular probably being dlv and smodels [4, 5]. We will indeed see that the extension of the approach by Rounds and Zhang by default negation is a natural generalization of answer set programming with extended disjunctive logic programs [6].

On the other hand, building on the work reported in [7], we establish a strong connection between the clausal logic on algebraic domains mentioned above, and fundamental notions from formal concept analysis [8]. More precisely, we will see that in certain cases the formation of formal concepts from formal contexts can be recast naturally via the notion of logical consequence in Rounds' and Zhang's clausal logic. Our default reasoning paradigm on domains can therefore be reinterpreted as a reasoning paradigm over conceptual knowledge, with potential applications to symbolic data analysis.

To the best of our knowledge, the results in this paper constitute the first proposal for a default reasoning paradigm on conceptual knowledge which is compatible with mainstream research developments in nonmonotonic reasoning. We focus on laying foundations for this, but will not pursue questions of applicability to data analysis at this stage. This will be done elsewhere.

The plan of the paper is as follows. In Section 2 we recall main notions and results on the clausal logic of Rounds and Zhang, and its extension to a logic programming paradigm. In Section 3 we will add a notion of default negation, and in Section 4 we will see that it naturally extends answer set programming for extended disjunctive programs. Section 5 is devoted to the study of conceptual knowledge related to our paradigm. Related work is being discussed in Section 6, while we will conclude and discuss further work in Section 7.

Proofs have been omitted for lack of space; they can be found on the author's webpage.

Acknowledgements. This work was supported by a fellowship within the Postdoc-Programme of the German Academic Exchange Service (DAAD) and carried out while the author was visiting the Department of Electrical Engineering and Computer Science at Case Western Reserve University, Cleveland, Ohio. I am grateful for inspiring discussions with Rainer Osswald, Matthias Wendt, and Guo-Qiang Zhang, and for the feedback of some anonymous referees on an earlier version of this paper.

2 Clausal Logic and Logic Programming in Algebraic Domains

The study of domain theory from a logical perspective has a long tradition, and originates from [9], where a logical characterization (more precisely, a categorical equivalence) of bounded complete algebraic cpo's (with Scott continuous functions as morphisms) was given. Rounds and Zhang [1] have recently devised a similar characterization of Smyth powerdomains. They use a clausal logic for this purpose, and have also shown that it extends naturally to a disjunctive logic programming paradigm. We recall necessary notation and terminology in order to make this paper self-contained.

A *partially ordered set* is a pair $(D, \sqsubseteq)$, where D is a nonempty set and $\sqsubseteq$ is a reflexive, antisymmetric, and transitive relation on D. A subset X of a partially ordered set is *directed* if for all $x, y \in X$ there is $z \in X$ with $x, y \sqsubseteq z$. Note that the empty set is directed. An *ideal* is a directed and downward closed set. A *complete partial order*, *cpo* for short, is a partially ordered set $(D, \sqsubseteq)$ with a least element $\bot$, called the *bottom element* of $(D, \sqsubseteq)$, and such that every directed set in D has a least upper bound, or supremum, $\bigsqcup D$. An element $c \in D$ is said to be *compact* or *finite* if whenever $c \sqsubseteq \bigsqcup L$ with L directed, then there exists $e \in L$ with $c \sqsubseteq e$. The set of all compact elements of a cpo D is written as $\mathsf{K}(D)$. An *algebraic cpo* is a cpo such that every $e \in D$ is the directed supremum of all compact elements below it. For $a, b \in D$ we write $a \not\uparrow b$ if a and b are *inconsistent*, i.e. if there does not exist a common upper bound of a and b.

A set $U \subseteq D$ is said to be *Scott open*, or just *open*, if it is upward closed and for any directed $L \subseteq D$ we have $\bigsqcup L \in U$ if and only if $U \cap L \neq \emptyset$. The *Scott topology* on D is the topology whose open sets are all Scott open sets. An open set is *compact open* if it is compact in the Scott topology. A *coherent algebraic cpo* is an algebraic cpo such that the intersection of any two compact open sets is compact open. We will not make use of many topological notions in the sequel. So let us just note that coherency of an algebraic cpo implies that the set of all minimal upper bounds of a finite number of compact elements is finite, i.e. if $c_1, \dots, c_n$ are compact elements, then the set $\mathsf{mub}\{c_1, \dots, c_n\}$ of minimal upper bounds of these elements is finite. As usual, we set $\mathsf{mub}\,\emptyset = \{\bot\}$, where $\bot$ is the least element of D.

In the following, $(D, \sqsubseteq)$ will always be assumed to be a coherent algebraic cpo. We will also call these spaces *domains*. All of the above notions are standard and can be found e.g. in [2].

The following notions are taken from [1].

Definition 1. *Let D be a coherent algebraic cpo with set $\mathsf{K}(D)$ of compact elements. A* clause *is a finite subset of $\mathsf{K}(D)$. We denote the set of all clauses over D by $\mathcal{C}(D)$. If X is a clause and $w \in D$, we write $w \models X$ if there exists $x \in X$ with $x \sqsubseteq w$, i.e. X contains an element below w. A* theory *is a set of clauses, which may be empty. An element $w \in D$ is a* model *of a theory T, written $w \models T$, if $w \models X$ for all $X \in T$ or, equivalently, if every clause $X \in T$ contains an element below w. A clause X is called a* logical consequence *of a theory T, written $T \models X$, if $w \models T$ implies $w \models X$. If $T = \{E\}$, then we write $E \models X$ for $\{E\} \models X$. Note that this holds if and only if for every $w \in E$ there is $x \in X$ with $x \sqsubseteq w$. For two theories T and S, we say that $T \models S$ if $T \models X$ for all $X \in S$. In order to avoid confusion, we will throughout denote the empty clause by $\{\}$, and the empty theory by $\emptyset$. A theory T is* closed *if $T \models X$ implies $X \in T$ for all clauses X. It is called* consistent *if $T \not\models \{\}$ or, equivalently, if there is w with $w \models T$.*

The clausal logic introduced in Definition 1 will henceforth be called the *logic RZ* for convenience.

A main technical result from [1], where the notions from Definition 1 were introduced, shows that the set of all consistent closed theories over D, ordered by inclusion, is isomorphic to the collection of all non-empty Scott-compact saturated subsets of D, ordered by reverse inclusion — and the latter is isomorphic to the Smyth powerdomain of D. This result rests on the Hofmann-Mislove theorem [10]. It is also shown that a theory is logically closed if and only if it is an ideal,[1] and also that a clause is a logical consequence of a theory T if and only if it is a logical consequence of a finite subset of T. The latter is a compactness theorem for clausal logic in algebraic domains.

Example 1. In [1], the following running example was given. Consider a countably infinite set of propositional variables, and the set $\mathbb{T} = \{\mathbf{f}, \mathbf{u}, \mathbf{t}\}$ of truth values ordered by $\mathbf{u} \leq \mathbf{f}$ and $\mathbf{u} \leq \mathbf{t}$. This induces a pointwise ordering on the space $\mathbb{T}^{\mathcal{V}}$ of all interpretations (or *partial truth assignments*). The partially ordered set $\mathbb{T}^{\mathcal{V}}$ is a coherent algebraic cpo[2] and has been studied e.g. in [11] in a domain-theoretic context, and in [12] in a logic programming context. Compact elements in $\mathbb{T}^{\mathcal{V}}$ are those interpretations which map all but a finite number of propositional variables to $\mathbf{u}$. We denote compact elements by strings such as $pq\overline{r}$, which indicates that p and q are mapped to $\mathbf{t}$ and r is mapped to $\mathbf{f}$. Clauses in $\mathbb{T}^{\mathcal{V}}$ can be identified with formulae in disjunctive normal form, e.g. $\{pq\overline{r}, \overline{p}q, r\}$ translates to $(p \land q \land \neg r) \lor (\neg p \land q) \lor r$.

In [1], it was shown that the logic RZ is compact. A proof theory for it was also given. An alternative version was reported in [13, 14].

The logic RZ provides a framework for reasoning with disjunctive information on a lattice which encodes background knowledge. Indeed it was shown [7] that it relates closely to formal concept analysis, which in turn has been applied successfully in data mining, and we will expand on this point later on in Section 5. Moreover, the system can be extended naturally to a disjunctive logic programming paradigm, as presented next, following [1].

Definition 2. *A (*disjunctive logic*)* program *over a domain D is a set P of* rules *of the form $Y \leftarrow X$, where X, Y are clauses over D. An element $e \in D$ is said to be a* model *of P if for every rule $Y \leftarrow X$ in P, if $e \models X$, then $e \models Y$. A clause Y is a* logical consequence *of P if every model of P satisfies Y. We write $\mathsf{cons}(P)$ for the set of all clauses which are logical consequences of P. If T is a theory, we write $\mathsf{cons}(T)$ for the set of all clauses which are logical consequences of T, i.e. $\mathsf{cons}(T)$ is the* logical closure *of T.*

Note that the notions of logical consequence differ for theories and programs. However, given a theory T, we have $\mathsf{cons}(T) = \mathsf{cons}(P_T)$, where $P_T = \{X \leftarrow \{\bot\} \mid X \in T\}$.

[1] An ideal with respect to the *Smyth preorder* $\sqsubseteq^\sharp$, where $X \sqsubseteq^\sharp Y$ if and only if for every $y \in Y$ there exists some $x \in X$ with $x \sqsubseteq y$.

[2] In fact it is also bounded complete.

The (*clause*) *propagation*[3] *rule*

$$\frac{X_1 \quad \dots \quad X_n; \quad a_i \in X_i \quad (\text{all } i); \quad Y \leftarrow Z \in P; \quad \mathsf{mub}\{a_1, \dots, a_n\} \models Z}{Y \cup \bigcup_{i=1}^{n} (X_i \setminus \{a_i\})},$$

denoted by $\mathsf{CP}(P)$, for given program P, was studied in [1]. Applying this rule, we say that $Y \cup \bigcup_{i=1}^{n} (X_i \setminus \{a_i\})$ is a $\mathsf{CP}(P)$-consequence of a theory T if $X_1, \dots, X_n \in T$. The following operator is based on the notion of $\mathsf{CP}(P)$-consequence and acts on logically closed theories. Let T be a logically closed theory over D and let P be a program and define

$$\mathcal{T}_P(T) = \mathsf{cons}\left(\{Y \mid Y \text{ is a } \mathsf{CP}(P)\text{-consequence of } T\}\right).$$

In [1], it was shown that $\mathcal{T}_P$ is a Scott-continuous function on the space of all logically closed theories under set-inclusion, hence has a least fixed point $\mathsf{fix}(\mathcal{T}_P) = \bigsqcup \{\mathcal{T}_P \uparrow n\}$, where $\mathcal{T}_P \uparrow 0 = \mathsf{cons}(\{\{\bot\}\})$ and recursively $\mathcal{T}_P \uparrow (n+1) = \mathcal{T}_P (\mathcal{T}_P \uparrow n)$. It was also shown that $\mathsf{fix}(\mathcal{T}_P) = \mathsf{cons}(P)$.

3 Default Negation

We intend to add a notion of default negation to the logic programming framework presented above. The extension is close in spirit to mainstream developments concerning knowledge representation and reasoning with nonmonotonic logics.

Definition 3. *Let D be a coherent algebraic domain. An* extended clause *is a pair (C, N) of clauses over D, which we also write as "$C, {\sim}N$". An extended clause (C, N) is called* trivially *extended if $N = \{\}$, and we may omit N in this case. A (*trivially*)* extended rule *is of the form $Y \leftarrow X$, where Y is a clause and X is a (trivially) extended clause. An (*extended disjunctive*)* program *consists of a set of extended rules. If $Y \leftarrow C, {\sim}N$ is an extended rule, then we call (C, N) the* body *of the rule and Y the* head *of the rule.*

Informally, we read an extended rule $Y \leftarrow C, {\sim}N$ as follows: If C holds, and N does not, then Y shall hold. This intuition gives rise to the following notions, akin to the answer set semantics [6], a point which we will discuss further in Section 4.

Definition 4. *Let D be a coherent algebraic domain, let P be an extended disjunctive program, and let $w \in D$. We define P/w to be the (non-extended) program obtained by applying the following two transformations: (1) Replace each body (C, N) of a rule by C if $w \not\models N$. (2) Delete all rules with a body (C, N) for which $w \models N$. An element $w \in D$ is an* answer model *of P if it satisfies $w \models \mathsf{fix}\left(\mathcal{T}_{P/w}\right)$. An element $w \in D$ is a* min-answer model *of P if it is minimal among all v satisfying $v \models \mathsf{fix}\left(\mathcal{T}_{P/w}\right)$.*

[3] This rule was called the *hyperresolution rule determined by P* in [1].

Note that every min-answer model is an answer model. Recall also from [1] that the set of all models of a theory is compact saturated, hence is the upper closure of its minimal elements.

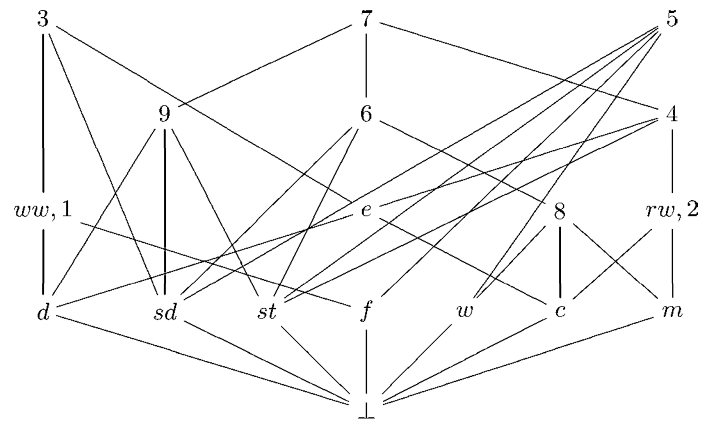
Example 2. Consider the (finite) domain D depicted in Figure 1. This example is taken from [7] and encodes restaurant menues via formal concept analysis, a point which will be discussed in more detail later on in Section 5. We can now encode the wishes of a customer by programs, e.g. as follows.

$$\begin{aligned}\{d\} &\leftarrow \{\bot\}\\ \{2,3,4\} &\leftarrow \{\bot\}\\ \{rw\} &\leftarrow \{\bot\}, \sim\!\{ww\}\end{aligned}$$

Informally, the first rule states that the customer definitely wants a dessert. The second rule states that the customer wants one of the set meals 2, 3 or 4. The third rule states that the customer will choose red wine in all cases in which he does not have a good reason to choose white wine.

The element $4 \in D$ is a min-answer model for P, since $P/4$ consists of the clauses $\{d\} \leftarrow \{\bot\}$, $\{2,3,4\} \leftarrow \{\bot\}$, and $\{rw\} \leftarrow \{\bot\}$, and 4 is a minimal model of $\{\{d\},\{2,3,4\},\{rw\}\}$. Likewise, $3 \in D$ is a min-answer model since $P/3$ consists of the first two clauses from above and 3 is a minimal model of these. $7 \in D$ is an answer model of P, but not a min-answer model.

Fig. 1. Figure for Example 2. Abbreviations are: *sd* salad, *st* starter, *f* fish, *m* meat, *rw* red wine, *ww* white wine, *w* water, *d* dessert, *c* coffee, *e* expensive. Numbers 1 to 9 stand for set meals.



4 Answer Set Programming

Answer set programming is an artificial intelligent reasoning paradigm which was devised in order to capture some aspects of commonsense reasoning. More precisely, it is based on the observation that humans constantly tend to *jump to conlusions* in real-life situations, and on the idea that this imprecise reasoning mechanism (amongst other things) allows us to deal with the world effectively. Formally, *jumping to conclusions* can be studied by investigating supraclassical logics, see [15], where *supraclassicality* means, roughly speaking, that under such a logic more conclusions can be drawn from a set of axioms (or knowledge base) than could be drawn using classical (e.g. propositional or first-order) logic. Answer set programming, as well as the related default logic [3], is also non-monotonic, in the sense that a larger knowledge base does not necessarily yield a larger set of conclusions.

We next describe the notion of answer set for extended disjunctive logic programs, as proposed in [6]. It forms the heart of answer set programming systems like dlv [4], which have become a standard paradigm in artificial intelligence.

Let $\mathcal{V}$ denote a countably infinite set of propositional variables. A *rule* is an expression of the form

$$L_1, \dots, L_n \leftarrow L_{n+1}, \dots, L_m, {\sim}L_{m+1}, \dots, {\sim}L_k,$$

where each of the L_i is a literal, i.e. either a propositional variable or of the form $\neg p$ for some $p \in \mathcal{V}$. Given such a rule r, we set $\mathtt{Head}(r) = \{L_1, \dots, L_n\}$, $\mathtt{Pos}(r) = \{L_{n+1}, \dots, L_m\}$, and $\mathtt{Neg}(r) = \{L_{m+1}, \dots, L_k\}$.

In order to describe the answer set semantics, or stable model semantics, for extended disjunctive programs, we first consider programs without $\sim$. Thus, let P denote an extended disunctive logic program in which $\mathtt{Neg}(r)$ is empty for each rule $r \in P$. A subset X of $\mathcal{V}^{\pm} = \mathcal{V} \cup \neg\mathcal{V}$ is said to be *closed by rules* in P if, for every $r \in P$ such that $\mathtt{Pos}(r) \subseteq X$, we have that $\mathtt{Head}(r) \cap X \neq \emptyset$. The set $X \in 2^{\mathcal{V}^{\pm}}$ is called an *answer set* for P if it is a minimal subset of $\mathcal{V}^{\pm}$ such that the following two conditions are satisfied.

1. If X contains complementary literals, then $X = \mathcal{V}^{\pm}$.
2. X is closed by rules in P.

We denote the set of answer sets of P by $\alpha(P)$. Now suppose that P is an extended disjunctive logic program that may contain $\sim$. For a set $X \in 2^{\mathcal{V}^{\pm}}$, consider the program P/X defined as follows.

1. If $r \in P$ is such that $\mathtt{Neg}(r) \cap X$ is not empty, then we remove r i.e. $r \notin P/X$.
2. If $r \in P$ is such that $\mathtt{Neg}(r) \cap X$ is empty, then the rule r' belongs to P/X, where r' is defined by $\mathtt{Head}(r') = \mathtt{Head}(r)$, $\mathtt{Pos}(r') = \mathtt{Pos}(r)$ and $\mathtt{Neg}(r') = \emptyset$.

The program transformation $(P, X) \mapsto P/X$ is called the *Gelfond-Lifschitz transformation* of P with respect to X.

It is clear that the program P/X does not contain $\sim$ and therefore $\alpha\,(P/X)$ is defined. We say that X is an *answer set* or *stable model* of P if $X \in \alpha\,(P/X)$. So, answer sets of P are fixed points of the operator GL_P introduced by Gelfond and Lifschitz in [6], where $\mathsf{GL}_P(X) = \alpha\,(P/X)$. We note that the operator GL_P is in general not monotonic, and call it the *Gelfond-Lifschitz operator* of P.

Now consider the coherent algebraic cpo $\mathbb{T}^{\mathcal{V}}$ from Example 1, and call an extended program over $\mathbb{T}^{\mathcal{V}}$ a *propositional program* if for each rule $Y \leftarrow (C, N)$ in P we have that Y, C and N contain only atoms in $\mathbb{T}^{\mathcal{V}}$ or $\bot$, i.e. propositional variables or their negations (with respect to $\neg$) or $\bot$, Y does not contain $\bot$, and C is a singleton clause.

Now if P is a propositional program, then let P' be the extended disjunctive logic program obtained from P by transforming each rule

$$\{p_1, \dots, p_n\} \leftarrow \{q_1 \dots q_m\}, \sim\{r_1, \dots, r_k\}$$

from P into the rule

$$p_1, \dots, p_n \leftarrow q_1, \dots, q_m, \sim r_1, \dots, \sim r_k,$$

in P', where p_i, q_j, r_l are atoms in $\mathbb{T}^{\mathcal{V}}$, i.e. literals over $\mathcal{V}$. If $q_1 \dots q_m = \bot$, then it is omitted. If $r_i = \bot$ for some i, then the rule will never play a role, so we can assume without loss of generality that this does not occur. This transformation can obviously be reversed. We say that P and P' are *associated with each other.*

Theorem 1. *Let P be a propositional program and P' be its associated extended disjunctive logic program. Then $w \in \mathbb{T}^{\mathcal{V}}$ is a min-answer model of P if and only if $w' = \{p \in \mathcal{V}^{\pm} \mid w \models \{p\}\}$, i.e. the set of all atoms for which w is a model, is an answer set for P'. Conversely, if $X \subseteq \mathcal{V}^{\pm}$ is an answer set for P' which does not contain complementary literals, then $x = \bigsqcup X \in \mathbb{T}^{\mathcal{V}}$ exists and is a min-answer model of P. If $\mathcal{V}^{\pm}$ is an answer set for P' (and hence the only answer set of P'), then P does not have any min-answer models.*

Theorem 1 shows that reasoning (or programming) with min-answer models encompasses answer set programming with extended disjunctive logic programs. More precisely, we obtain the classical answer set programming paradigm by restricting our attention to the domain $\mathbb{T}^{\mathcal{V}}$. What do we gain through this more general framework? One the one hand, we improve in conceptual clarity: Our results open up the possibility of a domain-theoretical (and domain-logical) treatment of answer set programming in the basic paradigm, and possibly also for some extensions recently being studied. On the other hand, we gain flexibility due to the possible choice of underlying domain, which we like to think of as background knowledge on which we program or which we query. The choice of $\mathbb{T}^{\mathcal{V}}$ corresponds to the language of propositional logic, and all order structures satisfying the requirements of being coherent algebraic cpos are suitable. These requirements are rather weak from a computational perspective, because among the computationally relevant order structures studied in domain theory, coherent algebraic cpos form a rather general class. In particular, they encompass all finite partial orders, and all complete algebraic lattices.

In Section 5 we will actually propose a very general way — using formal concept analysis — of obtaining suitable order structures.

The following theorem is an immediate corollary from Theorem 1.

Theorem 2. *Let P be a propositional program not containing $\sim$ and P' be its associated extended disjunctive program. If $w \in \mathbb{T}^{\mathcal{V}}$ is a minimal model of P then $w' = \{p \in \mathcal{V}^{\pm} \mid w \models \{p\}\}$ is minimally closed by rules in P'. Conversely, if $X \subseteq \mathcal{V}^{\pm}$ is minimally closed by rules in P' and does not contain complementary literals, then $x = \bigsqcup X \in \mathbb{T}^{\mathcal{V}}$ exists and is a minimal model of P. If $\mathcal{V}^{\pm}$ is minimally closed by rules in P' (and thus is the only answer set of P'), then P does not have any models.*

In particular, Theorem 1 shows that the minimal model semantics for definite logic programs [16] can be recovered using the original approach from [1] without default negation. Likewise, the same holds for the stable model semantics for normal logic programs [17], which are non-disjunctive ones without negation $\neg$.

5 Formal Concept Analysis

Formal concept analysis is a powerful lattice-based approach to symbolic data analysis. It was devised in the 1980s [18] and was originally inspired by ideas from philosophy, more precisely by *Port Royal Logic*, which describes a *concept* as consisting of a set of objects (the *extent* of the concept) and a set of attributes (the *intent* of the concept) such that these objects share exactly all these attributes and vice-versa. In the meantime, an active community is driving the field, covering mathematical foundations, logical aspects, and applications in data mining, ontology engineering, artificial intelligence, and elsewhere.

The formation of concepts can be viewed as logical closure in the sense that a set of attributes B implies an attribute m (which may or may not be contained in B), if all objects which fall under all attributes in B also share the attribute m. This will be made more precise below. We thus obtain a notion of logical consequence on attribute sets, respectively a natural implicative theory, which corresponds to so-called *association rules* in data mining. This implicative theory is intimately related to the logic RZ, a point which we mentioned earlier and will study formally in the following. The strong correspondence between the logic RZ and the formation of formal concepts from formal contexts has already been reported in [7] for the case of finite contexts. We will now supplement these results by a theorem which treats the case of infinite contexts.

We first introduce the notions of formal context and concept as used in formal concept analysis. We follow the standard reference [8].

A (*formal*) *context* is a triple (G, M, I) consisting of two sets G and M and a relation $I \subseteq G \times M$. Without loss of generality, we assume that $G \cap M = \emptyset$. The elements of G are called the *objects* and the elements of M are called the *attributes* of the context. For $g \in G$ and $m \in M$ we write gIm for $(g, m) \in I$, and say that g *has the attribute* m.

For a set $A \subseteq G$ of objects we set $A' = \{m \in M \mid gIm \text{ for all } g \in A\}$, and for a set $B \subseteq M$ of attributes we set $B' = \{g \in G \mid gIm \text{ for all } m \in B\}$. A *(formal) concept* of (G, M, I) is a pair (A, B) with $A \subseteq G$ and $B \subseteq M$, such that $A' = B$ and $B' = A$. We call A the *extent* and B the *intent* of the concept (A, B). For singleton sets, e.g. $B = \{b\}$, we simplify notation by writing b' instead of $\{b\}'$.

The set $\mathcal{B}(G, M, I)$ of all concepts of a given context (G, M, I) is a complete lattice with respect to the order defined by $(A_1, B_1) \leq (A_2, B_2)$ if and only if $A_1 \subseteq A_2$, which is equivalent to the condition $B_2 \subseteq B_1$. $\mathcal{B}(G, M, I)$ is called the *concept lattice* of the context (G, M, I).

Remark 1. For every set $B \subseteq M$ of attributes we have that $B' = B'''$, so that (B', B'') is a concept. Hence, the concept lattice of a context (G, M, I) can be identified with the set $\{B'' \mid B \subseteq M\}$, ordered by reverse subset inclusion.

Furthermore, if $m \in M$ is an attribute, then we call $(m', m'') = (\{m\}', \{m\}'')$ an *attribute concept*. Dually, if $g \in G$ is an object, then we call $(g'', g') = (\{g\}'', \{g\}')$ an *object concept*. The subposet L of $\mathcal{B}(G, M, I)$ consisting of all attribute and object concepts is called the *Galois subhierarchy* or *AOC* associated with (G, M, I). By abuse of notation, we denote members of L by elements from $G \cup M$. This is justified by the obvious possibility to identify the set L with $(G \cup M)/_\sim$, where $\sim$ is the equivalence relation identifying each two elements in $G \cup M$ whose associated concepts coincide. We denote the induced order on L by $\leq$.

Theorem 3. *Let (G, M, I) be a formal context, $\mathcal{B}(G, M, I)$ be the corresponding formal concept lattice, and $(L, \leq)$ be the Galois subhierarchy associated with (G, M, I). Let $(D, \sqsubseteq)$ be a coherent algebraic cpo and $\iota : L \to D$ be an order-reversing injective function which covers all of $\mathsf{K}(D)$, i.e. for each $c \in \mathsf{K}(D)$ there exists some $a \in L$ with $\iota(a) = c$. Furthermore, let $A = \{m_1, \ldots, m_n\} \subseteq M$ such that $\iota(m_i) \in \mathsf{K}(D)$ for all i. Then*

$$A'' = \{m \mid \{\{\iota(m_1)\}, \ldots, \{\iota(m_n)\}\} \models \{\iota(m)\}\}.$$

We remark that Theorem 3 applies to all finite contexts since the Galois subhierarchy of a finite context is always (and trivially) a coherent algebraic cpo where all elements are compact; a bottom element may have to be added, though. This finite case is also a corollary from [7, Theorem 3], taking Example 1 and Proposition 1 from [7] into account.

The following example is taken from [7]; it complements Example 2.

Example 3. Consider the formal context given in Table 1. It shall represent, in simplified form, a selection of set dinners from a restaurant menu. The Galois subhierarchy of its formal concept lattice is depicted in Figure 1. Concepts in this setting correspond to types of dinners, e.g. one may want to identify the concept with extent $\{4, 6, 7\}$ and intent $\{st, m, c\}$, using the abbreviations from Figure 1, to be the *heavy* meals, while the *expensive* ones are represented by the attribute concept of e, and turn out to always include coffee. Using the logic

Table 1. Formal context for Example 3.

	salad	starter	fish	meat	red wine	white wine	water	dessert	coffee	expensive
1			×			×		×		
2				×	×				×	
3	×		×			×		×	×	×
4		×		×	×			×	×	×
5	×	×	×				×			
6	×	×		×			×		×	
7	×	×		×	×		×	×	×	×
8				×			×		×	
9	×	×						×		

RZ, we can for example conclude that a customer who wants salad and fish will choose one of the meals 3 or 5, since these elements of the poset are exactly those which are both objects and models of the theory $\{\{sd\},\{f\}\}$. Also, he will always get a starter or a dessert, formally $\{\{sd\},\{f\}\} \models \{st,d\}$. To give a slightly more sophisticated example, suppose that a customer wants salad or a starter, additionally fish or a dessert, and drinks water. From this we can conclude that in any case he will get both a salad and a starter. Formally, we obtain $\{\{sd,st\},\{f,d\},\{w\}\} \models \{sd\}$ and $\{\{sd,st\},\{f,d\},\{w\}\} \models \{st\}$. A little bit of reflection on the context makes it clear that these inferences are indeed natural ones.

Let us stop for a moment and dwell on the significance of Theorem 3. We note first of all that the hypothesis is not very strong from a domain-theoretic perspective: we encompass all concept lattices for which some corresponding Galois subhierarchy forms at least an abstract basis for a coherent algebraic cpo. One could argue that such or similar conditions have to be satisfied in any case if one intends to perform computation on an infinite order structure. The conclusion of the theorem then says that concept closure (or in other words, the underlying implicative theory) basically coincides with consequence in RZ, restricted to finite sets of singleton clauses, which can be interpreted as conjunctions of elements or items from $G \cup M$. The logic RZ then lifts concept closure to become part of disjunctive reasoning, in a natural and intuitively appealing way. From this perspective we can say that the logic RZ *is* the implicative theory obtained from concept closure, naturally extended with a notion of disjunction.

What we gain from this perspective is not only a tight relationship between formal concept analysis and domain theory, but also a non-monotonic reasoning paradigm on conceptual knowledge, by utilizing our results in Section 4. Formal contexts can now be interpreted as providing *background knowledge* in elementary form, which can be queried, or programmed on, by using disjunctive logic programs with default negation, as described in Section 4. From this, we obtain a clear distinction between the (monotonic!) background knowledge or underlying database, and the program written on top of it, allowing for a clear separation of the nonmonotonic aspects which are diffcult to deal with efficiently and effectively.

6 Related Work

Logical aspects of formal concept analysis have certainly received ample attention in the literature, see e.g. [19, 20]. In particular, the *contextual attribute logic* due to Ganter and Wille [19] is closely related to our results in Section 5, and for the finite case this was spelled out in [7].

The study of relationships between formal concept analysis and domain theory has only recently received attention. Zhang and Shen [21, 22] approach the issue from the perspective of Chu spaces and Scott information systems. A category-theoretical setting was developed from these investigations in [23]. The work just mentioned has a different focus than our result in Section 5 and [7], but develops along similar basic intuitions and is mainly compatible with ours. Its flavour is more category-theoretical and targets categorial constructions which may be used for ontology engineering.

Osswald and Petersen [24, 25] study an approach to encoding knowledge in order structures which is inspired from linguistics. They obtain a framework which is more flexible than formal concept analysis, and appears to be compatible with our results in Section 5 and [7]. They also propose a default reasoning paradigm, but it remains to be worked out how it relates to ours.

Relationships between domain theory and nonmonotonic reasoning have hardly been studied in the literature, except from series of papers by Rounds and Zhang, e.g. [1, 26, 27], and Hitzler and Seda, e.g. [28–30]. This is remarkable since domain theory has become a respected paradigm in the theory of computing with widespread applications. We believe that this relationship deserves much more attention in order to understand the theoretical underpinnings of nonmonotonic reasoning and other artificial intelligence paradigms.

Default reasoning on concept hierarchies has also been studied before, for example in the form of default reasoning in semantic networks, e.g. [31], and as nonmonotonic reasoning with ontologies, e.g. [32, 33]. Since ontology creation is a currently evolving area of application for formal concept analysis, we expect that our paradigm will also be useful for similar purposes. Another related paradigm is logic programming with inheritance [34], where the underlying order structures are is-a hierarchies, which do not have a similarly rich logical structure as the logic RZ or Galois subhierarchies of formal concept lattices.

7 Conclusions and Further Work

The work presented in this paper touches domain theory, nonmonotonic reasoning, and symbolic data analysis. The contribution should mainly be considered as an inspiration for further investigations which grow naturally out of our observations. There are several starting points for such work, and some of them bear potential for full research projects which are interesting in their own right.

Concerning the relations worked out between the logic RZ and nonmonotonic reasoning, we have described a general reasoning framework which encompasses answer set programming with extended disjunctive programs as a special case, namely with the domain restricted to $\mathbb{T}^{\mathcal{V}}$. This opens up new ways for domain-theoretic analysis for nonmonotonic reasoning in this paradigm, with the hope that e.g. decidability aspects could be tackled — an issue which has so far received only little attention in the nonmonotonic reasoning community. On the other hand, by substituting $\mathbb{T}^{\mathcal{V}}$ by other domains, it should be possible to lift answer set programming out of the restricted syntax provided by the fragment of first-order logic usually considered.

Concerning the relations between the logic RZ and formal concept analysis displayed in Section 5, we can understand the logic RZ as a means of reasoning with conceptual knowledge, related to the approach presented in [19], as already mentioned in [7]. Indeed, the choice of $\mathbb{T}^{\mathcal{V}}$ as underlying domain relates to answer set programming, while the choice of other domains can be motivated by formal concept analysis. Of particular interest are also the infinitary aspects of this, and the potential of the domain-theoretic approach to deal with questions of computability and query-answering even on infinite contexts. From this perspective, it should be investigated under which conditions a context satisfies the hypotheses of Theorem 3. It would also be important to relate this result to those of [21], where domain theory and formal concept analysis are being related by means of Chu space theory, and [24, 25], where a general approach encompassing formal concept analysis is described for obtaining order structures carrying hierarchical knowledge.

Finally, we would like to emphasize that the results presented here lead to a nonmonotonic reasoning paradigm on conceptual knowledge. More precisely, starting from a given (and possibly infinite) context, we have provided means for doing nonmonotonic reasoning on the Galois subhierarchy of the context. Since the logic RZ captures the notion of concept closure, we obtain a reasoning paradigm dealing with conceptual knowledge in a way very natural to formal concept analysis. On the other hand, the nonmonotonic reasoning paradigm thus put in place is very close in spirit to mainstream developments in answer set programming, and can thus benefit from the experience gained within this field of research.

We believe that the resulting nonmonotonic reasoning paradigm with conceptual knowledge bears potential for applications. One could envisage background knowledge in the form of formal contexts, and sophisticated queries or planning tasks expressed by programs. We are not aware of any other work which proposes a default reasoning paradigm on conceptual knowledge compatible with mainstream research developments in nonmonotonic reasoning.

References

1. Rounds, W.C., Zhang, G.Q.: Clausal logic and logic programming in algebraic domains. Information and Computation **171** (2001) 156–182

2. Abramsky, S., Jung, A.: Domain theory. In Abramsky, S., Gabbay, D., Maibaum, T.S., eds.: Handbook of Logic in Computer Science. Volume 3. Clarendon, Oxford (1994)
3. Reiter, R.: A logic for default reasoning. Artificial Intelligence **13** (1980) 81–132
4. Eiter, T., Leone, N., Mateis, C., Pfeifer, G., Scarcello, F.: A deductive system for nonmonotonic reasoning. In Dix, J., Furbach, U., Nerode, A., eds.: Proceedings of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'97. Volume 1265 of Lecture Notes in Artificial Intelligence., Springer, Berlin (1997)
5. Simons, P., Niemelä, I., Soininen, T.: Extending and implementing the stable model semantics. Artificial Intelligence (200x) To appear.
6. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Computing **9** (1991) 365–385
7. Hitzler, P., Wendt, M.: Formal concept analysis and resolution in algebraic domains. In de Moor, A., Ganter, B., eds.: Using Conceptual Structures — Contributions to ICCS 2003, Shaker Verlag, Aachen (2003) 157–170
8. Ganter, B., Wille, R.: Formal Concept Analysis — Mathematical Foundations. Springer, Berlin (1999)
9. Scott, D.S.: Domains for denotational semantics. In Nielsen, M., Schmidt, E.M., eds.: Automata, Languages and Programming, 9th Colloquium, July 1982, Aarhus, Denmark, Proceedings. Volume 140 of Lecture Notes in Computer Science., Springer, Berlin (1982) 577–613
10. Hofmann, K.H., Mislove, M.W.: Local compactness and continuous lattices. In Banaschewski, B., Hofmann, R., eds.: Continuous Lattices, Proceedings. Volume 871 of Lecture Notes in Mathematics., Springer-Verlag (1981) 209–248
11. Plotkin, G.: T^ω as a universal domain. Journal of Computer and System Sciences **17** (1978) 209–236
12. Fitting, M.: A Kripke-Kleene-semantics for general logic programs. The Journal of Logic Programming **2** (1985) 295–312
13. Hitzler, P.: A resolution theorem for algebraic domains. In Gottlob, G., Walsh, T., eds.: Proceedings of the 18th International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 2003, Morgan Kaufmann Publishers (2003) 1339–1340
14. Hitzler, P.: A generalized resolution theorem. Journal of Electrial Engineering, Slovak Academy of Sciences **55** (2003) 25–30
15. Makinson, D.: Bridges between classical and nonmonotonic logic. Logic Journal of the IGPL **11** (2003) 69–96
16. Lloyd, J.W.: Foundations of Logic Programming. Springer, Berlin (1988)
17. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In Kowalski, R.A., Bowen, K.A., eds.: Logic Programming. Proceedings of the 5th International Conference and Symposium on Logic Programming, MIT Press (1988) 1070–1080
18. Wille, R.: Restructuring lattice theory: An approach based on hierarchies of concepts. In Rival, I., ed.: Ordered Sets. Reidel, Dordrecht-Boston (1982) 445–470
19. Ganter, B., Wille, R.: Contextual attribute logic. In Tepfenhart, W.M., Cyre, W.R., eds.: Conceptual Structures: Standards and Practices. Proceedings of the 7th International Conference on Conceptual Structures, ICCS '99, July 1999, Blacksburgh, Virginia, USA. Volume 1640 of Lecture Notes in Artificial Intelligence., Springer, Berlin (1999) 377–388

20. Wille, R.: Boolean judgement logic. In Delugach, H., Stumme, G., eds.: Conceptual Structures: Broadening the Base, Proceedings of the 9th International Conference on Conceptual Structures, ICCS 2001, July 2001, Stanford, LA, USA. Volume 2120 of Lecture Notes in Artificial Intelligence., Springer, Berlin (2001) 115–128
21. Zhang, G.Q.: Chu spaces, concept lattices, and domains. In: Proceedings of the Nineteenth Conference on the Mathematical Foundations of Programming Semantics, March 2003, Montreal, Canada. Volume 83 of Electronic Notes in Theoretical Computer Science. (2003)
22. Zhang, G.Q., Shen, G.: Approximable concepts, Chu spaces, and information systems. Theory and Applications of Categories (200x) To appear.
23. Hitzler, P., Zhang, G.Q.: A cartesian closed category of approximable concept structures. In Pfeiffer, H., Wolff, K., eds.: Proceedings of the International Conference On Conceptual Structures, Huntsville, Alabama, USA. Lecture Notes in Computer Science, Springer (2004) To appear.
24. Osswald, R.: Assertions, conditionals, and defaults. In: Proceedings of the 1st Workshop on Conditionals, Information, and Inference. Lecture Notes in Artificial Intelligence (200x) To appear.
25. Osswald, R., Petersen, W.: A logical approach to data driven classification. In Günter, A., Kruse, R., Neumann, B., eds.: KI-2003: Advances in Artificial Intelligence. Volume 2821 of Lecture Notes in Artificial Intelligence., Springer (2003) 267–281
26. Zhang, G.Q., Rounds, W.C.: Reasoning with power defaults (preliminary report). In Dix, J., Furbach, U., Nerode, A., eds.: Proceedings of the Fourth International Conference on Logic Programming and Non-Monotonic Reasoning, LPNMR'97, Dagstuhl, Germany. Volume 1265 of Lecture Notes in Computer Science., Springer (1997) 152–169
27. Zhang, G.Q., Rounds, W.C.: Semantics of logic programs and representation of Smyth powerdomains. In Keimel, K., et al., eds.: Domains and Processes. Kluwer (2001) 151–179
28. Hitzler, P., Seda, A.K.: Some issues concerning fixed points in computational logic: Quasi-metrics, multivalued mappings and the Knaster-Tarski theorem. In: Proceedings of the 14th Summer Conference on Topology and its Applications: Special Session on Topology in Computer Science, New York. Volume 24 of Topology Proceedings. (1999) 223–250
29. Hitzler, P., Seda, A.K.: Generalized metrics and uniquely determined logic programs. Theoretical Computer Science **305** (2003) 187–219
30. Seda, A.K., Hitzler, P.: Topology and iterates in computational logic. In: Proceedings of the 12th Summer Conference on Topology and its Applications: Special Session on Topology in Computer Science, Ontario, August 1997. Volume 22 of Topology Proceedings. (1997) 427–469
31. Shastri, L.: Default reasoning in semantic networks: A formalization of recognition and inheritance. Artificial Intelligence **39** (1989) 283–355
32. Baader, F., Hollunder, B.: Embedding defaults into terminological representation systems. J. Automated Reasoning **14** (1995) 149–180
33. Donini, F.M., Nardi, D., Rosati, R.: Description logics of minimal knowledge and negation as failure. ACM Trans. Comput. Logic **3** (2002) 177–225
34. Buccafurri, F., Leone, N.: Disjunctive logic programs with inheritance. Theory and Practice of Logic Programming **2** (2002) 293–321

Appendix: Proofs

Theorem 1 Let P be a propositional program and P' be its associated extended disjunctive logic program. Then $w \in \mathbb{T}^{\mathcal{V}}$ is a min-answer model of P if and only if $w' = \{p \in \mathcal{V}^{\pm} \mid w \models \{p\}\}$, i.e. the set of all atoms for which w is a model, is an answer set for P'. Conversely, if $X \subseteq \mathcal{V}^{\pm}$ is an answer set for P' which does not contain complementary literals, then $x = \bigsqcup X \in \mathbb{T}^{\mathcal{V}}$ exists and is a min-answer model of P. If $\mathcal{V}^{\pm}$ is an answer set for P' (and hence the only answer set of P'), then P does not have any min-answer models.

Proof. Note that $(P/w)' = (P'/w')$, so we first restrict our attention to programs without $\sim$. Let Q, Q' be such programs associated with each other. We show first that for every min-answer model v for Q we have that

$$v' = \{p \in \mathcal{V}^{\pm} \mid v \models \{p\}\}$$

is closed by rules in Q'. Then we show that for every answer set X of Q' we have that $x = \bigsqcup X$ is an answer model of Q. Then we proceed with showing that v' is an answer set if v is a min-answer model, and finally that $x = \bigsqcup X$ is a min-answer model for Q whenever X is an answer set for Q'.

So let $v \in \mathbb{T}^{\mathcal{V}}$ be a min-answer model for Q. Then v is minimal among all z with $z \models \mathsf{fix}(\mathcal{T}_{Q/v}) = \mathsf{fix}(\mathcal{T}_Q) = \mathsf{cons}(Q)$, i.e. v is a minimal model of Q. Note that $v = \bigsqcup v' = \bigsqcup\{p \in \mathcal{V}^{\pm} \mid v \models \{p\}\}$, so v' cannot contain complementary literals, since two complementary literals do not have a common upper bound in $\mathbb{T}^{\mathcal{V}}$. We show next that v' is closed by rules in Q'. So let $p_1, \dots, p_n \leftarrow q_1, \dots, q_m$ be a rule in Q' such that $q_1, \dots, q_m \in v'$. But then $v \models \bigsqcup\{q_1, \dots, q_m\} = q_1 \dots q_m$ and $\{p_1, \dots, p_n\} \leftarrow \{q_1 \dots q_m\}$ is a clause in Q. Since v is a model of Q we obtain $v \models \{p_1, \dots, p_n\}$ and hence $v \models \{p_i\}$ for some i, which implies $p_i \in v'$ as desired.

Now let $X \subseteq \mathcal{V}^{\pm}$ be an answer set for Q' which does not contain complementary literals. Then $x = \bigsqcup X$ exists, and we show that it is an answer model of Q. So let $\{p_1, \dots, p_n\} \leftarrow \{q_1 \dots q_m\}$ be a clause in Q with $x \models \{q_1 \dots q_m\}$. Then $\{q_1, \dots, q_m\} \subseteq X$, hence $p_i \in X$ for some i, and consequently $x \models \{p_i\}$ as desired.

For a min-answer model v for Q we know already that v' does not contain complementary literals and is closed by rules in Q'. Now let $Y \subseteq v'$ be an answer set for Q'. Then $\bigsqcup Y$ exists and $\bigsqcup Y \sqsubseteq \bigsqcup v' = v$, hence $\bigsqcup Y = v$ by minimality of v, and consequently $Y = v'$, so v' is an answer set for Q'.

For an answer set X for Q' we know already that $x = \bigsqcup X$ is an answer model of Q. Now let $y \sqsubseteq x$ be a min-answer model for Q. Then $y' = \{p \in \mathcal{V}^{\pm} \mid y \models \{p\}\} \subseteq X$ is an answer set for Q', hence $y' = X$ by minimality of X, and consequently $y = x$, so x is a min-answer model for Q.

This closes the proof for programs without $\sim$. Now let P be a propositional program including $\sim$.

Let w be a min-answer model for P, hence w is minimal among all v with $v \models \mathsf{fix}(\mathcal{T}_{P/w})$, and in particular w is a min-answer model for P/w. So $w' = \{p \in \mathcal{V}^{\pm} \mid w \models \{p\}\}$ is an answer set for $(P/w)' = P'/w'$ as desired.

Let X be an answer set for P' which does not contain complementary literals. Then X in particular is an answer set for $P'/X = (P/\bigsqcup X)'$, hence $x = \bigsqcup X$ is a min-answer model for P/x, and consequently also a min-answer set model P as desired.

If X is an answer set for P' containing complementary literals then $X = \mathcal{V}^{\pm}$, and X is the only answer set for P'. Now if P had a min-answer model w, then $w' = \{p \in \mathcal{V}^{\pm} \mid w \models \{p\}\} \subseteq X$ were an answer set for P. But then $w' = X$ which is impossible since this implies $w = \bigsqcup X$, but the supremum does not exist.

Theorem 3 Let (G, M, I) be a formal context, $\mathcal{B}(G, M, I)$ be the corresponding formal concept lattice, and $(L, \leq)$ be the Galois subhierarchy associated with (G, M, I). Let $(D, \sqsubseteq)$ be a coherent algebraic cpo and $\iota : L \to D$ be an order-reversing injective function which covers all of $\mathsf{K}(D)$, i.e. for each $c \in \mathsf{K}(D)$ there exists some $a \in L$ with $\iota(a) = c$. Furthermore, let $A = \{m_1, \ldots, m_n\} \subseteq M$ such that $\iota(m_i) \in \mathsf{K}(D)$ for all i. Then

$$A'' = \{m \mid \{\{\iota(m_1)\}, \ldots, \{\iota(m_n)\}\} \models \{\iota(m)\}\}.$$

Proof. Let m be such that $\{\{\iota(m_1)\}, \ldots, \{\iota(m_n)\}\} \models \{\iota(m)\}$. We have to show that $m \in A''$. So let $g \in G$ be such that gIm_i for all i, which implies $g \leq m_i$ for all i. So $\iota(g) \sqsupseteq \iota(m_i)$ for all i, i.e. $\iota(g) \models \{\{\iota(m_1)\}, \ldots, \{\iota(m_n)\}\}$, and consequently $\iota(g) \models \{\iota(m)\}$. It follows that $\iota(g) \sqsupseteq \iota(m)$ which implies $g \leq m$, hence gIm. Since g was chosen arbitrarily, we conclude $m \in A''$.

Conversely, let $m \in A''$ and let w be chosen arbitrarily with $w \in \mathsf{mub}\,\iota(A)$. Then it remains to show that $\iota(m) \sqsubseteq w$ since this implies $\iota(m) \sqsubseteq x$ for all x with $x \models \{\{\iota(m_1)\}, \ldots, \{\iota(m_n)\}\}$ by arbitrary choice of w, and hence

$$\{\{\iota(m_1)\}, \ldots, \{\iota(m_n)\}\} \models \{\iota(m)\}$$

as desired.

In order to show $\iota(m) \sqsubseteq w$ first note that $w \in \mathsf{K}(D)$ by coherency and our assumption on A. We consider the two cases (i) $\iota^{-1}(w) \in G$ and (ii) $\iota^{-1}(w) \in M$.

(i) If $\iota^{-1}(w) = g \in G$, then $g \leq m_i$ for all i, hence gIm_i for all i and consequently gIm. We obtain $g \leq m$, hence $w = \iota(g) \sqsupseteq \iota(m)$ as desired.

(ii) If $\iota^{-1}(w) = a \in M$, then $a \leq m_i$ for all i, hence $a' \subseteq m_i'$ for all i. So for all g with gIa we have gIm_i for all i, and by $m \in A''$ we obtain gIm. We have just shown that $g \in a'$ implies $g \in m'$, so $a' \subseteq m'$, which is equivalent to $a \leq m$. Hence $w = \iota(a) \sqsupseteq \iota(m)$ as desired.

Eingereicht bei Theoretical Computer Science

A Categorical View on Algebraic Lattices in Formal Concept Analysis

Pascal Hitzler[1]
Markus Krötzsch[2]
Guo-Qiang Zhang[3]

[1] Institut AIFB, Universität Karlsruhe, Germany.
[2] Fakultät für Informatik, Technische Universität Dresden, Germany.
[3] Department of Electrical Engineering and Computer Science, Case Western Reserve University, Cleveland, Ohio, U.S.A.

Abstract

Formal concept analysis has grown from a new branch of the mathematical field of lattice theory to a widely recognized tool in Computer Science and elsewhere. In order to fully benefit from this theory, we believe that it can be enriched with notions such as *approximation by computation* or *representability*. The latter are commonly studied in denotational semantics and domain theory and captured most prominently by the notion of *algebraicity*, e.g. of lattices. In this paper, we explore the notion of algebraicity in formal concept analysis from a category-theoretical perspective. To this end, we build on the the notion of *approximable concept* with a suitable category and show that the latter is equivalent to the category of algebraic lattices. At the same time, the paper provides a relatively comprehensive account of the representation theory of algebraic lattices in the framework of Stone duality, relating well-known structures such as Scott information systems with further formalisms from logic, topology, domains and lattice theory.

1 Introduction

Algebraic lattices conveniently represent computationally relevant properties. As partial orders they allow for the expression of amounts of *information content*. Distinguished elements — called *compact or finite* — stand for computationally representable information. Every element or information item not directly representable can be approximated by representable, i.e. compact, items.

So algebraic lattices can be identified as computationally relevant structures, and as such have found applications in Computer Science, most prominently in the theory of denotational semantics, domain theory (see, e.g. [AJ94]), but recently also in aspects regarding knowledge representation and reasoning (see e.g. [RZ01, ZR04, Hit04]).

As can be expected from rich mathematical structures such as algebraic lattices, a multitude of possible characterizations have been established, ranging from the classical correspondence between algebraic lattices and their semilattices of compacts [GHK$^+$03], over logical characterizations such as Scott information systems [Sco82a], to topological investigations via the Scott topology [Joh82, Abr91]. Following Abramsky's programme of *domain theory in logical form*, each of these representations is associated with either the *spacial* or the *localic* side of Stone duality: the former includes syntactical, logical, and axiomatic formalisms, while the latter typically incorporates semantical, observational, and denotational aspects. The equivalence of both worlds leads to rather pleasant results of soundness and completeness of corresponding proof systems and model theories.

We add to this collection by a representation of algebraic lattices based on the framework of formal concept analysis (FCA, [GW99]). Originally, FCA was conceived as an alternative formulation of the theory of complete lattices, motivated by philosophical considerations [Wil82]. In the meantime, FCA has grown from a new branch of lattice theory to a widely recognized tool in Computer Science (see, e.g., [Stu02]). Prominent applications concern areas such as Data- and Textmining, Knowledge Representation and Reasoning, Semantic Web, Computational Linguistics. FCA starts from *formal contexts*, syntactical descriptions of object-attribute relations, and lifts them to closure operators and complete lattices. While this suggests a logical viewpoint based on the given (deductive) closure, the derived logical entailment lacks the important property of compactness: some conclusions can only be drawn from infinite sets of premises [Zha03a]. This motivates a deviation from the classical definition of closures in FCA to ensure Scott continuity of the derived closure operators (the so called algebraic ones), thus recovering compactness and switching to complete lattices that are algebraic. We achieve this by introducing the (complete algebraic) lattice of *approximable concepts* [ZS0x], obtained from given object-attribute relations analogous to the classical construction used in FCA, but at the same time conforming to the insights concerning computationally relevant structures as studied in domain theory.

The strong interest in algebraic lattices indeed stems only in part from the appealing way in which these structures capture the possibility of approximating infinite computation by finite elements. The full strength of the theory only becomes apparent when Scott continuous functions are employed as morphisms of a category **Alg** of algebraic lattices. The interplay between the lattices and these morphisms is highly satisfactory: the set of all Scott continuous functions between two algebraic lattices can again be viewed as an algebraic lattice, and **Alg** is in fact cartesian closed. Consequently, we augment the above characterizations of algebraic lattices by suitable notions of morphisms, inducing in each case a category that is equivalent to **Alg**. We thus obtain a cartesian closed category of formal contexts corresponding to the new notion of approximable concepts.

At the same time, this article gives a relatively comprehensive account of the numerous representations of algebraic lattices by offering a fresh, unified, and largely self-contained treatment of the theory. The new approach via FCA provides additional insights into the nature of the well-known formalisms. In particular, we give a direct proof of the cartesian closedness of the new category of formal contexts, hence obtaining novel categorical product and function space constructions based on formal contexts. Particularly, the formulation of function spaceenhances our understanding of *approximable mappings*, the class of morphisms Scott conceived for his information systems [Sco82a]. Indeed, these relations turn out to be immediate descriptions of sets of step functions, sufficient to capture all Scott continuous functions between the corresponding algebraic lattices.

Our discussion will also expose the connections between algebraic lattices and the conjunctive fragment of propositional logic — an approach that appears to be rather intuitive from the viewpoint of Computer Science and also brings to bear on the results from [HW03, Hit04]. As encompassed in [Abr91], this is achieved through the Lindenbaum algebras of these logics. Our profit, besides finding a simple access to Scott information systems, is an alternative view on approximable mappings as *multilingual sequent calculi*, as considered in [JKM99] for more expressive logics.

The structure of this paper is as follows. In the next Section 2 the most fundamental definitions from order theory, topology, and category theory are recalled. Section 3 starts the discussion of algebraic lattices from a domain theoretic perspective, with special emphasis on the role of the semilattice of compact elements. Thereafter, Section 4 introduces appropriate notions of morphisms for such semilattices, which are shown to be equivalent to Scott continuous functions between the corresponding algebraic lattices. Section 5 then introduces a category of formal contexts equivalent to the algebraic lattices and Scott continuous functions, and gives an explicit proof of the cartesian closure of this new category. Building on the prototypical categorical equivalences established earlier, Section 6 introduces further representation theorems from logic and topology, which are then connected using Stone duality. Finally, Section 7 gives pointers to further literature and hints at possible extensions of given results.

A very preliminary report on some of the results in this paper has appeared as [HZ04]. The notion of *approximable concept* has first been proposed in [ZS0x], but without exploring its category-theoretical content.

Acknowledgements. The first author acknowledges support by the German Ministry of Education and Research under the SmartWeb project, and by the European Union under the KnowledgeWeb network of excellence. The second author gratefully acknowledges support by *Case Western Reserve University*, Cleveland/Ohio, where most of his work was carried out, and sponsorship by the *German Academic Exchange Service* (DAAD) and by the *Gesellschaft von Freunden und Förderern der TU Dresden e.V.*

2 Preliminaries and Notation

We first give some basic definitions of order theory at least *to fix notations*. Our main reference will be [GHK+03]. A more gentle first introduction is given in [DP02].

A partially ordered set (poset) is a set P with a reflexive, symmetric, transitive relation $\leq \subseteq P \times P$. If $(P, \leq)$ is a poset, then its dual is the poset $(P, \geq)$. We denote posets by their carrier set as long as the partial order is clear from the context.

Definition 2.1 Consider a poset L. A non-empty subset $D \subseteq L$ is *directed* if, for any $x, y \in D$, there is some element $z \in D$ such that $x \leq z$ and $y \leq z$. If every directed subset $D \subseteq L$ has a least upper bound (supremum, join) $\bigvee D$ in L, then L is a *directed complete partial order* (dcpo).

L is a *complete lattice* if every subset $S \subseteq L$ has a least upper bound $\bigvee S$ and a greatest lower bound (infimum, meet) $\bigwedge S$. For a set X, 2^X denotes the powerset lattice, i.e. the complete lattice of all subsets of X under inclusion.

We recall that a poset that has all infima also has all suprema, and vice versa, so that one of these conditions is in fact sufficient. Furthermore we fix some basic terminologies about lattices.

Definition 2.2 A poset L is a *lattice* if every two elements of L have a supremum and an infimum. These meets and joins of binary sets will be written in infix: $\bigvee\{x, y\} = x \vee y$ and $\bigwedge\{x, y\} = x \wedge y$. L is distributive if, for all $x, y, z \in L$, one finds $x \wedge (y \vee z) = (x \vee y) \wedge (x \vee z)$.

An element $x \in L$ is called

- *meet-irreducible* if $y \wedge z = x$ implies $y = x$ or $z = x$,
- *meet-prime* if $y \wedge z \leq x$ implies $y \leq x$ or $z \leq x$.

Join-irreducible and join-prime elements are defined dually.

In a distributive lattice, the meet-irreducibles are exactly the meet-primes, and this will be the only case considered in this paper. Furthermore we want to talk about functions between partially ordered sets.

Definition 2.3 Consider posets P and Q, and a function $f : P \to Q$. Then f is *monotone* if it preserves the order of P, i.e. $x \leq y$ in P implies $f(x) \leq f(y)$ in Q. Moreover, *f preserves (directed) suprema* if, for any (directed) $S \subseteq P$ such that $\bigvee S$ exists, one finds that $\bigvee f(S) = \bigvee\{f(s) \mid s \in S\}$ exists and that $f(\bigvee S) = \bigvee f(S)$. Preservation of infima is defined dually. A function that preserves directed suprema is also called *Scott continuous*. An *order-isomorphism* is a bijective monotone function that has a monotone inverse.

Note that preservation of directed suprema (infima) always entails monotonicity, since every pair of elements $x \leq y$ induces a directed set $\{x, y\}$ for which preservation of suprema (infima) implies $f(x) \leq f(y)$ as required.

We also need a little general topology. Our view on topology largely agrees with [Smy92].

Definition 2.4 Let X be a set and let $\tau \subseteq 2^X$ be a system of subsets of X. (X, τ) is a *topological space* if τ contains X and the empty set, and is closed under arbitrary unions and finite intersections. The members of such a system τ are called *open sets* and the complete lattice $(\tau, \subseteq)$ is called the *open set lattice*. The complements of open sets are the *closed sets*. If confusion is unlikely, we will denote topological spaces by their sets of points. For a topological space X, we also use $\Omega(X)$ to denote its open set lattice.

A subset B of τ is a *base* of τ if every open set is equal to the union of all members of B it contains.

The appropriate mappings between topological spaces are *continuous functions*.

Definition 2.5 Consider topological spaces X and Y, and a function $f : X \to Y$. Then f is *continuous* if its inverse image preserves open sets, i.e. for every open set $O \subseteq Y$, the set $f^{-1}(O) = \{x \in X \mid f(x) \in O\}$ is open in X. If f is bijective and both f and f^{-1} are continuous then f is a *homeomorphism*. The topological spaces X and Y are said to be *homeomorphic* if a homeomorphism between them exists.

Finally, a minimum amount of category theory is utilized in this paper, in order to present relationships of the different concepts to their full extent. Our terminology follows [Bor94]. Other good references include [Mac71], and the more easy-paced introductions [LR03] and [McL92]. A *category* **C** consists of

(i) a class $|\mathbf{C}|$ of *objects* of the category,

(ii) for all $A, B \in |\mathbf{C}|$, a set $\mathbf{C}(A, B)$ of *morphisms* from A to B,

(iii) for all $A, B, C \in |\mathbf{C}|$, a composition operation
$\circ : \mathbf{C}(B, C) \times \mathbf{C}(A, B) \to \mathbf{C}(A, C)$,

(iv) for all $A \in |\mathbf{C}|$, an *identity morphism* $\mathrm{id}_A \in \mathbf{C}(A, A)$,

such that for all $f \in \mathbf{C}(A, B)$, $g \in \mathbf{C}(B, C)$, $h \in \mathbf{C}(C, D)$, the associativity axiom $h \circ (g \circ f) = (h \circ g) \circ f$, and the identity axioms $\mathrm{id}_B \circ f = f$ and $g \circ \mathrm{id}_B = g$ are satisfied. As usual, we write $f : A \to B$ for morphisms $f \in \mathbf{C}(A, B)$. The *opposite* $\mathbf{C}^{\mathsf{op}}$ of a category $\mathbf{C}$ is defined by setting $|\mathbf{C}^{\mathsf{op}}| = |\mathbf{C}|$ and $\mathbf{C}^{\mathsf{op}}(A, B) = \mathbf{C}(B, A)$. A morphism $f : A \to A'$ is an *isomorphism*, if it has an inverse, i.e. if there is a (necessarily unique) morphism $g : A' \to A$ with $g \circ f = \mathrm{id}_A$ and $f \circ g = \mathrm{id}_{A'}$.

A *functor* F from a category $\mathbf{A}$ to a category $\mathbf{B}$ consists of

(i) a mapping $|\mathbf{A}| \to |\mathbf{B}|$ of objects, where the image of an object $A \in |\mathbf{A}|$ is denoted by $\mathsf{F}A$,

(ii) for all $A, A' \in |\mathbf{A}|$, a mapping $\mathbf{A}(A, A') \to \mathbf{B}(\mathsf{F}A, \mathsf{F}A')$, where the image of a morphism $f \in \mathbf{A}(A, A')$ is denoted by $\mathsf{F}f$,

such that for all $A, B, C \in |\mathbf{A}|$ and all $f \in \mathbf{A}(A, B)$ and $g \in \mathbf{A}(B, C)$ we have $\mathsf{F}(f \circ g) = \mathsf{F}f \circ \mathsf{F}g$ and $\mathsf{F}\,\mathrm{id}_A = \mathrm{id}_{\mathsf{F}A}$.

The third basic ingredient of category theory are *natural transformations*. Given two functors $\mathsf{F}, \mathsf{G} : \mathbf{A} \to \mathbf{B}$, a family of morphisms $\eta = (\eta_A : \mathsf{F}A \to \mathsf{G}A)_{A \in |\mathbf{A}|}$ is a natural transformation from F to G, if, for all morphisms $f : A \to A'$ of $\mathbf{A}$, one has that $\eta_{A'} \circ \mathsf{F}f = \mathsf{G}f \circ \eta_A$. This situation is denoted by $\eta : \mathbf{A} \Rightarrow \mathbf{B}$. A natural transformation $(\eta_A)_{A \in |\mathbf{A}|}$ is a *natural isomorphism* if all of its members are isomorphisms.

More specific notions will be introduced as they are needed.

3 Algebraic lattices

In this section we introduce algebraic lattices and review their most well-known characterizations in terms of the sub-poset of compact elements and closure systems of Scott continuous closure operators. The material basically follows [GHK+03], to which we refer for the details of the proofs which we omit to avoid replication. We start with a basic definition.

Definition 3.1 Consider a dcpo P. An element $c \in P$ is *compact* if, for every directed set $D \subseteq P$ we have that $c \leq \bigvee D$ implies $c \leq d$ for some $d \in D$. The set of all compact elements of P is denoted by $\mathsf{K}(P)$. We usually consider $\mathsf{K}(P)$ to be a sub-poset of P.

We note the following

Proposition 3.2 Let L be a complete lattice with compact elements $a, b \in \mathsf{K}(L)$ and least element $\bot$. Then $a \vee b$ and $\bot$ are compact.

Proposition 3.2 contains important information about the structure of the sub-poset of compact elements of a complete lattice. The following definition makes the properties of $\mathsf{K}(L)$ precise.

Definition 3.3 A poset S is a *join-semilattice*, if any two elements a, b in S have a least upper bound $a \vee b$. Dually, in a meet-semilattice any two elements have a greatest lower bound.

We conclude that the poset $\mathsf{K}(L)$ of compact elements of a complete lattice is a join-semilattice with least element under the order of L. However, for a full characterization we shall also be interested in the opposite direction, i.e. given a join-semilattice, we would like to construct a complete lattice. The right tool for this endeavor is that of *ideal completion*, introduced next. Given a set X we define $\downarrow X = \{y \mid \text{there is } x \in X \text{ such that } y \leq x\}$ and $\uparrow X = \{y \mid \text{there is } x \in X \text{ such that } x \leq y\}$; a set is called an *upper* (respectively, *lower*) set if $X = \uparrow X$ (respectively, $X = \downarrow X$). Upper and lower sets of singleton sets $\{x\}$ are denoted by $\uparrow x$ and $\downarrow x$, respectively.

Definition 3.4 Consider a partially ordered set P. A subset $I \subseteq P$ is an *ideal* if it is a directed lower set. The *ideal completion* $\mathsf{Idl}(P)$ is the collection of all ideals of P partially ordered via subset inclusion.

Note that lower sets $\downarrow x$ are always ideals — the *principle ideals* generated by the element x. On the other hand, the empty set is not an ideal, since directed sets need to be non-empty. We see below that the ideal completion of any join-semilattice with least element is a complete lattice. However, not all complete lattices arise in this way. The next definition provides the appropriate characterization.

Definition 3.5 A complete lattice L is an *algebraic lattice*, if for every element $x \in L$, we have $x = \bigvee (\downarrow x \cap \mathsf{K}(L))$.

One can easily see from Proposition 3.2 that any set of the form $\downarrow x \cap \mathsf{K}(L)$ is necessarily directed. Now we are ready to state the important

Theorem 3.6 ([GHK$^+$03] Proposition I-4.10) Let L be an algebraic lattice and let S be a join-semilattice with least element.

(i) $\mathsf{K}(L)$ is a join-semilattice with least element, where the order is induced by that given on L.

(ii) $\mathsf{Idl}(S)$ is a an algebraic lattice, where join is given by set-intersection.

(iii) S is order-isomorphic to $\mathsf{K}(\mathsf{Idl}(S))$ via the isomorphism $f : S \to \mathsf{K}(\mathsf{Idl}(S)) : a \mapsto {\downarrow}a$.

(iv) L is order-isomorphic to $\mathsf{Idl}(\mathsf{K}(L))$ via the isomorphism $g : L \to \mathsf{Idl}(\mathsf{K}(L)) : x \mapsto {\downarrow}x \cap \mathsf{K}(L)$.

This result demonstrates that we can represent any algebraic lattice — up to isomorphism — by an appropriate semilattice and vice versa. We subsequently obtain a number of alternative characterizations from this statement and its proof. A first observation is that Theorem 3.6 assures that every algebraic lattice is isomorphic to a lattice of sets. More precisely, for an algebraic lattice L, we established an isomorphism to a subset of the powerset of its compact elements $2^{\mathsf{K}(L)}$. Now one may ask how to characterize those substructures of powersets which yield algebraic lattices. The tool for this purpose are closure operators.

Definition 3.7 Consider a poset P and a function $c : P \to P$. Then c is a *closure operator* if the following hold for all elements $x, y \in P$

(i) $c(x) = c(c(x))$ (c is idempotent)

(ii) $x \leq c(x)$ (c is inflationary)

(iii) $x \leq y$ implies $c(x) \leq c(y)$ (c is monotone)

An important result about this kind of operators is that they can be characterized completely by their images, the *closure systems*. Explicitly, we have the following.

Proposition 3.8 ([GHK$^+$03] Proposition O-3.13) Let L be a complete lattice and let c be a closure operator on L. Then c preserves arbitrary infima. Especially, its image $c(L) = \{c(x) \mid x \in L\}$ is closed under arbitrary infima in L. Conversely, any subset C of L that is closed under arbitrary infima in L induces a unique closure operator c with image C, given by $c : L \to L : x \mapsto \bigwedge\{y \in C \mid x \leq y\}$.

In Theorem 3.6(ii) it was shown that the set of ideals is closed under arbitrary intersections. By the above proposition this assures that $\mathsf{Idl}(S)$ is a closure system on 2^S, which can be uniquely characterized by a closure operator. However, not every closure system is algebraic, such that a further restriction on the class of closure operators is required. It turns out that Scott continuity (see Definition 2.3) is what is needed to further extend the representation of algebraic lattices.

Theorem 3.9 ([GHK+03] Corollary I-4.14) Any algebraic lattice L is isomorphic to the image of a Scott continuous closure operator on the powerset $2^{\mathsf{K}(L)}$. The operator is given by assigning to any set of compacts the least ideal which contains this set. Conversely, the image of any such closure is an algebraic lattice, where the compacts are exactly the images of finite sets of compacts.

This gives us a third characterization of algebraic lattices. One is tempted to develop a similar statement for join-semilattices with least element. Indeed, any closure operator on the semilattice of finite elements of a powerset can uniquely be extended to a Scott continuous closure on the powerset. However, it is not true that all join-semilattices are images of closure operators on the semilattice of finite subsets of some set. This is easy to see by noting that any collection of finite sets can only have finite descending chains, i.e. it satisfies the *descending chain condition* (see [DP02]). Yet there are join-semilattices with least element that do not have this property, like for example the non-negative rational numbers in their natural order. What we can say is the following.

Corollary 3.10 For any join-semilattice S with least element, there is a closure operator $c : 2^S \to 2^S$, such that S is isomorphic to the image of the finite elements of 2^S under c. Conversely, the finite-set image of any closure operator on a powerset is a join-semilattice with least element.

Proof. Note that any closure operator c on a powerset induces a unique Scott continuous closure c' by setting $c'(X) = \bigcup\{c(A) \mid A \subseteq X,\ A \text{ finite}\}$, where c' agrees with c on all finite sets. Then combine Theorems 3.6 and 3.9, especially the characterization of compact closed subsets. □

The significance of this statement will become apparent in Section 5.

4 Approximable mappings

So far we have only provided object-level correspondences between algebraic lattices and join-semilattices. We supplement this with suitable morphisms which turn these relations into an equivalence of the respective categories. On the side of algebraic lattices, one typically employs Scott continuous functions to form a category **Alg**. This definition leads to a rather advantageous property, namely *cartesian closedness*, which will be discussed in the next section. The aim of this section is to identify a notion of morphism for join-semilattices that produces a category which is *equivalent* to **Alg**.

Definition 4.1 Consider categories **A** and **B**. An *equivalence of categories* **A** and **B** is constituted by a pair of functors $\mathsf{F} : \mathbf{A} \to \mathbf{B}$ and $\mathsf{G} : \mathbf{B} \to \mathbf{A}$, together with a pair of natural isomorphisms $\eta : \mathsf{GF} \Rightarrow \mathrm{id}_\mathbf{A}$ and $\epsilon : \mathsf{FG} \Rightarrow \mathrm{id}_\mathbf{B}$, where $\mathrm{id}_\mathbf{A}$ and $\mathrm{id}_\mathbf{B}$ denote the identity functors on the respective categories.

It is well-known that a functor $\mathsf{F} : \mathbf{A} \to \mathbf{B}$ that is part of an equivalence of categories must be *full* and *faithful*, i.e. there must be a bijection between the hom-sets $\mathbf{A}(A, A')$ (the set of all morphisms from A to A') and $\mathbf{B}(\mathsf{F}A, \mathsf{F}A')$. Thus our next goal is to define a set of morphisms between each pair of join-semilattices which corresponds bijectively to the set of Scott continuous mappings between the associated algebraic lattices. It is easy to see that we cannot expect to use functions for this purpose for mere cardinality reasons: the set of compacts can be significantly smaller than its algebraic lattice. This problem was already solved by Scott in the closely related case of his *information systems* [Sco82a], which we shall also encounter later on. The idea is to shift to a special set of relations, called *approximable mappings*. To our knowledge, the notion of approximable mappings has not yet been introduced to the study of join-semilattices, so we spell out the details.

Definition 4.2 Consider join-semilattices S and T with least elements $\bot_S$ and $\bot_T$, respectively. A relation $\rightsquigarrow \subseteq S \times T$ is an *approximable mapping* if the following hold:

(am1) $a \rightsquigarrow \bot_T$ (non-emptiness)

(am2) $a \rightsquigarrow b$ and $a \rightsquigarrow b'$ implies $a \rightsquigarrow b \vee b'$ (directedness)

(am3) $a \leq a'$, $a \rightsquigarrow b$, and $b' \leq b$ imply $a' \rightsquigarrow b'$ (monotonicity and downward closure)

for all elements $a, a' \in S$ and $b, b' \in T$. This situation is denoted by writing $S \rightsquigarrow T$.

The labels for the above properties already indicate their purpose: for every element $a \in S$ the set $\{b \in T \mid a \rightsquigarrow b\}$ is an ideal of T and the resulting assignment $S \to \mathsf{Idl}(T)$ is monotone. It is now rather obvious how this encodes Scott continuous functions: The image of a compact element is given explicitly via the ideal of compacts which approximates it. The image of a non-compact element is obtained by representing it as directed supremum of compacts and applying Scott continuity.

Some easy checks show that join-semilattices with least element together with approximable mappings indeed constitute a category $\mathbf{Sem}_\vee$, where composition of morphisms is defined as the usual composition of relations. Thus for two approximable mappings $S \rightsquigarrow_1 R$ and $R \rightsquigarrow_2 T$, one defines

$$\rightsquigarrow_2 \circ \rightsquigarrow_1 = \{(s,t) \mid \text{there is } r \in R \text{ such that } (s,r) \in \rightsquigarrow_1 \text{ and } (r,t) \in \rightsquigarrow_2\}.$$

Clearly, $\rightsquigarrow_2 \circ \rightsquigarrow_1$ satisfies (am1) since $a \rightsquigarrow_1 \bot_R$ and $\bot_R \rightsquigarrow_2 \bot_T$. Likewise, under the assumptions of (am2), one finds intermediate values $r, r' \in R$ with $a \rightsquigarrow_1 r \rightsquigarrow_2 b$ and $a \rightsquigarrow_1 r' \rightsquigarrow_2 b'$. By (am2) $a \rightsquigarrow_1 r \vee r'$, and by (am3) $r \vee r' \rightsquigarrow_2 b$ and $r \vee r' \rightsquigarrow_2 b'$. Hence $a \rightsquigarrow_1 r \vee r' \rightsquigarrow_2 b \vee b'$ by another application of (am2). Finally, suppose the assumptions for (am3) hold for $\rightsquigarrow_2 \circ \rightsquigarrow_1$. Then there is $r \in R$ such that $a \rightsquigarrow_1 r \rightsquigarrow_2 b$ and hence $a' \rightsquigarrow_1 r \rightsquigarrow_2 b'$ as required. The identity morphism on a semilattice $S \in |\mathbf{Sem}_\vee|$ is just its greater-or-equal relation $\geq_S$. The fact that this yields an identity under relational composition is just statement (am3). Associativity is inherited from relational composition.

Lemma 4.3 The object mappings Idl and K from Section 3 can be extended to morphisms as follows. For any approximable mapping $\rightsquigarrow \subseteq S \times T$, define $\mathsf{Idl}(\rightsquigarrow) : \mathsf{Idl}(S) \to \mathsf{Idl}(T)$ as $\mathsf{Idl}(\rightsquigarrow)(I) = \{b \mid \text{there is } a \in I \text{ with } a \rightsquigarrow b\}$. For any Scott continuous mapping $f : L \to M$, define $\mathsf{K}f \subseteq \mathsf{K}L \times \mathsf{K}M$ by setting $\mathsf{K}f = \{(a,b) \mid b \leq f(a)\}$. These definitions produce functors $\mathsf{Idl} : \mathbf{Sem}_\vee \to \mathbf{Alg}$ and $\mathsf{K} : \mathbf{Alg} \to \mathbf{Sem}_\vee$.

Proof. To see that Idl is indeed well-defined, observe that for any $a \in S$, $\mathsf{Idl}(\rightsquigarrow)(\downarrow a) = \{b \mid a \rightsquigarrow b\}$, by (am3). This set has already be recognized as an ideal, and hence $\mathsf{Idl}(\rightsquigarrow)$ is well-defined for the compact elements of $\mathsf{Idl}(S)$. By algebraicity, any ideal I is equal to the directed union $\bigcup_{a\in I} \downarrow a$, and hence, observing that $\mathsf{Idl}(\rightsquigarrow)$ preserves all unions, $\mathsf{Idl}(\rightsquigarrow)(I) = \bigcup_{a\in I} \mathsf{Idl}(\rightsquigarrow)(\downarrow a)$. This observation shows that, as a directed union of ideals, $\mathsf{Idl}(\rightsquigarrow)(I)$ is an ideal, and that $\mathsf{Idl}(\rightsquigarrow)$ is Scott continuous.

It is immediate that $\mathsf{Idl}(\rightsquigarrow)$ maps the identity approximable mapping $\geq$ to the identity function. To see that it also preserves composition, note that Scott continuity allows us to restrict to the case of principal ideals. Thus consider two approximable mappings $S \rightsquigarrow_1 R$ and $R \rightsquigarrow_2 T$ and some principal ideal $\downarrow a$, $a \in S$. We compute $(\mathsf{Idl}(\rightsquigarrow_2) \circ \mathsf{Idl}(\rightsquigarrow_1))(\downarrow a) = \mathsf{Idl}(\rightsquigarrow_2)\{r \mid a \rightsquigarrow_1 r\} = \{b \mid \text{there is } r \text{ with } a \rightsquigarrow_1 r \text{ and } r \rightsquigarrow_2 b\} = \{b \mid a(\rightsquigarrow_2 \circ \rightsquigarrow_1)b\} = \mathsf{Idl}(\rightsquigarrow_2 \circ \rightsquigarrow_1)(\downarrow a)$.

Now clearly $\mathsf{K}f$ has properties (am1) to (am3). For functoriality consider Scott continuous functions $f_1 : L \to M$ and $f_2 : M \to N$. It is easy to see that for $a \in \mathsf{K}L$ and $c \in \mathsf{K}N$, whenever there is $b \in \mathsf{K}M$ with $b \leq f_1(a)$ and $c \leq f_2(b)$, one has $c \leq f_2(f_1(a))$. Since the converse also holds, we find that $\mathsf{K}(f_2 \circ f_1) = \{(a,c) \mid c \leq f_2(f_1(a))\} = \{(a,c) \mid \text{there is } b \in \mathsf{K}M \text{ with } b \leq f_1(a) \text{ and } c \leq f_2(b)\} =$

$\mathsf{K}f_2 \circ \mathsf{K}f_1$. Finally, applying K to the identity function clearly yields the identity approximable mapping. □

We finish this section by showing the expected categorical equivalence:

Theorem 4.4 The functors Idl and K of Section 3 yield an equivalence of the categories **Alg** and $\mathbf{Sem}_\vee$.

Proof. For an algebraic lattice L let $\eta_L : L \to \mathsf{Idl}(\mathsf{K}(L)) : x \mapsto {\downarrow}x \cap \mathsf{K}(L)$ be the isomorphism as established in Theorem 3.6. Now consider an algebraic lattice M and a Scott continuous function $f : L \to M$. For any element $x \in L$, $\mathsf{Idl}(\mathsf{K}(f))$ maps the ideal $\eta_L(x)$ to the ideal $\{b \mid \text{there is } a \in \mathsf{K}(L) \text{ with } a \leq x \text{ and } b \leq f(a)\}$. Since Scott continuity guarantees that the supremum of all $f(a)$ is $f(x)$, this is just the set $\eta_M(f(x))$ of all compacts below $f(x)$. Consequently, $\mathsf{Idl}(\mathsf{K}(f))(\eta_L(x)) = \eta_M(f(x))$, i.e. η is natural.

For a join-semilattice S with least element, we define $\epsilon_S \subseteq S \times \mathsf{K}(\mathsf{Idl}(S))$ by setting $\epsilon_S = \{(a, I) \mid I \subseteq {\downarrow}a\}$. From Theorem 3.6 we derive that every compact ideal I is of the form ${\downarrow}b$, hence $\epsilon_S = \{(a, {\downarrow}b) \mid b \leq a\}$. It should now be obvious that ϵ_S is an isomorphism whose inverse is given by $\{({\downarrow}b, a) \mid a \leq b\}$. For naturality of ϵ, consider some approximable mapping $S \rightsquigarrow T$. We compute $\mathsf{K}(\mathsf{Idl}(\rightsquigarrow)) \circ \epsilon_S = \{(a, {\downarrow}b) \mid \text{there is } a' \in S \text{ with } a' \leq a \text{ and } ({\downarrow}a', {\downarrow}b) \in \mathsf{K}(\mathsf{Idl}(\rightsquigarrow))\}$. Expanding the condition $({\downarrow}a', {\downarrow}b) \in \mathsf{K}(\mathsf{Idl}(\rightsquigarrow))$, we find it equivalent to ${\downarrow}b \subseteq \mathsf{Idl}(\rightsquigarrow)({\downarrow}a')$, which in turn is true iff ${\downarrow}b \subseteq \{t \mid a' \rightsquigarrow t\}$, exploiting the fact that ${\downarrow}a'$ is compact. Finally, by (am3) this is equivalent to $a' \rightsquigarrow b$, and we obtain $\mathsf{K}(\mathsf{Idl}(\rightsquigarrow)) \circ \epsilon_S = \{(a, {\downarrow}b) \mid a \rightsquigarrow b\}$, again by (am3). On the other hand, $\epsilon_T \circ \rightsquigarrow = \{(a, {\downarrow}b) \mid \text{there is } b' \in T \text{ with } a \rightsquigarrow b' \text{ and } b \leq b'\}$. Using (am3) once more, this evaluates to $\{(a, {\downarrow}b) \mid a \rightsquigarrow b\}$, which finishes the proof of naturality of ϵ. □

5 A cartesian closed category of formal contexts

Formal concept analysis (FCA, [GW99]) is a powerful lattice-based tool for symbolic data analysis. In essence, it is based on the extraction of a lattice — called *formal concept lattice* — from a binary relation called *formal context* consisting of a set of objects, a set of attributes, and an incidence relation. The transformation from a two-dimensional incidence table to a lattice structure is a crucial *paradigm shift* from which FCA derives much of its power and versatility as a modeling tool. The concept lattices obtained this way turn out to be exactly the complete lattices, and the particular way in which they structure and represent knowledge is very appealing and natural from the perspective of many scientific disciplines.

The successful applications of FCA, however, are mainly restricted to finite contexts and finite concept lattices, since infinite complete lattices generally do not lend themselves for practical implementations. Yet, infinite structures are highly relevant for numerous concrete tasks in knowledge representation and reasoning: model theories of logic programs, computation domains in functional programming, and class hierarchies in ontology research are some typical examples. In order to make methods from FCA available in these application areas, we suggest an interpretation of formal contexts based solely on finitely representable knowledge, thereby obtaining a canonical and computationally feasible representation of infinite data-structures. In effect, we establish a systematic connection between formal concept analysis and algebraic lattices, and thus with domain theory [AJ94], as a categorical equivalence, enriching the link between the two areas as outlined in [Zha03a]. This leads to a category of formal contexts that we now show directly to be cartesian closed.

Definition 5.1 A *formal context* is a structure $\mathbb{P} = (O, A, \models)$, where O and A are sets, and $\models \subseteq O \times A$ is a binary relation. In this case the members of O are called *objects*, the members of A are called *attributes*, and $\models$ is viewed as an *incidence relation* between these two. Accordingly, one says that an object *o has property a* whenever $o \models a$, i.e. $(o, a) \in \models$.

Functions $\alpha_{\mathbb{P}} : 2^O \to 2^A$ and $\omega_{\mathbb{P}} : 2^A \to 2^O$ are defined by setting $\alpha_{\mathbb{P}}(X) = \{a \in A \mid o \models a \text{ for all } o \in X\}$ and $\omega_{\mathbb{P}}(Y) = \{o \in O \mid o \models a \text{ for all } a \in Y\}$.[1] If the context is clear, we omit the subscript from these maps. We also abbreviate $\alpha \circ \omega$ by $\alpha\omega$ etc. as is customary in category theory.

Intuitively, α yields all attributes common to a set of objects. Conversely, ω maps a set of attributes to all objects that fall under all of these attributes. It is straightforward to show that α and ω form an antitone Galois connection between the powerset lattices. This is usually exploited for constructing closure operators $\alpha \circ \omega : 2^A \to 2^A$ and $\omega \circ \alpha : 2^O \to 2^O$. It turns out that the closure systems for both of these are dually isomorphic, the isomorphisms being given by α and ω.

For studying these closure systems, we can therefore focus our attention on the map $\alpha \circ \omega$. Sets of attributes that are closed with respect to this operator are called (attribute) concepts in the literature. FCA builds on the fact that the collection of all concepts of any given formal context is a complete lattice, and that all complete lattices can be obtained this way. This relationship is mediated by the closure system on 2^A induced by the mapping $\alpha\omega$. We take a slightly different approach and focus our attention on the operation of $\alpha\omega$ on $\mathsf{K}(2^A)$, the join-semilattice with least element given by the finite subsets of A. It turns out that this way we obtain

[1]In FCA, $\alpha_{\mathbb{P}}(X)$ is usually written as X', and $\omega_{\mathbb{P}}(Y)$ is similarly written as Y'. We feel that for our treatment a more explicit notation is more convenient.

all complete algebraic lattices instead of all complete ones. Now Corollary 3.10 suggests the following.

Corollary 5.2 For every formal context $\mathbb{P} = (O, A, \models)$, the set $\mathsf{Sem}(\mathbb{P}) = \alpha\omega(\mathsf{K}(2^A))$ is a join-semilattice with least element. Conversely, every such semilattice can (up to isomorphism) be represented in this way.

Proof. In spite of our earlier considerations, we give the easy direct proof. For two finite sets X and Y, $\alpha\omega(X \cup Y)$ is the least closed set that contains X and Y, and thus also $\alpha\omega(X)$ and $\alpha\omega(Y)$. Hence $\alpha\omega(X) \vee \alpha\omega(Y) = \alpha\omega(X \cup Y)$. The first part of the proof is finished by noting that $\alpha\omega(\emptyset)$ is the least closed set.

Conversely, for a join-semilattice with least element S, consider the context $(S, S, \geq)$. Then for any finite $X \subseteq S$, $\alpha\omega(X)$ is the set of all lower bounds of all upper bounds of X. But this is easily recognized as $\downarrow \bigvee X$. Note that the least upper bound of the empty set is just the least element. The obvious isomorphism between S and the semilattice $(\{\downarrow s \mid s \in S\}, \subseteq)$ suffices to complete the proof. □

By Theorem 3.6 the above shows that every algebraic lattice can be represented by some formal context and vice versa. To make this explicit, we can extend the closure operator of Corollary 5.2 to a Scott continuous closure operator on 2^A, as done before in the proof of Corollary 3.10. In this way we can recover the following result from [ZS0x].

Corollary 5.3 Consider a formal context $\mathbb{P} = (O, A, \models)$ and the mapping $c : 2^A \to 2^A : x \mapsto \bigcup\{\alpha\omega(X) \mid X \subseteq x,\ X \text{ finite}\}$. Then $\mathsf{Alg}(\mathbb{P}) = c(2^A)$ is an algebraic lattice and every algebraic lattice is of this form (up to isomorphism).

Proof. Clearly, c is just the unique Scott continuous closure operator induced by $\alpha \circ \omega$ as in Corollary 3.10. By Theorem 3.9 its closure system is indeed an algebraic lattice. For the other direction combine Theorem 3.9 and Theorem 3.6 to see that $c(2^A)$ is isomorphic to the ideal completion of $\mathsf{Sem}(\mathbb{P})$. Since every algebraic lattice is of this form for some join-semilattice with least element, the claim follows from Corollary 5.2. □

Closed sets with respect to the operator c from the above proposition have been termed *approximable concepts* in [ZS0x]. Naturally, it is also possible to extend this result to a categorical equivalence. For this purpose we define a category **Cxt** of formal contexts. The morphisms between two contexts $\mathbb{P}$ and $\mathbb{Q}$ are defined by setting $\mathbf{Cxt}(\mathbb{P}, \mathbb{Q}) = \mathbf{Sem}_{\vee}(\mathsf{Sem}(\mathbb{P}), \mathsf{Sem}(\mathbb{Q}))$.[2] The following is readily seen.

[2]In [HZ04] a slightly different definition of morphisms is given. In the formulation given there, the corresponding approximable mapping is not defined on the closed sets $\mathsf{Sem}(\mathbb{P})$ but on all finite attribute sets. We get a context morphism in this sense by extending our approximable mappings, relating two finite sets iff their closures are related.

$$\begin{tikzcd}
 & & \mathbb{P} \\
\mathbb{R} \arrow[r, dashed, "\langle \rightsquigarrow_P, \rightsquigarrow_Q \rangle"] \arrow[urr, bend left, "\rightsquigarrow_P"] \arrow[drr, bend right, "\rightsquigarrow_Q"'] & \mathbb{P} \times \mathbb{Q} \arrow[ur, "\pi_P"'] \arrow[dr, "\pi_Q"] & \\
 & & \mathbb{Q}
\end{tikzcd}$$

Figure 1: The product construction in **Cxt**.

Theorem 5.4 The categories $\mathbf{Sem}_\vee$ and **Cxt** are equivalent.

The functors needed for this result are obvious: on the object level, we obtain suitable mapping from Corollary 5.2, and the situation for morphisms is trivial. The construction of the natural isomorphisms is similar to the one of ϵ in Theorem 4.4, where the identity approximable mapping was modified using the given order-isomorphism of the semilattices.

In the remainder of this section we investigate the categorical constructions that are possible within the categories **Alg**, $\mathbf{Sem}_\vee$, and **Cxt**, where the latter will be the explicit object of study. Because **Cxt** is equivalent to **Alg**, we know that it is *cartesian closed*. We make the required constructions explicit in the sequel, and thus give a mostly self-contained proof of cartesian closedness of **Cxt**.

Definition 5.5 A category $\mathbf{C}$ is *cartesian closed* if it has all finite products, and there is a functor $\mathbf{C}^{\mathsf{op}} \times \mathbf{C} \to \mathbf{C} : (A, B) \mapsto B^A$ and a natural bijection between the hom-sets $\mathbf{C}(A \times B, C)$ and $\mathbf{C}(A, C^B)$.

Exact requirements for showing each of these properties will be given in the respective proofs and statements. We first consider the empty product, i.e. the terminal object, which turns out to be given by the formal context $\mathbf{1} = (\emptyset, \emptyset, \emptyset)$. Indeed, for every formal context $\mathbb{P} = (O, A, \models)$ there is a unique approximable mapping $\mathbb{P} \rightsquigarrow \mathbf{1}$ that relates every finite subset of A to the empty set. The situation for binary products is not much more difficult.

Proposition 5.6 Consider two formal contexts $\mathbb{P} = (O_P, A_P, \models_P)$ and $\mathbb{Q} = (O_Q, A_Q, \models_Q)$, and define a formal context $\mathbb{P} \times \mathbb{Q} = (O_P \uplus O_Q, A_P \uplus A_Q, (\models_Q) \uplus (\models_P) \uplus (O_P \times A_Q) \uplus (O_Q \times A_P))$, where $\uplus$ denotes disjoint union.

Then $\mathbb{P} \times \mathbb{Q}$ is the categorical product of $\mathbb{P}$ and $\mathbb{Q}$, i.e. there are approximable mappings $\pi_P : \mathbb{P} \times \mathbb{Q} \to \mathbb{P}$ and $\pi_Q : \mathbb{P} \times \mathbb{Q} \to \mathbb{Q}$ such that, given approximable mappings $\rightsquigarrow_P$ and $\rightsquigarrow_Q$ as in Figure 1, there is a unique approximable mapping $\langle \rightsquigarrow_P, \rightsquigarrow_Q \rangle$ that makes this diagram commute.

Proof. Since context morphisms were defined with reference to the induced semilattices, we first look at $\mathsf{Sem}(\mathbb{P}\times\mathbb{Q})$. It is easy to see that concept closure in $\mathbb{P}\times\mathbb{Q}$ is computed by taking disjoint unions of closures in $\mathbb{P}$ and $\mathbb{Q}$, i.e. for sets $X \subseteq A_P$ and $Y \subseteq A_Q$, one finds that $\alpha\omega(X \uplus Y) = \alpha\omega(X) \uplus \alpha\omega(Y)$. Hence every element of $\mathsf{Sem}(\mathbb{P}\times\mathbb{Q})$ corresponds to a unique disjoint union of elements of $\mathsf{Sem}(\mathbb{P})$ and $\mathsf{Sem}(\mathbb{Q})$.

We can now define the projections by setting $(X \uplus Y, X') \in \pi_P$ iff $X' \subseteq X$ and $(X \uplus Y, Y') \in \pi_Q$ iff $Y' \subseteq Y$, for all X, $X' \in \mathsf{Sem}(\mathbb{P})$ and Y, $Y' \in \mathsf{Sem}(\mathbb{Q})$. It is readily seen that these morphisms satisfy the properties of Definition 4.2.

Now consider $\rightsquigarrow_P$ and $\rightsquigarrow_Q$ as in Figure 1. We define the relation $\langle\rightsquigarrow_P,\rightsquigarrow_Q\rangle$ by setting $(Z, X \uplus Y) \in \langle\rightsquigarrow_P,\rightsquigarrow_Q\rangle$ iff $Z \rightsquigarrow_P X$ and $Z \rightsquigarrow_Q Y$, for all concepts X, Y, Z from the corresponding semilattices. Again it is easy to check the conditions of Definition 4.2, since they follow immediately from the corresponding properties of $\rightsquigarrow_P$ and $\rightsquigarrow_Q$. Furthermore, if there is $X \uplus Y \in \mathsf{Sem}(\mathbb{P}\times\mathbb{Q})$ with $(Z, X \uplus Y) \in \langle\rightsquigarrow_P,\rightsquigarrow_Q\rangle$ and $(X \uplus Y, X') \in \pi_P$ then $Z \rightsquigarrow_P X'$ by the definition of π_P and (am3). Conversely, if $Z \rightsquigarrow_P X'$ then one finds that $X' \uplus \alpha\omega(\emptyset) \in \mathsf{Sem}(\mathbb{P}\times\mathbb{Q})$ yields the required intermediate element to show that $(Z, X') \in \pi_P \circ \langle\rightsquigarrow_P,\rightsquigarrow_Q\rangle$. Since a similar reasoning applies to $\rightsquigarrow_Q$, Figure 1 commutes as required.

Finally, for uniqueness of $\langle\rightsquigarrow_P,\rightsquigarrow_Q\rangle$ consider $\mathbb{R} \rightsquigarrow \mathbb{P}\times\mathbb{Q}$ with $\pi_P \circ \rightsquigarrow\; = \;\rightsquigarrow_P$ and $\pi_Q \circ \rightsquigarrow\; = \;\rightsquigarrow_Q$. If $Z \rightsquigarrow X \uplus Y$, then $(Z, X) \in \pi_P \circ \rightsquigarrow$ and hence $Z \rightsquigarrow_P X$ and, by a similar reasoning, $Z \rightsquigarrow_Q Y$. Conversely, if $Z \rightsquigarrow_P X$ then there must be some X' and Y' such that $X \subseteq X'$ and $Z \rightsquigarrow X' \uplus Y'$. By (am3) this implies $Z \rightsquigarrow X \uplus Y'$. The same argument can be applied to $\rightsquigarrow_Q$. Thus whenever $Z \rightsquigarrow_P X$ and $Z \rightsquigarrow_Q Y$, there are X' and Y' with $Z \rightsquigarrow X \uplus Y'$ and $Z \rightsquigarrow X' \uplus Y$. Invoking properties (am2) and (am3) for $\rightsquigarrow$, this shows that $Z \rightsquigarrow X \uplus Y$. We have just shown that $Z \rightsquigarrow X \uplus Y$ iff $Z \rightsquigarrow_P X$ and $Z \rightsquigarrow_Q Y$, and hence that $\rightsquigarrow\; = \langle\rightsquigarrow_P,\rightsquigarrow_Q\rangle$ as required. □

The above product construction is also known in formal concept analysis as the *direct sum* of two contexts [GW99]. However, it is not the only possible specification of the products in **Alg**. For each formal context $\mathbb{P} = (O_P, A_P, \models_P)$, we define a context $\mathbb{P}^+ = (O_P^+, A_P^+, \models_{P^+})$, where $O_P^+ = O_P \cup \{g\}$ and $A_P^+ = A_P \cup \{m\}$, with g and m being fresh elements: $g \notin O_P$ and $m \notin A_P$. For defining the incidence relation, we set $o \models_{P^+} a$ whenever $o \models_P a$ (requiring that $a \in A_P$ and $o \in O_P$) or $o = g$ or $a = m$. Thus $\mathbb{P}^+$ emerges from $\mathbb{P}$ by "adding a full row and a full column."

Now let $\mathbb{P} = (O_P, A_P, \models_P)$ and $\mathbb{Q} = (O_Q, A_Q, \models_Q)$ be formal contexts. Define a new formal context $\mathbb{P}\otimes\mathbb{Q} = (O_P^+ \times O_Q^+, A_P^+ \times A_Q^+, \models_{P\times Q})$ of $\mathbb{P}$ and $\mathbb{Q}$ by setting $(o_1, o_2) \models_{P\times Q} (a_1, a_2)$ iff $o_1 \models_{P^+} a_1$ and $o_2 \models_{Q^+} a_2$. This turns out to be an alternative description of the products in **Cxt**.

Proposition 5.7 Given arbitrary formal contexts $\mathbb{P} = (O_P, A_P, \models_P)$ and $\mathbb{Q} = (O_Q, A_Q, \models_Q)$, the contexts $\mathbb{P}\times\mathbb{Q}$ and $\mathbb{P}\otimes\mathbb{Q}$ are isomorphic in **Cxt**. Equivalently,

$\mathbb{P} \otimes \mathbb{Q}$ is the object part of the categorical product of $\mathbb{P}$ and $\mathbb{Q}$ in **Cxt**.

Proof. The required isomorphism corresponds to an iso approximable mapping between the semilattices $\mathsf{Sem}(\mathbb{P}\times\mathbb{Q})$ and $\mathsf{Sem}(\mathbb{P}\otimes\mathbb{Q})$. The elements of the former were already recognized as disjoint unions of concepts from $\mathbb{P}$ and $\mathbb{Q}$. In the latter case, concepts are easily recognized as products of concepts from $\mathbb{P}^+$ and $\mathbb{Q}^+$. Adding the additional elements m and g guarantees that neither of these extended formal contexts allows for the empty set as a concept, so that each element of $\mathsf{Sem}(\mathbb{P} \otimes \mathbb{Q})$ is indeed of the form $X \times Y$ for two uniquely determined concepts $X = \alpha\omega(X) \in \mathsf{Sem}(\mathbb{P}^+)$ and $Y = \alpha\omega(Y) \in \mathsf{Sem}(\mathbb{Q}^+)$.

We define a relation $\rightsquigarrow^+ \subseteq \mathsf{Sem}(\mathbb{P} \times \mathbb{Q}) \times \mathsf{Sem}(\mathbb{P} \otimes \mathbb{Q})$ by setting $X \rightsquigarrow^+ Y$ whenever $p_1(Y) \cap A_P \subseteq X$ and $p_2(Y) \cap A_Q \subseteq X$, where p_i denotes the projection to the ith components in a set of pairs. Conversely, a relation $\rightsquigarrow^- \subseteq \mathsf{Sem}(\mathbb{P} \otimes \mathbb{Q}) \times \mathsf{Sem}(\mathbb{P} \times \mathbb{Q})$ is specified by setting $Y \rightsquigarrow^- X$ whenever $X \cap A_P \subseteq p_1(Y)$ and $X \cap A_Q \subseteq p_2(Y)$.

We claim that $\rightsquigarrow^+$ and $\rightsquigarrow^-$ are mutually inverse approximable mappings between $\mathsf{Sem}(\mathbb{P} \times \mathbb{Q})$ and $\mathsf{Sem}(\mathbb{P} \otimes \mathbb{Q})$. The properties of Definition 4.2 follow immediately from our use of set-theoretic operations in the definitions. Furthermore it is easy to see that $X(\rightsquigarrow^- \circ \rightsquigarrow^+)X'$ implies $X' \subseteq X$ for any two elements $X, X' \in \mathsf{Sem}(\mathbb{P} \times \mathbb{Q})$. The converse implication also holds, which can be concluded from the obvious relationships $X \rightsquigarrow^+ \alpha\omega(X \cap A_P) \times \alpha\omega(X \cap A_Q)$, $\alpha\omega(X' \cap A_P) \times \alpha\omega(X' \cap A_Q) \rightsquigarrow^- X'$, and $\alpha\omega(X' \cap A_P) \times \alpha\omega(X' \cap A_Q) \subseteq \alpha\omega(X \cap A_P) \times \alpha\omega(X \cap A_Q)$. Hence $\rightsquigarrow^- \circ \rightsquigarrow^+$ is indeed the identity approximable mapping. A similar reasoning shows that the same is true for $\rightsquigarrow^+ \circ \rightsquigarrow^-$, thus finishing the proof.

Finally, the assertion that this makes $\otimes$ an alternative product construction is a basic fact from category theory. The required projections are obtained by composing $\rightsquigarrow^-$ with the projections from the proof of Proposition 5.6. $\square$

The construction of exponentials in **Cxt** turns out to be slightly more intricate. To fully understand the following definition, it is helpful to look at the function spaces in **Alg**. These are just the sets of all Scott continuous maps between two algebraic lattices under the pointwise order of functions. The standard technique for describing the compact elements of this lattice are so-called *step functions*. Given two algebraic lattices L and M and two compacts $a \in \mathsf{K}(L)$ and $b \in \mathsf{K}(M)$, one defines a function $|a \Rightarrow b| : L \to M$, that maps an element x to b whenever $a \leq x$, and to $\perp_M$ otherwise. It is well-known that any such step function is Scott continuous and compact in the function space of L and M (see [GHK+03]). However, not all compacts are of this form, since finite joins of step functions are also compact maps that can usually take more than two different values.

Our goal is to construct a formal context that represents the join-semilattice of all compact Scott continuous functions in the sense of Corollary 5.2. Intuitively, the collection of all step functions suggests itself as the set of attributes. Finitely generated concepts should represent finite joins of step functions, which in turn correspond bijectively to lower sets with respect to the pointwise order of step functions. In order to obtain a formal context that yields this lower closure, one is tempted to take some subset of Scott continuous functions for objects, and to employ the inverted pointwise order as an entailment relation. This is indeed feasible, but our supply of step functions unfortunately is insufficient to serve as object set in this case. We end up with the following definition:

Definition 5.8 Consider two formal contexts $\mathbb{P}$ and $\mathbb{Q}$, and the sets $A = \mathsf{Sem}(\mathbb{P}) \times \mathsf{Sem}(\mathbb{Q})$ and $O = \mathsf{Fin}(A)$. A formal context $[\mathbb{P} \leadsto \mathbb{Q}] = (O, A, \models)$ is defined by setting $\{(a_i, b_i)\} \models (a, b)$ iff $b \subseteq \bigvee\{b_i \mid a_i \subseteq a\}$, where $\bigvee$ is the join operation from the semilattice $\mathsf{Sem}(\mathbb{Q})$.

This definition derives from the above discussion by representing step functions $|a \Rightarrow b|$ via pairs (a, b).[3] Hence, the approximable concepts of $[\mathbb{P} \leadsto \mathbb{Q}]$ as obtained in Corollary 5.3 are sets of such pairs, i.e. relations between $\mathsf{Sem}(\mathbb{P})$ and $\mathsf{Sem}(\mathbb{Q})$. The reader's suspicion about the true nature of these relations shall be confirmed:

Lemma 5.9 Given contexts $\mathbb{P}$ and $\mathbb{Q}$, the algebraic lattice $L = \mathsf{Alg}[\mathbb{P} \leadsto \mathbb{Q}]$ of approximable concepts of $[\mathbb{P} \leadsto \mathbb{Q}]$ coincides with the lattice of all approximable mappings from $\mathbb{P}$ to $\mathbb{Q}$, ordered by subset inclusion.

Proof. Consider any approximable concept $x \in L$. Definition 5.8 implies that the pairs of arbitrary elements $a \in \mathsf{Sem}(\mathbb{P})$ and the least element of $\mathsf{Sem}(\mathbb{Q})$ are modelled by any object of $[\mathbb{P} \leadsto \mathbb{Q}]$, i.e. (am1) of Definition 4.2 holds for x. For (am2), assume $(a, b_1) \in x$ and $(a, b_2) \in x$. Following the construction in Corollary 5.3, one finds that $\alpha\omega(\{(a, b_1), (a, b_2)\}) \subseteq x$. However, for any object o of $[\mathbb{P} \leadsto \mathbb{Q}]$, $o \models (a, b_1)$ and $o \models (a, b_2)$ clearly implies $o \models (a, b_1 \vee b_2)$, by expanding the definition of $\models$, and thus $(a, b_1 \vee b_2) \in x$. Finally, for (am3) consider some $(a, b) \in x$, $a' \supseteq a$, and $b' \subseteq b$. Clearly, we have $\alpha\omega(\{(a, b)\}) \subseteq x$. The definition of $\models$ shows immediately that every object that models (a, b) must also model (a', b'), and thus $(a', b') \in \alpha\omega(\{(a, b)\})$ as required.

For the converse consider any approximable mapping $\mathbb{P} \leadsto \mathbb{Q}$. We show that $\leadsto \in L$. Given any finite subset $X = \{(a_i, b_i)\} \subseteq \leadsto$, one finds that $X \models (a_n, b_n)$ for all $(a_n, b_n) \in X$. Thus $X \in \omega(X)$ and, whenever $(a, b) \in \alpha\omega(X)$, one also has $X \models (a, b)$, i.e. $b \subseteq \bigvee\{b_j \mid a_j \subseteq a\}$. Defining $J = \{j \mid a_j \subseteq a\}$, one finds that for every $n \in J$, $a_n \subseteq \bigvee\{a_j \mid j \in J\}$ and hence $\bigvee\{a_j \mid j \in J\} \leadsto b_n$ by (am3). Since J is finite, one can employ an easy induction to show that $\bigvee\{a_j \mid j \in J\} \leadsto \bigvee\{b_j \mid j \in$

[3]This correspondence is not injective. In fact, the context $[\mathbb{P} \leadsto \mathbb{Q}]$ in general contains both duplicate rows and duplicate columns.

$J\}$, where the case $J = \emptyset$ follows from (am1) and the induction step uses (am2). Obviously $\bigvee\{a_j \mid j \in J\} \subseteq a$ and $b \subseteq \bigvee\{b_j \mid j \in J\}$, and hence $a \rightsquigarrow b$ by (am3). This shows that $\rightsquigarrow$ is an approximable concept. □

The above considerations shed additional light on approximable mappings in general: they can in fact be viewed as lower sets of step functions, the joins of which uniquely determine an arbitrary Scott continuous map between the induced algebraic lattices. We remark that this also hints at an alternative formulation of the constructions in Lemma 4.3.

It remains to show that the above construction does indeed yield a function space in the sense of category theory:

Proposition 5.10 The construction $[- \rightsquigarrow -]$ yields the categorical function space of the two contexts, i.e. for all contexts $\mathbb{P}$, $\mathbb{Q}$, and $\mathbb{R}$, there is a bijection between the sets $\mathbf{Cxt}(\mathbb{P} \times \mathbb{Q}, \mathbb{R})$ and $\mathbf{Cxt}(\mathbb{P}, [\mathbb{Q} \rightsquigarrow \mathbb{R}])$, and this bijection is natural in all arguments.

Proof. Our earlier results can be employed to simplify this proof. The algebraic lattices associated with the above contexts is denoted by $L = \mathsf{Alg}(\mathbb{P})$, $M = \mathsf{Alg}(\mathbb{Q})$, and $N = \mathsf{Alg}(\mathbb{R})$, and we write $[M \to N]$ for the lattice of all Scott continuous functions from M to N, ordered pointwise. The categorical equivalences between **Cxt**, $\mathbf{Sem}_\vee$, and **Alg** (Theorem 4.4 and Theorem 5.4) and the categorical role of the product construction $\mathbb{Q} \times \mathbb{R}$ (Proposition 5.6) establish natural bijections between the sets $\mathbf{Cxt}(\mathbb{P} \times \mathbb{Q}, \mathbb{R})$ and $\mathbf{Alg}(L \times M, N)$, where $L \times N$ is the standard product order. Likewise, using the same equivalences and the bijection of function spaces from Lemma 5.9, one finds another natural bijection between $\mathbf{Cxt}(\mathbb{P}, [\mathbb{Q} \rightsquigarrow \mathbb{R}])$ and $\mathbf{Alg}(L, [N \to M])$.

The proof is completed by providing the well-known natural bijection of the sets $\mathbf{Alg}(L \times M, N)$ and $\mathbf{Alg}(L, [N \to M])$. This standard proof can for example be found in [GHK+03]. □

Summing up these results, we obtain:

Theorem 5.11 The categories **Alg**, $\mathbf{Sem}_\vee$, and **Cxt** are cartesian closed.

Proof. **Cxt** was shown cartesian closed in Proposition 5.6 and Proposition 5.10. Closure of the other categories follows by their categorical equivalence (Theorem 4.4 and Theorem 5.4). □

The cartesian closed category **Cxt** which we propose here is tailored to the needs of Computer Science. It differs from the categories normally considered in formal concept analysis by emphasizing *algebraicity*, whereas morphisms listed e.g. in [GW99] are suitable for complete, but not necessarily algebraic, lattices.

We also stress the fact that our novel interpretation of formal contexts perfectly agrees with the classical one, as long as finite contexts or lattices are considered, which covers most of the current FCA applications in Computer Science. On the other hand, the different treatment of infinite data structures displays a deviation from the classical philosophically motivated viewpoint towards one that respects the practical constraints of finiteness and computability.

6 Further representations

So far, we encountered three equivalent representations for algebraic lattices. Clearly, the hard part was to establish the equivalence of the rather diverse categories **Alg** and $\mathbf{Sem}_\vee$. Many other equivalent categories can now be recognized by relating them to one of these two — an objective that will in general be accomplished rather easily. A typical example for this has already been given in form of the category **Cxt**, that was easily seen to be equivalent to $\mathbf{Sem}_\vee$.

The representations given below are grouped according to these observations: we start with "logical" descriptions that have their closest relationships to the categories **Cxt** and $\mathbf{Sem}_\vee$, and then proceed to formulations that can be connected to **Alg** in a more natural way. Classifying representations in this way is by no means arbitrary: as we will see the end of this section, our arrangement reflects the "localic" respectively "spacial" side of a very specific case of Stone duality.

6.1 Logic and information systems

The representation of join-semilattices via formal contexts did already incorporate some logical flavor: approximable concepts can be viewed as sets closed under a certain entailment relation. Scott continuity of this closure is reminiscent of the compactness property of a logic. However, we will see that a much closer connection to some very well-known logics can be made. The reader is referred to [DH01] for related considerations.

Definition 6.1 Given a set A of propositions, the set of well-formed *conjunctive propositional formulae* $\mathscr{S}(A)$ over A is given by the following expression:

$$\mathscr{S}(A) \ ::= \ \top \ | \ a \in A \ | \ (\mathscr{S}(A) \wedge \mathscr{S}(A))$$

A relation $\vdash \ \subseteq \mathscr{S}(A) \times \mathscr{S}(A)$ is a *consequence relation* of conjunctive propositional logic (CCP logic) if it is closed under application of the following rules:

$$F \vdash \top \quad (\mathrm{T}) \qquad F \vdash F \quad (\mathrm{R}) \qquad \frac{F \vdash G,\ G \vdash H}{F \vdash H} \quad (\mathrm{Cut})$$

$$\frac{F \vdash (G \wedge H)}{F \vdash G}\ \text{(W1)} \qquad \frac{F \vdash (G \wedge H)}{F \vdash H}\ \text{(W2)} \qquad \frac{F \vdash G,\ F \vdash H}{F \vdash (G \wedge H)}\ \text{(And)}$$

In this case $(\mathscr{S}(A), \vdash)$ is called a *deductive system* (of CCP logic). For any two formulae $F, G \in \mathscr{S}(A)$, the situation where $F \vdash G$ and $G \vdash F$ is denoted $F \approx G$.

Hence deductive systems are logical systems of the conjunctive fragment of propositional logic, together with a (not necessarily minimal) consequence relation. The following properties are easily verified.

Lemma 6.2 Consider a deductive system $(\mathscr{S}(A), \vdash)$. The following hold for all formulae F, G, and $H \in \mathscr{S}(A)$:

- $((F \wedge G) \wedge H) \approx (F \wedge (G \wedge H))$
- $(F \wedge G) \approx (G \wedge F)$
- $F \approx (F \wedge F)$
- $F \approx (F \wedge \top)$

Hence we see that the rules (W1), (W2), and (And) imply associativity, commutativity, and idempotency of $\wedge$. Furthermore, occurrences of $\top$ can be eliminated. Consequently, we henceforth write formulae of CCP in the form $a_1 \wedge a_2 \wedge \ldots \wedge a_n$ ($a_i \in A$), knowing that this determines a set of "real" formulae up to proof-theoretic equivalence. Additionally, for the case $n = 0$ the above expression is interpreted as the singleton set $\{\top\}$. Any statement about formulae in this notation represents the corresponding set of statements about the original formulae. We can now consider the algebraic semantics (see [DH01]) of these logics. This is based largely on the following notion:

Definition 6.3 Consider a deductive system $(\mathscr{S}(A), \vdash)$. The *Lindenbaum algebra* of $(\mathscr{S}(A), \vdash)$ is the poset obtained from the preorder $(\mathscr{S}(A), \vdash)$ through factorization by the equivalence relation $\approx$, i.e. $[F]_\approx \leq [G]_\approx$ iff $F \vdash G$. The Lindenbaum algebra is denoted by $\mathsf{LA}(\mathscr{S}(A), \vdash)$.

Hence the Lindenbaum algebra is a partially ordered set of $\approx$-equivalence classes of formulae, ordered by syntactic entailment. Since it can cause hardly any confusion, we take the freedom to denote equivalence classes by one of their representatives or even by the simplified notation introduced above. Of course, this creates possible ambiguity between the conjunction symbol and the meet operation within the Lindenbaum algebra. The following lemma shows that this is not a problem.

Lemma 6.4 Consider a deductive system $(\mathscr{S}(A), \vdash)$ and formulae $F, G \in \mathscr{S}(A)$. Then $[F]_\approx \wedge [G]_\approx = [F \wedge G]_\approx$.

Proof. We have to show that $F \wedge G \vdash F$, $F \wedge G \vdash G$, and that for any formula H such that $H \vdash F$ and $H \vdash G$, we find $H \vdash F \wedge G$. These assertions are obvious consequences of the proof rules of CCP. □

Since the meet operation yields a unique result, this shows that $F \approx F'$ and $G \approx G'$ imply $F \wedge G \approx F' \wedge G'$, which is just the *Replacement Theorem* [DH01] for CCP logics. We state the now obvious representation theorem:

Theorem 6.5 For any deductive system $(\mathscr{S}(A), \vdash)$, the Lindenbaum algebra $\mathsf{LA}(\mathscr{S}(A), \vdash)$ is a meet-semilattice with greatest element. Conversely, every such semilattice is isomorphic to the Lindenbaum algebra of some deductive system.

Proof. Lemma 6.4 already showed the existence of binary meets. We conclude the first part of the proof by noting that $[\top]_\approx$ is the required greatest element.

For the converse let S be a meet-semilattice with greatest element. We define a consequence relation $\vdash$ on $\mathscr{S}(S)$ by setting, for all $a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_m \in S$, $a_1 \wedge a_2 \wedge \ldots \wedge a_n \vdash b_1 \wedge b_2 \wedge \ldots \wedge b_m$ whenever $a_1 \wedge a_2 \wedge \ldots \wedge a_n \leq b_1 \wedge b_2 \wedge \ldots \wedge b_m$. One can easily check that this definition satisfies all of the required rules. Note that (T) follows by our convention to represent $\top$ by the empty conjunction. To reduce confusion, we denote meets in S by $\bigwedge$ and meets in $\mathsf{LA}(\mathscr{S}(S), \vdash)$ by $\bigwedge_\approx$.

We claim that S is isomorphic to $\mathsf{LA}(\mathscr{S}(S), \vdash)$. Indeed, one can define mappings $f : S \to \mathsf{LA}(\mathscr{S}(S), \vdash)$ and $g : \mathsf{LA}(\mathscr{S}(S), \vdash) \to S$ by setting $f(a) = [a]_\approx$ and, for propositions a_i, $1 \leq i \leq n$, $g[\bigwedge_\approx a_i]_\approx = \bigwedge a_i$. To see that g is well-defined, note that for any two formulae $\bigwedge_\approx a_i, \bigwedge_\approx b_j \in \mathscr{S}(S)$ we have that $\bigwedge_\approx a_i \approx \bigwedge_\approx b_j$ (in $\mathscr{S}(S)$) implies $\bigwedge a_i = \bigwedge b_j$ (in S) by the definition of $\vdash$.

Finally, we show that g and f are inverse to each other. By what was said above, $g(f(a)) = a$ is immediate. On the other hand, any formula $\bigwedge_\approx a_i$ is syntactically equivalent to $\bigwedge a_i$ by the definition of $\vdash$. This shows bijectivity of f and g. Monotonicity of both functions is obvious from their definition. □

This relationship closes the gap to our prior category $\mathbf{Sem}_\vee$, since the above meet-semilattices are just the order duals of the objects within this category. By an approximable mapping between two meet-semilattices with least element or two deductive systems of CCP logic, we mean an approximable mapping between the induced join-semilattices. The following is immediate.

Theorem 6.6 Consider the categories $\mathbf{Sem}_\wedge$ and $\mathbf{CCP}$ of meet-semilattices with greatest element and deductive systems of CCP logic, respectively, together with approximable mappings as morphisms. Then $\mathbf{Sem}_\vee$, $\mathbf{Sem}_\wedge$, and $\mathbf{CCP}$ are equivalent.

The insights just obtained allow to relate our study with results obtained in [HW03, Hit04], where the conjunctive fragment of the logic RZ (introduced in

[RZ01]), was found to be closely related to concept closure in FCA. We derive a very similar result, but some preparations are needed first.

An *algebraic cpo D* is a dcpo with least element $\bot$ such that every $e \in D$ is the directed supremum of all compact elements below it. A *coherent algebraic cpo* is an algebraic cpo such that, with respect to the Scott topology (see Definition 6.10), the intersection of any two compact open sets is compact open.

These notions can be found in [RZ01], along with a characterization of the Smyth Powerdomain of any given coherent algebraic cpo D by means of a logic defined on D, which we call the *logic RZ*. We will only be concerned with the conjunctive fragment of RZ, which can be given as follows. For compact elements $c_1, \ldots, c_n, d_1, \ldots, d_m$ we write $c_1 \wedge \ldots \wedge c_n \vdash d_1 \wedge \ldots \wedge d_m$ iff any minimal upper bound of $\{c_1, \ldots, c_n\}$ is above all d_i. This way, we obtain a deductive system $(\mathsf{K}(D), \vdash)$, and the following result, which is related to those in [HW03, Hit04], and such considerations were put to use in [Hit04] for developing a generic non-monotonic rule-based reasoning paradigm over hierarchical knowledge.

Theorem 6.7 Let $\mathbb{P} = (O, A, \models)$ be any formal context. Then there is a coherent algebraic cpo D and a mapping $\iota : A \to D$ such that for every finite set $X = \{a_1, \ldots, a_n\} \subseteq A$ we have $\alpha\omega(X) = \{a \mid \iota(a_1) \wedge \ldots \wedge \iota(a_n) \vdash \iota(a)\}$.

Proof. Define $D = \mathsf{Alg}(\mathbb{P})$ and set $\iota(a) = \alpha\omega(\{a\})$ for $a \in A$. Since D is a complete algebraic lattice, it is a coherent algebraic cpo.

Now consider the finite set X as above. Using the completeness of the lattice, we obtain that $\iota(X)$ has $\alpha\omega(X)$ as supremum, which suffices. $\square$

The difference between Theorem 6.7 and the results in [HW03, Hit04] lies in the fact that the latter were proven by taking D to be a sublattice of the (classical) formal concept lattice, instead of $\mathsf{Alg}(\mathbb{P})$, which facilitates reasoning with formal contexts in a natural way.

Finally, we come to another popular description of algebraic lattices, that fits well into the above discussion, and will also shed additional light on morphisms of **CCP**.

Definition 6.8 Consider a structure $(A, \Vdash)$, where A is a set, and $\Vdash \subseteq \mathsf{Fin}(A) \times A$ is a relation between finite subsets of A and elements of A. Then $(A, \Vdash)$ is a *Scott information system* (with trivial consistency predicate) if the following hold:

(ISi) $a \in X$ implies $X \Vdash a$,

(ISii) if $X \Vdash y$ for all $y \in Y$ and $Y \vdash a$, then $X \Vdash a$.

Scott information systems were introduced in [Sco82a] as a logical characterization of order structures arising in denotational semantics. The connection to CCP logic is as follows.

Proposition 6.9 There is a bijective relationship between Scott information systems and deductive systems of CCP logic.

Proof. Consider a Scott information system $(A, \Vdash)$. Using the set A as propositions, we obtain the set of CCP formulae $\mathscr{S}(A)$. A consequence relation $\vdash$ for $\mathscr{S}(A)$ is defined by setting $a_1 \wedge a_2 \wedge \ldots \wedge a_n \vdash b_1 \wedge b_2 \wedge \ldots \wedge b_m$ whenever $\{a_1, a_2, \ldots, a_n\} \Vdash b_i$ for all $i = 1, \ldots m$. We have to verify that $\vdash$ is closed under the rules given in Definition 6.1. For the case $m = 0$ the condition is obviously true so that we obtain axiom (T). Likewise, the conditions for axiom (R) are satisfied due to condition (ISi) in Definition 6.8. Similarly, the (Cut) rule follows immediately from (ISii). For the rules (W1), (W2), and (And), we simply notice that these are direct consequences from our definition of $\vdash$.

Now for the opposite direction, consider a deductive system $(\mathscr{S}(A), \vdash)$. Using the set of propositions of $\mathscr{S}(A)$ as attributes, we construct a Scott information system $(A, \Vdash)$, where we define $\{a_1, a_2, \ldots, a_n\} \Vdash b$ whenever $a_1 \wedge a_2 \wedge \ldots \wedge a_n \vdash b$. Again it is straightforward to check that this is indeed an information system. (ISi) can be deduced from the rules (R) and iterated applications of (W1) and (W2). Under the assumption of (ISii), we see that the (And) rule allows us to construct a conjunction that corresponds to the premise Y of the second rule. By (Cut) this yields the required entailment.

To complete the proof, we note that these two constructions are in fact inverse to each other. The identity on Scott information systems is trivial. For CCP logics, we note that any sequent $a_1 \wedge a_2 \wedge \ldots \wedge a_n \vdash b_1 \wedge b_2 \wedge \ldots \wedge b_m$ induces via (W1)/(W2) the existence of sequents $a_1 \wedge a_2 \wedge \ldots \wedge a_n \vdash b_i$, for all $i = 1, \ldots, m$. The original sequent can then be reconstructed from the entailment of the Scott information system induced from these relations. □

Note that this proposition yields a bijective correspondence, not just a relationship up to isomorphism. Indeed Scott information systems are essentially an efficient formulation of conjunctive propositional logic, where the properties of $\wedge$ are obtained implicitly by using sets in the first place. The category of Scott information systems and approximable mappings between the induced semilattices is denoted **SIS**[4]. From 6.9 one easily concludes that **SIS** is *isomorphic* to **CCP**, and hence also equivalent to all categories mentioned earlier.

Furthermore, approximable mappings between CCP logics need not be expressed on the level of their Lindenbaum algebras, but could be formulated directly on formulae. From this viewpoint, approximable mappings appear as consequence relations between different logical languages. Indeed, all the requirements of Definition 4.2 do still have a very intuitive reading under this interpreta-

[4]Historically, this is indeed the first context for which *approximable mappings* were defined [Sco82a].

tion: (am1) and (am2) correspond to (T) and (And) of Definition 6.1, respectively, while (am3) can be viewed as a modified (Cut) rule, that also subsumes (W1) and (W2). Hence we recognize approximable mappings as a simple form of *multilingual sequent calculi* as introduced in [JKM99] for the more complicated case of *positive logic* (i.e., logics including conjunction and disjunction). Further details and motivation can be found therein.

We remark that one could as well have connected CCP logic or information systems directly to algebraic lattices, instead of presenting the ideal completion for semilattices of compacts. In the case of logics, algebraic lattices are obtained directly as sets of *models* of a deductive system, where models are considered as deductively closed sets of (true) formulae. These turn out to be exactly the filters[5] within the corresponding Lindenbaum algebras, and the duality to ideal completion is immediate. The reader may care to consult [DH01] for a general treatment of such matters. For Scott information systems, algebraic lattices are constructed similarly as sets of *elements*. As defined in [Sco82a], an element of an information system $(A, \Vdash)$ is a subset $x \subseteq A$ such that $a \in x$ whenever there is some finite set $X \subseteq x$ with $X \Vdash a$.

Our logical considerations can also be put to practical use by noting that every *definite logic program* (see, e.g., [Llo87]) can be expressed by a deductive system in the above sense. This has also been mentioned in [Zha03b]. Considering the fact that the theory of definite logic programs is quite well-developed, these insights are merely providing some further explanation for the situation in this field. In the light of the connections to Stone duality outlined below and the immediate connection to algebraic semantics of logical systems, one could also further analyze the situation for more expressive logical languages from this perspective.

Note that only a small portion of Scott information systems and algebraic lattices can be obtained from definite logic programs. The reason is that there are only countably many different programs, but uncountably many Scott information systems (even for countable sets of generators). We also remark that, while algebraicity always makes fixed point computation possible in theory, the specific structure of the information systems of logic programs is employed to ensure that the semantic operator suitable for logic programs is indeed effectively computable.

We do not bother to give a category of logic programs, although this could be done by adjusting the formalism of approximable mappings. However, it is not clear at the moment how the subcategory of algebraic lattices that arises in this way can be characterized.

[5]A filter F is the dual of an ideal: an upper set $F = \uparrow F$ such that $a, b \in F$ implies the existence of some $c \in F$ such that $c \leq a$ and $c \leq b$.

6.2 The Scott topology

Next we want to study the spacial side of Stone duality. It is here where we find the models and their semantic entailment, while the *localic* side is inhabited by syntactic representations and their proof theory. We already mentioned that models in our case take the specific form of algebraic lattices, and thus it is natural to ask which subsets of models correspond to a logical theory. The appropriate collection of sets turns out to be the following well-known topology:

Definition 6.10 Consider a dcpo P. A subset $O \subseteq P$ is *Scott open* if the following hold:

(i) $x \in O$ and $x \leq y$ imply $y \in O$ (O is an upper set),

(ii) for any directed set $D \subseteq P$, $\bigvee D \in O$ implies $D \cap O \neq \emptyset$ (O is inaccessible by directed suprema).

The *Scott topology* on P is the collection of Scott open sets. We use $\sigma(P)$ to denote this collection and $\Sigma(P) = (P, \sigma(P))$ for the resulting topological space.

But one can also reverse the process to obtain orders from topologies:

Definition 6.11 Consider a topological space (X, τ). Then τ defines a *specialization (pre-)order* $\leq$ on X by setting $x \leq y$ whenever $x \in O$ implies $y \in O$ for any $O \in \tau$. A topology on a partially ordered set is called *order consistent* if its specialization order coincides with the order of the poset.

For an algebraic lattice, the Scott topology has some more specific properties. Recall that an open set is *compact* if it is a compact element of the open set lattice, and that a topology is *coherent* if the intersection of any two compact open sets is compact. Proof for the following statements can be found in [AJ94, GHK+03, Joh82].

Proposition 6.12 Consider an algebraic lattice L. We have the following:

(i) $\Sigma(L)$ is order consistent.

(ii) The set $B = \{\uparrow c \mid c \in \mathsf{K}(L)\}$ is a base for $\sigma(L)$.

(iii) The compact opens of $\Sigma(L)$ are exactly the finite unions of members of B.

(iv) $\sigma(L)$ is coherent.

(v) $\sigma(L)$ is sober.[6]

[6]We did not define sobriety in this document. Readers who are not familiar with this concept may safely ignore this statement.

Order consistency insures that algebraic lattices and their Scott topologies uniquely characterize each other. A category $\mathbf{\Sigma_{Alg}}$ of Scott topologies on algebraic lattices is readily obtained by employing continuous maps between topologies as morphisms.

Theorem 6.13 The categories **Alg** and $\mathbf{\Sigma_{Alg}}$ are isomorphic, hence equivalent.

Proof. The required functors are defined on objects by taking the Scott topology and the specialization order of the arguments, respectively. By order consistency of the topologies, this yields a bijection between the classes of objects. Since the carrier sets of lattices and topologies remain unchanged, one can consider every function between algebraic lattices directly as a function between spaces and vice versa. To finish the proof, one needs to show that a function between algebraic lattices is Scott continuous iff it is continuous with respect to the Scott topologies. This standard result can for example be found in [AJ94]. □

In the next section, we see that the topological spaces of $\mathbf{\Sigma_{Alg}}$ are indeed very specific.

6.3 Stone duality

Since the very beginning of the theory, Stone duality has been recognized as a tool for relating proof theory, algebraic semantics, and model theory of logical systems (see [Sto37]). One direction of this investigation has already been mentioned in Section 6.1: Lindenbaum algebras can be represented by corresponding model theories, where models are characterized as subsets (filters) of formulae. Dually, one could also have presented every formula by the set of its models. The conceptual step from such systems of specific subsets to *topological spaces* was the key to the strength and utility of Stone's original representation theorems.

However, it still took decades to recognize that it would be even more advantageous to undo this step to the spacial side of Stone duality and to return to the more abstract world of partially ordered sets. It became apparent that topologies could not only serve as a representation for specific ordered structures, but that conversely orders could serve as a general substitute for topological spaces. Indeed, the leap to the spacial side is usually not an easy one — in many cases it cannot be made within classical Zermelo-Fraenkel set theory (ZF). The localic side on the other hand can mimic most of the features of the original topological setting, while being freed from the weight of points which often prevent purely constructive reasoning.

In what follows we embed our specific scenery into the setting of Stone duality. However, it turns out that the special case we consider does not justify to present the theory in its common generality. Hence we give explicit proofs for the object level relationships in our specialized setting and hint at the connections to more abstract versions of Stone duality where appropriate. Other than providing the merit of a more self-contained presentation, this also enables us to work exclusively in ZF, with no additional choice principles whatsoever. As a general reference on Stone duality, we recommend [Joh82].

The passage from spaces to orders is a particularly simple one: the open set lattice of a topology is already a poset. The class of posets arising in this way are the *spacial locales*.

Definition 6.14 A complete lattice L is a *locale* if the following infinite distributive law holds for all $S \subseteq L$ and $x \in L$:

$$x \wedge \bigvee S = \bigvee \{x \wedge s \mid s \in S\}.$$

A *point of a locale* is a principal prime ideal of L, i.e. a subset $p \subseteq L$ such that $p = \downarrow \bigwedge p$ and, for any $x \wedge y \in p$, $x \in p$ or $y \in p$. The set of all points of L is denoted by $\mathsf{pt}(L)$.

A locale is *spacial* if, for any two elements $x, y \in L$ with $x \not\leq y$, there is a point $p \subseteq L$ such that $x \in p$ and $y \notin p$. L is *spectral* if L is algebraic, its greatest element is compact, and the meet of any two compact elements of L is compact.

We remark that locales are also called *frames*, and that structures of this kind are equivalently characterized as *complete Heyting algebras*.[7]

It is now easy to see that any open set lattice yields a locale, where distributivity follows from the corresponding distributivity of finite intersections over infinite unions. Furthermore, Proposition 6.12 (ii), (iii), and (iv) show that, for an algebraic lattice L, $(\sigma(L), \subseteq)$ is even a spectral locale. We shall find that these locales are even more specific than this.

Our starting point for investigating topologies were algebraic lattices, which we have earlier recognized as the model theories of deductive systems of CCP logics. The abstraction to (certain) spectral locales brings us back to proof theory. We now characterize the above locales by relating them to Lindenbaum algebras of CCP logic, and reobtain topological spaces from this data.

We consider arbitrary meet-semilattices with greatest element, knowing that they are up to isomorphism just the Lindenbaum algebras of CCP (Theorem 6.5). Furthermore, we already mentioned that the collections of all filters (the order-dual concepts of the ideals) of such semilattices are just the algebraic lattices, which follows immediately from Theorem 3.6. We can now give a characterization for the locale of Scott open sets of algebraic lattices:

[7]The interested reader will find definitions and treatment in [Joh82, GHK$^+$03].

Theorem 6.15 Consider a meet-semilattice S with greatest element and the corresponding algebraic lattice $(\mathsf{Flt}(S), \subseteq)$ of filters of S. The collection of lower sets of S, ordered by subset inclusion, is isomorphic to $\sigma(\mathsf{Flt}(S))$. Every Scott open set lattice of an algebraic lattice is of this form.

Proof. Theorem 3.6 shows the bijective correspondence between the elements of S and the compacts of $\mathsf{Flt}(S)$, since S is dually order-isomorphic to $\mathsf{K}(\mathsf{Flt}(S))$. Proposition 6.12 demonstrated that every Scott open set is characterized by the compact elements it contains. Now it is obvious that such sets of compacts correspond to upper sets in the join-semilattice of compacts, and thus to lower sets in its dual meet-semilattice. The other direction is also immediate from the according part of Theorem 3.6. □

Hence the spectral locales of the form $\sigma(L)$ for some algebraic lattice L are more precisely characterized as the lower set topologies of meet-semilattices with greatest element, i.e. as the *Alexandrov topologies* of join-semilattices with least element. Note also that meets and joins within these locales are really given by the corresponding set operations. By $\sigma_{\mathbf{Alg}}$ we denote the category of all locales isomorphic to the collection of lower sets on some meet-semilattice with greatest element together with functions that preserve finite meets and arbitrary joins.[8]

Next we want to connect up with the common constructions of Stone duality.

Lemma 6.16 Consider a meet-semilattice with greatest element S and its locale of lower sets σ. Then the meet-prime elements of σ are exactly the complements of the filters of S.

Proof. Let $F \subseteq S$ be a filter and set $A = S \setminus F \in \sigma$. Now assume there are lower sets $B_1, B_2 \in \sigma$ such that $B_1 \cap B_2 = A$. For a contradiction, assume that there are elements $b_1 \in B_1 \cap F$ and $b_2 \in B_2 \cap F$. Then $b_1 \wedge b_2 \in F$ and $b_1 \wedge b_2 \in B_1 \cap B_2 = A$ — a contradiction. Hence, one of B_1, B_2 contains just the elements of A as required.

Conversely, let $A \in \sigma$ be meet-prime and consider the upper set $F = S \setminus A$. For any two elements $a, b \in F$ it is easy to see that $\downarrow a \cap \downarrow b = \downarrow(a \wedge b)$. Hence, if $a \wedge b \in A$ then $\downarrow a \cup A$ and $\downarrow b \cup A$ are elements of σ with intersection A, which cannot be. Hence $a \wedge b \in F$ as required. □

This gives us all necessary information about the points of these locales (see Definition 6.14), since these were defined to be just the principal ideals generated by meet-prime elements. We can thus identify the set of points $\mathsf{pt}(\sigma)$ with the set

[8]This is of course rather a category of frames and frame homomorphisms than a category of locales (which would be described by its dual). We have chosen to trade some terminological precision for conciseness of the presentation.

of all meet-prime elements of σ.[9] Our insights allow us to give a direct description of the topological spaces associated with semilattices:

Corollary 6.17 Let S be a meet-semilattice with greatest element, let L be an algebraic lattice, and let σ be a spectral locale, such that

- S^{op} is isomorphic to $\mathsf{K}(L)$ and
- σ is isomorphic to $\sigma(L)$.

Then the following are homeomorphic:

(i) $(L, \sigma(L))$, the Scott topology on L;

(ii) the topology on $\mathsf{Flt}(S)$ generated from the basic open sets

$$O_a = \{F \in \mathsf{Flt}(S) \mid a \in F\} \qquad \text{for all } a \in S;$$

(iii) the topology on $\mathsf{pt}(\sigma)$ given by the open sets

$$P_A = \{p \in \mathsf{pt}(\sigma) \mid A \notin p\} \qquad \text{for all } A \in \sigma.$$

Proof. Most of the above should be obvious at this stage, so we spare out some details. Suitable bijections between L, $\mathsf{Flt}(S)$, and $\mathsf{pt}(\sigma)$ have been obtained in 3.6 and 6.16. First we show the homeomorphism between (i) and (ii) (which induces also that (O_a) is indeed a base). For this we only have to note that $O_a = \{F \in \mathsf{Flt}(S) \mid {\uparrow}a \subseteq F\}$. Using the bijection between (principal) filters and (compact) elements from Theorem 3.6, one sees that O_a corresponds to an open set ${\uparrow}c$, $c \in \mathsf{K}(L)$, of (i). The fact that these subsets are open and form the basis for the Scott topology has been shown in Proposition 6.12.

For the homeomorphism between (ii) and (iii), we consider the locale of lower sets of S, which is isomorphic to σ by Theorem 6.15. Clearly this affects the topology of (iii) only up to homeomorphism. Now in the locale of lower sets, a point (principal prime ideal) $p = {\downarrow}B$ is in P_A iff the corresponding meet-prime B does not contain A. But this is the case iff the complement of B intersects A. Hence, by Lemma 6.16, P_A corresponds exactly to the collection of those filters of S that contain some element of A, i.e. to the set $\bigcup\{O_a \mid a \in A\}$. But these are precisely the open sets of the topology of (ii). $\square$

[9]Furthermore, we remark that this guarantees a sufficient supply of prime elements without invoking any additional choice principles, i.e. we are dealing with a class of locales that is spacial in Zermelo-Fraenkel set theory. This contrasts with the class of all spectral locales, which is only spacial when the existence of prime ideals is explicitly postulated, i.e. when the *Boolean prime ideal theorem* [DP02] is assumed to hold.

With respect to the given preconditions on the relationship between S, L, and σ, note that the various transformations between semilattices, algebraic lattices, and locales established earlier yield a variety of equivalent ways to state that the three given objects describe “the same thing”.

To complete the targeted categorical equivalence between the dual category of $\sigma_{\mathbf{Alg}}$ and $\mathbf{\Sigma}_{\mathbf{Alg}}$ (**Alg**, $\mathbf{Sem}_{\vee}$, ...), one still needs to prove a suitable bijection of hom-sets. This correspondence between inverse frame homomorphisms and continuous functions is a basic result of Stone duality which we will not repeat here. See [Joh82] for details.

7 Summary and further results

We provided characterizations of the category of algebraic lattices by means of structures from logic, topology, domain theory, and formal concept analysis. More precisely, we characterized algebraic lattices by certain semilattices, formal contexts, and deductive systems of the conjunctive fragment of propositional logic. The novel category **Cxt** of formal contexts and approximable mappings was used to establish the cartesian closure of these categories, and the categorical constructions needed for this were explicitly given. Other representations referred to special classes of closure systems, Scott topologies, locales, and definite logic programs. An overview of the major equivalences given herein is displayed in Figure 2.

Although this treatment is quite comprehensive, one could still add some more equivalent formalisms. Especially, we left out the *coverage technique* of [Joh82] (see also [Sim04]), which represents locales in a syntactical way that relates closely to Scott information systems. Furthermore, we deliberately ignored Scott's earlier approach to presenting domains via *neighborhood systems* [Sco82b], since these structures are not much more than a mixture of the later (token-set based) information systems and continuous closure operators. Finally, one could also identify the classes of distributive lattices that arise as the compact elements of the spectral locales we considered as the free distributive lattices over the underlying semilattice.

In this article we have also presented a unified treatment of the basic techniques and mechanisms that are applied to join domain theory, algebra, logic, and topology. Algebraic lattices turn out to be the simplest case where such a discussion is feasible. Part of the given results have been generalized in various ways, some of which are subject to current research. A common way to generalize the above results is to extend the logic under consideration. A technique for including “negation-like” constraints without need for an internal negation operation has been employed by Scott in his original formulation of information systems

$$\begin{tikzcd}
\sigma_{\mathbf{Alg}} \arrow[rrrr, leftrightarrow, "(6.17)" description, "\Omega"{pos=0.03}, "\mathsf{pt}"'{pos=0.97}] \arrow[ddd, leftrightarrow, "(6.15)" description, "\mathsf{Down}"'{pos=0.03}] \arrow[rrrrd, leftrightarrow, bend right=10, "(6.16)" description, "\sigma"'{pos=0.03}, "\mathsf{Prm}"{pos=0.97}] & & & & \Sigma_{\mathbf{Alg}} \arrow[d, leftrightarrow, "(6.12)" description, "\Sigma"'{pos=0.05}, "\leq"{pos=0.95}] \\
& & & & \mathbf{Alg} \arrow[dl, leftrightarrow, "(5.3)" description, "\mathsf{Alg}"{pos=0.05}] \arrow[dd, leftrightarrow, "(3.6)" description, "\mathsf{Idl}"{pos=0.05}, "\mathsf{K}"{pos=0.95}] \\
& \mathbf{CCP} \arrow[r, leftrightarrow, "(6.9)" description] & \mathbf{SIS} & \mathbf{Cxt} \arrow[dr, leftrightarrow, "(5.2)" description, "\mathsf{Sem}"'{pos=0.95}] & \\
\mathbf{Sem}_\wedge \arrow[rrrruu, bend left=30, "(3.6)^{\mathsf{op}}" description, "\mathsf{Flt}"'{pos=0.97}] \arrow[ur, leftrightarrow, "(6.5)" description, "\mathsf{LA}"'{pos=0.03}] \arrow[rrrr, leftrightarrow, ".^{\mathsf{op}}"'{pos=0.03}, ".^{\mathsf{op}}"'{pos=0.97}] & & & & \mathbf{Sem}_\vee
\end{tikzcd}$$

Figure 2: Summary of all established equivalences with reference to the corresponding (object-level) statements. Labels at the arrow tips specify the name of the functor that was used in a construction, where Down denotes the construction of the lower set topology from a meet-semilattice, Prm yields the set of principal prime ideals of a locale, ordered by subset inclusion, and ≤ denotes the construction of the specialization order from a topological space.

[Sco82a]. There he introduced a collection of finite subsets of propositions that are consistent, assuming that no inconsistent sets can be mapped to true by any model. This procedure can be viewed as an extension of the deductive system that allows statements of the form "$X \Vdash$", interpreted as $\bigwedge X \Vdash \bot$, where $\bot$ is the constant *false* — a construction well-known under the notion of *integrity constraint* in database theory. Clearly, $\bot$ will then represent the least element in the resulting Lindenbaum algebras. However, as important as introducing the constant $\bot$ into the logical language is a change of the model theory: models now have to be *proper* filters, i.e. the case that all (including $\bot$) formulae evaluate to true is excluded. The posets of models for such logics turn out to be exactly the Scott domains (the bounded complete algebraic cpos).

As another step, one can include disjunction into the formalism. This already leads to a substantial complication of the theory: choice principles are now needed to obtain models. Since logical conjunction and disjunction are classically assumed to distribute over each other, one obtains all (bounded) distributive lattices as Lindenbaum algebras. In place of algebraic lattices one finds a curious class of dcpos that have been termed *information domains* in [DG90]. Later the direct construction of distributive lattices and locales from such deductive systems was studied in [CC00] and [CZ00], and in [RZ01] Smyth powerdomains were characterized by similar means using a clausal logic which was also extended to non-monotonic reasoning paradigms on hierarchical knowledge [RZ01, Hit04]. Other than this, one can apply all the representation machinery that has been set up for distributive lattices, including both Stone's and Priestley's representation theorems for these structures ([Joh82]).

Further strengthening of the logic is possible by including some internal negation operation. Intuitionistic negation yields Heyting algebras as Lindenbaum algebras. The resulting topologies are already studied in [Sto37], though the significance of specialization orders and domain theoretic concepts were not yet recognized at this time. If classical negation is preferred instead, thus yielding classical propositional logic, the class of Boolean algebras provides the well-known algebraic semantics. While topological representation via Stone's theorem is rather pleasant in this case, the domain theoretic aspects are quite disappointing: the specialization order of models is discrete. Related approaches nevertheless have been taken for the context of negation in logic programming [Sed95, Hit04], but the domain-theoretic content of these investigations remains to be determined.

For reasons as those just described, internal negation is usually not considered in domain-theoretical studies. However both inconsistency of finite subsets and finite disjunctions can be employed with various restrictions to obtain classes of domains that are more general than the Scott domains. A slight constraint on either the logical ([DG90]) or the localic level ([Abr91]) restricts the obtained class of dcpos (of models) to the coherent algebraic dcpos. However, while this is a well-known concept in domain theory, it results in rather unusual restrictions on the logics (Lindenbaum algebras, locales). Further conditions will lead to SFP-domains [Abr91, Zha91]. On the other hand, conditions that characterize a class of deductive systems that produces exactly the L-domains have been studied in [Zha92].

References

[Abr91] S. Abramsky. Domain theory in logical form. *Annals of Pure and Applied Logic*, 51:1–77, 1991.

[AJ94] S. Abramsky and A. Jung. Domain theory. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume III. Oxford University Press, 1994.

[Bor94] F. Borceux. *Handbook of Categorical Algebra 1: Basic Category Theory*, volume 53 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 1994.

[CC00] J. Cederquist and T. Coquand. Entailment relations and distributive lattices. In S. Buss, P. Hájek, and P. Pudlák, editors, *Proceedings of the Annual European Summer Meeting of the Association for Symbolic Logic, Prague, Czech Republic, 1998*, volume 13 of *Lecture Notes in Logic*. Association for Symbolic Logic, 2000.

[CZ00] T. Coquand and G.-Q. Zhang. Sequents, frames and completeness. In P. Clote and H. Schwichtenberg, editors, *Proceedings of the Annual Conference of the European Association for Computer Science Logic (CLS2000), Fischbachau/Munich, Germany*, volume 1862 of *Lecture Notes in Computer Science*. Springer, 2000.

[DG90] M. Droste and R. Göbel. Non-deterministic information systems and their domains. *Theoretical Computer Science*, 75:289–309, 1990.

[DH01] J. M. Dunn and G. M. Hardegree. *Algebraic methods in philosophical logic*. Clarendon Press, 2001.

[DP02] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, second edition, 2002.

[GHK$^+$03] G. Gierz, K. H. Hofmann, K. Keimel, J. D. Lawson, M. Mislove, and D. S. Scott. *Continuous Lattices and Domains*, volume 93 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 2003.

[GW99] B. Ganter and R. Wille. *Formal Concept Analysis — Mathematical Foundations*. Springer, 1999.

[Hit04] P. Hitzler. Default reasoning over domains and concept hierarchies. In S. Biundo, T. Frühwirth, and G. Palm, editors, *Proceedings of the 27th German conference on Artificial Intelligence, KI'2004, Ulm, Germany, September 2004*, volume 3238 of *Lecture Notes in Artificial Intelligence*, pages 351–365. Springer, Berlin, 2004.

[HW03] P. Hitzler and M. Wendt. Formal concept analysis and resolution in algebraic domains. In A. de Moor and B. Ganter, editors, *Using Conceptual Structures — Contributions to ICCS 2003*, pages 157–170. Shaker Verlag, Aachen, 2003.

[HZ04] P. Hitzler and G.-Q. Zhang. A cartesian closed category of approximable concept structures. In K.-E. Wolff, H. D. Pfeiffer, and H. S. Delugach, editors, *Proceedings of the International Conference On Conceptual Structures, Huntsville, Alabama, USA*, Lecture Notes in Computer Science, pages 170–185. Springer, July 2004.

[JKM99] A. Jung, M. Kegelmann, and M. A. Moshier. Multi lingual sequent calculus and coherent spaces. *Fundamenta Informaticae*, XX:1–42, 1999.

[Joh82] P. T. Johnstone. *Stone spaces*. Cambridge University Press, 1982.

[Llo87] J. W. Lloyd. *Foundations of Logic Programming*. Springer Verlag, 2nd extended edition, 1987.

[LR03] F. W. Lawvere and R. Rosebrugh. *Sets for mathematics*. Cambridge University Press, 2003.

[Mac71] S. Mac Lane. *Categories for the Working Mathematician*. Springer, 1971.

[McL92] C. McLarty. *Elementary categories, elementary toposes*. Clarendon Press, 1992.

[RZ01] W. C. Rounds and G.-Q. Zhang. Clausal logic and logic programming in algebraic domains. *Information and Computation*, 171(2):156–182, 2001.

[Sco82a] D. S. Scott. Domains for denotational semantics. In M. Nielsen and E. M. Schmidt, editors, *Proceedings of the 9th Colloquium on Automata, Languages and Programming, Aarhus, Denmark (ICALP'82)*, volume 140 of *Lecture Notes in Computer Science*. Springer, 1982.

[Sco82b] D. S. Scott. Lectures on a mathematical theory of computation. In M. Broy and G. Schmidt, editors, *Theoretical Foundations of Programming Methodology*, pages 145–292. Carnegie-Mellon University, Department of Computer Science, Pittsburgh, D. Reidel Publishing Company, 1982.

[Sed95] A. K. Seda. Topology and the semantics of logic programs. *Fundamenta Informaticae*, 24(4):359–386, 1995.

[Sim04] H. Simmons. The coverage technique for enriched posets. *Available from the author's homepage* `www.cs.man.ac.uk/~hsimmons`, 2004.

[Smy92] M. B. Smyth. Topology. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume I. Oxford University Press, 1992.

[Sto37] H. M. Stone. Topological representations of distributive lattices and Brouwerian logics. *Časopis pro Pěstování Matematiky a Fysiky*, 67:1–25, 1937.

[Stu02] G. Stumme. Formal concept analysis on its way from mathematics to computer science. In U. Priss, D. Corbett, and G. Angelova, editors, *Conceptual Structures: Integration and Interfaces, Proc. ICCS 2002*, LNAI, pages 2–19. Springer, 2002.

[Wil82] R. Wille. Restructuring lattice theory: An approach based on hierarchies of concepts. In I. Rival, editor, *Ordered Sets*, pages 445–470. Reidel, Dordrecht-Boston, 1982.

[Zha91] G.-Q. Zhang. *Logic of Domains*. Birkhauser, Boston, 1991.

[Zha92] G.-Q. Zhang. Disjunctive systems and L-domains. In W. Kuich, editor, *Proceedings of the 19th International Colloquium on Automata, Languages, and Programming (ICALP'92), Vienna, Austria*, volume 623 of *Lecture Notes in Computer Science*. Springer, 1992.

[Zha03a] G.-Q. Zhang. Chu spaces, concepts lattices, and domains. In *Proceedings of the 19th Conference of the Mathematical Foundations of Programming Semantics, Montreal, Canada, 2003*, volume 83 of *Electronic Notes in Theoretical Computer Science*, 2003.

[Zha03b] G.-Q. Zhang. Topology, lattices, and logic programming. Presentation at the DIMACS Lattice Workshop, Juli 8–10, 2003.

[ZR04] G.-Q. Zhang and W. Rounds. Reasoning with power defaults. *Theoretical Computer Science*, 323(1–3):321–350, 2004.

[ZS0x] G.-Q. Zhang and G. Shen. Approximable concepts, Chu spaces, and information systems. *Theory and Applications of Categories*, 200x. To appear.

Journal of ELECTRICAL ENGINEERING, VOL. 55, NO. 1-2, 2004, 25–30

A GENERALIZED RESOLUTION THEOREM

Pascal Hitzler [*]

W.C. Rounds and G.-Q. Zhang have recently proposed to study a form of resolution on algebraic domains [2]. This framework allows reasoning with knowledge which is hierarchically structured and forms a (suitable) domain, more precisely, a coherent algebraic cpo as studied in domain theory. In this paper, we give conditions under which a resolution theorem — in a form underlying resolution-based logic programming systems — can be obtained. The investigations bear potential for engineering new knowledge representation and reasoning systems on a firm domain-theoretic background.

K e y w o r d s: domain theory; automated theorem proving; domain logics; resolution

1 INTRODUCTION

Domain Theory [2] is an abstract mathematical theory for programming semantics and has grown into a respected field on the borderline between mathematics and computer science. Relationships between domain theory and logic were noted early on by Scott [3], and subsequently developed by many authors, including Smyth [4], Abramsky [5], and Zhang [6]. There has been much work on the use of domain logics as logics of types and of program correctness, with a focus on functional and imperative languages. However, there has been only little work relating domain theory to logic programming or other AI paradigms, two exceptions being the application of methods from quantitative domain theory to the semantic analysis of logic programming paradigms studied by Hitzler and Seda [7, 8], and the work of Rounds and Zhang on the use of domain logics for disjunctive logic programming and default reasoning [1 9].

The latter authors, in [1], introduced a form of clausal logic generalized to coherent algebraic domains, motivated by theoretical investigations into the logical nature of ordered spaces occuring in domain theory. In essence, they propose to interpret finite sets of compact elements as abstract formal clauses, yielding a theory which links standard domain-theoretic notions to corresponding logical notions. Amongst other things, they establish a sound and complete proof theory based on a generalized resolution rule, and a form of disjunctive logic programming in domains. A corresponding semantic operator turns out to be Scott-continuous.

In this paper, we study this clausal logic, henceforth called *logic RZ* for convenience. The occurrence of a proof theory based on a generalized resolution rule poses the question whether results underlying resolution-based logic programming systems can be carried over to the logic RZ. One of the most fundamental results underlying these systems is the *resolution theorem* which states that a clause X is a logical consequence of a theory T if and only if it is possible to derive a contradiction, *ie*the empty clause, via resolution from the theory $T \cup \{\neg X\}$ [10, 11].

What we just called *resolution theorem* is certainly an immediate consequence of the fact that resolution is sound and complete for classical logic. However, it is not obvious how it can be transfered to the logic RZ, mainly because it necessitates negating a clause, and negation is not available in the logic RZ in explicit form. This observation will lead our thoughts, and in the end we will develop conditions on the underlying domain which ensure that a negation is present which allows to prove an analogon of the theorem.

The paper is structured as follows. In Section 2 we review the most fundamental definitions from the logic RZ, as laid out in [1]. In Section 2.2 we recall the corresponding proof theory, based on a form of resolution for this framework. In Section 3 we will simplify the proof theory and provide a rule system which is simpler and easier to work with. The remainder of the paper is devoted to determining conditions under which a resolution theorem, in the form mentioned above, can be proven for the logic RZ. These conditions will involve atomicity of the underlying domain, studied in Section 4, and a form of negation for these spaces, studied in Section 5. We will conclude in Section 6.

An extended abstract of this paper appeared in [12].

2 PRELIMINARIES

2.1 The Logic RZ

A *partially ordered set* is a pair $(D, \sqsubseteq)$, where D is a nonempty set and $\sqsubseteq$ is a relexive, antisymmetric, and transitive relation on D. A subset X of a partially ordered set is *directed* if for all $x, y \in X$ there is $z \in X$ with $x, y \sqsubseteq z$. An *ideal* is a directed and downward closed set. A *complete partial order*, *cpo* for short, is a partially ordered set $(D, \sqsubseteq)$ with a least element $\bot$, called the *bottom element* of $(D, \sqsubseteq)$, and such that every directed set in D has a least upper bound, or supremum, $\bigsqcup D$. An element $c \in D$ is said to be *compact* or *finite* if whenever $c \sqsubseteq \bigsqcup L$ with L directed, then there exists $e \in L$ with $c \sqsubseteq e$. The set of all compact elements of a cpo D is denoted by $\mathsf{K}(D)$. An *algebraic cpo* is a cpo such that every $e \in D$ is the directed supremum of all compact elements below it.

* Artificial Intelligence Institute, Department of Computer Science, Dresden University of Technology, Germany, E-mail: phitzler@t-online.de

A set $U \subseteq D$ is said to be *Scott open*, or just *open*, if it is upward closed and for any directed $L \subseteq D$ we have $\bigsqcup L \in U$ if and only if $U \cap L \neq 0$. The *Scott topology* on D is the topology whose open sets are all Scott open sets. An open set is *compact open* if it is compact in the Scott topology. A *coherent algebraic cpo* is an algebraic cpo such that the intersection of any two compact open sets is compact open. This coincides with the coherency notion defined in [2], which may be consulted as basic reference for domain theory. We will not make use of many topological notions in the sequel. So let us note that coherency of an algebraic cpo implies that the set of all minimal upper bounds of a finite number of compact elements is finite, *ie*if $c_1, \dots, c_n$ are compact elements, then the set $\operatorname{mub}\{c_1, \dots, c_n\}$ of minimal upper bounds of these elements is finite. Note that $\operatorname{mub}\emptyset = \{\bot\}$, where $\bot$ is the least element of D.

In the following, $(D, \sqsubseteq)$ will always be assumed to be a coherent algebraic cpo. We will also call these spaces *domains*. Two elements $c, d \in D$ are called *inconsistent*, symbolically $c \not\uparrow d$, if c and d have no common upper bound.

Following [13], an element $a \in D$ is called an *atom*, or an *atomic element*, if whenever $x \sqsubseteq a$ we have $x = a$ or $x = \bot$. The set of all atoms of a domain is denoted by $\mathsf{A}(D)$.

Definition 2.1. Let D be a coherent algebraic cpo with set $\mathsf{K}(D)$ of compact elements. A *clause* is a finite subset of $\mathsf{K}(D)$. We denote the set of all clauses over D by $\mathcal{C}(D)$. If X is a clause and $w \in D$, we write $w \models X$ if there exists $x \in X$ with $x \sqsubseteq w$, *ie* X contains an element below w.

A *theory* is a set of clauses, which may be empty. An element $w \in D$ is a *model* of a theory T, written $w \models T$, if $w \models X$ for all $X \in T$ or, equivalently, if every clause $X \in T$ contains an element below w.

A clause X is called a *logical consequence* of a theory T, written $T \models X$, if $w \models T$ implies $w \models X$. If $T = \{E\}$, then we write $E \models X$ for $\{E\} \models X$. Note that this holds if and only if for every $w \in E$ there is $x \in X$ with $x \sqsubseteq w$.

For two theories T and S, we say that $T \models S$ if $T \models X$ for all $X \in S$. We say that T and S are (*logically*) *equivalent*, written $T \sim S$, if $T \models S$ and $S \models T$. In order to avoid confusion, we will throughout denote the empty clause by $\{\}$, and the empty theory by $\emptyset$. A theory T is (*logically*) *closed* if $T \models X$ implies $X \in T$ for all clauses X. It is called *consistent* if $T \not\models \{\}$ or, equivalently, if there is w with $w \models T$.

Rounds and Zhang originally set out to characterize logically the notion of *Smyth powerdomain* of coherent algebraic cpos. It naturally lead to the clausal logic RZ from Definition 2.1. Indeed, as was shown in [1], the Smyth powerdomain of any coherent algebraic domain is isomorphic to the set of all consistent closed theories over the domain, ordered by set-inclusion. A corollary from the proof is that a clause is a logical consequence of a theory if and only if it is a logical consequence of a finite subset of the theory, which is a compactness theorem for the logic RZ.

Example 2.2. In [1], the domain $\mathbb{T}^\omega$ from [14], here denoted $\mathbb{T}^\mathcal{V}$, was given as a running example. Consider some three-valued logic in the propositional case, with the usual (knowledge)-ordering on the set $\mathbb{T} = \{\mathbf{f}, \mathbf{u}, \mathbf{t}\}$ of truth values given by $\mathbf{u} < \mathbf{f}$ and $\mathbf{u} < \mathbf{t}$. This induces a pointwise ordering on the space $\mathbb{T}^\mathcal{V}$ of all interpretations (or *partial truth assignments*), where $\mathcal{V}$ is the (countably infinite) set of all propositional variables in the language under consideration. The partially ordered set $\mathbb{T}^\mathcal{V}$ is a coherent algebraic cpo. Compact elements in $\mathbb{T}^\mathcal{V}$ are those interpretations which map all but a finite number of propositional variables to $\mathbf{u}$. We denote compact elements by strings such as $pq\overline{r}$, which indicates that p and q are mapped to $\mathbf{t}$ and r is mapped to $\mathbf{f}$.

We note that $\{e \mid e \models \phi\}$ is upward-closed for any logical formula ϕ if considering *eg* Kleene's strong three-valued logic, which has been recognized as being important in a logic programming context [15]. A clause in $\mathbb{T}^\mathcal{V}$ is a formula in disjunctive normal form, *eg* $\{pq\overline{r}, \overline{p}q, r\}$ translates to $(p \wedge q \wedge \neg r) \vee (\neg p \wedge q) \vee r$.

We also note that every compact element in $\mathbb{T}^\mathcal{V}$ can be uniquely expressed as the supremum of a finite number of atomic elements, and the set of all atomic elements is $\mathsf{A}\left(\mathbb{T}^\mathcal{V}\right) = \mathcal{V} \cup \{\overline{v} \mid v \in \mathcal{V}\}$. Furthermore, there exists a bijective function $^- : \mathsf{A}\left(\mathbb{T}^\mathcal{V}\right) \to \mathsf{A}\left(\mathbb{T}^\mathcal{V}\right) : p \to \overline{p}$ which extends naturally to a Scott-continuous involution on all of $\mathbb{T}^\mathcal{V}$ via $\overline{p_1 \dots p_n} = \overline{p_1} \dots \overline{p_n}$. In the following, a clause over a domain D will be called an *atomic clause* if it is a finite subset of $\mathsf{A}(D)$. Atomic clauses on $\mathbb{T}^\mathcal{V}$ correspond to propositional clauses in the classical sense. Note that $p \not\uparrow \overline{p}$ for $p \in \mathsf{A}\left(\mathbb{T}^\mathcal{V}\right)$ and in general for all $c \in \mathsf{K}\left(\mathbb{T}^\mathcal{V}\right)$ we have $c \not\uparrow \overline{c}$.

The following example shows how knowledge can be represented in algebraic domains. For convenience, examples will be presented as subsets of $\mathbb{T}^\mathcal{V}$, in the notation from Example 2.2.

Example 2.3. Consider the subspace of $\mathbb{T}^\mathcal{V}$ constituted by the elements $\bot$, b (is a bird), f (flies), $\overline{f}$ (does not fly), a (lives in australia), s (lives near south pole), $b\overline{f}s$ (is a penguin), and $b\overline{f}a$ (is an ostrich). Then *eg* $\{\{b\}, \{\overline{f}\}\} \models \{a, s\}$.

As to the knowledge representation capabilities of the logic RZ, we remark that some first investigations have exhibited a strong link to formal concept analysis [16, 17].

2.2 Resolution in the logic RZ

In [1], a sound and complete proof theory, using *clausal hyperresolution*, was given as follows, where $\{X_1, \dots, X_n\}$ is a clause set and Y a clause.

$$\frac{X_i; \quad a_i \in X_i \quad (i \le n); \quad \operatorname{mub}\{a_i \mid i \le n\} \models Y}{Y \cup \bigcup_{i \le n} (X_i \setminus \{a_i\})} \qquad \text{(hr)}$$

This rule is sound in the following sense: Whenever $w \models X_i$ for all i, then for any admissible choice of the a_i and Y in the antecedent, we have $w \models Y \cup \bigcup_{i=1}^{n}(X_i \setminus \{a_i\})$.

For completeness, it is necessary to adjoin to the above clausal hyperresolution rule a special rule which allows the inference of any clause from the empty clause. We indicate this rule as follows.

$$\frac{\{\}; \qquad Y \in \mathcal{C}(D)}{Y} \quad (\mathsf{spec})$$

With this addition, given a theory T and a clause X with $T \models X$, we have that $T \vdash^* X$, where $\vdash^*$ stands for a finite number of applications of the clausal hyperresolution rule together with the special rule.

Furthermore, [1, Remark 4.6] shows that binary hyperresolution, together with (spec), is already complete, *ie* the system consisting of the *binary clausal hyperresolution* rule

$$\frac{X_1 \quad X_2; \quad a_i \in X_i; \quad \mathrm{mub}\{a_1, a_2\} \models Y}{Y \cup (X_1 \setminus \{a_1\}) \cup (X_2 \setminus \{a_2\})} \quad (\mathsf{bhr})$$

together with the special rule is sound and complete.

If the set $\{a_1, a_2\}$ is inconsistent, then $\mathrm{mub}\{a_1, a_2\} = \{\}$. Since $\{\} \models \{\}$, clausal hyperresolution generalizes the usual notion of resolution, given by the following rule.

$$\frac{X_1 \quad X_2; \quad a_i \in X_i; \qquad a_1 \not\uparrow a_2}{(X_1 \setminus \{a_1\}) \cup (X_2 \setminus \{a_2\})} \quad (\mathsf{r})$$

Example 2.4. Returning to Example 2.3, note that *eg* $\{\{b\}, \{\overline{f}\}\} \vdash \{b\overline{f}s, b\overline{f}a\}$ using (bhr).

3 SIMPLIFYING THE RESOLUTION SYSTEM

Note that two special instances of the clausal hyperresolution rule are as follows, which we call the *reduction rule* and the *extension rule*.

$$\frac{X; \quad \{a, y\} \subseteq X; \quad y \sqsubseteq a}{X \setminus \{a\}} \quad (\mathsf{red}),$$

$$\frac{X; \quad y \in \mathsf{K}(D)}{\{y\} \cup X} \quad (\mathsf{ext})$$

Indeed, the first rule follows from (hr) since $a \in X$ and $\{a\} \models \{y\}$, while the latter rule follows since $\{a\} \models \{a, y\}$ for all $y \in \mathsf{K}(D)$. The special rule (spec) can be understood as an instance of (ext). Note also that resolution (r) together with (ext) and (red) is not complete. In order to see this, we refer again to Example 2.2. Let $T = \{\{p\}, \{q\}\}$ and $X = \{pq\}$. Then $T \models X$ but there is no way to produce X from T using (r), (ext) and (red) alone. Indeed, it is easy to show by induction that any X which can be derived from T by using only (r), (ext) and (red), contains either p or q, which suffices.

It is our desire to provide a sound and complete system whose rules are as simple as possible. Consider the following rule, which we call *simplified hyperresolution.* It is easy to see that it is an instance of (hr) and more general than (r).

$$\frac{X_1 \quad X_2; \quad a_i \in X_i}{\mathrm{mub}\{a_1, a_2\} \cup (X_1 \setminus \{a_1\}) \cup (X_2 \setminus \{a_2\})} \quad (\mathsf{shr})$$

Theorem 3.1. *The system consisting of* (shr), (ext) *and* (red) *is complete.*

P r o o f . In order to show completeness, we derive (bhr) from (shr), (ext) and (red). Let X_1, X_2 be given with $a_1 \in X_1$ and $a_2 \in X_2$ with $a_1 \uparrow a_2$. Furthermore, let Y be a clause with $\mathrm{mub}\{a_1, a_2\} \models Y$. Let $\mathrm{mub}\{a_1, a_2\} = \{b_1, \ldots, b_n\}$. Then for every b_i there exists $y_i \in Y$ with $y_i \sqsubseteq b_i$. Using (shr), from X_1 and X_2 we can derive $X_3 = \mathrm{mub}\{a_1, a_2\} \cup (X_1 \setminus \{a_1\}) \cup (X_2 \setminus \{a_2\})$, and with repeated application of (ext) and (red) we obtain from this $X_4 = \{y_1, \ldots, y_n\} \cup (X_1 \setminus \{a_1\}) \cup (X_2 \setminus \{a_2\})$. Finally, using (ext) repeatedly, we can add to X_4 all remaining elements from Y. The argumentation for $a_1 \not\uparrow a_2$ is similar. This completes the proof.

We note that a rule with weaker preconditions than (red) suffices, which we call the *weakening rule*:

$$\frac{X; \qquad a \in X; \qquad y \sqsubseteq a}{\{y\} \cup (X \setminus \{a\})} \quad (\mathsf{w})$$

Indeed, (red) can be derived from (w) as follows. Let $\{a, y\} \subseteq X$ with $y \sqsubseteq a$. Then in particular $a \in X$, *ie*using (w) we can derive $\{y\} \cup (X \setminus \{a\})$ which is equal to $X \setminus \{a\}$ since y is already contained in X. On the other hand, (w) can be derived from (red) and (ext) as follows. Let $a \in X$ and $y \sqsubseteq a$. If $a = y$ then there is nothing to show, so assume $a \neq y$. Then $X \vdash X \cup \{y\}$ by the extension rule, so the reduction rule can be applied, yielding $(X \cup \{y\}) \setminus \{a\}$ as required.

The following technical result is inspired by [18, Theorem 7].

Proposition 3.2. *For clauses* $X_1, \ldots, X_n$ *we have* $\{X_1, \ldots, X_n\} \models X$ *if and only if* $\{\{a_1\}, \ldots, \{a_n\}\} \models X$ *for all* $(a_1, \ldots, a_n) \in X_1 \times \ldots \times X_n$.

P r o o f . Assume $\{X_1, \ldots, X_n\} \models X$ and let $a_i \in X_i$ be arbitrarily chosen for $i = 1, \ldots, n$. Then $\{a_i\}$ X_i for all $i = 1, \ldots n$ by (ext) and therefore $\{\{a_1\}, \ldots, \{a_n\}\} \models \{X_1, \ldots, X_n\} \models X$.

Conversely, assume that $\{\{a_1\}, \ldots, \{a_n\}\} \models X$ for all $(a_1, \ldots, a_n) \in X_1 \times \ldots \times X_n$ and let $w \in D$ with $w \models \{X_1, \ldots, X_n\}$, *ie*$w \models X_i$ for all $i = 1, \ldots, n$. Then for all $i = 1, \ldots, n$ there is $a_i \in X_i$ with $a_i \sqsubseteq w$. So for all $i = 1, \ldots, n$ choose a_i with $a_i \sqsubseteq w$. Then $w \models \{\{a_1\}, \ldots, \{a_n\}\}$ and by assumption we obtain $w \models X$.

We call the system consisting of the rules (red), (ext) and (shr) the RAD system, from *Resolution in Algebraic Domains.* For two theories T and S, we write $T \vdash^* S$

if $T \vdash^* A$ for each $A \in S$, and for clauses X and Y we write $X \vdash^* Y$, respectively $X \vdash^* T$, for $\{X\} \vdash^* Y$, respectively $\{X\} \vdash^* T$. The symbol $\vdash$ denotes derivation by a single application of one of the rules in RAD. With slight abuse of notation, for two theories T and S we allow to write $T \vdash S$ if $T \vdash X$ for some clause X and $S \subseteq T \cup \{X\}$.

We interpret the RAD rules in the setting of Example 2.2. We already know that clauses correspond to formulas in disjunctive normal form (DNF), and theories to sets of DNF formulas. The weakening rule acts on single clauses and replaces a conjunction contained in a DNF formula by a conjunction which contains a subset of the propositional variables contained in the original conjunction, *eg* $(p \wedge q) \vee r$ becomes $p \vee r$. The extension rule disjunctively extends a DNF formula by a further conjunction of propositional variables, *eg* $(p \wedge q) \vee r$ becomes $(p \wedge q) \vee r \vee (s \wedge q)$. The simplified hyperresolution rule finally takes two DNF formulas, deletes one conjunction from each of them, and forms a disjunction from the resulting formulas together with the conjunction of the deleted items, *eg* $(p \wedge q) \vee r$ and $\neg p \vee (s \wedge r)$ can be resolved to $(p \wedge q) \vee (r \wedge \neg p) \vee (s \wedge r)$.

A more abstract interpretation of the RAD system comes from a standard intuition underlying domain theory. Elements of the domain D are interpreted as pieces of information, and if $x \sqsubseteq y$, this represents that y contains more information than x. Compact elements are understood as items which are computationally accessible. From this point of view, RAD gives a calculus for reasoning about disjunctive information in computation, taking a clause, *ie*a finite set of computationally accessible information items as disjunctive knowledge about these items. The rules from RAD yield a system for deriving further knowledge from the given disjunctive information. The weakening rule states that we can replace an item by another one which contains less information. The extension rule states that we can always extend our knowledge disjunctively with further bits of information. Both rules decrease our knowledge. The simplified hyperresolution rule states that we can disjunctively merge two collections of disjunctive information, while strengthening our knowledge by replacing two of the items from the collections by an item which contains both pieces of information, and deleting the original items.

EXAMPLE 3.3. For Example 2.3, note that $\{\{b\}, \{\overline{f}\}\} \vdash \{b\overline{f}s, b\overline{f}a\}$ using (shr), $\{b\overline{f}s, b\overline{f}a\} \vdash \{s, b\overline{f}a\}$ using (w), and finally $\{s, b\overline{f}a\} \vdash \{s, a\}$ using (w) again.

4 ATOMIC DOMAINS

We simplify proof search via resolution by requiring stronger conditions on the domain.

DEFINITION 4.1. An *atomic domain* is a coherent algebraic cpo D with the following property: For all $c \in \mathsf{K}(D)$, the set $\mathsf{A}(c) = \{p \in \mathsf{A}(D) \mid p \sqsubseteq c\}$ is finite and $c = \bigsqcup \mathsf{A}(c)$.

The domain $\mathbb{T}^{\mathcal{V}}$ from Example 2.2 is an example of an atomic domain. In the remainder of this section, D will always be an atomic domain.

We seek to represent a clause X by a finite set $\mathsf{A}(X)$ of atomic clauses which is logically equivalent to X. Given $X = \{a_1, \ldots, a_n\}$, we define $\mathsf{A}(X)$ as follows.

$$\mathsf{A}(X) = \{\{b_1, \ldots, b_n\} \mid b_i \in \mathsf{A}(a_i) \text{ for all } i = 1, \ldots, n\}$$

Then the following theorem holds.

THEOREM 4.2. *For any clause X we have $\mathsf{A}(X) \sim \{X\}$.*

P r o o f . For a clause $X = \{a_1, \ldots, a_n\}$ set $X/a_1 = \{\{b, a_2, \ldots, a_n\} \mid b \in \mathsf{A}(a_1)\}$. Then $X/a_1 \models X$. Indeed, since $\bigsqcup \mathsf{A}(a_1) = a_1$ we obtain $\operatorname{mub} \mathsf{A}(a_1) \models \{a_1\}$, and therefore $X/a_1 \vdash^* X$ from (hr).

Now let $X = \{a_1, \ldots, a_n\}$ and let $Y = \{b_1, \ldots, b_n\} \in \mathsf{A}(X)$ with $b_i \in \mathsf{A}(a_i)$ for all i. Then $b_i \sqsubseteq a_i$ for all i and hence $X \vdash^* Y$ by repeated application of the weakening rule. Conversely, define for any compact element a and any set T of clauses: $T/a = \{Z \in T \mid a \notin Z\} \cup \{\{b\} \cup (Z \setminus \{a\}) \mid b \in \mathsf{A}(a), a \in Z \in T\}$. So for any clause Z and $a \in Z$ we have $\{Z\}/a = Z/a$ and we obtain that $T/a \models T$ for all sets of clauses T and $a \in \mathsf{K}(D)$. Now let $X = \{a_1, \ldots, a_n\}$. Then $(\ldots(X/a_1)/a_2 \ldots)/a_n = \mathsf{A}(X)$ and consequently $\mathsf{A}(X) \models X$, which completes the proof.

In view of Theorem 4.2, it suffices to study $T \vdash^* X$ for theories T and atomic clauses X. We can actually obtain a stronger result, as follows, which provides some kind of normal forms of derivations. For a theory T, define $\mathsf{A}(T) = \{\mathsf{A}(X) \mid X \in T\}$.

THEOREM 4.3. *Let D be an atomic domain, T be a theory, X be a clause and*

$$T \vdash T_1 \vdash \cdots \vdash T_N \vdash X$$

be a derivation in RAD. Then there exists a derivation

$$\mathsf{A}(T) \vdash^* \mathsf{A}(T_1) \vdash^* \cdots \vdash^* \mathsf{A}(T_N) \vdash^* \mathsf{A}(X)$$

using only the atomic extension rule

$$\frac{X; \quad y \in \mathsf{A}(D)}{\{y\} \cup X} \quad (\mathsf{axt})$$

and the multiple atomic shift rule (mas)*, as follows.*

$$\frac{a_i \in X_i; \quad \operatorname{mub}\{a_i \mid i \leq n\} = \{x_j \mid j \leq m\}; \quad b_i \in \mathsf{A}(x_i)}{\{b_1, \ldots, b_m\} \cup \bigcup_{i \leq n}(X_i \setminus \{a_i\})}$$

Furthermore, all clauses occuring in the derivation are atomic.

P r o o f . Let X_1, X_2, X be clauses. We distinguish three cases, from which the assertion follows easily by induction on N.

1. $X_1 \vdash X$ using the reduction rule. First note that the following *atomic shift rule* (ash) is a special instance of the multiple atomic shift rule.

$$\frac{a_1 \in X_1 \quad a_2 \in X_2; \quad a \in \mathsf{A}(x) \text{ for all } x \in \mathrm{mub}\{a_1, a_2\}}{\{a\} \cup (X_1 \setminus \{a_1\}) \cup (X_2 \setminus \{a_2\})}$$

Indeed, (ash) follows from (mas) with $n = 2$ and $a = b_1 = \ldots = b_k$. Now let $a, y \in X_1$ with $y \sqsubseteq a$ and $X = X_1 \setminus \{a\} = \{y, x_1, \ldots, x_n\}$. Let $A \in \mathsf{A}(X)$, say $A = \{y', x'_1, \ldots, x'_n\}$ with $y' \in \mathsf{A}(y)$ and $x'_i \in \mathsf{A}(x_i)$ for all i. Without loss of generality we can assume that $\mathsf{A}(y) \subset \mathsf{A}(a)$, so there is $\{a'\} \cup A \in \mathsf{A}(X_1)$ for some $a' \in \mathsf{A}(a) \setminus \mathsf{A}(y)$. So we now have $a', y' \sqsubseteq a$ and $y' \sqsubseteq y$, *ie* $\{y', a', x'_1, \ldots, x'_n\} \in \mathsf{A}(X_1)$ and $\{y', y', x'_1, \ldots, x'_n\} = A \in \mathsf{A}(X_1)$. So $a' \in \{y', a', x'_1, \ldots, x'_n\}$, $y' \in \{y', y', x'_1, \ldots, x'_n\}$ and since $y' \sqsubseteq x$ for all $x \in \mathrm{mub}\{y', a'\}$ we can derive $\{y'\} \cup (\{y', a', x'_1, \ldots, x'_n\} \setminus \{a'\}) \cup (\{y', y', x'_1, \ldots, x'_n\} \setminus \{y'\}) = \{y', x'_1, \ldots, x'_n\} = A$ using the atomic shift rule.

2. $X_1 \vdash X$ using the extension rule, *ie* $X = X_1 \cup \{y\}$ for some y. Let $A \in \mathsf{A}(X)$. Then $A = \{y'\} \cup Y$ for some $y' \in \mathsf{A}(y)$ and $Y \in \mathsf{A}(X_1)$. Using the atomic extension rule we can derive $Y \vdash A$ and therefore $\mathsf{A}(X_1) \vdash A$ using the atomic extension rule only, which suffices.

3. $\{X_1, X_2\} \vdash X$ using the simplified hyperresolution rule. Let $a_1 \in X_1$, $a_2 \in X_2$ and $X = \mathrm{mub}\{a_1, a_2\} \cup (X_1 \setminus \{a_1\}) \cup (X_2 \setminus \{a_2\})$. Furthermore, let $M = \mathrm{mub}\{a_1, a_2\} = \{m_1, \ldots, m_k\}$ and let $A \in \mathsf{A}(X)$, *ie* $A = \{m'_1, \ldots, m'_k\} \cup B_1 \cup B_2$, where $m'_i \in \mathsf{A}(m_i)$ for all i, $B_1 \in \mathsf{A}(X_1 \setminus \{a_1\})$ and $B_2 \in \mathsf{A}(X_2 \setminus \{a_2\})$. Note that for all $a'_1 \in \mathsf{A}(a_1)$ we have that $B_1 \cup \{a'_1\} \in \mathsf{A}(X_1)$ and for all $a'_2 \in \mathsf{A}(a_2)$ we have that $B_2 \cup \{a'_2\} \in \mathsf{A}(X_2)$. Let $\mathsf{A}(a_1) = \{a'_1, \ldots, a'_{k_1}\}$ and $\mathsf{A}(a_2) = \{a'_{k_1+1}, \ldots, a'_{k_1+k_2}\}$. For $i = 1, \ldots, k_1$ let $Y_i = B_1 \cup \{a'_i\} \in \mathsf{A}(X_1)$ and for $i = k_1, \ldots, k_1 + k_2$ let $Y_i = B_2 \cup \{a'_i\} \in \mathsf{A}(X_2)$. Since $a_1 = \bigsqcup \mathsf{A}(a_1)$ and $a_2 = \bigsqcup \mathsf{A}(a_2)$ we have $\mathrm{mub}\,(\mathsf{A}(a_1) \cup \mathsf{A}(a_2)) = \mathrm{mub}\{a_1, a_2\} = \{m_1, \ldots, m_k\} = M$. From the multiple atomic shift rule we obtain (with $i \le k_1 + k_2$ and $j \le k$)

$$\frac{a_i \in Y_i \quad \mathrm{mub}\{a'_1, \ldots, a'_{k_1+k_2}\} = M, \quad m'_j \in \mathsf{A}(m_j)}{\{m'_1, \ldots, m'_k\} \cup \bigcup_{i \le k_1+k_2} (Y_i \setminus \{a_i\})}$$

Since $Y_i \setminus \{a'_i\} \subseteq B_1$ for $i = 1, \ldots, k_1$ and $Y_i \setminus \{a'_2\} \subseteq B_2$ for $i = k_1, \ldots, k_1 + k_2$, we obtain $\{m'_1, \ldots, m'_k\} \cup \bigcup (Y_i \setminus \{a_i\}) \subseteq A$ which suffices by the atomic extension rule.

Note that the atomic extension rule is a special case of the extension rule, and that the multiple atomic shift rule can be obtained as a subsequent application of first the hyperresolution rule (with $Y = \mathrm{mub}\{a_1, \ldots, a_n\}$) and then multiple instances of the reduction rule, hence both rules are sound.

R e m a r k 4.4. We note that Theorem 4.3 does not hold if (mas) is replaced by its binary version (bas), as follows.

$$\frac{a_1 \in X_1, a_2 \in X_2; \mathrm{mub}\{a_1, a_2\} = \{x_1 \mid i \le k\}; b_i \in \mathsf{A}(x_i)}{\{b_1, \ldots, b_k\} \cup (X_1 \setminus \{a_1\}) \cup (X_2 \setminus \{a_2\})}$$

In order to see this, consider three atomic elements a_1, a_2, a_3 which are mutually consistent with supremum $\sup\{a_i, a_j\} = a_{ij}$, but do not have a common upper bound. Then $\{\{a_1\}, \{a_2\}, \{a_3\}\} \models \{\}$, but the empty clause $\{\}$ cannot be derived from the theory $T = \{\{a_1\}, \{a_2\}, \{a_3\}\}$ using (axt) and (bas) alone. Indeed it is easy to show by induction that every clause which is derived from T using applications of (axt) and (bas) always contains one of the elements a_1, a_2 or a_3.

5 DOMAINS WITH NEGATION

We introduce and investigate a notion of negation on domains, motivated by classical negation as in Example 2.2.

DEFINITION 5.1. An atomic domain is called an *atomic domain with negation* if there exists an involutive and Scott-continuous *negation function* $^{-} : D \to D$ with the following properties:

(i) $^{-}$ maps $\mathsf{A}(D)$ onto $\mathsf{A}(D)$.

(ii) For all $p, q \in \mathsf{A}(D)$ we have $p \not\uparrow q$ if and only if $q = \overline{p}$.

(iii) For every finite subset $A \subseteq \mathsf{A}(D)$ such that $p \uparrow q$ for all $p, q \in A$, the supremum $\bigsqcup A$ exists.

$\mathbb{T}^{\mathcal{V}}$ from Example 2.2 is an example of an atomic domain with negation.

PROPOSITION 5.2. *Let D be an atomic domain with negation. Then for all $c \in \mathsf{K}(D)$ we have* $\overline{c} = \bigsqcup\{\overline{a} \mid a \in \mathsf{A}(c)\}$.

P r o o f . Let $c \in \mathsf{K}(D)$. Then $c = \bigsqcup \mathsf{A}(c)$, hence $\mathsf{A}(c)$ is consistent. By (ii) of Definition 5.1, we obtain that every pair of elements from $\{\overline{a} \mid a \in \mathsf{A}(c)\}$ is consistent, and by (iii) the supremum $d = \bigsqcup\{\overline{a} \mid a \in \mathsf{A}(c)\}$ exists. From monotonicity of $^{-}$, we obtain first $d \sqsubseteq \overline{c}$, and then $\overline{d} \sqsubseteq \overline{\overline{c}} = c$. But, again by monotonicity of $^{-}$, we know that $\overline{d}$ is an upper bound of $\mathsf{A}(c)$, hence $c \sqsubseteq \overline{d}$, and consequently $c = \overline{d}$ and $\overline{c} = d = \bigsqcup\{\overline{a} \mid a \in \mathsf{A}(c)\}$ as required.

The following result, an analogon to the resolution theorem mentioned in the introduction, allows one to replace the search for derivations by search for contradiction.

THEOREM 5.3. *Let D be an atomic domain with negation. Let T be a theory and X be an atomic clause. Then $T \models X$ if and only if $T \cup \{\{\overline{a}\} \mid a \in X\} \vdash^* \{\}$.*

P r o o f . Assume $T \models X$.
Then $T \vdash^* X$ and $\{X\} \cup \{\{\overline{a}\} \mid a \in X\} \vdash^* \{\}$ follows easily by repeated application of the resolution rule (r).

Conversely, assume $T \cup \{\{\overline{a}\} \mid a \in X\} \vdash^* \{\}$, *ie* $T \cup \{\{\overline{a}\} \mid a \in X\} \models \{\}$. If $T \models \{\}$ then $T \vdash^* \{\} \vdash^* X$. So assume that $T \not\models \{\}$, *ie*there exists $w \in D$ with $w \models T$. We have to show that $w \models X$ for every such w. Since $w \models T$ but $w \not\models T \cup \{\{\overline{a}\} \mid a \in X\}$, we have that there is $a \in X$ with $\overline{a} \not\sqsubseteq w$. Hence there exists $x \in \mathsf{A}(w)$ with

$x \not\sqsubseteq \overline{a}$. From the hypothesis we obtain $x = a$. Hence $a \sqsubseteq w$ and therefore, by the weakening rule, $w \vdash^* X$, *ie* $w \models X$.

On atomic domains with negation, we can therefore establish the following sound and complete proof principle.

THEOREM 5.4. *Let T be a theory and X a clause. Consider $T' = \mathsf{A}(T)$. For every atomic clause $A \in \mathsf{A}(X)$ attempt to show $T' \cup \{\{\overline{a}\} \mid a \in A\} \vdash^* \{\}$ using* (axt) *and* (mas)*. If this succeeds, then $T \models X$. Conversely, if $T \models X$ then there exists a derivation $T' \cup \{\{\overline{a}\} \mid a \in A\} \vdash^* \{\}$ for each $A \in \mathsf{A}(X)$ using only the above mentioned rules.*

Proof. If $T' \cup \{\{\overline{a}\} \mid a \in A\} \vdash^* \{\}$, then by Theorem 4.3 the derivation can be carried out using only the mentioned rules and we obtain $T' \cup \{\{\overline{a}\} \mid a \in A\} \models \{\}$. By Theorem 5.3 we obtain $T' \models A$, so $T' \models A$ for all $A \in \mathsf{A}(X)$. By Theorem 4.2 this yields $T' \models X$ and finally we obtain $T \models X$ by application of Theorem 4.2, noting that $T' = \mathsf{A}(T) \sim T$.

Conversely, if $T \models X$ then we have $T' \models A$ for all $A \in \mathsf{A}(X)$, again by Theorems 4.2. Theorem 5.3 then yields $T' \cup \{\{\overline{a}\} \mid a \in A\} \vdash^* \{\}$ for all $A \in \mathsf{A}(X)$, and finally from Theorem 4.3 we obtain that this derivation can be done using only the designated rules.

EXAMPLE 5.5. We give an abstract example, again using notation from Example 2.2, which shows that reasoningin atomic domains with negation does not lead directly back to resoning in $\mathbb{T}^{\mathcal{V}}$. Consider the subcpo constituted by the elements $\{\bot, p, q, r, \overline{p}, \overline{q}, \overline{r}, pqr, \overline{pqr}, p\overline{q}, p\overline{r}, q\overline{p}, q\overline{r}, r\overline{p}, r\overline{q}\}$, which is an atomic domain with negation. Then *eg* $\{\{p\}, \{q\}\} \models \{r\}$. Indeed, $\{\{p\}, \{q\}, \{\overline{r}\}\} \vdash \{\}$ by (mas) because $\mathrm{mub}\{p, q, \overline{r}\} = \{\}$.

6 CONCLUSIONS

We have shown that for certain domains logical consequence in the logic RZ can be reduced to search for contradiction, a result which yields a proof mechanism similar to that underlying the resolution principle used in resolution-based logic programming systems. The result should be understood as foundational for establishing logic programming systems on hierarchical knowledge — like *eg*in formal concept analysis — built on a firm domain-theoretic background. Further research is being undertaken to substantiate this.

REFERENCES

[1] ROUNDS, W. C.—ZHANG, G.-Q.: Clausal Logic and Logic Programming in Algebraic Ddomains, Information and Computation **171** No. 2 (2001), 156–182.

[2] ABRAMSKY, S.—JUNG, A.: Domain theory, Handbook of Logic in Computer Science, volume 3 (Samson Abramsky, Dov Gabbay, and Thomas S.E. Maibaum, eds.), Clarendon, Oxford, 1994.

[3] SCOTT, DANA S.: Domains for Denotational Ssemantics, Proceedings of Automata, Languages and Programming, 9th Colloquium, July 1982, Aarhus, Denmark, Lecture Notes in Computer Science (Magens Nielsen and Erik M. Schmidt, eds.), vol. 140, Springer, Berlin, 1982, pp. 577–613.

[4] SMYTH, M. B.: Powerdomains and Predicate Transformers: A Topological View, Proceedings of Automata, Languages and Programming, 10th Colloquium, July 1989, Barcelona, Spain, Lecture Notes in Computer Science (Josep Díaz, ed.), vol. 298, Springer, Berlin,, 1989, pp. 662–675.

[5] ABRAMSKY, S.: Domain Theory in Logical Form, Annals of Pure and Applied Logic **51** (1991), 1–77.

[6] ZHANG, G.-Q.: Logic of Domains, Birkhäuser, Boston, 1991.

[7] HITZLER, P.: Generalized Metrics and Topology in Logic Programming Semantics, PhD Thesis, Department of Mathematics, National University of Ireland, University College Cork, 2001.

[8] HITZLER, P.—SEDA, A.-K.: Generalized Metrics and Uniquely Determined Logic Programs, Theoretical Computer Science **305** No. 1-3 (2003), 187–219.

[9] ZHANG, G.-Q.—ROUNDS, W. C.: Semantics of Logic Programs and Representation of Smyth Powerdomains., Domains and Processes (Klaus Keimel *et al*, eds.), Kluwer, 2001, pp. 151–179.

ROBINSON, J. A.: A Machine-Oriented Logic Based on the Resolution Principle, Journal of the ACM **12** No. 1 (1965), 23–41.

[11] LLOYD, J. W.: Foundations of Logic Programming, Springer, Berlin, 1988.

[12] HITZLER, P.: A Resolution Theorem for Algebraic Domains, Proceedings of the 18th International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 2003 (Georg Gottlob and Toby Walsh, eds.), Morgan Kaufmann Publishers, 2003, pp. 1339–1340.

[13] JOHNSTONE, P. T.: Stone Spaces, Number 3 in Cambridge Studies in Advanced Mathematics, Cambridge University Press, 1982.

[14] PLOTKIN, G.: T^{ω} as a Universal Domain, Journal of Computer and System Sciences **17** (1978), 209–236.

[15] FITTING, M.: A Kripke-Kleene-Semantics for General Logic Programs, The Journal of Logic Programming **2** (1985), 295-312.

[16] HITZLER, P.—WENDT, M.: Formal Concept Analysis and Resolution in Algebraic Domains, Using Conceptual Structures — Contributions to ICCS 2003 (Aldo de Moor and Bernhard Ganter, eds.), Shaker Verlag, Aachen, 2003, pp. 157–170.

[17] ZHANG, G.-Q.: Chu Spaces, Concept Lattices, and Domains, Proceedings of the Nineteenth Conference on the Mathematical Foundations of Programming Semantics, March 2003, Montreal, Canada, volume 83 of Electronic Notes in Theoretical Computer Science, 2003.

[18] COQUAND, T.—ZHANG, G.-Q.: Sequents, Frames, and Completeness, 14th International Workshop on Computer Science Logic, Fischbachau, Germany, August 2000, Lecture Notes in Computer Science, vol. 1862, Springer, 2000, pp. 277–291.

Received 6 October 2003

Pascal Hitzler (PhD, Dipl-Math), born 1971 in Germany, studied Mathematics and Computer Science at the University of Tübingen, completed with distinction in 1998. He then was a PhD student of Dr. Anthony K. Seda at the National University of Ireland, University College Cork, where he finished his dissertation in Mathematics in 2001. Since then, he is a research assistant of Prof. Dr. Steffen Hölldobler at the Artificial Intelligence Institute at the Department of Computer Science of Dresden University. His published papers *eg* in pure mathematics (topology and fixed-point theory), foundations of artificial intelligence (domain-theoretic aspects of knowledge representation and reasoning; logic and connectionism) and in programming language semantics (in particular logic programming). He is also actively involved in running enhancement programmes for skilled high-school students. Homepage: www.wv.inf.tu-dresden.de/∼pascal/

In: Aldo de Moor and Bernhard Ganter (Eds.): Using Conceptual Structures - Contributions to ICCS 2003.
Shaker Verlag, Aachen, ISBN 3-8322-1705-3, pp. 157-170.

Formal Concept Analysis and Resolution in Algebraic Domains

Pascal Hitzler and Matthias Wendt

Artificial Intelligence Institute, Department of Computer Science
Dresden University of Technology, Dresden, Germany
{phitzler,mw177754}@inf.tu-dresden.de

Abstract. We relate two formerly independent areas: Formal concept analysis and logic of domains. We will establish a correspondene between contextual attribute logic on formal contexts resp. concept lattices and a clausal logic on coherent algebraic cpos. We show how to identify the notion of formal concept in the domain theoretic setting. In particular, we show that a special instance of the resolution rule from the domain logic coincides with the concept closure operator from formal concept analysis. The results shed light on the use of contexts and domains for knowledge representation and reasoning purposes.

1 Introduction

Domain theory was introduced in the 1970s by Scott as a foundation for programming semantics. It provides an abstract model of computation using order structures and topology, and has grown into a respected field on the borderline between Mathematics and Computer Science [1]. Relationships between domain theory and logic were noted early on by Scott [2], and subsequently developed by many authors, including Smyth [3], Abramsky [4], and Zhang [5]. There has been much work on the use of domain logics as logics of types and of program correctness, with a focus on functional and imperative languages.

However, there has been only little work relating domain theory to logical aspects of knowledge representation and reasoning in artificial intelligence. Two exceptions were the application of methods from quantitative domain theory to the semantic analysis of logic programming paradigms studied by Hitzler and Seda [6–8], and the work of Rounds and Zhang on the use of domain logics for disjunctive logic programming and default reasoning [9–11]. The latter authors developed a notion of clausal logic in coherent algebraic domains, for convenience henceforth called *logic RZ*, based on considerations concerning the Smyth powerdomain, and extended it to a disjunctive logic programming paradigm [11]. A notion of default negation, in the spirit of answer set programming [12] and Reiter's default logic [13], was also added [14].

The notion of *formal concept* evolved out of the philosophical theory of concepts. Wille [15] proposed the main ideas which lead to the development of formal concept analysis as a mathematical field [16]. The underlying philosophical rationale is that a concept is determined by its *extent*, i.e. the collection of objects which fall under this concept, and its *intent*, i.e. the collection of properties or attributes covered by this concept. Thus, a formal concept is usually distilled out of an incidence relation, called a *formal context*, between a set of objects and a set of attributes via some concept closure operator, see Section 2 for details. The set of all concepts is then a complete lattice under some natural order, called a *concept lattice.*

The concept closure operator can naturally be represented by an implicational theory of attributes, e.g. the attribute "is a dog" would imply the attribute "is a mammal", to give a simple example. Thus, contexts and concepts determine logical structures, which are investigated e.g. in [17–19]. In this paper, we establish a close relationship between the logical consequence relation in the logic RZ and the construction of concepts from contexts via the mentioned concept closure operator. We will show that finite contexts can be mapped naturally to certain partial orders such that the concept closure operator coincides with a special instance of a resolution rule in the logic RZ, and that the concept lattice of the given context arises as a certain set of logically closed theories. Conversely, we will see how the logic RZ on finite pointed posets finds a natural representation as a context. Finally, we will also see how the contextual attribute logic due to Ganter and Wille [17] reappears in our setting.

Due to the natural capabilities of contexts and concepts for knowledge representation, and the studies by Rounds and Zhang on the relevance of the logic RZ for reasoning mentioned above, the result shows the potential of using domain logics for knowledge representation and reasoning. As such, the paper is part of our investigations concerning the use of domain theory in artificial intelligence, where domains shall be used for knowledge representation, and domain logic for reasoning. The contribution of this paper is on the knowledge representation aspect, more precisely on using domains for representing knowledge which is implicit in formal contexts. Aspects of reasoning, building on the clausal logic of Rounds and Zhang and its extensions, as mentioned above, are being pursued and will be presented elsewhere, and some general considerations can be found in the conclusions. We also note that our results may make way for the use of formal concept analysis for domain-theoretic program analysis, and this issue is also to be taken up elsewhere.

The plan of the paper is as follows. In Section 2 we provide preliminaries and notation from lattice theory, formal concept analysis, and domain theory, which will be needed throughout the paper. In Section 3, we will identify certain logically closed theories from the logic RZ, called *singleton-generated theories*, and show that the set of all these coincides with the Dedekind-MacNeille completion of the underlying finite poset. This sets the stage for the central Section 4 where we will present the main results on the correspondence between concept closure and logical consequence in the logic RZ, as mentioned above. In Section 5 we shortly exhibit how the contextual attribute logic relates to our setting. Finally, in Section 6, we conclude with a general discussion on knowledge representation and reasoning perspectives of our work, and display some of the difficulties involved in carrying over the results to the infinite case.

Acknowledgements. We thank Sergej Kuznetsov and Guo-Qiang Zhang for inspiring discussions, and Pawel Waszkiewicz for valuable feedback on an earlier version of this paper. We are very grateful for detailed discussions with Bernhard Ganter about our work and for the comments of three anonymous referees, which helped to improve our presentation substantially.

2 Preliminaries and Notation

Our general reference for lattice theory is [20], assuming basic knowledge about partially ordered sets (posets) and (complete) lattices. For a poset P, $\uparrow X = \{y \mid x \leq y \text{ for some } x \in X\}$ denotes the *upper closure* of X, $X^{\uparrow} = \{y \mid x \leq y \text{ for all } x \in X\}$ denotes the (set of) *upper bounds* of X. The set of *minimal upper bounds* of a set X is denoted by $\operatorname{mub} X$. The notions of *lower closure* and *lower bounds* are obtained dually.

We furthermore assume basic terminology from formal concept analysis, such as formal context, the derivation operators, and concept lattices, the standard reference being [16]. For reference, we recall a part of the basic theorem of concept lattices [16, Theorem 3], which we will use in the sequel.

Theorem 1. *Given a formal context (G, M, I), a complete lattice L is isomorphic to the concept lattice $\underline{\mathfrak{B}}(G, M, I)$ if and only if there are mappings $\overline{\gamma} : G \to L$ and $\overline{\mu} : M \to L$ such that $\overline{\gamma}(G)$ is join-dense and $\overline{\mu}(M)$ is meet-dense in L and gIm is equivalent to $\overline{\gamma}(g) \leq \overline{\mu}(m)$.*

Given a poset P, the smallest complete lattice into which P can be embedded is called the *Dedekind-MacNeille completion* of P, which can in turn be identified with the concept lattice of the formal context $(P, P, \leq)$.

We will now recall in more detail some basic notions from domain theory and the ideas underlying domain logics, good references being [1, 4, 5].

We call a subset $X \subseteq P$ of a poset P *directed* if for all $x, y \in X$ there exists $z \in X$ with $x \leq z$ and $y \leq z$. A poset P is called *pointed* if it has a least element $\bot$, and is called a *cpo* (*complete partial order*) if it is pointed and limits of all directed subsets exist. An element $c \in P$ is called *compact* (or finite) if for all directed $D \subseteq P$ with $c \leq \bigvee D$, there exists $x \in D$ with $c \leq x$. The set of all compact elements of P is denoted by $K(P)$. A cpo is called *algebraic* if every element is the directed join of the compact elements below it. A subset $U \subseteq P$ of a cpo is called *Scott open* if $\uparrow U = U$ and for any directed $D \subseteq P$ we have $\bigvee D \in U$ if and only if $U \cap D \neq \emptyset$. These sets form the so-called *Scott topology.* A cpo is called *coherent* if the intersection of any two compact open sets is compact open. A set $U \subseteq D$ is called *saturated* if it is the intersection of all Scott opens containing it. In the following, we will call coherent algebraic cpos *domains.*

In [11], Rounds and Zhang developed a clausal logic on domains, in the following called *logic RZ*, which bears potential for establishing a disjunctive logic programming paradigm with a clear domain-theoretic semantics. The next definition shortly recaptures their work. Our discussion, however, will mostly be restricted to the finite case of finite pointed posets. A short discussion of this is deferred to Section 6.

Definition 1. *Let $(P, \leq)$ be a domain. A* clause *over P is a finite subset of $K(P)$. A* theory *over P is a set of clauses over P. For a clause X and $m \in P$, we say that m is a* model *of X, written $m \models X$, if there exists some $x \in X$ with $x \leq m$. For a theory T and $m \in P$, we set $m \models T$ if $m \models X$ for all $X \in T$, in which case we call m a* model *of T. For a theory T and a clause X we say that X is a* logical consequence *of T, written $T \models X$, if for all $m \in P$ we have that $m \models T$ implies $m \models X$. A theory T is said to be* logically closed *if $T \models X$ implies $X \in T$ for all clauses X. Given a theory S, we say that T is the* logical closure *of S if T is the smallest logically closed theory containing S. A theory is called* consistent *if it does not have the empty clause as a logical consequence.*

For a theory T, we will denote the set of models of T by $\mathrm{Mod}(T)$. Similarly, given a set $M \subseteq P$ of models, we define the corresponding theory $\mathrm{Th}(M)$ to be the set of all clauses which have all elements of M as model. Note that for every $M \subseteq P$ the corresponding theory $\mathrm{Th}(M)$ contains the clause $\{\bot\}$, hence is non-empty. The original rationale for studying the logic RZ was to obtain a characterization of the Smyth powerdomain of coherent algebraic cpos by means of a domain logic. The Smyth powerdomain is used in denotational semantics for modelling nondeterminism, and it can be characterized as the set of all nonemtpy compact saturated subsets of P, ordered by reverse subset inclusion. For details we refer to [1]. The mentioned characterization from [11] now is that the Smyth powerdomain of P is isomorphic to the set of all consistent and logically closed theories over P, under subset inclusion.

3 Singleton-Generated Theories and Poset Completion

In this section, we will show a strong relationship between the logic RZ and poset completion. In particular we show that a set of certain theories is isomorphic to the Dedekind-MacNeille completion of the given poset. Due to the strong link between concept lattices and the Dedekind-MacNeille completion exhibited by Theorem 1, this will provide the necessary tool for our main results, presented in Section 4.

First of all, we describe the domain-theoretic Smyth powerdomain construction by means of formal concept analysis. Note however, that although the Smyth powerdomain of a coherent algebraic cpo is always a lattice (just missing a top element), it is in general not a complete lattice. Thus the assumption of P being a finite pointed poset is crucial to our formal concept analysis reformulation of this powerdomain construction. Now in the finite case, we can provide two representations of the Smyth powerdomain of a pointed poset P: The first one uses the definition of the Smyth powerdomain as the set of all compact saturated subsets of P ordered by reverse inclusion. In the finite case, a set is compact saturated if and only if it is upward closed. This yields that the Smyth powerdomain of a finite pointed poset is isomorphic to $\mathfrak{B}(P, P, \not\geq)$, having as intents the order filters, ordered by reverse inclusion. The second representation uses the logical characterization from [11] mentioned above, according to which the

Smyth powerdomain of a given domain P is isomorphic to the set of all logically closed consistent theories over P, under subset inclusion. Since clauses in the finite case are just subsets of P, the Smyth powerdomain is isomorphic to $\underline{\mathfrak{B}}(\mathfrak{P}(P), P, =\!\mid)$, where $\mathfrak{P}(P)$ denotes the powerset of P. Corresponding concepts are of the form (A, B), where A is a logically closed theory and B is its set of models, i.e., $A = \mathrm{Th}(B)$ and $B = \mathrm{Mod}(A)$.

We will now investigate conjunctive assertions in the logic RZ. According to the intuition that theories are conjunctions of their clauses, we have to consider theories which contain only singleton clauses. Theories which are the closure of a set containing only singleton clauses will be called *singleton-generated theories*, and a set $\mathcal{G} \subseteq P$ will be called a *generator* of the theory $\mathrm{Th}(\{\{d\} \mid d \in \mathcal{G}\})$. These definitions formalize the idea of considering only inferences of the form $\{\{d_1\}, \ldots, \{d_n\}\} \models \{d\}$, which was also done in [21].

These singleton-generated theories can be obtained by restricting the objects of the Smyth powerdomain context $(\mathfrak{P}(P), P, =\!\mid)$ to singleton clauses. Thus, the singleton-generated theories are obtained as the extents of the formal context $(P, P, =\!\mid)$. Noting that for a singleton clause $\{d\}$ and a model m we have $\{d\} =\!\mid m$ if and only if $d \leq m$, the context for the singleton-generated theories can be written as $(P, P, \leq)$. We have just shown the following.

Theorem 2. *Given a finite pointed poset P, the set of all singleton-generated theories over P, ordered by subset inclusion, is isomorphic to the Dedekind-MacNeille completion $\mathcal{N}(P) \cong \underline{\mathfrak{B}}(P, P, \leq)$ of P.*

Before we make use of Theorem 2 in the next section, let us briefly reflect on what we have achieved so far. Identifying singleton-generated theories with elements of the Dedekind-MacNeille completion yields the possibility of representing finite lattices — which are always complete — by means of finite pointed posets. From an order-theoretic point of view this idea appears to be rather straightforward. Relating this setting to a logic of domains, however, provides a novel aspect. On the one hand, we now have the possibility to use a restricted form of resolution on ordered sets — as will be explained in Section 4 — in order to represent elements of the corresponding Dedekind-MacNeille completion. On the other hand, we obtain a new perspective on the logic RZ, namely that underlying posets can be interpreted from a knowledge representation point of view. More precisely, in the next section we will show how Theorem 2 can be employed for relating the logic RZ to formal concept analysis. The following corollary to Theorem 2 will also be helpful as it provides a logical representation of finite lattices by theories on finite pointed posets.

Corollary 1. *Let L be a finite lattice. Then for every finite pointed poset P which can be embedded join- and meet-densely into L, the set of all singleton-generated theories over P is isomorphic to L.*

We know that every complete lattice is the concept lattice of some formal context. We can thus interpret the elements of the Dedekind-MacNeille completion of a pointed poset P, which can in turn be identified with singleton-generated theories over P, as concepts in the corresponding concept lattice. This indicates that the logic RZ might be used as a knowledge representation formalism. This, and details of the relationship between the logic RZ on finite pointed posets and concept lattices will be explained in more detail in the next section.

4 Representation of Formal Contexts by Finite Posets

In the previous section we have shown that each finite pointed poset P can be interpreted as a formal context whose concept lattice $\underline{\mathfrak{B}}(P, P, \leq)$ is isomorphic to the set of all singleton-generated theories over P. In this formal context, each element of the poset is both object and attribute, resembling the fact that in the domain logic of a finite pointed poset, each element can be both a singleton clause and a model.

Since we consider finite lattices, [16, Proposition 12] implies that there is a unique reduced context, the standard context of $(P, P, \leq)$, having a concept lattice isomorphic to $\underline{\mathfrak{B}}(P, P, \leq)$. The objects of the standard context are those elements of P whose object concept is join-irreducible in $\underline{\mathfrak{B}}(P, P, \leq)$, the attributes are those whose attribute concept is meet-irreducible. Considering the principal ideal embedding of P into $\underline{\mathfrak{B}}(P, P, \leq)$, we find that the irreducible objects of $(P, P, \leq)$ are those which are not the join of all elements strictly below them, whereas the irreducible attributes are those which are not the meet of all elements strictly above them.

From these considerations it follows immediately that any singleton-generated theory over P is completely determined by the set of irreducible objects it contains as singletons: any singleton which is not an irreducible object can be represented as the join of all the irreducible objects below it, hence is derivable from these objects in the logic RZ. Formally, we can state the following lemma.

Lemma 1. *Let P be a finite pointed poset and T_1, T_2 be two singleton-generated theories over P which coincide on all irreducible objects of P. Then $T_1 = T_2$.*

Proof. Assume $\{x\} \in T_1$ for some reducible object $x \in P$. Now x is the join of all the irreducible objects below it, and by logical closure $\{m\} \in T_1$ for all elements $m \leq x$, hence $\{m\} \in T_2$ for all irreducible objects $m \leq x$. So by logical closure of T_2 we obtain $\{x\} \in T_2$. The argument clearly reverses and therefore suffices.

At this stage our way of interpreting singleton-generated theories as concepts becomes almost obvious. An object $g \in P$ is in the extent of a concept (i.e. of a singleton-generated theory) T if $\{g\} \in T$, and an attribute $m \in P$ is in the intent of a concept T if $m \models T$. The latter means that every object contained as singleton in T necessarily has attribute m. Furthermore, if an object g is a model for a theory, then it is necessary for any other object in the corresponding concept extent to have all the attributes g has. If an attribute is contained as a singleton in a theory, then any object having this attribute is also contained as a singleton in the theory.

When reasoning about the knowledge represented in a poset P, we can — according to Lemma 1 — restrict our attention to the irreducible objects. But we can also incorporate the attributes into the reasoning, if desired, as a kind of *macros* for describing collections of objects. This perspective will be employed later on when discussing the logic programming framework developed by Rounds and Zhang [11] in terms of formal concept analysis. Reasoning with objects probably seems unusual from the point of view of formal concept analysis, since it is more common to consider the logic behind the attributes, focusing on the implicational theory of the attributes. The emphasis on objects in this paragraph stems from the domain theoretic intuition on which the logic RZ is based, namely that $x \leq y$ stands for the situation in which y contains more information than x. When representing concept lattices by singleton generated theories in the logic RZ, it is therefore more intuitive to consider the concept lattice in its reverse order, or equivalently, the dual context in which the attributes are considered to be the new objects, with which we reason. We will do this in the following.

The next theorem makes explicit the representation of finite formal contexts by finite pointed posets, such that the concept lattice of the formal context is isomorphic (in the reverse order) to the set of all singleton generated theories of the finite pointed poset.

Theorem 3. *Let (G, M, I) be a reduced finite formal context. Then $\underline{\mathfrak{B}}(G, M, I)$, under the reverse order, is isomorphic to the set of all singleton-generated theories of a finite pointed poset P, under subset inclusion, if and only if there exist bijections $\gamma : G \to M(P)$ and $\mu : M \to J(P)$, where $(J(P), M(P), \leq)$ is the reduced context of $(P, P, \leq)$, such that for all $g \in G$ and $m \in M$ we have gIm if and only if $\gamma(g) \geq \mu(m)$.*

Proof. Immediate from [16, Proposition 12], the Basic Theorem of Concept Lattices (Theorem 1) and Theorem 2.

So the singleton-generated theories in the logic RZ have the same implicational logic as the attributes of the represented context (G, M, I), whenever we restrict to the irreducible objects.

Next, we will give a specific example for a construction of a finite pointed poset from a given context (G, M, I). Note that also $\underline{\mathfrak{B}}(G, M, I)$, reversely ordered, is a finite pointed poset trivially satisfying the conditions from Theorem 3.

Example 1. Let (G, M, I) be a formal context, where G and M are finite and disjoint. Define the following ordering on $G \mathbin{\dot\cup} M$:

(i) For $m_1, m_2 \in M$ let $m_1 \leq m_2$ if $m_1' \supseteq m_2'$.
(ii) For $g_1, g_2 \in G$ let $g_1 \leq g_2$ if $g_1' \subseteq g_2'$.
(iii) For $g \in G$ and $m \in M$ let $m \leq g$ if gIm.
(iv) For $g \in G$ and $m \in M$ let $g \leq m$ if for all $h \in G$ and all $n \in M$ we have that gIn and hIm imply hIn.

The above construction yields a preorder on $G \mathbin{\dot\cup} M$. We obtain from this a partial order, also denoted by $\leq$, by taking the quotient order in the usual way.

Table 1. Formal context for Example 2.

	salad	starter	fish	meat	red wine	white wine	water	dessert	coffee	expensive
1			×			×		×		
2				×	×			×		
3	×		×			×		×	×	×
4		×		×	×			×	×	×
5	×	×	×				×			
6	×	×		×			×		×	
7	×	×		×	×		×	×	×	×
8				×			×		×	
9	×	×						×		

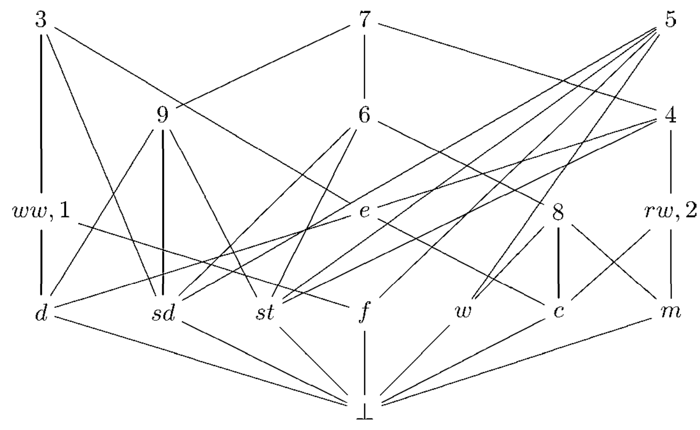
If $(G \dot{\cup} M/_{\sim}, \leq)$ does not have a least element, we add $\bot$ to $G \dot{\cup} M/_{\sim}$ and set $\bot \leq x$ for all $x \in G \dot{\cup} M/_{\sim}$. The latter amounts to adding an additional attribute m with $m' = G$ to the context.

The main intuition behind this construction is to use the set consisting of all objects and attributes as a join- and meet-dense subset of the concept lattice and to supply the induced order by constructions directly available from the formal context. The first three items do exactly this. However, we have to take care of those elements which are both join- and meet-irreducible in the concept lattice. This is achieved with (iv) and the quotient order construction, where those object-attribute pairs are identified which will result in doubly irreducible elements. This construction of endowing $G \dot{\cup} M/_{\sim}$ with the induced order from $\underline{\mathfrak{B}}(G, M, I)$ is also known as the Galois subhierarchy introduced in [22], see also [23] and the references contained therein. Formally, we can state the following.

Proposition 1. *Given a finite formal context (G, M, I) and the poset $P = (G \dot{\cup} M/_{\sim}, \leq)$ as defined in Example 1, we have $\underline{\mathfrak{B}}(G, M, I) \cong \underline{\mathfrak{B}}(P, P, \leq)$.*

Example 2. Consider the formal context given in Table 1. It shall represent, in simplified form, a selection of set dinners from a restaurant menu. Using Example 1, we obtain the finite pointed poset depicted in Figure 1. Concepts in this setting correspond to types of dinners, e.g. one may want to identify the concept with extent $\{4, 6, 7\}$ and intent $\{st, m, c\}$, using the abbreviations from Figure 1, to be the *heavy* meals, while the *expensive* ones are represented by the attribute concept of e, and turn out to always include coffee. Using the logic RZ, we can for example conclude that a customer who wants salad and fish will choose one of the meals 3 or 5, since these elements of the poset are exactly those which are both objects and models of the theory $\{\{sd\}, \{f\}\}$. Also, he will always get a starter or a dessert, formally $\{\{sd\}, \{f\}\} \models \{st, d\}$. To give a slightly more sophisticated example, suppose that a customer wants salad or a starter, additionally fish or a dessert, and drinks water. From this we can conclude that in any case he will get both a salad and a starter. Formally, we

Fig. 1. Figure for Example 2. Abbreviations are: *sd* salad, *st* starter, *f* fish, *m* meat, *rw* red wine, *ww* white wine, *w* water, *d* dessert, *c* coffee, *e* expensive.



obtain $\{\{sd, st\}, \{f, d\}, \{w\}\} \models \{sd\}$ and $\{\{sd, st\}, \{f, d\}, \{w\}\} \models \{st\}$. A little bit of reflection on the context makes it clear that these inferences are indeed natural ones, in fact also follow from the implicational theory of the context.

Having established Theorem 3 as a link between the logic RZ and formal concept analysis, we will now discuss how the different techniques on both sides embed. In particular, we will shortly consider a proof theory for the logic RZ, discussed next, and also the contextual attribute logic of formal concept analysis, discussed in Section 5.

In [11], the following hyperresolution rule was presented:

$$\frac{X_1\ X_2 \dots X_n; \quad a_i \in X_i \quad \text{for } 1 \le i \le n; \quad \text{mub}\{a_1, \dots, a_n\} \models Y}{Y \cup \bigcup_{1 \le i \le n}(X_i \setminus \{a_i\})}$$

In words, this rule says that from clauses $X_1, \dots, X_n$, $a_i \in X_i$ for all i, and $\text{mub}\{a_1, \dots, a_n\} \models Y$ with respect to the logic RZ, the clause $Y \cup \bigcup_{1 \le i \le n}(X_i \setminus \{a_i\})$ may be derived. This rule, together with two special rules treating the cases of an empty selection of clauses resp. an empty clause in the premise of the rule, yields a proof theory resp. entailment relation $\vdash$ which is sound and complete w.r.t. the model theory given in Definition 1. From our results, in particular from Theorem 3, we obtain that the following restriction of the hyperresolution rule to singleton clauses induces an entailment relation $\vdash_s$ which is equivalent to the concept closure operator $(\cdot)'' : \mathfrak{P}(G) \to \mathfrak{P}(G)$ which maps any set of objects B to the extent B'' of the corresponding concept (B'', B'):

$$\frac{\{a_1\}\;\{a_2\}\ldots\{a_n\};\quad \mathrm{mub}\{a_1,\ldots,a_n\}\models\{a\}}{\{a\}}$$

Thus we can conclude that the logic RZ can be used for knowledge representation in much the same way as formal concept analysis: There is a correspondence between finite formal contexts and finite pointed posets and, moreover, both the proof and the model theory of [11] lend themselves to an easy characterization of concept closure. This is probably not too surprising from the viewpoint of formal concept analysis resp. lattice theory. However, from the viewpoint of domain theory it is certainly interesting that there is such a close correspondence between domain logics developed for reasoning about program semantics [4] and a knowledge representation mechanism like formal concept analysis.

5 Contextual Attribute Logic and the Logic RZ

In this section, we will show that the correspondence between the logic RZ and formal concept analysis is not exhausted by the relationship between singleton-generated theories and concept closure. In particular, we will show how to identify part of the contextual attribute logic due to [17] in a finite pointed poset P by means of the logic RZ. We first show how clauses and theories resemble constructions of compound attributes in the poset.

In [17], compound attributes are defined to be compositions of attributes w.r.t. their extent. More precisely, for any set $A \subseteq M$ of attributes of a formal context (G, M, I), the compound attribute $\bigvee A$ has the extent $\bigcup\{m' \mid m \in A\}$, and the compound attribute $\bigwedge A$ has the extent $\bigcap\{m' \mid m \in A\}$. For an attribute $m \in M$, the compound attribute $\neg m$ has the extent $G \setminus m'$.

Now we can relate compound attributes and theories in the logic RZ by the following proposition, which is in fact a straightforward consequence of our previous results, so we skip the proof.

Proposition 2. *Let P be a finite pointed poset and consider the formal context (G, M, I) obtained from it as indicated in Theorem 3, and let γ, μ be as in the same theorem. Then for all $A \subseteq M$, $g \in G$, and $m \in M$ the following hold.*

- *g is in the extent of $\bigvee A$ if and only if $\gamma(g) \models \mu(A)$.*
- *g is in the extent of $\bigwedge A$ if and only if $\gamma(g) \models \{\{\mu(a)\} \mid a \in A\}$.*
- *g is in the extent of $\neg m$ if and only if $\gamma(g) \not\models \{\mu(m)\}$.*

We thus see, that the formation of conjunction and disjunction of attributes to compound attributes corresponds exactly to the formation of singleton generated theories resp. clauses. Negation, however, is more difficult to represent in the logic RZ, since the set of all models of $\neg m$ is not an upper set, but a lower set, more precisely it is the complement of a principal filter in P. Thus it seems that the Scott topology, on which the logic RZ is implicitly based, is not appropriate for handling this kind of negation — which could be a candiate for a strong negation in the logic programming paradigm discussed in Section 6. It remains to be investigated whether the results presented in [11] carry over to the Lawson topology and the Plotkin powerdomain, see [1] for definitions, which according to what has been said above may be the correct setting for handling this negation.

In [17], sequents of the form (A, S), where $A, S \subseteq M$, were introduced as a possible reading of compound attributes $\bigvee(S \cup \{\neg m \mid m \in A\})$. A sequent (A, S) may thus be interpreted as an implication $\bigvee S \leftarrow \bigwedge A$. A *clause set* over M is a set of sequents over M. The clause logic, called *contextual attribute logic*, of a finite pointed poset P is then the set of all sequents that are *all-extensional* in P, i.e. all sequents whose extent contains the set of all objects of P. This means that the implication represented by the sequent holds for all the objects in P.

Due to the difficulties with negation discussed above we restrict our attention, for the time being, to all-extensional sequents (A, S) with $A = \emptyset$. So consider again the setting of Proposition 2, and let $X \subseteq M$. Then $(\emptyset, X)$ is an all-extensional sequent if and only if $\gamma(x) \models \mu(X)$ for all $x \in G$. This is easily verified using Theorem 3 and Proposition 2.

Apart from investigating compound attributes involving negation — as discussed above — it also remains to be determined whether there exists a way of identifying the contextual attribute logic by means of the proof theory defined by Rounds and Zhang [11]. This will be subject to further research.

6 Conclusions and Further Work

We have displayed a strong relationship between formal concept analysis and the domain logic RZ. The restriction of inference to singleton clauses yields the concept closure operator of formal concept analysis. Furthermore, any logically closed theory in the logic RZ can be understood as a clause set over a formal context, in the sense of contextual attribute logic, and the hyperresolution rule of [11] can be used to reason about such knowledge present in a given formal context in much the same way as the resolution rule proposed in [17]. This of course can be a foundation for logic programming over formal contexts, i.e. logic programming with background knowledge which is taken from a formal context and used as "hard constraints".

The appropriate way of doing this on domains was also studied by Rounds and Zhang. In their logic programming paradigm on coherent algebraic cpos a logic program is a set of rules of the form $\theta \leftarrow \tau$, where θ and τ are clauses over the respective domain. The models of such a rule $\theta \leftarrow \tau$ are exactly those elements w of the domain which satisfy $w \models \theta$ whenever $w \models \tau$. The rule

$$\frac{X_1\, X_2 \ldots X_n; \quad a_i \in X_i \quad \text{for } 1 \le i \le n; \quad \theta \leftarrow \tau \in P; \quad \text{mub}\{a_1, \ldots, a_n\} \models \tau}{\theta \cup \bigcup_{1 \le i \le n}(X_i \setminus \{a_i\})}$$

corresponds to inference taking the clause $\theta \leftarrow \tau$ into account. By adjoining to the usual proof theory the inference rules for all the clauses in a given program, one can define a monotonic and continuous operator T_P on the set of all logically closed theories, whose least fixed point yields a very satisfactory semantics for the considered program P.

This logic programming paradigm can be understood as logic programming with background knowledge, since the semantics of the program does not only satisfy the program, in a reasonable sense, but also takes into account those implications which are hidden in the underlying domain, i.e. in the context. It is interesting to note that the knowledge implicit in the context need not be made explicit, e.g. by computing the stem base of the context. This implicational knowledge is implicitly represented by the inference rules constituting the proof theory of the logic RZ. The authors are currently investigating the potential of this approach.

Example 3. Consider again the setting from Example 2 and suppose that a customer's wishes can be expressed by the following rules.

$$\begin{aligned} \{rw\} &\leftarrow \{m\} \\ \{c, ww\} &\leftarrow \{d\} \\ \{sd, st\} &\leftarrow \{\bot\} \end{aligned}$$

We understand, that the customer does not want meat without red wine, that for him a dessert should always go with white wine or coffee, and that in any case he wants a starter or a salad. The models for this program are 3, 4, 5, 7, sd, and st, which do not constitute an upper set. Possible choices by the customer are again those models which are also objects, i.e. 3, 4, 5, and 7. Drawing inferences from this program alone, however, yields counterintuitive results, e.g. because every clause which is a logical consequence must contain both st and sd (or the bottom element). Consequently, $\{e, f\}$ is not a logical consequence although each possible choice of meal by the customer will include fish or be expensive. This situation is caused by the fact that objects and attributes are no longer distinguished in the domain-theoretic setting, and that the information that salad or starter are part of every meal which is satisfactory for the customer, is necessarily part of every inference drawn. We can rectify this by adding another rule

$$\{1, 2, 3, 4, 5, 6, 7, 8, 9\} \leftarrow \{\bot\}$$

to the program, which can be interpreted as saying that the customer also wants a meal (obviously). The logical consequences from this program are then as can be expected from the context, e.g. $\{e, f\}$ as discussed above. So evaluating the set of rules amounts to querying the background knowledge represented by the context from Table 1. We suspect a strong relationship to inferences from the contextual attribute logic underlying the context, but details remain to be worked out.

In this paper, we have restricted our considerations to the case of finite pointed posets. So let us shortly discuss some of the difficulties involved in carrying over our results to the case of arbitrary coherent algebraic cpos. The correspondence between singleton-generated theories and Dedekind cuts underlying

Theorem 2 carries over to the infinite case without major restrictions — one just has to correctly adjust it to compact elements and to keep in mind that any non-compact element can be represented as the supremum of all compact elements below it. Difficulties occur when trying to characterize the lattices which arise as Dedekind-MacNeille completions of coherent algebraic cpos, since on the domain-theoretic side one has to deal with the topological notion of coherence, which is not really present on the lattice-theoretic side. Furthermore, the Scott topology we are implicitly dealing with when working with the logic RZ is not completion invariant, which means that the properties defined in terms of the Scott topology, e.g. continuity of the poset, do not carry over to the completion [24]. These issues will also have to be subject to further research. A construction similar to Example 1 carries over to a restricted infinite case, and details can be found in [21].

We finally note the very recent paper by Zhang [25], which also studies relationships between domain theory and formal concept analysis, though from a very different perspective involving Chu spaces.

References

1. Abramsky, S., Jung, A.: Domain theory. In Abramsky, S., Gabbay, D., Maibaum, T.S., eds.: Handbook of Logic in Computer Science. Volume 3. Clarendon, Oxford (1994)
2. Scott, D.S.: Domains for denotational semantics. In Nielsen, M., Schmidt, E.M., eds.: Automata, Languages and Programming, 9th Colloquium, July 1982, Aarhus, Denmark, Proceedings. Volume 140 of Lecture Notes in Computer Science., Springer, Berlin (1982) 577–613
3. Smyth, M.B.: Powerdomains and predicate transformers: A topological view. In Díaz, J., ed.: Automata, Languages and Programming, 10th Colloquium, July 1989, Barcelona, Spain, Proceedings. Volume 298 of Lecture Notes in Computer Science., Springer, Berlin (1989) 662–675
4. Abramsky, S.: Domain theory in logical form. Annals of Pure and Applied Logic **51** (1991) 1–77
5. Zhang, G.Q.: Logic of Domains. Birkhauser, Boston (1991)
6. Seda, A.K., Hitzler, P.: Topology and iterates in computational logic. In: Proceedings of the 12th Summer Conference on Topology and its Applications: Special Session on Topology in Computer Science, Ontario, August 1997. Volume 22 of Topology Proceedings. (1997) 427–469
7. Hitzler, P.: Generalized Metrics and Topology in Logic Programming Semantics. PhD thesis, Department of Mathematics, National University of Ireland, University College Cork (2001)
8. Hitzler, P., Seda, A.K.: Generalized metrics and uniquely determined logic programs. Theoretical Computer Science (200x) To appear.
9. Zhang, G.Q., Rounds, W.C.: Reasoning with power defaults (preliminary report). In Dix, J., Furbach, U., Nerode, A., eds.: Proceedings of the Fourth International Conference on Logic Programming and Non-Monotonic Reasoning, LPNMR'97, Dagstuhl, Germany. Volume 1265 of Lecture Notes in Computer Science., Springer (1997) 152–169

10. Zhang, G.Q., Rounds, W.C.: Semantics of logic programs and representation of Smyth powerdomains. In Keimel, K., et al., eds.: Domains and Processes. Kluwer (2001) 151–179
11. Rounds, W.C., Zhang, G.Q.: Clausal logic and logic programming in algebraic domains. Information and Computation **171** (2001) 156–182
12. Marek, V.W., Truszczyński, M.: Stable models and an alternative logic programming paradigm. In Apt, K.R., Marek, V.W., Truszczyński, M., Warren, D.S., eds.: The Logic Programming Paradigm: A 25-Year Persepective. Springer, Berlin (1999) 375–398
13. Reiter, R.: A logic for default reasoning. Artificial Intelligence **13** (1980) 81–132
14. Hitzler, P.: Towards nonmonotonic reasoning on hierarchical knowledge. In: Proceedings of the 17th Workshop Logische Programmierung, WLP02, December 2002, Dresden, Germany. (200x) To appear.
15. Wille, R.: Restructuring lattice theory: An approach based on hierarchies of concepts. In Rival, I., ed.: Ordered Sets. Reidel, Dordrecht-Boston (1982) 445–470
16. Ganter, B., Wille, R.: Formal Concept Analysis — Mathematical Foundations. Springer, Berlin (1999)
17. Ganter, B., Wille, R.: Contextual attribute logic. In Tepfenhart, W.M., Cyre, W.R., eds.: Conceptual Structures: Standards and Practices. Proceedings of the 7th International Conference on Conceptual Structures, ICCS '99, July 1999, Blacksburgh, Virginia, USA. Volume 1640 of Lecture Notes in Artificial Intelligence., Springer, Berlin (1999) 377–388
18. Ganter, B.: Attribute exploration with background knowledge. Theoretical Computer Science **217** (1999) 215–233
19. Wille, R.: Boolean judgement logic. In Delugach, H., Stumme, G., eds.: Conceptual Structures: Broadening the Base, Proceedings of the 9th International Conference on Conceptual Structures, ICCS 2001, July 2001, Stanford, LA, USA. Volume 2120 of Lecture Notes in Artificial Intelligence., Springer, Berlin (2001) 115–128
20. Davey, B., Priestley, H.: Introduction to Lattices and Order. Cambridge University Press (1990)
21. Hitzler, P.: Contexts, concepts and logic of domains. Technical Report WV-02-12, Knowledge Representation and Reasoning Group, Department of Computer Science, Dresden University of Technology, Dresden, Germany (2002)
22. Godin, R., Mineau, G., Missaoui, R., Mili, H.: Méthodes de classification conceptuelle basées sur les treillis de galois et applications. Revue d'Intelligence Artificielle **9** (1995) 105–137
23. Huchard, M., Roume, C., Valtchev, P.: When concepts point at other concepts: the case of UML diagram reconstruction. In Liquière, M., et al., eds.: Proceedings of the Second International Workshop on Advances in Formal Concept Analysis for Knowledge Discovery in Databases (FCAKDD), Lyon, France, July 2002. (2002) 32–43
24. Erné, M.: A completion-invariant extension of the concept of continuous lattices. In Banaschewski, B., Hoffmann, R.E., eds.: Continuous Lattices: Proceedings of the Conference on Topological and Categorical Aspects of Continuous Lattices. Volume 871 of Lecture Notes in Mathematics., Berlin, Springer (1979) 43–60
25. Zhang, G.Q.: Chu spaces, concept lattices, and domains. In: Proceedings of the Nineteenth Conference on the Mathematical Foundations of Programming Semantics, March 2003, Montreal, Canada. Electronic Notes in Theoretical Computer Science (2003) To appear.