Querying formal contexts with answer set programs*

Pascal Hitzler and Markus Krötzsch

AIFB, University of Karlsruhe, Germany

Abstract. Recent studies showed how a seamless integration of formal concept analysis (FCA), logic of domains, and answer set programming (ASP) can be achieved. Based on these results for combining hierarchical knowledge with classical rule-based formalisms, we introduce an expressive common-sense query language for formal contexts. Although this approach is conceptually based on order-theoretic paradigms, we show how it can be implemented on top of standard ASP systems. Advanced features, such as default negation and disjunctive rules, thus become practically available for processing contextual data.

1 Introduction

At the heart of formal concept analysis (FCA) lies the formation of formal concepts from formal contexts. As such, formal concept analysis is a powerful tool for extracting conceptual hierarchies from raw data. The resulting lattices structure the knowledge hidden in the raw data, i.e. formal contexts, in a way which appeals to human experts, and allows them to navigate the data in a new way in order to understand relationships or create new hypotheses.

Formal concept analysis thus has a commonsense knowledge representation aspect. This also becomes apparent by its multiple uses in the creation of ontologies for the semantic web. It serves as a basic tool for the conceptualization of data, and the conceptual hierarchies obtained from it can often form a modelling base for ontologies in more expressive logical languages.

In this paper, we address the question of querying the conceptual knowledge hidden within a formal context. We present a query language based on commonsense reasoning research in artificial intelligence. It allows to query formal contexts by means of logic programs written over attributes and objects, and features default negation in the sense of the knowledge representation and reasoning systems known as answer set programming (ASP).

Our results will also show how the resulting query system can be implemented on top of standard ASP systems, which allows to utilize the highly optimized systems currently available. Our work also sheds some foundational light on ASP

* The authors acknowledge support by the German Federal Ministry of Education and Research (BMBF) under the SmartWeb project, and by the European Commission under contract IST-2003-506826 SEKT and under the KnowledgeWeb Network of Excellence.

itself, whose theoretical underpinnings, in particular in relation to order-theoretic perspectives, are not yet understood in a satisfactory way.

The paper will be structured as follows. We first review the foundational results from [1] which serve as a base for our contribution. Section 2 introduces the logical formalism of reasoning on hierarchical knowledge that we will build upon, and recalls some basics of formal concept analysis. Section 3 discusses default negation for conceptual hierarchies. Section 4 describes the query language which we introduce. In Section 5 we present the theoretical results which enable the implementation of the querying system on top of the dlv ASP system. We close with the discussion of related and future work in Section 6.

2 Logic of domains and FCA

We need to establish a certain amount of formal terminology in order to be able to motivate our contribution. This will be done in this and the next section. Following [2], we first introduce the logic of domains, and then recall the basics of formal concept analysis.

We assume the reader to be familiar with the basic notions of order theory, and recall only the relevant notions of domain theory. Thus let $(D, \sqsubseteq)$ be a partially ordered set. A subset $X \subseteq D$ is *directed* if, for all x, $y \in X$, there is $z \in X$ with $x \sqsubseteq z$ and $y \sqsubseteq z$. We say that D is a *complete partial order* (*cpo*) if every directed set $X \subseteq D$ has a least upper bound $\bigsqcup X \in D$. Note that we consider the empty set to be directed, and that any cpo thus must have a least element $\bigsqcup \emptyset$ that we denote by $\bot$.

An element $c \in D$ is *compact* if, whenever $c \sqsubseteq \bigsqcup X$ for some directed set X, there exists $x \in X$ with $c \sqsubseteq x$. The set of all compact elements of D is written as $\mathsf{K}(D)$. An *algebraic cpo* is a cpo in which every element $d \in D$ is the least upper bound of the – necessarily directed – set $\{c \sqsubseteq d \mid c \in \mathsf{K}(D)\}$ of compact elements below it.

A set $O \subseteq D$ is *Scott open* if it is upward closed, and *inaccessible by directed suprema*, i.e., for any directed set $X \subseteq D$, we have $\bigsqcup X \in O$ if and only if $O \cap X \neq \emptyset$. The *Scott topology* on D is the collection of all Scott open sets of D, and a Scott open set is *compact*, if it is compact as an element of the Scott topology ordered under subset inclusion. A *coherent algebraic cpo* is an algebraic cpo such that the intersection of any two compact open sets is compact open. 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 $c_1, \ldots, c_n$ are compact elements, then the set $\mathsf{mub}\{c_1, \ldots, c_n\}$ 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 [3].

We can now define the basic notions of domain logic. The following is taken from [2], where further details can be found.

Definition 1. *Let D be a coherent algebraic cpo with set $\mathsf{K}(D)$ of compact elements. A* clause *of D is a finite subset of $\mathsf{K}(D)$. Given a clause X over D, and an element $m \in D$, we write $m \models X$ if there exists $x \in X$ with $x \sqsubseteq m$, i.e. X contains an element below m. In this case, we say that m is a model of X.*

The clausal logic introduced in Definition 1 will henceforth be called the *Logic RZ* for convenience.

Example 1. In [2], the following running example was given. Consider a countably infinite set of propositional variables $\mathcal{V}$, and the set $\mathbb{T} = \{\mathbf{f}, \mathbf{u}, \mathbf{t}\}$ of truth values ordered 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*). The partially ordered set $\mathbb{T}^{\mathcal{V}}$ is a coherent algebraic cpo[1] and has been studied, e.g., in [4] in a domain-theoretic context, and in [5] 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 $(p \wedge q \wedge \neg r) \vee (\neg p \wedge q) \vee r$.

The Logic RZ provides a framework for reasoning with disjunctive information. However, it is also possible to encode conjunctive information: given a finite set X of compact elements of a domain D, the "conjunction" of the elements of D can be expressed by the clause $\mathsf{mub}(D)$. Indeed, whenever an element models all members of X, it is greater or equal than one of the minimal upper bounds of X.

Example 2. Consider the domain $\mathbb{T}^{\mathcal{V}}$ of Example 1. The set of minimal upper bounds of every finite set of compact elements in $\mathbb{T}^{\mathcal{V}}$ is either singleton or empty. For instance, the only minimal (and therefore least) upper bound of $p\overline{r}$ and pq is $pq\overline{r}$.

The Logic RZ enables logical reasoning with respect to a background theory of hierarchical knowledge that is encoded in the structure of the domain. Formal concept analysis (FCA), in contrast, provides techniques for representing data in form of conceptual hierarchies, that allow for simple relational descriptions. We quickly review the basic notions of FCA, and refer to [6] for an in-depth treatment.

A (*formal*) *context* $\mathbb{K}$ is a triple (G, M, I) consisting of a set G of *objects*, a set M of *attributes*, and an *incidence relation* $I \subseteq G \times M$. Without loss of generality, we assume that $G \cap M = \emptyset$. For $g \in G$ and $m \in M$ we write $g\ I\ m$ for $(g, m) \in I$, and say that g *has the attribute* m.

For a set $O \subseteq G$ of objects, we set $O' = \{m \in M \mid g\ I\ m \text{ for all } g \in O\}$, and for a set $A \subseteq M$ of attributes we set $A' = \{g \in G \mid g\ I\ m \text{ for all } m \in A\}$. A (*formal*) *concept* of $\mathbb{K}$ is a pair (O, A) with $O \subseteq G$ and $A \subseteq M$, such that $O' = A$ and $A' = O$. We call O the *extent* and A the *intent* of the concept (O, A).

[1] In fact it is also bounded complete.

The set $\mathcal{B}(\mathbb{K})$ of all concepts of $\mathbb{K}$ is a complete lattice with respect to the order defined by $(O_1, A_1) \leq (O_2, A_2)$ if and only if $O_1 \subseteq O_2$, which is equivalent to the condition $A_2 \subseteq A_1$. $\mathcal{B}(\mathbb{K})$ is called the *concept lattice* of $\mathbb{K}$.

The mappings $(\cdot)'$ are closure operators, which, under appropriate conditions, can be regarded as logical closures of theories in the Logic RZ. Details on this relationship between Logic RZ and formal concept analysis can be found in [7].

3 Logic programming in the Logic RZ

In this section, we discuss how the Logic RZ can be extended to a logic programming paradigm, thus adding disjunctive rules and default negation to the expressive features of the formalism. In addition, we review the classical approach of answer set programming that will be related to logic programming in the Logic RZ in Section 5.

Following [2], we first explain how the Logic RZ can be extended naturally to a disjunctive logic programming paradigm.

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 $w \in D$ is a* model *of P if, for every rule $Y \leftarrow X$ in P, if $w \models X$, then $w \models Y$. We write $w \models P$ in this case. 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.*

Note that the condition $w \models P$ is strictly stronger than $w \models \mathsf{cons}(P)$. For an example, consider the domain $\{p, q, \bot\}$ defined by $\bot < p$ and $\bot < q$, and the program P consisting of the single rule $\{p\} \leftarrow \{q\}$. Then $\mathsf{cons}(P)$ contains only tautologies, and thus is modelled by the element q. Yet q is not a model for P.

In [1], a notion of *default negation* was added to the logic programming framework presented above. The extension is close in spirit to mainstream developments concerning knowledge representation and reasoning with nonmonotonic logics. It will serve as the base for our query language.

Since the following definition introduces a nonmonotonic negation operator $\sim$ into the logic, we wish to emphasize that the negation $\bar{\cdot}$ from Example 1 is not a special symbol of our logic but merely a syntactical feature to denote the elements of one particular example domain. Similar situations are known in FCA: in some formal contexts, every attribute has a "negated" attribute that relates to exactly the opposite objects, but FCA in general does not define negation. Analogously, by choosing appropriate domains, Logic RZ can be used to model common monotonic negations. In contrast, the semantic extension introduced next cannot be accounted for in this way.

Definition 3. *Consider a coherent algebraic domain D. An* extended rule *is a rule of the form $H \leftarrow X, \sim Y$, where H, X, and Y are clauses over D. An extended rule is* trivially extended *if $Y = \{\}$, and we may omit Y in this case. We call the tuple (X, Y) the* body *of the rule and H the* head *of the rule. An (*extended disjunctive*)* program *is a set of extended rules.*

Informally, we read an extended rule $H \leftarrow X, {\sim}Y$ as follows: if X holds, and Y does not, then H shall hold. As usual in logic programming, the formal semantics of $\sim$ is defined by specifying the semantics of logic programs in which $\sim$ is contained. But in contrast to classical negation, semantics is not defined by specifying the effect of $\sim$ on logical interpretations, e.g. by using *truth tables*. The reason is that we want to enrich our reasoning paradigm with nonmonotonic features, which are characterized by the fact that previously drawn conclusions might become invalid when adding additional knowledge (i.e. program rules). But such semantics clearly cannot be defined *locally* by induction on the structure of logical formulae – the whole program must be taken into account for determining logical meaning. We consider the following formal definition, akin to the answer set semantics that will be introduced later on in this section.

Definition 4. *Consider a coherent algebraic domain D, an element $w \in D$, and an extended disjunctive program P. We define P/w to be the (non-extended) program obtained by applying the following two transformations:*

1. *Replace each body (X, Y) of a rule by X whenever $w \not\models Y$.*
2. *Delete all rules with a body (X, Y) for which $w \models Y$.*

An element $w \in D$ is an answer model *of P if it satisfies $w \models \mathsf{cons}(P/w)$. It is a* min-answer model *of P if it is minimal among all v satisfying $v \models \mathsf{cons}(P/w)$.*

Note that every min-answer model is an answer model. We do not require answer models w to satisfy the rules of the program P/w. However, one can show the following lemma.

Lemma 1. *Consider a coherent algebraic domain D and a disjunctive program (i.e. without default negation) P over D. If $w \in D$ is minimal among all elements v with property $v \models \mathsf{cons}(P)$, then $w \models P$.*

Proof. This was shown in [2, Lemma 5.3]. □

The proof of the following statement refers to [2, Lemma 5.1] which uses Zorn's Lemma (or, equivalently, the Axiom of Choice).

Lemma 2. *Consider a coherent algebraic domain D, and a disjunctive program P over D. If $w \in D$ is such that $w \models \mathsf{cons}(P)$, then there is an element $w' \sqsubseteq w$ such that $w' \models P$ and which is minimal among all v that satisfy $v \models \mathsf{cons}(P)$.*

Proof. By [2], $\mathsf{cons}(P)$ is a logically closed theory. By [2, Proof of Theorem 3.2], the set M of all models of $\mathsf{cons}(P)$ is compact and upwards closed. By [2, Lemma 5.1] we have that M is the upper closure of its finite set $C(M)$ of minimal compact elements. Consequently, for any $w \models \mathsf{cons}(P)$ we have $w \in M$ and there is a $w' \in C(M)$ with the desired properties. □

Intuitively, the above lemma enables us to conclude that there is a minimal model below any model of the program P. The rationale behind the definition of

min-answer model is that it captures the notion of answer set as used in answer set programming which we will introduce next.

Answer set programming (ASP) is a reasoning paradigm in artificial intelligence which was devised in order to capture some aspects of commonsense reasoning. We now briefly review the basic concepts of ASP so that we can make the relationship to nonmonotonic logic programming in the Logic RZ explicit in Section 5.

ASP is based on the observation that humans tend to *jump to conclusions* 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 [8], 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 [9], is also *nonmonotonic*, in the sense that a larger knowledge base might yield a smaller set of conclusions.

We next describe the notion of answer set for extended disjunctive logic programs, as proposed in [10]. It forms the heart of answer set programming systems like dlv[2] or smodels[3] [11, 12], which have become a standard paradigm in artificial intelligence.

Let $\mathcal{V}$ denote a countably infinite set of propositional variables. An *ASP-rule* is an expression of the form

$$L_1, \ldots, L_n \leftarrow L_{n+1}, \ldots, L_m, {\sim}L_{m+1}, \ldots, {\sim}L_k,$$

where each L_i is a literal, i.e. either of the form p or $\neg p$ for some propositional variable $p \in \mathcal{V}$. Given such an ASP-rule r, we set $\texttt{Head}(r) = \{L_1, \ldots, L_n\}$, $\texttt{Pos}(r) = \{L_{n+1}, \ldots, L_m\}$, and $\texttt{Neg}(r) = \{L_{m+1}, \ldots, L_k\}$.

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 disjunctive logic program in which $\texttt{Neg}(r)$ is empty for each ASP-rule $r \in P$. A set $W \subseteq \mathcal{V}^{\pm} = \mathcal{V} \cup \neg\mathcal{V}$ is said to be *closed by rules* in P if, for every $r \in P$ such that $\texttt{Pos}(r) \subseteq W$, we have that $\texttt{Head}(r) \cap W \neq \emptyset$. W 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 W contains complementary literals, then $W = \mathcal{V}^{\pm}$.
2. W 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 $W \subseteq \mathcal{V}^{\pm}$, consider the program P/W defined as follows.

1. If $r \in P$ is such that $\texttt{Neg}(r) \cap W$ is not empty, then we remove r i.e. $r \notin P/W$.

[2] http://www.dbai.tuwien.ac.at/proj/dlv/
[3] http://www.tcs.hut.fi/Software/smodels/

```
object(1).          object(2).        ...
attribute(a).       attribute(b).     ...
incidence(1,b).     ...

in_extent(G) :- object(G), not outof_ext(G).
outof_ext(G) :- object(G), attribute(M), in_intent(M), not incidence(G,M).

in_intent(M) :- attribute(M), not outof_int(M).
outof_int(M) :- object(G), attribute(M), in_extent(G), not incidence(G,M).
```

Fig. 1. Computing formal concepts using answer set programming.

2. If $r \in P$ is such that $\mathtt{Neg}(r) \cap W$ is empty, then the ASP-rule r' belongs to P/W, 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 $(P, W) \mapsto P/W$ is called the *Gelfond-Lifschitz transformation* of P with respect to W.

It is clear that the program P/W does not contain $\sim$ and therefore $\alpha(P/W)$ is defined. We say that W is an *answer set* or *stable model* of P if $W \in \alpha(P/W)$. So, answer sets of P are fixed points of the operator GL_P introduced by Gelfond and Lifschitz in [10], where $\mathsf{GL}_P(W) = \alpha(P/W)$.[4] We note that the operator GL_P is in general not monotonic, and call it the *Gelfond-Lifschitz operator* of P.

Example 3. We illustrate answer set programming by means of a program due to Carlos Damasio [13] given in Fig. 1. It computes all formal concepts for a given formal context. The program consists of declarations of the objects, attributes, and incidence relation in the form of facts, hinted at in the first three lines of Fig. 1. The remaining four lines suffice to describe the problem – run in an answer set programming system, the program will deliver several answer sets, which coincide with the formal concepts of the given context if restricted to the predicates `in_extent` and `in_intent`. Note that "`not`" stands for default negation $\sim$, and "`:-`" stands for $\leftarrow$. Variables are written uppercase.

We follow common practice in allowing variables to occur in programs, but we need to explain how the syntax of Fig. 1 relates to the answer set semantics given earlier. This is done by *grounding* the program by forming all ground instances of the rules by making all possible substitutions of variables by the constants occurring in the program. Variable bindings within a rule have to be respected. For example, the first rule

```
in_extent(G) :- object(G), not outof_ext(G).
```

has the ASP-rules

[4] GL_P being a multi-valued map, we speak of W as a fixed point of GL_P if $W \in \mathsf{GL}_P(W)$.

```
in_extent(1) :- object(1), not outof_ext(1).    and
in_extent(b) :- object(b), not outof_ext(b).
```

as two examples of ground instances. The resulting ground atoms, i.e. atomic formulae such as `object(b)` and `outof_ext(1)`, can then be understood as propositional variables, and the semantics given earlier can be derived.

It was shown in [1] that extended disjunctive programs over the domain $\mathbb{T}^{\mathcal{V}}$ from Example 1 can be closely related to classical answer set programming over a set of ground atoms $\mathcal{V}$. We will generalize this result in Theorem 1 below, where we incorporate information from formal contexts as well.

4 Querying formal contexts

In this section, we integrate hierarchical background knowledge specified by a formal context with the generalized programming paradigm for the Logic RZ. The result can be considered as a query language for formal contexts.

Definition 5. *Let $\mathbb{K}$ be a finite formal context with concept lattice $\mathcal{B}(\mathbb{K})$, and let $\mathbb{T}^{\mathcal{V}}$ be the domain from Example 1. A* query *over $\mathbb{K}$ is any extended disjunctive program over the domain $\mathcal{B}(\mathbb{K}) \times \mathbb{T}^{\mathcal{V}}$.*

The fact that $\mathcal{B}(\mathbb{K}) \times \mathbb{T}^{\mathcal{V}}$ is a domain follows since both factors of this product are domains as well. That $\mathcal{B}(\mathbb{K})$ is a domain is ensured by restricting the above definition to finite contexts. Concepts of $\mathbb{K}$ can be viewed as conjunctions of attributes of $\mathbb{K}$ (or, similarly, as conjunctions of objects), thus allowing for a straightforward intuitive reading of the rules of a query. When formulating a rule, however, the restriction to concepts can be unwanted, and one might prefer to state arbitrary conjunctions over attributes and objects. To this end, we now develop a more convenient concrete syntax for queries.

Definition 6. *Given a context $\mathbb{K} = (G, M, I)$, a* literal over $\mathbb{K}$ *is either an element of $G \cup M$, or a formula of the form $p(t_1, \ldots, t_n)$ or $\neg p(t_1, \ldots, t_n)$, where p is an n-ary predicate symbol and $t_1, \ldots, t_n$ are terms over some first-order language. A* rule over $\mathbb{K}$ *then is of the form*

$$L_1, \ldots, L_n \leftarrow L_{n+1}, \ldots, L_m, {\sim}L_{m+1}, \ldots, {\sim}L_k,$$

where each of the L_i is a literal over $\mathbb{K}$. A simplified query *for $\mathbb{K}$ is a set of rules over $\mathbb{K}$.*

The intention is that rules over a context allow for an intuitive reading which is similar to that for classical ASP-rules, and that finite sets of rules unambiguously represent queries over the given context. We consider rules with variables as a short-hand notation for the (possibly infinite) set of all ground instances (with respect to the considered first-order signature), and thus can restrict our attention to rules that do not contain first-order variables.

When relating Definition 6 to Definition 5, we have to be aware that both definitions are somewhat implicit about the considered logical languages. Namely, the notion of a simplified query depends on the chosen first order language, and, similarly, queries employ the domain $\mathbb{T}^{\mathcal{V}}$ that depends on choosing some concrete set of propositional variables $\mathcal{V}$. Given standard cardinality constraints, the choice of these invariants is not relevant for our treatment, but notation is greatly simplified by assuming that $\mathcal{V}$ is always equal to the set of ground atoms (i.e. atomic logic formulae without variables) over the chosen language. Thus, we can also view ground atoms as elements of $\mathbb{T}^{\mathcal{V}}$ mapping exactly the specified element of $\mathcal{V}$ to true, and leaving everything else undetermined.

Moreover, note that both $\mathcal{B}(G, M, I)$ and $\mathbb{T}^{\mathcal{V}}$ have least elements (M', M'') and $\bot$, respectively. We exploit this to denote elements of $\mathcal{B}(G, M, I) \times \mathbb{T}^{\mathcal{V}}$ by elements of $G \cup M \cup \mathcal{V} \cup \overline{\mathcal{V}}$. Namely, each element $o \in G$ denotes the element $((\{o\}'', \{o\}'), \bot)$, $a \in M$ denotes $((\{a\}', \{a\}''), \bot)$, and $p \in \mathcal{V} \cup \overline{\mathcal{V}}$ denotes $((M', M''), p)$. This abbreviation is helpful since the atomic elements are supremum-dense in $\mathcal{B}(G, M, I) \times \mathbb{T}^{\mathcal{V}}$ and thus can be used to specify all other elements.

Definition 7. *Consider a context $\mathbb{K} = (G, M, I)$, and a rule over $\mathbb{K}$ of the form*

$$L_1, \ldots, L_n \leftarrow L_{n+1}, \ldots, L_m, {\sim}L_{m+1}, \ldots, {\sim}L_k.$$

The associated *extended disjunctive rule over $\mathcal{B}(\mathbb{K}) \times \mathbb{T}^{\mathcal{V}}$ is defined as*

$$\{L_1, \ldots, L_n\} \leftarrow \bigsqcup\{L_{n+1}, \ldots, L_m\}, {\sim}\{L_{m+1}, \ldots, L_k\}.$$

Given a simplified query P, its associated query $\widehat{P}$ *is obtained as the set of rules associated to the rules of P in this sense.*

Conversely, it is also possible to find an appropriate simplified query for arbitrary queries. The problem for this transformation is that the simplified syntax does not permit disjunctions in the bodies of rules, and generally restricts to atomic expressions. It is well-known, however, that disjunctions in bodies do usually not increase expressiveness of a rule language. Indeed, consider the extended disjunctive rule

$$\{l_1, \ldots, l_n\} \leftarrow \{l_{n+1}, \ldots, l_m\}, {\sim}\{l_{m+1}, \ldots, l_k\},$$

where l_i are elements of $\mathcal{B}(\mathbb{K}) \times \mathbb{T}^{\mathcal{V}}$ as in Definition 5. Using a simple form of the so-called Lloyd-Topor transformation, we can rewrite this rule into the set of rules

$$\left\{\{l_1, \ldots, l_n\} \leftarrow \{l_j\}, {\sim}\{l_{m+1}, \ldots, l_k\} \mid j \in \{n+1, \ldots, m\}\right\}.$$

It is straightforward to show that this transformation preserves answer models and min-answer models, and we omit the details.

Similarly, it is possible to treat heads $\{l_1, \ldots, l_n\}$, where the l_i are not necessarily literals. Indeed, each l_i can be written as $l_i = \bigsqcup A_i$, where $A_i = \{a_{i1}, \ldots, a_{in_i}\}$ is a finite set of literals, and a rule of the form

$$\{l_1, \ldots, l_n\} \leftarrow \{l\}, {\sim}\{l_{m+1}, \ldots, l_k\}$$

can thus be transformed into the set of rules

$$\{\{e_1, \ldots, e_n\} \leftarrow \{l_j\}, {\sim}\{l_{m+1}, \ldots, l_k\} \mid e_i \in A_i\}\,.$$

Intuitively, this transformation is obtained by bringing the head into conjunctive normal form (using a distributivity law) and subsequent splitting of conjunctions into different clause heads. This constitutes another Lloyd-Topor transformation, and it is again straightforward – but also quite tedious – to show that the transformation preserves answer models and min-answer models.

Similar techniques could be applied to transform complex expressions in the default negated part of the body. Therefore, we can restrict our subsequent considerations to simplified queries without loss of generality.

5 Practical evaluation of queries

Based on the close relationship to answer set programming discussed in Section 3, we now present a way to evaluate queries within standard logic programming systems. This has the huge advantage that we are able to employ highly optimized state of the art systems for our reasoning tasks. Furthermore, the connection to standard answer set programming creates further possibilities to combine contextual knowledge with other data sources that have been integrated into answer set programming paradigms.

Our goal is to reduce queries to (classical) answer set programs. For this it is necessary to translate both the rules of the program and the data of the underlying formal context into the standard paradigm. On a syntactic level, we already established a close correspondence based on the notion of a simplified query. We now show how to take this syntactic similarity to a semantic level.

Definition 8. *Given a simplified query P for a context $\mathbb{K} = (G, M, I)$, consider the syntactic representation of P as an extended disjunctive logic program over the set of variables $G \cup M \cup \mathcal{V}$. Furthermore, define a program $\mathsf{ASP}(\mathbb{K})$ over this set of variables to consist of the union of*

1. *all rules of the form $o \leftarrow a_1, \ldots, a_n$, with $a_1, \ldots, a_n \in M$, $o \in \{a_1, \ldots, a_n\}'$,*
2. *all rules of the form $a \leftarrow o_1, \ldots, o_n$, with $o_1, \ldots, o_n \in G$, $a \in \{o_1, \ldots, o_n\}'$.*

By $\mathsf{ASP}(P)$, we denote the extended disjunctive logic program $P \cup \mathsf{ASP}(\mathbb{K})$ that is thus associated *with the simplified query P.*

Obviously, $\mathsf{ASP}(\mathbb{K})$ (and therefore also $\mathsf{ASP}(P)$) will in general contain redundancies, which could be eliminated if desired by using stem base techniques

(see [6]). We will not discuss this optimization in detail as it is not necessary for our exhibition.

On the semantic level of min-answer models and answer sets, the relationship between queries and logic programs is described by the following function.

Definition 9. *Consider the domain $\mathbb{T}^{\mathcal{V}}$ and a context $\mathbb{K} = (G, M, I)$. A mapping ι from elements of $\mathcal{B}(\mathbb{K}) \times \mathbb{T}^{\mathcal{V}}$ to subsets of $G \cup M \cup \mathcal{V}^{\pm}$ is defined by setting $\iota(w) = \{p \mid w \models p\}$.*

Note that the above definition is only meaningful when employing our convention of using elements $p \in G \cup M \cup \mathcal{V}^{\pm}$ to denote (not necessarily atomic) elements of $\mathcal{B}(\mathbb{K}) \times \mathbb{T}^{\mathcal{V}}$, as discussed in the previous section. This relationship need not be injective, but elements from $\mathcal{V}^{\pm}$ and $G \cup M$ are never associated with the same element of $\mathcal{B}(\mathbb{K}) \times \mathbb{T}^{\mathcal{V}}$, and this suffices to eliminate any confusion in our following considerations.

Lemma 3. *Consider a simplified query P for a context $\mathbb{K}$ with associated query $\widehat{P}$. For any element $w \in \mathcal{B}(\mathbb{K}) \times \mathbb{T}^{\mathcal{V}}$, we find that $\mathsf{ASP}(P)/\iota(w) = P/\iota(w) \cup \mathsf{ASP}(\mathbb{K})$.*

Furthermore, an ASP-rule $L_1, \ldots, L_n \leftarrow L_{n+1}, \ldots, L_m$ is in $P/\iota(w)$ iff the corresponding rule $\{L_1, \ldots, L_n\} \leftarrow \bigsqcup\{L_{n+1}, \ldots, L_m\}$ is in $\widehat{P}/w$.

Proof. The first part of the claim is immediate, since there are no default negated literals in $\mathsf{ASP}(\mathbb{K})$. For the second part, note that any rule in either $P/\iota(w)$ or $\widehat{P}/w$ stems from a rule of the form $L_1, \ldots, L_n \leftarrow L_{n+1}, \ldots, L_m, {\sim}L_{m+1}, \ldots, {\sim}L_k$ in P. To finish the proof, we just have to observe that $w \models \{L_{m+1}, \ldots, L_k\}$ iff $\iota(w) \models L_i$ for some $i = m+1, \ldots, k$. □

Restricting to non-extended disjunctive programs only, the next lemma establishes the basis for the main result of this section.

Lemma 4. *Consider a simplified query P over some context $\mathbb{K} = (G, M, I)$, such that no default negation appears in P. If $w \in \mathcal{B}(\mathbb{K}) \times \mathbb{T}^{\mathcal{V}}$ is such that $w \models \widehat{P}$, then $\iota(w)$ is closed under rules of $P \cup \mathsf{ASP}(\mathbb{K})$.*

Conversely, assume there is a consistent set $W \subset (G \cup M \cup \mathcal{V})^{\pm}$ closed under rules of $P \cup \mathsf{ASP}(\mathbb{K})$. Then $W \supseteq \iota(w)$ for some element $w \in \mathcal{B}(\mathbb{K}) \times \mathbb{T}^{\mathcal{V}}$ with property $w \models \widehat{P}$. In particular, if W is minimal among all sets closed under said rules, then W is equal to $\iota(w)$.

Proof. For the first part of the claim, let $w \models \widehat{P}$ be as above, and consider an ASP-rule $r \in P \cup \mathsf{ASP}(\mathbb{K})$ given by $L_1, \ldots, L_n \leftarrow L_{n+1}, \ldots, L_m$. First consider the case $r \in P$. By Definition 7, $\{L_1, \ldots, L_n\} \leftarrow \bigsqcup\{L_{n+1}, \ldots, L_m\}$ is in $\widehat{P}$. If $L_{n+1}, \ldots, L_m \in \iota(w)$, then $w \models L_i$, $i = n+1, \ldots, m$ by the definition of ι. Hence $w \models \bigsqcup\{L_{n+1}, \ldots, L_m\}$ and thus $w \models L_j$ for some $j \in \{1, \ldots, n\}$. But then $L_j \in \iota(w)$ and so $\iota(w)$ satisfies the rule r.

On the other hand, that $\iota(w)$ satisfies any ASP-rule $r \in \mathsf{ASP}(\mathbb{K})$ is obvious from the fact that $\iota(w) \cap G$ is an extent with corresponding intent $\iota(w) \cap M$. This finishes the proof that $\iota(w)$ is closed under rules of $\mathsf{ASP}(P)/\iota(w)$.

For the second part of the claim, consider a set W as in the assumption. First note that elements of $\neg G \cup \neg M$ do not occur in the rules of $P \cup \mathsf{ASP}(\mathbb{K})$. Consequently, whenever W is closed under rules of $P \cup \mathsf{ASP}(\mathbb{K})$, we find that $V = W \cap (G \cup M \cup \mathcal{V}^{\pm})$ has this property as well.

Now consider sets $O = V \cap G$ and $A = V \cap M$. We claim that (O, A) is a concept of $\mathbb{K}$, i.e. $O' = A$. Since V is closed under $\mathsf{ASP}(\mathbb{K})$, whenever $a \in O'$ for some $a \in M$, we find $a \in W$, and hence $A \supseteq O'$. Analogously, we derive $O \supseteq A'$. Using standard facts about concept closure $(\cdot)'$ [6], we obtain $O'' \subseteq A' \subset O$ and $A'' \subseteq O' \subset A$ which establishes the claim.

Given that $V \cap (G \cup M)^{\pm}$ is the (disjoint) union of the extent and intent of a concept, it is obvious that $V = \iota(w)$ for some $w \in \mathcal{B}(\mathbb{K}) \times \mathbb{T}^{\mathcal{V}}$. Here we use the assumed consistency of W and V to ensure that the $\mathbb{T}^{\mathcal{V}}$-part of V can indeed be expressed by an appropriate w.

We still have to show that $w \models \widehat{P}$. For any rule $\{L_1, \ldots, L_n\} \leftarrow \bigsqcup\{L_{n+1}, \ldots, L_m\}$ in $\widehat{P}$, there is an ASP-rule $L_1, \ldots, L_n \leftarrow L_{n+1}, \ldots, L_m$ in P. By the definition of ι, it is clear that $\iota(w) = V$ models this ASP-rule iff w models the corresponding rule, which establishes the claim, since V models all ASP-rules. □

Theorem 1. *Consider a simplified query P with associated query $\widehat{P}$ and program $\mathsf{ASP}(P)$. If $\widehat{P}$ has any min-answer models, then the function ι from Definition 9 is a bijection between min-answer models of P and answer sets of $\mathsf{ASP}(P)$.*

Proof. Consider an answer set W of $\mathsf{ASP}(P)$ such that $W \neq (G \cup M \cup \mathcal{V})^{\pm}$. Considering P/W as a simplified query, we can apply Lemma 4, to find some element $w \models \widehat{P/W}$ such that $W = \iota(w)$. By Lemma 3, an ASP-rule is in P/W iff a corresponding rule is in $\widehat{P}/w$, and we conclude $w \models \widehat{P}/w$. We claim that w additionally is a min-answer model. Indeed, by Lemma 2, there is an element $w' \sqsubseteq w$ such that $w' \models \widehat{P}/w$ and which is minimal among all $v \models \mathsf{cons}(P)$. For a contradiction, suppose that $w \neq w'$, i.e. w is not a min-answer model. Using Lemmas 3 and 4, we find that $w' \models \widehat{P}/w$ implies that $\iota(w')$ is closed under rules of $P/\iota(w)$. Closure of $\iota(w')$ under rules of $\mathsf{ASP}(\mathbb{K})$ is immediate since the first component of w' is required to be a concept. Thus, we obtain a model $\iota(w')$ for $\mathsf{ASP}(P)$ which is strictly smaller than $\iota(w)$. This contradicts the assumed minimality of $\iota(w) = W$, so that w must be a min-answer model.

Conversely, we show that $\iota(w)$ is an answer set of $\mathsf{ASP}(P)$ whenever w is a min-answer model of $\widehat{P}$. Combining Lemmas 3 and 4, we see that $\iota(w)$ is closed under rules of $\mathsf{ASP}(P)/\iota(w)$. For a contradiction, suppose that $\iota(w)$ is not minimal, i.e. there is some model $V \subseteq W$ that is also closed under rules of $\mathsf{ASP}(P)$. Then, by Lemma 4, we have $V \supseteq \iota(v)$ for some v for which $v \models \widehat{P}/w$. Clearly, $\iota(v) \subseteq V \subseteq W$ implies $v \sqsubset w$, thus contradicting our minimality assumption on w. □

The above result can be compared to the findings in [1], where programs over $\mathbb{T}^{\mathcal{V}}$ were related to answer set programs. The corresponding result can be obtained from Theorem 1 by restricting to the empty context $(\emptyset, \emptyset, \emptyset)$.

6 Conclusions and further work

We have shown how artificial intelligent commonsense reasoning in the form of answer set programming can be merged with conceptual knowledge in the sense of formal concept analysis. We have utilized this in order to develop a commonsense query answering system for formal contexts, which features the full strength of disjunctive answer set programming.

Based on our results, it is straightforward to implement this e.g. on top of the dlv system[5] [11], which has recently been supplemented to support extensions like ours in a hybrid fashion [14].

The dlv system also provides modules for interfacing with conceptual knowledge in other paradigms like OWL [15] or RDF [16], resulting in a hybrid reasoning system. These features become available for us by way of dlv and thus allow for an integrated querying and reasoning paradigm over heterogeneous knowledge. This could also be further enhanced with query-based multicontext browsing capabilities in the sense of [17].

Finally, let us remark that our work sheds some light on recently discussed issues concerning the interplay between conceptual knowledge representation and rule-based reasoning.[6] Indeed, our approach realizes a strong integration between paradigms, with the disadvantage that it is restricted to hierarchical conceptual knowledge in the sense of formal concept analysis. It may nevertheless be a foundation for further investigations into the topic from an order-theoretic perspective.

References

1. Hitzler, P.: Default reasoning over domains and concept hierarchies. In Biundo, S., Frühwirth, T., Palm, G., eds.: Proceedings of the 27th German conference on Artificial Intelligence, KI'2004, Ulm, Germany, September 2004. Volume 3238 of Lecture Notes in Artificial Intelligence., Springer, Berlin (2004) 351–365
2. Rounds, W.C., Zhang, G.Q.: Clausal logic and logic programming in algebraic domains. Information and Computation **171** (2001) 156–182
3. 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)
4. Plotkin, G.: T^ω as a universal domain. Journal of Computer and System Sciences **17** (1978) 209–236
5. Fitting, M.: A Kripke-Kleene-semantics for general logic programs. The Journal of Logic Programming **2** (1985) 295–312
6. Ganter, B., Wille, R.: Formal Concept Analysis – Mathematical Foundations. Springer, Berlin (1999)
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

[5] `http://www.dbai.tuwien.ac.at/proj/dlv/`

[6] See e.g. the work of the W3C Rule Interchange Format working group at `http://www.w3.org/2005/rules/wg.html`.

8. Makinson, D.: Bridges between classical and nonmonotonic logic. Logic Journal of the IGPL **11** (2003) 69–96
9. Reiter, R.: A logic for default reasoning. Artificial Intelligence **13** (1980) 81–132
10. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Computing **9** (1991) 365–385
11. 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)
12. Simons, P., Niemelä, I., Soininen, T.: Extending and implementing the stable model semantics. Artificial Intelligence **138** (2002) 181–234
13. Damasio, C.: Personal communication (2004)
14. Eiter, T., Ianni, G., Schindlauer, R., Tompits, H.: A uniform integration of higher-order reasoning and external evaluations in answer set programming. In Kaelbling, L.P., Saffiotti, A., eds.: Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-05). (2005)
15. Smith, M.K., McGuinness, D.L., Welty, C.: OWL Web Ontology Language Guide. W3C Recommendation 10 February 2004 (2004) available at `http://www.w3.org/TR/owl-guide/`.
16. Manola, F., Miller, E.: Resource Description Framework (RDF) Primer. W3C Recommendation 10 February 2004 (2004) available at `http://www.w3.org/TR/rdf-primer/`.
17. Tane, J.: Using a query-based multicontext for knowledge base browsing. In: Formal Concept Analysis, Third International Conf., ICFCA 2005-Supplementary Volume, Lens, France, IUT de Lens, Universite d'Artois (2005) 62–78