From Trace Sets to Modal-Transition Systems by Stepwise Abstract Interpretation

David A. Schmidt [1]

Computing and Information Sciences Department
Kansas State University
Manhattan, KS 66506 USA.

Abstract

Following and expanding upon the philosophy set down by Cousot and Cousot, this tutorial paper uses stepwise abstract interpretation to transform a system's naive trace-set semantics into a format that is readily analyzable by temporal logic. The abstraction interpretations first transform a concrete trace-set semantics, where the traces are characterized by a state-transition relation, into an abstracted trace-set semantics that uses abstracted states such that linear-time properties are soundly (and in some cases, completely) preserved. Conditions under which universally and existentially quantified linear-time properties are preserved are also stated, and these conditions motivate the universal and existential abstractions of the abstracted trace-set semantics into state-set semantics upon which quantified properties can be soundly and completely checked, producing simple branching-time logics.

Finally, the combination of universal and existential quantification in a single logic motivates mixed and modal transition systems, where a concrete system is abstracted by two transition relations, one that abstracts universal behaviors and one that abstracts existential behaviors. The definitions, constructions, and technical results are illustrated by examples.

1 Introduction

A standard way of understanding a hardware or software system is by executing it and studying the execution traces that result. *Linear-time temporal logic* [16,37] was designed for stating and checking patterns of correct behavior of individual execution traces, and as argued most strongly by M. Vardi

[1] Ph: +1-785-532-6350. schmidt@cis.ksu.edu. Supported by NSF CCR-9970679, INT-9981558, ITR-0085949, and ITR-0086154.

[50] but also by many others, it is the most natural format for this purpose. But many testing and analysis frameworks are oriented towards another format, branching-time temporal logic [4,16,18], which lets one write correctness properties in terms of the potential execution steps ("branches") that might be taken from each state in a system.

The standard formulations of linear-time logic (*LTL* [16,37]) and branching-time logic (*CTL* [6,16]) are often presented as incomparable, because of theoretical results that assert that the two logics are incomparable in terms of expressivity [3,17,27]. Indeed, in a landmark paper, Lamport presented linear-time logic and branching-time logic as competing semantical interpretations of the same temporal-logic notation [27]. Although these insights are valuable, they have suggested that the linear-time and branching-time approaches are "competitors," locked in a dual where one must vanquish the other [18,50].

A significant paper by Cousot and Cousot shows that this scenario need not be the case [12]: By employing *abstract interpretation*, a theory of abstraction of structure [9,10,25], Cousot and Cousot show that a system's concrete operational semantics generates execution traces whose behaviors are neatly described in terms of linear-time temporal logics. Application of abstract interpretation to both the execution-trace sets *and* the linear-temporal logic generates natural branching-time temporal logics. Rather than being incomparable competitors, linear-time and branching-time logics are related by abstract interpretation mappings in sound and complete ways.

This tutorial paper applies these insights to a pedagogical, stepwise, abstract-interpretation-based development of frameworks for sound (and in some cases, complete) extraction of temporal-logic properties of systems that are understood in terms of execution traces. The stages proceed as follows:

(1) A system's semantics is defined by its set of execution traces, called its *model*. For computability purposes, the model must be characterized by a (not necessarily finite) binary transition relation.

    A linear-time temporal logic, LMC (*linear mu-calculus*), is used to express correctness properties of traces, and LMC's semantics defines when a trace satisfies a correctness property. Correctness properties can be quantified as *universal* (the property holds for all traces) or *existential* (the property holds for a trace).

    This material appears in Section 2.

(2) For reasons of finite computability, the states that appear in execution traces are abstracted to states with simpler structure. The abstraction is performed by means of a Galois connection [9,31], which is a special form of binary abstraction relation [40].

    Although the abstraction is done on the system's states, it affects the generation of the system's traces. Specifically, the transition relation that

generates the concrete system's model is abstracted to a transition relation that generates the abstract system's model. The abstraction is also applied to the linear-time logic, synthesizing an abstracted logic that one uses to check the abstracted traces.

Using the classic techniques introduced by Cousot and Cousot [11], a sound, and in some cases, complete abstract logic is derived— abstract properties that are validated as true on an abstract trace must hold true for the corresponding concrete property applied to the corresponding concrete trace.

The cases of universally and existentially quantified properties are given special attention, because the standard abstraction technique soundly reflects only universally quantified properties on the corresponding concrete system. This motivates the formulation of a dual abstracted transition relation that soundly reflects existentially quantified properties.

Sections 3 and 4 present these developments.

(3) For historical and pragmatic reasons, we abstract the linear-time logic further so that it checks states, rather than traces: A trace to be checked is abstracted to just one state that represents the entire trace, and the logic is abstracted to a logic that checks properties of states. As noted by Cousot and Cousot [12], this abstraction transforms linear-time logic into branching-time logic.

But there are two such abstractions from linear-time logic to branching-time logic: a universal abstraction and an existential abstraction, showing that classic linear-time logic (LTL) abstracts to either universal branching-time logic (ACTL) or existential branching-time logic (ECTL), each of which must be checked on correspondingly abstracted transition systems. For restricted versions of the abstracted logics, state checking of properties is complete with respect to the corresponding trace checking of the related trace property.

Section 5 explains these techniques.

(4) Finally, the universal and existential quantifiers can be added directly into linear-time logic, giving the logic, *mu-CTL**. But abstraction of the logic's quantifiers must be done in *two* ways, by means of a universal abstraction and an existential abstraction. This produces a *mixed transition system*, which allows both trace and state checking, of mu-CTL*. Specifically, the state-checking abstraction of a subset of the logic yields the *modal mu-calculus*, a standard branching-time logic which is complete for state checking with respect to the restricted form of mu-CTL*. A linear-time logic, mu-XCTL*, is proposed for describing properties of *modal transition systems*, which are a well-behaved variant of mixed transition systems.

Sections 6 and 7 develop these results.

At the conclusion of this development, we have a roadmap that takes us from a naive and natural trace semantics with its linear-time logic to a systematically

abstracted version based on multiple transition relations and corresponding linear- and branching-time logics.

2 Traces and Linear-Time Logic

We assume that the operational semantics of a program, a hardware component, or a software system is based on incremental, discrete updates of *state*: A program begins in some starting state, and each execution step transforms the current state into a new state. A state traditionally consists of an instruction counter paired with the values of storage cells, registers, or input data, but any configuration that supports incremental updates is acceptable. We use Σ to name the set of states that might be encountered during executions. An execution of a program generates a countably infinite sequence of states, called an *execution trace*:

Definition 1 *For state set, Σ, $\Sigma^\omega = Nat \to \Sigma$ is the set of (infinite) execution traces consisting of states from Σ. For $\pi \in \Sigma^\omega$, $i \in Nat$, $\pi(i) \in \Sigma$ is the ith state in π, and $\pi^k = \lambda i.\pi(k+i)$ is the trace that results from forgetting the first k states of π and treating $\pi(k)$ as the initial state.*

We will informally write a trace as a sequence of states, e.g., $s_0 s_1 \cdots s_i \cdots$, and use the abbreviation, s^ω, to represent s repeated infinitely, $s\ s\ s \cdots$.

We assume that traces are infinite primarily for technical convenience; to model a finite execution as an infinite trace, repeat the final state infinitely or append a special "halt" state infinitely to the end of the sequence.

The notion of trace in Definition 1 can be adapted to display the inputs or events that generate state updates. A sequence, $s_0 \xrightarrow{e_0} s_1 \xrightarrow{e_1} s_2 \cdots$, where event e_i triggers a transition from s_i to s_{i+1}, can be expressed as the sequence, $(s_0)(e_0, s_1)(e_1, s_2) \cdots$, where event-state pairs are used in the sequence. We retain Definition 1 primarily for convenience.

Of course, a program might be executed with many distinct start states, s_0, and a nondeterministic program might generate distinct execution traces from the same s_0. For this reason, one "execution" might generate a set of traces. Or, the traces generated by distinct executions might be grouped into one trace set. For technical reasons, it is important that a trace set of executions be *suffix closed*:

Definition 2 *A trace set, $T \subseteq \mathcal{P}(\Sigma^\omega)$, is* suffix closed *if, for all $\pi \in T$ and $i \in Nat$, $\pi^i \in T$ also.*

The trace set is "closed" in its expression of the "futures" of the executions it portrays. When a suffix-closed set of traces contains all the executions of interest to us, we call the set a *model* and use $\mathcal{M} \subseteq \mathcal{P}(\Sigma^\omega)$ to denote it.

2.1 Linear-time logic

Given a model, $\mathcal{M}$, we might examine its traces for correctness properties. Such an examination might check each trace in the model for desirable (or undesirable) state sequences. For example, a trace that documents the usage of a resource should contain desirable sequences of the form,

$$acquire \cdots use \cdots release$$

that is, a resource must be acquired before it is used, and it must be released some time thereafter. If a trace deviates from the desired behavior, the deviations should be reported.

To help us discuss such behaviors without digging into the internal structure of the states, we define a set, *AtomProp*, that names the primitive, atomic properties that we care about (e.g., *acquire, use, release*), and we employ an interpretation function, $\mathcal{I} : \Sigma \to \mathcal{P}(AtomProp)$, that maps each state to those atomic properties that hold true for it. For example, we might define $AtomProp = \{acquire, use, release\}$ and then define $\mathcal{I}$ so that those program states, s, that acquire the resource have $acquire \in \mathcal{I}(s)$ and those states that use the resource have $use \in \mathcal{I}(s)$, and so on. In this way, $\mathcal{I}$ catalogs the atomic behaviors of individual states. [2]

Linear-time temporal logic is a notation for stating behavioral patterns within traces. Regular-expression notation is one example of linear-time temporal logic; to see this, say that we have a model, $\mathcal{M}$, whose states, Σ, possess properties from $AtomProp = \{acquire, use, release\}$. Then, the regular expression, $E = acquire; \mathsf{true}^*; release$, is one way of stating that an acquired resourse must be released some finite time later. [3] We check whether a trace, $\pi \in \mathcal{M}$, matches E as follows:

- $acquire \in \mathcal{I}(\pi(0))$
- $release \in \mathcal{I}(\pi(n))$, for some $n > 0$

[2] This development can be taken to an extreme by *defining* $\Sigma = \mathcal{P}(AtomProp)$ and making $\mathcal{I}$ the identity mapping. This characterization is sometimes called a *predicate abstraction* [22].

[3] This example is somewhat informal, where true^* represents a finite state sequence where atomic properties are unimportant.

$$[\![\cdot]\!]^{\mathcal{M}} \subseteq \mathcal{M}$$

$$[\![p]\!]^{\mathcal{M}} = \{\pi \in \mathcal{M} \mid \pi(0) \in \mathcal{I}(p)\}$$
$$[\![\neg\phi]\!]^{\mathcal{M}} = \mathcal{M} - [\![\phi]\!]^{\mathcal{M}}$$
$$[\![\phi_1 \vee \phi_2]\!]^{\mathcal{M}} = [\![\phi_1]\!]^{\mathcal{M}} \cup [\![\phi_2]\!]^{\mathcal{M}}$$
$$[\![\mathsf{X}\phi]\!]^{\mathcal{M}} = \{\pi \in \mathcal{M} \mid \pi^1 \in [\![\phi]\!]^{\mathcal{M}}\}$$
$$[\![\phi_1 \mathsf{U} \phi_2]\!]^{\mathcal{M}} = \{\pi \in \mathcal{M} \mid \text{there exists } k \geq 0 \text{ such that for all } 0 \leq j < k, \pi^j \in [\![\phi_1]\!]^{\mathcal{M}}, \text{ and } \pi^k \in [\![\phi_2]\!]^{\mathcal{M}}\}$$

Fig. 1. LTL semantics

If the check succeeds, we write $\pi \models E$ to denote that π has property E.

Regular expression notation includes union ($\vee$), intersection ($\wedge$), and complement ($\neg$), giving a propositional logic. For example, to assert, "for all subtraces within a trace, property E holds," we might write the expression, $\neg(\mathsf{true}^*; \neg E)$. But regular-expression notation is clumsy to use for specifying behavioral properties,[4] so we move to an alternative notation, namely, linear-time logic as proposed by Pnueli [16,37].

Here is a syntax for a linear-time logic that we title *LTL*:

$$p \in \mathit{AtomProp} \quad \phi \in \mathit{PathProp}$$
$$\phi \ ::= \ p \mid \phi_1 \vee \phi_2 \mid \neg\phi \mid \mathsf{X}\phi \mid \phi_1 \mathsf{U} \phi_2$$

Specifications in LTL consist of atomic propositions, p, the propositional connectives, and two *modalities*:

- $\mathsf{X}\phi$ ("next ϕ"), which asserts that ϕ holds true for the subtrace that begins at the next time instance;
- $\phi_1 \mathsf{U} \phi_2$ ("ϕ_1 until ϕ_2"), which asserts that ϕ_1 holds true for some finite sequence of time until ϕ_2 becomes true.

Figure 1 presents the logic's semantics, where the meaning of a proposition is defined as that subset of model $\mathcal{M}$ for which the proposition holds true. (Examples soon follow.) Classical propositional reasoning lets us employ the abbreviations, $\phi_1 \wedge \phi_2$ for $\neg(\neg\phi_1 \vee \neg\phi_2)$ and $\phi_1 \supset \phi_2$ for $\neg(\phi_1 \wedge \neg\phi_2)$. When $\pi \in [\![\phi]\!]^{\mathcal{M}}$, we write $\pi \models^{\mathcal{M}} \phi$ (similarly for $\pi \not\models^{\mathcal{M}} \phi$).

[4] But there has been sustantial effort in employing ω-regular expressions in this context [16,48].

Some examples best explain the semantics:

Example 3 *Let* $\Sigma = \{s_i \mid i \geq 0\}$ *and say that the atomic properties for the states are defined as* $\mathcal{I}(s_{2i}) = \{even\}$ *and* $\mathcal{I}(s_{2i+1}) = \{odd\}$*, for* $0 \leq i \leq 49$*, and finally* $\mathcal{I}(s_j) = \{even, halt\}$*, for* $i \geq 100$*. Now, consider the trace,* $\pi = s_0 s_1 \cdots s_i \cdots$*, and define model* $\mathcal{M} = \{\pi^i \mid i \geq 0\}$*;* $\mathcal{M}$ *is the suffix closure of* π*.*

Here are some examples for the traces in $\mathcal{M}$:

- $\pi^0 \models^{\mathcal{M}} even \wedge \mathsf{X}odd$, because the trace's initial state has property *even* and the subtrace π^1 has property *odd*. Similarly, $\pi^1 \not\models^{\mathcal{M}} even \wedge \mathsf{X}odd$, implying $\pi^1 \models^{\mathcal{M}} \neg(even \wedge \mathsf{X}odd)$. Also, $\pi^1 \models^{\mathcal{M}} \mathsf{X}(even \wedge \mathsf{X}odd)$. These examples show how to use the X modality to list sequences of properties/events.
- $\pi^0 \models^{\mathcal{M}} (odd \vee \mathsf{X}odd)\mathsf{U}halt$, because the trace maintains $odd \vee \mathsf{X}odd$ until time 100, when π^{100} has property *halt*. Indeed, $\pi^i \models^{\mathcal{M}} (odd \vee \mathsf{X}odd)\mathsf{U}halt$ for all $i \geq 0$.

  Let true represent some tautology, e.g., $even \vee \neg even$. Then $\pi^0 \models^{\mathcal{M}} \mathsf{true}\mathsf{U}halt$ holds. This particular LTL pattern is abbreviated by the modality, F:

$$\mathsf{F}\phi \text{ iff } \mathsf{true}\mathsf{U}\phi$$

  It can be read as, "in the finite future, ϕ holds."
- Another commonly used pattern is

$$\mathsf{G}\phi \text{ iff } \neg\mathsf{F}\neg\phi$$

  which is read as "ϕ holds generally (globally) true." We see that $\pi^{100} \models^{\mathcal{M}} \mathsf{G}halt$ and $\pi^0 \not\models^{\mathcal{M}} \mathsf{G}even$.

  Propositions like $\mathsf{GF}even$ ("*even* holds infinitely often") and $\mathsf{FG}even$ ("in the future, *even* holds generally") are terse but important examples of behaviors expressible in LTL.

A typical LTL proposition will state a *safety*[5] or invariance property of the form $\mathsf{G}\phi$, e.g., $\mathsf{G}(\neg use\mathsf{U}acquire)$—"it is generally true that a resource is not used until it is first acquired." Or, an LTL proposition might state a *liveness* [6] property of the form $\mathsf{F}\phi$, e.g., $acquire \supset \mathsf{F}release$—"if the resource is acquired now, then in the future it is released."

Example 4 *Consider a system consisting of a shared file,* n *processes that read the file, and 1 process that writes to it. States in such a system take the form,* $\langle r_1, \cdots, r_n, w \rangle$*, where each* $r_i \in \{sleep, read\}$ *is the state of the* i*th*

[5] A safety property is sometimes described as an assertion stating, "something bad never happens" [27].

[6] The intuition is, "something good must eventually happen" [27].

reader, and $w \in \{wait, write\}$ *is the state of the writer. Traces are generated by applying these transition schemes:*

$$\langle \cdots, sleep, \cdots wait \rangle \rightarrow \langle \cdots, read, \cdots wait \rangle$$
$$\langle \cdots, read, \cdots \rangle \rightarrow \langle \cdots, sleep, \cdots \rangle$$
$$\langle sleep, sleep, \cdots, sleep, wait \rangle \rightarrow \langle sleep, sleep, \cdots, sleep, write \rangle$$
$$\langle \cdots, write \rangle \rightarrow \langle \cdots, wait \rangle$$

For example, for two readers and one writer, a trace might begin as

$$\langle sleep, sleep, wait \rangle \rightarrow \langle sleep, read, wait \rangle \rightarrow \langle read, read, wait \rangle$$
$$\rightarrow \langle sleep, read, wait \rangle \rightarrow \langle sleep, sleep, wait \rangle \rightarrow \langle sleep, sleep, write \rangle$$
$$\rightarrow \cdots$$

Atomic properties might be assigned to the states depending on whether the file is currently read or written: $AtomProp = \{read, write\}$, $read \in \mathcal{I}\langle \cdots, read, \cdots \rangle$; $write \in \mathcal{I}\langle \cdots, write \rangle$. *For example,* $\mathcal{I}\langle sleep, read, write \rangle = \{read, write\}$.

Model $\mathcal{M}$ *is the set of all traces generated from initial state,* $\langle sleep, \cdots, sleep, wait \rangle$, *made suffix closed.*

For the previous example, $\mathsf{G}(\neg(read \wedge write))$ expresses the safety property that the file is never simultaneously read and written; the property can be verified true for every trace starting from the initial state. Another example is the liveness property, $\mathsf{F}write$. This property *fails* to hold for those traces in the model that allow the reader processes to unfairly use the file forever— an assumption of fair execution must be added to rule out the unfair traces, e.g., $fairness \supset \mathsf{F}write$, where $fairness$ is an LTL-like coding of a pattern of fair execution.

As noted by Wolper and others [50,52], LTL is not expressive enough to state notions of fair execution;[7] as shown by Thomas [47], LTL can express exactly the star-free ω-regular expressions. For this reason among others, we consider a more expressive variant, the *linear mu-calculus*.

[7] See Clarke, et al. [6] for an introduction to coding fairness assumptions.

If LTL is propositional logic with the X modality and "iteration" (U), then the linear mu-calculus [2,49] is propositional logic with X and recursion (least-fixed-point propositions, $\mu Z.\phi_Z$, and greatest-fixed-point propositions, $\nu Z.\phi_Z$). Here is the syntax of the linear mu-calculus we call *LMC*:

$$p, \neg p \in \mathit{AtomProp} \qquad Z \in \mathit{Var} \qquad \phi \in \mathit{PathProp}$$

$$\phi \;::=\; p \;\mid\; \neg p \;\mid\; \phi_1 \vee \phi_2 \;\mid\; \phi_1 \wedge \phi_2 \;\mid\; \mathsf{X}\phi \;\mid\; \mu Z.\phi \;\mid\; \nu Z.\phi \;\mid\; Z$$

The recursion operators let us easily express safety and liveness propositions: $\mathsf{G}\phi$ is written $\nu Z.\phi \wedge \mathsf{X}Z$, and $\mathsf{F}\phi$ is $\mu Z.\phi \vee \mathsf{X}Z$. Also, $\phi_1 \mathsf{U} \phi_2$ is $\mu Z.\phi_2 \vee (\phi_1 \wedge \mathsf{X}Z)$. These can be understood by considering the infinite unfoldings of the recursions:

$$\mu Z.\phi \text{ iff } \bigvee_{i \geq 0} \phi_i, \text{ where } \begin{cases} \phi_0 = \mathsf{false} \\ \phi_{i+1} = [\phi_i / Z]\phi \end{cases}$$

$$\nu Z.\phi \text{ iff } \bigwedge_{i \geq 0} \phi_i, \text{ where } \begin{cases} \phi_0 = \mathsf{true} \\ \phi_{i+1} = [\phi_i / Z]\phi \end{cases}$$

where $[\phi'/Z]\phi$ defines the replacement of all free occurrences of Z by ϕ' within ϕ.

More formally, the recursion operators define the least- and greatest-fixed points of monotonic functionals defined over $\mathcal{P}(\mathcal{M})$; their semantics appear in Figure 2. As the Figure shows, an environment, ρ, remembers the trace sets denoted by variables. The classic Tarski-fixed-point theorems give the semantics of μ and ν. The monotonicity of the functional, F, used to calculate the fixed point depends crucially on the absence of negation, which is antimonotonic on the complete lattice, $\mathcal{P}(\mathcal{M})$. For this reason, LMC allows negations on atomic propositions only.

For the denotations of atomic propositions, we demand the following *consistency property* for the interpretation map, $\mathcal{I}$ [14]:

$$\text{for all } s \in \Sigma, \text{ for all } p \in \mathit{AtomProp}, \{p, \neg p\} \not\subseteq \mathcal{I}(s)$$

We can also demand *completeness* of $\mathcal{I}$:

$$\text{for all } s \in \Sigma \text{ and } p \in \mathit{AtomProp}, \text{ either } p \in \mathcal{I}(s) \text{ or } \neg p \in \mathcal{I}(s)$$

$[\![\cdot]\!]^{\mathcal{M}} \in Env \to \mathcal{P}(\mathcal{M})$, where $\rho \in Env = Var \to \mathcal{P}(\mathcal{M})$:

$$[\![p]\!]^{\mathcal{M}}\rho = \{\pi \in \mathcal{M} \mid p \in \mathcal{I}(\pi(0))\}$$
$$[\![\neg p]\!]^{\mathcal{M}}\rho = \{\pi \in \mathcal{M} \mid \neg p \in \mathcal{I}(\pi(0))\}$$
$$[\![\phi_1 \wedge \phi_2]\!]^{\mathcal{M}}\rho = and([\![\phi_1]\!]^{\mathcal{M}}\rho, [\![\phi_2]\!]^{\mathcal{M}}\rho)$$

where $and : \mathcal{P}(\mathcal{M}) \times \mathcal{P}(\mathcal{M}) \to \mathcal{P}(\mathcal{M})$ is defined as

$$and(S_1, S_2) = S_1 \cap S_2$$
$$[\![\phi_1 \vee \phi_2]\!]^{\mathcal{M}}\rho = or([\![\phi_1]\!]^{\mathcal{M}}\rho, [\![\phi_2]\!]^{\mathcal{M}}\rho)$$

where $or : \mathcal{P}(\mathcal{M}) \times \mathcal{P}(\mathcal{M}) \to \mathcal{P}(\mathcal{M})$ is defined as

$$or(S_1, S_2) = S_1 \cup S_2$$
$$[\![\mathsf{X}\phi]\!]^{\mathcal{M}}\rho = next([\![\phi]\!]^{\mathcal{M}}\rho)$$

where $next : \mathcal{P}(\mathcal{M}) \to \mathcal{P}(\mathcal{M})$ is defined as

$$next(S) = \{\pi \in \mathcal{M} \mid \pi^1 \in S\}$$
$$[\![\mu Z.\phi]\!]^{\mathcal{M}}\rho = \bigcap\{S \subseteq \mathcal{M} \mid F(S) \subseteq S\}$$
$$[\![\nu Z.\phi]\!]^{\mathcal{M}}\rho = \bigcup\{S \subseteq \mathcal{M} \mid S \subseteq F(S)\},$$

where $F : \mathcal{P}(\mathcal{M}) \to \mathcal{P}(\mathcal{M})$ is defined as

$$F(S) = [\![\phi]\!]^{\mathcal{M}}(\rho + [Z \mapsto S])$$
$$[\![Z]\!]^{\mathcal{M}}\rho = \rho(Z)$$

Fig. 2. LMC semantics

Returning to Example 3, its consistent and complete interpretation, $\mathcal{I}$, reads:

$$\mathcal{I}(s_{2i}) = \{even, \neg odd, \neg halt\}, 0 \leq i \leq 49$$
$$\mathcal{I}(s_{2i+1}) = \{odd, \neg even, \neg halt\}, 0 \leq i \leq 49$$
$$\mathcal{I}(s_j) = \{even, \neg odd, halt\}, j \geq 100$$

We use the standard conversion rules to attempt translation of propositions into LMC syntax:

$$\neg(\phi_1 \vee \phi_2) \Rightarrow \neg\phi_1 \wedge \neg\phi_2 \qquad \neg(\mu Z.\phi) \Rightarrow \nu Z.\neg([\neg Z/Z]\phi)$$
$$\neg(\phi_1 \wedge \phi_2) \Rightarrow \neg\phi_1 \vee \neg\phi_2 \qquad \neg(\nu Z.\phi) \Rightarrow \mu Z.\neg([\neg Z/Z]\phi)$$
$$\neg(\mathsf{X}\phi) \Rightarrow \mathsf{X}\neg\phi \qquad \neg\neg\phi \Rightarrow \phi$$

For example, $\neg(\nu Z.\neg p \wedge \mathsf{X}Z) \Rightarrow^* \mu Z.p \vee \mathsf{X}Z$, but $\mu Z.Z \supset p \Rightarrow^* \mu Z.\neg Z \vee p$, which is not in LMC. With the semantics of negation in Figure 1, the soundness of the translation rules is argued in the usual way:

Proposition 5 *[16,45] If $\phi \Rightarrow^* \phi'$, and $\phi' \in LMC$, then for all $\mathcal{M}$ and ρ:*

(1) if $\mathcal{I} : AtomProp \rightarrow \mathcal{P}(\Sigma)$ is consistent, then $[\![\phi]\!]^{\mathcal{M}}_{\rho} \supseteq [\![\phi']\!]^{\mathcal{M}}_{\rho}$
(2) if $\mathcal{I}$ is consistent and complete, then $[\![\phi]\!]^{\mathcal{M}}_{\rho} = [\![\phi']\!]^{\mathcal{M}}_{\rho}$.

Note that the soundness of the translation rule for X depends on all traces being infinite.

2.3 Checking linear-time properties

Given a model, $\mathcal{M}$, and property, $\phi \in LMC$, we might check individual traces, $\pi \in \mathcal{M}$, to see if $\pi \models^{\mathcal{M}} \phi$ holds.[8]

More likely, we are interested in those traces in $\mathcal{M}$ that begin at some state, $s \in \Sigma$. We define

$$\mathcal{M} \downarrow s = \{\pi \in \mathcal{M} \mid \pi(0) = s\}$$

Two commonly used notions are

$$s \models^{\mathcal{M}} \forall\phi \text{ iff } \pi \models^{\mathcal{M}} \phi \text{ for all } \pi \in \mathcal{M} \downarrow s$$
$$s \models^{\mathcal{M}} \exists\phi \text{ iff there exists } \pi \in \mathcal{M} \downarrow s \text{ and } \pi \models^{\mathcal{M}} \phi$$

We say that ϕ *universally holds at* s if $s \models^{\mathcal{M}} \forall\phi$, and ϕ *existentially holds at* s if $s \models^{\mathcal{M}} \exists\phi$. Further, we have these forms of properties, for $\phi \in LMC$ [14]:

universal safety: $s \models^{\mathcal{M}} \forall\mathsf{G}\phi$

universal liveness: $s \models^{\mathcal{M}} \forall\mathsf{F}\phi$

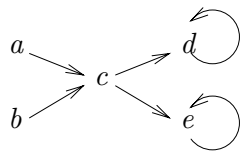
existential safety: $s \models^{\mathcal{M}} \exists\mathsf{G}\phi$

existential liveness: $s \models^{\mathcal{M}} \exists\mathsf{F}\phi$

[8] Of course, to make the check effective, we require a finite representation of π. This issue is addressed shortly in the guise of a (finite-)state-transition system that generates all traces in $\mathcal{M}$, including π. With this in hand, we can apply, for example, the tableau technique of Stirling and Walker [46] to systematically decompose ϕ and decide $\pi \models^{\mathcal{M}} \phi$.

If we plan to check universal or existential properties of a state, s, then we require a manageable (i.e., finite) representation of $\mathcal{M} \downarrow s$ and indeed, of $\mathcal{M}$ itself. The standard approach is to encode $\mathcal{M}$ as a *(finite) transition relation*, $\tau \subseteq \Sigma \times \Sigma$, such that $(s, s') \in \tau$ iff there is some $\pi \in \mathcal{M}$ such that $s = \pi(i)$ and $s' = \pi(i+1)$, for some $i \geq 0$. Let $wft(\tau)$ represent the set of all traces generated by τ.

Obviously, we desire, for all $s \in \Sigma$, that $\mathcal{M} \downarrow s = wft(\tau) \downarrow s$. *And we will assume this is so*, because it is easy to devise models where this is not the case: consider $\pi = acd^\omega$, $\pi' = bce^\omega$, and $\mathcal{M} = \{\pi^i, \pi'^i \mid i \geq 0\}$. Then τ is graphically depicted as



and we see that $ace^\omega \in wft(\tau)$, but this trace was not in $\mathcal{M}$.

When we wish to discuss the transition relation along with its state set and interpretation, we call it a *transition system*:

Definition 6 *A* (Kripke) transition system *is a tuple,* $\mathcal{K} = \langle \Sigma, \tau, \mathcal{I} \rangle$*, where*

- Σ *is a set of states*
- $\tau \subseteq \Sigma \times \Sigma$ *is a transition relation, and*
- $\mathcal{I} : \Sigma \to \mathcal{P}(AtomProp)$ *is an interpretation mapping.*

The model defined by $\mathcal{K}$ *is* $wft(\tau)$*.*

Transition systems are automata like, and it is not surprising that we can check universal and existential linear-time properties of states by applying automata-theoretic techniques [51]. Crudely stated, a transition system can be translated into an automaton whose language consists of $wft(\tau)$ annotated with atomic propositions. Next, a linear-time proposition, ϕ, can be translated into an automaton that accepts the language, $[\![\phi]\!]^{\mathcal{P}(\Sigma^\omega)}\emptyset$, where $\emptyset$ denotes the empty environment. Now, the task of checking, say, $s \models^{wft(\tau)} \forall\phi$, is performed as a language-inclusion check on the two automata: $wft(\tau) \downarrow s \subseteq [\![\phi]\!]^{\mathcal{P}(\Sigma^\omega)}\emptyset$. Simple introductions to this approach are found in Clarke, et al. [6] and Müller-Olm, et al. [34].

3 Abstracting the states within traces

Often, the task of formulating a finite-state transition system makes us replace the state set, Σ, with a significantly smaller— usually finite— set of *abstract states*, Σ_A, where every $s \in \Sigma$ is modelled (*abstracted*) by at least one $a \in \Sigma_A$. We write $s \mathcal{R} a$ to denote that state s is abstracted by a. Call Σ the *concrete state set*, Σ_A the *abstract state set*, and $\mathcal{R} \subseteq \Sigma \times \Sigma_A$ the *abstraction relation*.

Here are some examples:

Example 7 *For Example 3 seen earlier, we might abstract all the states to just these three: isEven, isOdd, isHalted; the abstraction relation is*

$$s_{2i} \mathcal{R} \, isEven, \;\; when \; 0 \leq i \leq 49$$
$$s_{2i+1} \mathcal{R} \, isOdd, \;\; when \; 0 \leq i \leq 49$$
$$s_j \mathcal{R} \, isHalted, \;\; when \; j \geq 100$$

A simpler abstraction uses just two states, isE and isO, and the relation, $s_{2i} \mathcal{R} \, isE$, $s_{2i+1} \mathcal{R} \, isO$, for $i \geq 0$.

A trivial abstraction of Example 3 would use just one state, a_0, such that $s_i \mathcal{R} \, a_0$, for all $i \geq 0$.

Example 8 *A program that manipulates variables* `x` *and* `y` *might use states of the form, $\langle p, v_x, v_y \rangle$, where p is the instruction counter and v_x and v_y are the respective values of the variables. The states might be abstracted to merely their counters: $\langle p, v_x, v_y \rangle \mathcal{R} \; p$. The abstract states monitor the program's control flow.*

Example 9 *A program that manipulates variable* `x` *uses states of form, $\langle p, v_x \rangle$, as above. The states are abstracted to $\langle p, u \rangle$, where $u \in \{odd, even, either\}$, and*

$$\begin{aligned} &\langle p, v_x \rangle \mathcal{R} \langle p, u \rangle \; iff \\ &\quad v_x mod 2 = 1 \; implies \; u \in \{odd, either\} \\ &and \quad v_x mod 2 = 0 \; implies \; u \in \{even, either\} \end{aligned}$$

The abstract states monitor the polarity of variable `x`*, with loss of precision in execution of statements like p_k:* `x := x div2`*, where the successor state to $\langle p_k, u \rangle$ would be $\langle p_{k+1}, either \rangle$.*

Example 10 *The system of Example 4, consisting of a shared file, n-readers, and one writer, can be abstracted to a system whose states are* $\{readOnly, writeOnly, atRest, readWrite\}$*, where*

$$\langle \cdots, read, \cdots, wait \rangle \; \mathcal{R} \; readOnly$$

$$\langle sleep, sleep, \cdots, sleep, write \rangle \; \mathcal{R} \; writeOnly$$

$$\langle sleep, sleep, \cdots, sleep, wait \rangle \; \mathcal{R} \; atRest$$

$$\langle \cdots, read, \cdots, write \rangle \; \mathcal{R} \; readWrite$$

The abstract states monitor the mode of file usage.

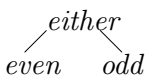
An abstract state, $a \in \Sigma_A$, acts as a representative for the concrete states that it abstracts. Based on the abstraction relation, $\mathcal{R} \subseteq \Sigma \times \Sigma_A$, we define a's *concretization* by this *concretization function*, $\gamma : \Sigma_A \to \mathcal{P}(\Sigma)$:

$$\gamma(a) = \{s \in \Sigma \mid s \; \mathcal{R} \; a\}$$

For Example 7, we see that $\gamma(isOdd) = \{s_{2i+1} \mid 0 \leq i \leq 49\}$.

Example 9 shows that a state in Σ can be abstracted by multiple states in Σ_A. In such a case, it is common to *partially order* Σ_A, where for $a, a' \in \Sigma_A$, $a \sqsubseteq_A a'$ if a conveys more precise information than does a'.

For Example 9, we see that a reasonable ordering for Σ_A is $\langle p, u \rangle \sqsubseteq_A \langle p', u' \rangle$ iff $p = p'$ and u falls below u' in this Hasse diagram:



A "reasonable" partial ordering for Σ_A should make the following two properties hold true:

Definition 11 *[38,40] For set P and partially ordered set Q, a binary relation, $\mathcal{R} \subseteq P \times Q$, is*

- U-closed *("upper closed") iff for all $p \in P$, $q \in Q$, $p \; \mathcal{R} \; q$ and $q \sqsubseteq_Q q'$ imply $p \; \mathcal{R} \; q'$*
- G-closed *("greatest-lower-bound closed") iff for all $p \in P$, $p \; \mathcal{R} \; (\sqcap\{q \mid p \; \mathcal{R} \; q\})$*

The U-closure property is equivalent to the statement, $a \sqsubseteq_A a'$ implies $\gamma(a) \subseteq \gamma(a')$, meaning that the concretization function γ is monotonic. The G-closure

property implies that each $s \in \Sigma$ has a most precise abstraction, $\beta(s) = \sqcap\{a \mid s \mathcal{R} a\}$, where the meet, $\sqcap$, is guaranteed to be defined.

When Σ_A is ordered as a complete lattice [9], even stronger results follow:

Theorem 12 *[40] For set P, complete lattice $\langle Q, \sqsubseteq_Q \rangle$, and UG-closed relation, $\mathcal{R} \subseteq P \times Q$, define $\gamma_{\mathcal{R}} : Q \to \mathcal{P}(P)$ and $\alpha_{\mathcal{R}} : \mathcal{P}(P) \to Q$ as follows:*

$$\gamma_{\mathcal{R}}(a) = \{s \mid s \mathcal{R} a\}$$
$$\alpha_{\mathcal{R}}(S) = \sqcup_{s \in S} \beta_{\mathcal{R}}(s), \text{ where } \beta_{\mathcal{R}}(s) = \sqcap\{a \mid s \mathcal{R} a\}$$

Then,

(1) $\alpha_{\mathcal{R}}$ and $\gamma_{\mathcal{R}}$ are monotone with respect to $\langle \mathcal{P}(P), \subseteq \rangle$ and $\langle Q, \sqsubseteq_Q \rangle$;
(2) $S \subseteq (\gamma_{\mathcal{R}} \circ \alpha_{\mathcal{R}})(S)$, for all $S \subseteq P$;
(3) $(\alpha_{\mathcal{R}} \circ \gamma_{\mathcal{R}})(a) \sqsubseteq_Q a$, for all $a \in Q$.

That is, $(\alpha_{\mathcal{R}}, \gamma_{\mathcal{R}})$ form a Galois connection *[9,31] for the complete lattices, $\langle \mathcal{P}(P), \subseteq \rangle$ and $\langle Q, \sqsubseteq_Q \rangle$.*

The mapping, $\alpha_{\mathcal{R}}$, defined above, generalizes $\beta_{\mathcal{R}}$ so that every set of concrete states has a best approximation as an abstract state—this is a crucial property for generating traces directly upon abstract states. [10]

Recall that a *Galois connection* is a pair of monotone mappings, $(\alpha : X \to Y, \gamma : Y \to X)$, for complete lattices X and Y, such that $id_X \sqsubseteq \gamma \circ \alpha$ and $\alpha \circ \gamma \sqsubseteq id_Y$ under the usual pointwise ordering of functions. The two mappings are "weak inverses" [11] that possess a wealth of useful properties [9,10,31]; we will enumerate some of these properties the narrative that follows.

3.1 Atomic properties of abstract states

Recall that the concrete state set, Σ, is accompanied by an interpretation, $\mathcal{I} : \Sigma \to \mathcal{P}(AtomProp)$, which maps states to the atomic properties that hold true for it. When Σ is abstracted to Σ_A, then $\mathcal{I}$ must be abstracted to some $\mathcal{I}_A : \Sigma_A \to \mathcal{P}(AtomProp)$ as well. There are two ways to abstract $\mathcal{I}$; the first is the standard one:

[9] If Σ_A is an unordered set, we can make a complete lattice from it by adding extra elements, $\bot$ and $\top$, such that $\bot \sqsubseteq a$ and $a \sqsubseteq \top$, for all $a \in \Sigma_A \cup \{\bot, \top\}$. We set $a \mathcal{R} \top$, for all $s \in \Sigma$; no s is abstracted by $\bot$.
[10] Note that $\beta_{\mathcal{R}}(s) = \alpha_{\mathcal{R}}\{s\}$.
[11] More precisely, the images of α and γ are order-isomorphic [31].

Definition 13 *For an abstraction relation, $\mathcal{R} \subseteq \Sigma \times \Sigma_A$, and interpretations $\mathcal{I}_A : \Sigma_A \to \mathcal{P}(AtomProp)$ and $\mathcal{I} : \Sigma \to \mathcal{P}(AtomProp)$, $\mathcal{R}$* reflects atomic properties *iff*

$$s \,\mathcal{R}\, a \text{ implies } \mathcal{I}(s) \supseteq \mathcal{I}_A(a)$$

(Equivalently, a concretization map, $\gamma : \Sigma_A \to \mathcal{P}(\Sigma)$, reflects atomic properties if $\mathcal{I}(s) \supseteq \mathcal{I}_A(a)$ where $s \in \gamma(a)$.)

Reflection sets the stage for checking temporal-logic properties of abstract states (and traces)—when p holds true for a and $a \,\mathcal{R}\, s$, then p holds true for s as well.

The best definition for $\mathcal{I}_A$ that makes $\mathcal{R}$ reflect atomic properties goes, $\mathcal{I}_A(a) = \bigcap_{s|s \,\mathcal{R}\, a} \mathcal{I}(s)$. (Equivalently, replace $s \in \gamma(a)$ for $s \,\mathcal{R}\, a$.)

Returning to Example 7, when the abstract state set is $\{isEven, isOdd, isHalted\}$, then we should define [12]

$$\mathcal{I}_A(isEven) = \{even, \neg odd, \neg halt\}$$
$$\mathcal{I}_A(isOdd) = \{odd, \neg even, \neg halt\}$$
$$\mathcal{I}_A(isHalted) = \{even, halt, \neg odd\}$$

and there is no loss of atomic properties in the state abstraction. Of course, precision can be lost; in Example 7, when we use the abstract state set $\{isE, isO\}$, then the best definition of $\mathcal{I}_A$ we can devise is merely

$$\mathcal{I}_A(isE) = \{even, \neg odd\}$$
$$\mathcal{I}_A(isO) = \{\}$$

because both halting and nonhalting concrete states are abstracted by the two abstract states. In particular, $halt \notin \mathcal{I}_A(isE)$ does *not* assert that all the states represented by $isEven$ are nonhalting—rather, we lack precision to decide this property.

When Σ_A is partially ordered, we expect that the ordering reflects information content: when $s \,\mathcal{R}\, a$, then $a \sqsubseteq_A a'$ should imply $\mathcal{I}_A(a) \supseteq \mathcal{I}_A(a')$. This property indeed holds when the abstraction relation, $\mathcal{R}$, is U-closed and $\mathcal{I}_A$ is defined as stated above.

There is a second way of abstracting $\mathcal{I}$—the dual of the above:

[12] We continue to assume that atomic properties have the form, p or $\neg p$.

Definition 14 *For an abstraction relation, $\mathcal{R} \subseteq \Sigma \times \Sigma_A$, and interpretations $\mathcal{I}_A : \Sigma_A \to \mathcal{P}(AtomProp)$ and $\mathcal{I} : \Sigma \to \mathcal{P}(AtomProp)$,* $\mathcal{R}$ preserves atomic properties *iff*

$$s \; \mathcal{R} \; a \text{ implies } \mathcal{I}(s) \subseteq \mathcal{I}_A(a)$$

(Equivalently, concretization map, $\gamma : \Sigma_A \to \mathcal{P}(\Sigma)$ preserves atomic properties if $\mathcal{I}(s) \subseteq \mathcal{I}_A(a)$ where $s \in \gamma(a)$.)

Preservation helps one *refute* temporal-logic properties: when p fails to hold for a and $s \; \mathcal{R} \; a$, then p fails to hold for s as well. The best definition of $\mathcal{I}_A$ that gives preservation of $\mathcal{I}$ goes, $\mathcal{I}_A(a) = \bigcup_{s|s \; \mathcal{R} \; a} \mathcal{I}(s)$.

3.2 Lifting a state abstraction to a trace abstraction

Now that Σ_A and $\mathcal{I}_A$ are fixed, we focus on the traces one generates from them: From Definition 1, we define $\mathcal{A} = Nat \to \Sigma_A = \Sigma_A^\omega$ as the set of abstract traces. We relate abstract traces to concrete ones by lifting the abstraction relation, $\mathcal{R} \subseteq \Sigma \times \Sigma_A$, to $\mathcal{R}^\omega \subseteq \mathcal{M} \times \mathcal{A}$; let $\kappa \in \mathcal{A}$ be an abstract trace:

$$\pi \; \mathcal{R}^\omega \; \kappa \text{ iff for all } i \geq 0, \pi(i) \; \mathcal{R} \; \kappa(i)$$

Given $\gamma(a) = \{s \in \Sigma \mid s \; \mathcal{R} \; a\}$, the corresponding concretization map is $\gamma^{Trace} : \mathcal{A} \to \mathcal{P}(\mathcal{M})$:

$$\gamma^{Trace}(\kappa) = \{\pi \in \mathcal{M} \mid \pi(i) \in \gamma(\kappa(i)), \text{ for all } i \geq 0\}$$

If the abstraction relation on states is formalized by a Galois connection, $\langle \alpha : \langle \mathcal{P}(\Sigma), \subseteq \rangle \to \langle \Sigma_A, \sqsubseteq_A \rangle, \; \gamma : \langle \Sigma_A, \sqsubseteq_A \rangle \to \langle \mathcal{P}(\Sigma, \subseteq) \rangle \rangle$, we can lift the Galois connection into one that defines which how concrete traces are abstracted by abstract traces. First, let $\sqsubseteq$ represent the pointwise ordering on traces in $\mathcal{A}$.[13]

$$\begin{aligned} &\langle \alpha^{Trace} : \langle \mathcal{P}(\mathcal{M}), \subseteq \rangle \to \langle \mathcal{A}, \sqsubseteq \rangle, \; \gamma^{Trace} : \langle \mathcal{A}, \sqsubseteq \rangle \to \langle \mathcal{P}(\mathcal{M}, \subseteq) \rangle \rangle \\ &\quad \gamma^{Trace}(\kappa) = \{\pi \in \mathcal{M} \mid \pi(i) \in \gamma(\kappa(i)), \text{ for all } i \geq 0\} \\ &\quad \alpha^{Trace}(S) = \sqcup \{\lambda i. \alpha\{\pi(i)\} \mid \pi \in S\} \end{aligned}$$

As before, $\gamma^{Trace}(\kappa)$ maps an abstract trace to all the concrete traces it represents. The dual map, α^{Trace}, synthesizes one abstract trace from a set of

[13] For $\kappa, \kappa' \in \mathcal{A}$, $\kappa \sqsubseteq \kappa'$ iff for all $i \geq 0$, $\kappa(i) \sqsubseteq_A \kappa'(i)$.

concrete traces by, first, building an abstract trace for each concrete trace and, second, joining the abstract traces by pointwise join.

Here are some examples: Recall model $\mathcal{M}$ from Example 3 and its three-state abstraction in Example 7. For these three example abstract traces, we calculate that

$$\gamma^{Trace}(isEven\ isOdd\ isHalted^{\omega}) = \{s_{98}s_{99}s_{100}\cdots\}$$
$$\gamma^{Trace}(isHalted^{\omega}) = \{s_i s_{i+1}\cdots \mid i \geq 100\}$$
$$\gamma^{Trace}(isEven^{\omega}) = \{\}$$

4 Checking properties of abstract traces

Given a trace, $\kappa \in \mathcal{A}$, we wish to check whether linear-time properties, ϕ, hold for κ, and we wish to use the outcomes to deduce properties of the concrete traces that κ represents. Assume that the abstraction relation, $\mathcal{R} \subseteq \Sigma \times \Sigma_A$, reflects atomic properties. By Definition 13, $p \in \mathcal{I}_A(a)$ implies $p \in \mathcal{I}(s)$, when $s \in \gamma(a)$ (equivalently, when $s\ \mathcal{R}\ a$). We wish to generalize this result to propositions in LMC:

$$\kappa \in [\![\phi]\!]^{\mathcal{A}}\emptyset \text{ implies } \gamma^{Trace}(\kappa) \subseteq [\![\phi]\!]^{\mathcal{M}}\emptyset$$

where $[\![\cdot]\!]^{\mathcal{A}} : AbsEnv \to \mathcal{P}(\mathcal{A})$, represents the *abstract interpretation* of the concrete linear-time semantics, $[\![\cdot]\!]^{\mathcal{M}} : Env \to \mathcal{P}(\Sigma^{\omega})$; $AbsEnv = Var \to \mathcal{P}(\mathcal{A})$, is the abstracted variable environment; and $\emptyset$ represents an empty environment. The inclusion states, when $\kappa \models^{\mathcal{A}} \phi$ holds, then it must be that $\pi \models^{\mathcal{M}} \phi$ holds, if $\pi\ \mathcal{R}^{\omega}\ \kappa$.

Ideally, we would like to strengthen the implication in the above assertion into an equivalence, that is,

$$[\![\phi]\!]^{\mathcal{A}}\emptyset = \{\kappa \mid \gamma^{Trace}(\kappa) \subseteq [\![\phi]\!]^{\mathcal{M}}\emptyset\}$$

This result makes the abstract interpretation, $[\![\cdot]\!]^{\mathcal{A}}$, as precise as possible regarding the abstract traces in $\mathcal{A}$ and concretization map γ^{Trace}.

Note that a precise-as-possible abstract interpretation is not the same as an equal one: for an $[\![\cdot]\!]^{\mathcal{A}}$ that satisfies the above equality, it may still be the case that $\pi\ \mathcal{R}^{\omega}\ \kappa$ and $\pi \models^{\mathcal{M}} \phi$, yet $\kappa \not\models^{\mathcal{A}} \phi$; the loss of precision due to use of abstract states leads to loss of precision in deciding linear-time properties of traces built from the abstract states.

Although it would be convenient to *define* $[\![\cdot]\!]^{\mathcal{A}}$ by the above equation, this is not pragmatic, because it makes calculating $[\![\cdot]\!]^{\mathcal{A}}$ depend on calculating $[\![\cdot]\!]^{\mathcal{M}}$, which is what we are trying to avoid. Instead, we must define $[\![\cdot]\!]^{\mathcal{A}}$ inductively on the syntax of LMC (see Figure 2), that is, for each syntax constructor, $\mathsf{op}\phi$, in LMC, we wish to define an abstract operation, op_A, so that

$$[\![\mathsf{op}\phi]\!]^{\mathcal{A}}\rho = op_A[\![\phi]\!]^{\mathcal{A}}\rho$$

for $op_A : \mathcal{P}(\mathcal{A}) \to \mathcal{P}(\mathcal{A})$. Figure 2 shows there are three such constructors in LMC that need such abstract operations: $\mathsf{X}\phi$, $\phi_1 \vee \phi_2$, and $\phi_1 \wedge \phi_2$. (The semantics of p, $\neg p$, $\mu Z.\phi$, $\nu Z.\phi$, and Z will be fixed in the format as they appear in Figure 2.)

Some properties from Galois connections can help us derive appropriate op_As.

4.1 Soundness and completeness of abstract operations

Definition 15 *Let* $(\alpha : X \to Y,\ \gamma : Y \to X)$ *be a Galois connection and* $op_C : X \to X$ *a function. A function,* $op_A : Y \to Y$*, is* sound with respect to op_C *iff* $\alpha \circ op_C \sqsubseteq op_A \circ \alpha$.[14] op_A *is* complete with respect to op_C *iff* $\alpha \circ op_C = op_A \circ \alpha$.

An op_A that is sound with respect to op_C approximates the latter's behavior on arguments from Y. A complete op_A mimics op_C so that the latter is indistinguishable from the former (when mapped through the Galois connection).

Proposition 16 *Given* $(\alpha : X \to Y,\ \gamma : Y \to X)$ *and* $op_C : X \to X$*, the most precise* op_A *that is sound with respect to* op_C *is* $op_A = \alpha \circ op_C \circ \gamma$.

The Proposition suggests that we should begin our search for sound op_As by formulating $\alpha \circ op_C \circ \gamma$.

We must define the abstract counterparts for *next*, *and*, and *or* in Figure 2. But these operations have arity $\mathcal{P}(\mathcal{M}) \to \mathcal{P}(\mathcal{M})$, implying that we require a Galois connection between the two sets $\mathcal{P}(\mathcal{M})$ and $\mathcal{P}(\mathcal{A})$. Alas, we have only the Galois connection that relates individual abstract traces to concrete sets of traces: $\langle \alpha^{Trace} : \mathcal{P}(\mathcal{M}) \to \mathcal{A},\ \gamma^{Trace} : \mathcal{A} \to \mathcal{P}(\mathcal{M}) \rangle$. The following proposition repairs the situation.

[14] As usual, $\sqsubseteq$ represents the pointwise ordering on functions. Also, Giacobazzi and Quintarelli [21] call the soundness property *forwards soundness*. There is also backwards soundness, $op_C \circ \gamma \sqsubseteq \gamma \circ op_A$, which is a nonequivalent notion.

Proposition 17 *Given a concretization function, $g : Y \to \mathcal{P}(X)$ (or, an abstraction relation, $\mathcal{R} \subseteq X \times Y$, such that $g(a) = \{s \mid s \mathcal{R} a\}$), the functions,*

$$\begin{aligned}
&\gamma_g : \mathcal{P}(Y) \to \mathcal{P}(X) \\
&\quad \gamma_g(T) = \{s \in X \mid s \in g(a), a \in T\} \\
&\alpha_g : \mathcal{P}(X) \to \mathcal{P}(Y) \\
&\quad \alpha_g(S) = \{a \in Y \mid g(a) \subseteq S\}
\end{aligned}$$

form the Galois connection,

$$\langle \alpha_g : \langle \mathcal{P}(X), \supseteq \rangle \to \langle \mathcal{P}(Y), \supseteq \rangle,\ \gamma_g : \langle \mathcal{P}(Y), \supseteq \rangle, \to \langle \mathcal{P}(X), \supseteq \rangle \rangle$$

γ_g merely unions the g-images of all elements in its argument. α_g behaves like g^{-1}, where each image set must fall within the argument set.

Proposition 17 is surprising because it merely requires that g be a function (and not half of a Galois connection). Also, note that the orderings on the sets of the powersets are reversed. We have this simple, useful Corollary:

Corollary 18 *For model $\mathcal{M} \subseteq \mathcal{P}(\Sigma^\omega)$ and concretization $\gamma^{Trace} : \mathcal{A} \to \mathcal{P}(\mathcal{M})$,*

$$\begin{aligned}
&\langle \alpha^{TrSet} : \langle \mathcal{P}(\mathcal{M}), \supseteq \rangle \to \langle \mathcal{P}(\mathcal{A}), \supseteq \rangle,\ \gamma^{TrSet} : \langle \mathcal{P}(\mathcal{A}), \supseteq \rangle, \to \langle \mathcal{P}(\mathcal{M}), \supseteq \rangle \rangle \\
&\quad \gamma^{TrSet}(T) = \{\pi \in \mathcal{M} \mid \pi \in \gamma^{Trace}(a), a \in T\} \\
&\quad \alpha^{TrSet}(S) = \{a \in \mathcal{A} \mid \gamma^{Trace}(a) \subseteq S\}
\end{aligned}$$

is a Galois connection.

Because the orderings on the powersets are reversed, soundness is correctly expressed as $\alpha^{TrSet} \circ op_C \supseteq op_A \circ \alpha^{TrSet}$, and completeness is the equality, $\alpha^{TrSet} \circ op_C = op_A \circ \gamma^{TrSet}$.

Using the Corollary, we can formulate the abstract semantics of Figure 2; see Figure 3.

The soundness of the abstract operators leads to a proof of soundness for $[\![\cdot]\!]^{\mathcal{A}}$: First, for $\rho \in Env = Var \to \mathcal{P}(\mathcal{A})$, define the environment's abstraction as $\alpha\rho = \lambda Z \in Var.\ \alpha^{TrSet}(\rho(Z))$. We have this result:

Theorem 19 *[12] Assume that γ reflects atomic properties (Definition 13).*

$\rho \in AbsEnv = Var \to \mathcal{P}(\mathcal{A})$
$[\![\phi]\!]^{\mathcal{A}} \in AbsEnv \to \mathcal{P}(\mathcal{A})$

$$
\begin{aligned}
&[\![p]\!]^{\mathcal{A}}\rho = \{\kappa \in \mathcal{A} \mid p \in \mathcal{I}^A(\kappa(0))\} \\
&[\![\neg p]\!]^{\mathcal{A}}\rho = \{\kappa \in \mathcal{A} \mid \neg p \in \mathcal{I}^A(\kappa(0))\} \\
&[\![\phi_1 \wedge \phi_2]\!]^{\mathcal{A}}\rho = and_A([\![\phi_1]\!]^{\mathcal{A}}\rho, [\![\phi_2]\!]^{\mathcal{A}}\rho) \\
&\quad \text{where } and_A(S_1, S_2) = \alpha^{TrSet}(\gamma^{TrSet}(S_1) \cap \gamma^{TrSet}(S_2)) \\
&[\![\phi_1 \vee \phi_2]\!]^{\mathcal{A}}\rho = or_A([\![\phi_1]\!]^{\mathcal{A}}\rho, [\![\phi_2]\!]^{\mathcal{A}}\rho) \\
&\quad \text{where } or_A(S_1, S_2) = \alpha^{TrSet}(\gamma^{TrSet}(S_1) \cup \gamma^{TrSet}(S_2)) \\
&[\![\mathsf{X}\phi]\!]^{\mathcal{A}}\rho = next_A([\![\phi]\!]^{\mathcal{A}}\rho) \\
&\quad \text{where } next_A(S) = \alpha^{TrSet}(next(\gamma^{TrSet}(S))) \\
&[\![\mu Z.\phi]\!]^{\mathcal{A}}\rho = \bigcap\{S \subseteq \mathcal{A} \mid F(S) \subseteq S\} \\
&[\![\nu Z.\phi]\!]^{\mathcal{A}}\rho = \bigcup\{S \subseteq \mathcal{A} \mid S \subseteq F(S)\}, \\
&\quad \text{where } F(S) = [\![\phi]\!]^{\mathcal{A}}(\rho + [Z \mapsto S]) \\
&[\![Z]\!]^{\mathcal{A}}\rho = \rho(Z)
\end{aligned}
$$

Fig. 3. Abstracted LMC semantics

For all $\phi \in LMC$, $\rho \in Env$,

$$[\![\phi]\!]^{\mathcal{A}}(\alpha\rho) \subseteq \alpha^{TrSet}[\![\phi]\!]^{\mathcal{M}}\rho = \{\kappa \mid \gamma^{Trace}(\kappa) \subseteq [\![\phi]\!]^{\mathcal{M}}\rho\}$$

Further, when the abstract operations, op_A, are complete with respect to the concrete operations, op_C, then the first $\subseteq$ becomes $=$.

PROOF. The proof is by induction on the structure of ϕ and is a simple variation of a similar result by Cousot and Cousot [12].

The cases for p and $\neg p$ follow from the reflection property for $\mathcal{I}_A$. When $\mathcal{I}_A(a) = \bigcap_{s|s\in\gamma(a)} \mathcal{I}(s)$, we have completeness:

$$
\begin{aligned}
&[\![p]\!]^{\mathcal{A}}\alpha\rho = \{\kappa \mid p \in \mathcal{I}_A(\kappa(0))\} \\
&= \{\kappa \mid p \in \textstyle\bigcap_{s|s\in\gamma(a)} \mathcal{I}(\pi(0))\} \\
&= \{\kappa \mid p \in \mathcal{I}(\pi(0)), \text{ for all } \pi \in \gamma^{Trace}(\kappa)\} \\
&= \{\kappa \mid \gamma^{Trace}(\kappa) \subseteq \{\pi \mid p \in \mathcal{I}(\pi(0))\}\} \\
&= \alpha^{TrSet}\{\pi \mid p \in \mathcal{I}(\pi(0))\} = \alpha^{TrSet}[\![p]\!]^{\mathcal{M}}\rho
\end{aligned}
$$

The cases for Z, $\mu Z.\phi$, and $\nu Z.\phi$ are proved in [12].

The cases for abstract operators, $op_A : \mathcal{A} \to \mathcal{A}$, proceed as follows:

$$
\begin{aligned}
&[\![\mathsf{op}\phi]\!]^{\mathcal{A}}\alpha\rho = (\alpha^{TrSet} \circ op_C \circ \gamma^{TrSet})[\![\phi]\!]^{A}\alpha\rho \\
&\subseteq (\alpha^{TrSet} \circ op_C \circ \gamma^{TrSet})\{\kappa \mid \gamma^{Trace}(\kappa) \subseteq [\![\phi]\!]^{\mathcal{M}}\rho\}, \text{ by the inductive hypothesis} \\
&\quad \text{(when completeness holds for } \phi\text{, then } \subseteq \text{ becomes } =) \\
&= (\alpha^{TrSet} \circ op_C \circ \gamma^{TrSet} \circ \alpha^{TrSet})[\![\phi]\!]^{\mathcal{M}}\rho \\
&\subseteq (\alpha^{TrSet} \circ op_C)[\![\phi]\!]^{\mathcal{M}}\rho, \text{ by soundness of } \alpha^{TrSet} \circ op_C \circ \gamma^{TrSet} \\
&\quad \text{(when completeness holds, then } \subseteq \text{ becomes } =) \\
&= \alpha^{TrSet}[\![\mathsf{op}\phi]\!]^{\mathcal{M}}\rho
\end{aligned}
$$

Corollary 20 $\gamma^{TrSet}[\![\phi]\!]^{\mathcal{A}}(\alpha\rho) \subseteq [\![\phi]\!]^{\mathcal{M}}\rho$.

PROOF. By the previous theorem, $[\![\phi]\!]^{\mathcal{A}}\alpha\rho \subseteq \alpha^{TrSet}[\![\phi]\!]^{\mathcal{M}}\rho$, and by monotonicity, $\gamma^{TrSet}[\![\phi]\!]^{\mathcal{A}}\alpha\rho \subseteq (\gamma^{TrSet} \circ \alpha^{TrSet})[\![\phi]\!]^{\mathcal{M}}\rho$. By definition of Galois connection, $(\gamma^{TrSet} \circ \alpha^{TrSet})[\![\phi]\!]^{\mathcal{M}}\rho \subseteq [\![\phi]\!]^{\mathcal{M}}\rho$, implying the result.

Note that completeness does *not* imply equality in the preceding Corollary—the results are always "filtered" through the Galois connection. To see this, consider the model $\mathcal{M} = \{s_0^\omega, s_1^\omega\}$ where $\mathcal{I}(s_0) = \{p\}$, $\mathcal{I}(s_1) = \{q\}$. Say that we employ the trivial abstraction, where $\Sigma_A = \{a\}$, so $\mathcal{A} = \{a^\omega\}$. There is a Galois connection between $\mathcal{P}(\Sigma)$ and Σ_A, where $\gamma_0(a) = \{s_0, s_1\}$. This forces $\mathcal{I}_A(a) = \{\}$. It is easy to check that $[\![p]\!]^{\mathcal{M}}\emptyset = \{s_0^\omega\} \neq \{\} = \gamma^{TrSet}[\![p]\!]^{\mathcal{A}}\emptyset$, where $[\![p]\!]^{\mathcal{A}}\emptyset = \{\kappa \mid \gamma^{Trace}(\kappa) \subseteq [\![p]\!]^{\mathcal{M}}\emptyset\}$.

4.2 Checking abstract transition relations

Despite our substantial efforts at moving from the world of concrete states, Σ, to abstract states, Σ_A, we have done little to formalize the set of abstract traces, $\mathcal{A}$, themselves, merely assuming that $\mathcal{A} = {\Sigma_A}^\omega$.

In practice, we must devise a finite representation of $\mathcal{A}$; this usually takes the form of a finite transition relation, $\tau_A \subseteq \Sigma_A \times \Sigma_A$, which we use to define an abstract transition system, $\langle \Sigma_A, \tau_A, \mathcal{I}_A \rangle$. The model of the abstract transition system is $\mathcal{A} = wft(\tau_A)$. Now we must define abstract operations, $op_A : \mathcal{A} \to \mathcal{A}$, such that soundness holds, that is, $\alpha^{TrSet} \circ op \circ \gamma^{TrSet} \sqsubseteq op_A$.

One would expect that τ_A must be derived from $\tau \subseteq \Sigma \times \Sigma$, the transition relation for concrete states. But we will see the surprising result that soundness of the standard definitions for the abstract operations, $next_A$, and_A, and or_A, are unaffected by the choice of abstract transition relation—*any* τ_A will suffice for abstracting τ soundly.

We begin with the technical results and follow with examples. Let $\tau_A \subseteq \Sigma_A \times \Sigma_A$ be an arbitrary transition relation. We write $a \to a' \in \tau_A$ to denote that $(a, a') \in \tau_A$. Here are the definitions of the three abstract operations we use to define $[\![\cdot]\!]^{\mathcal{A}}$:

Definition 21 *For $\tau_A \subseteq \Sigma_A \times \Sigma_A$, define $\mathcal{A} = wft(\tau_A)$, and define $next_{A'} : \mathcal{P}(\mathcal{A}) \to \mathcal{P}(\mathcal{A})$, $and_{A'} : \mathcal{P}(\mathcal{A}) \times \mathcal{P}(\mathcal{A}) \to \mathcal{P}(\mathcal{A})$, $or_{A'} : \mathcal{P}(\mathcal{A}) \times \mathcal{P}(\mathcal{A}) \to \mathcal{P}(\mathcal{A})$, as follows:*

$$next_{A'}(T) = \{\kappa \in \mathcal{A} \mid \kappa(0) \to \kappa(1) \in \tau_A,\ \kappa^1 \in T\}$$
$$and_{A'}(T_1, T_2) = T_1 \cap T_2$$
$$or_{A'}(T_1, T_2) = T_1 \cup T_2$$

Figure 4 documents the semantics with the abstract operations commonly used. Alas, these operations are not guaranteed to be complete [21,35], but we can readily prove their soundness:

Theorem 22 *Given the Galois connection, $\langle \alpha^{TrSet} : \mathcal{P}(wft(\tau)) \to \mathcal{P}(wft(\tau_A)), \gamma^{TrSet} : \mathcal{P}(wft(\tau_A)) \to \mathcal{P}(wft(\tau))\rangle$ from Corollary 18, the operations in Definition 21 are sound with respect to their concrete counterparts in Figure 2.*

Hence, Theorem 19 holds for the abstract semantics of Figure 4.

PROOF. For operation $next_{A'}$, consider some $\kappa' \in next_{A'}(\alpha^{TrSet}(S))$; it must have form, $a\kappa$, for some $a \in \Sigma_A$, $\kappa \in \alpha^{TrSet}(S)$, and $a \to \kappa(0) \in \tau_A$.

To show that $a\kappa$ is contained in $(\alpha^{TrSet} \circ next)(S)$, consider an arbitrary $s\pi \in wft(\tau)$ such that $s\pi\ \mathcal{R}^\omega\ a\kappa$. Of course, $\pi\ \mathcal{R}^\omega\ \kappa$, and by hypothesis, all π' such that $\pi'\ \mathcal{R}^\omega\ \kappa$ are in S. Also, $s\ \mathcal{R}\ a$ holds, and $s \to \pi(0) \in \tau$. This places $s\pi \in next(S)$. Hence, the image of $\gamma^{Trace}(a\kappa) \subseteq next(S)$, and $a\kappa \in \alpha^{TrSet}(next(S))$.

For $or_{A'}$, $\kappa \in \alpha^{TrSet}(S_1) \cup \alpha^{TrSet}(S_2)$ iff $\gamma^{TrSet}(\kappa) \subseteq S_1$ or $\gamma^{TrSet}(\kappa) \subseteq S_2$. This implies $\gamma^{TrSet}(\kappa) \subseteq S_1 \cup S_2$.

The proof for $and_{A'}$ is equally simple and shows an equality.

$\rho \in AbsEnv = Var \to \mathcal{P}(\mathcal{A})$
$[\![\phi]\!]^{\mathcal{A}} \in AbsEnv \to \mathcal{P}(\mathcal{A})$

$$[\![p]\!]^{\mathcal{A}}\rho = \{\kappa \in \mathcal{A} \mid p \in \mathcal{I}^A(\kappa(0))\}$$
$$[\![\neg p]\!]^{\mathcal{A}}\rho = \{\kappa \in \mathcal{A} \mid \neg p \in \mathcal{I}^A(\kappa(0))\}$$
$$[\![\phi_1 \wedge \phi_2]\!]^{\mathcal{A}}\rho = and_{A'}([\![\phi_1]\!]^{\mathcal{A}}\rho, [\![\phi_2]\!]^{\mathcal{A}}\rho)$$
where $and_{A'}(S_1, S_2) = S_1 \cap S_2$
$$[\![\phi_1 \vee \phi_2]\!]^{\mathcal{A}}\rho = or_{A'}([\![\phi_1]\!]^{\mathcal{A}}\rho, [\![\phi_2]\!]^{\mathcal{A}}\rho)$$
where $or_{A'}(S_1, S_2) = S_1 \cup S_2$
$$[\![\mathsf{X}\phi]\!]^{\mathcal{A}}\rho = next_{A'}([\![\phi]\!]^{\mathcal{A}}\rho)$$
where $next_{A'}(T) = \{\kappa \in \mathcal{A} \mid \kappa(0) \to \kappa(1),\ \kappa^1 \in T\}$
$$[\![\mu Z.\phi]\!]^{\mathcal{A}}\rho = \bigcap\{S \subseteq \mathcal{A} \mid F(S) \subseteq S\}$$
$$[\![\nu Z.\phi]\!]^{\mathcal{A}}\rho = \bigcup\{S \subseteq \mathcal{A} \mid S \subseteq F(S)\},$$
where $F(S) = [\![\phi]\!]^{\mathcal{A}}(\rho + [Z \mapsto S])$
$$[\![Z]\!]^{\mathcal{A}}\rho = \rho(Z)$$

Fig. 4. Abstracted LMC semantics with finitely computable operations

Because the proof of Theorem 22 does not refer to the structure of τ_A, the result holds for any τ_A. This happens because the concretization map, $\gamma^{Trace} : \mathcal{P}(wft(\tau_A)) \to \mathcal{P}(wft(\tau))$ maps nonsensical (*spurious* [5,15,21]) abstract traces to the empty set. A badly chosen τ_A generates many spurious traces.

Here are some examples, based on the model in Example 3. For $\Sigma = \{s_i \mid 0 \leq i \leq 100\}$, the transition relation for that example can be defined as

$$\tau = \{(s_i, s_{i+1}) \mid 0 \leq i \leq 99\} \cup \{(s_{100}, s_{100})\}$$

Recall that the state interpretation is

$$\mathcal{I}(s_{2i}) = \{even, \neg odd, \neg halt\}, 0 \leq i \leq 49$$
$$\mathcal{I}(s_{2i+1}) = \{odd, \neg even, \neg halt\}, 0 \leq i \leq 49$$
$$\mathcal{I}(s_{100}) = \{even, \neg odd, halt\}$$

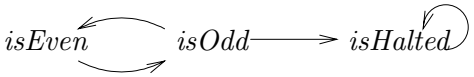
This model's abstraction is in Example 7, where $\Sigma_A = \{isEven, isOdd, isHalted\}$, and $\mathcal{I}_A(isEven) = \mathcal{I}(s_0)$, $\mathcal{I}_A(isOdd) = \mathcal{I}(s_1)$, and $\mathcal{I}_A(isHalted) = \mathcal{I}(s_{100})$.

The abstraction relation for the state sets goes

$$s_{2i}\ \mathcal{R}\ isEven,\ \text{when } 0 \leq i \leq 49$$
$$s_{2i+1}\ \mathcal{R}\ isOdd,\ \text{when } 0 \leq i \leq 49$$
$$s_{100}\ \mathcal{R}\ isHalted$$

Here are four possible transition relations for Σ_A; each is drawn in diagrammatic form.

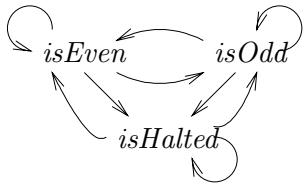
Example 23

$\tau_A =$ 

This definition is appealing because, for all $\pi \in wft(\tau)$, there exists some $\kappa \in wft(\tau_A)$ such that $\pi\ \mathcal{R}^\omega\ \kappa$, and τ_A is "minimal" in the sense that, for any other $wft(\tau_{A'})$ that "covers" $wft(\tau)$, $wft(\tau_A) \subseteq wft(\tau_{A'})$.

$wft(\tau_A)$ also contains spurious traces, e.g., $(isEven\ isOdd)^\omega$, but soundness is unharmed when $(isEven\ isOdd)^\omega \in [\![\phi]\!]^{wft(\tau_A)}\rho$ because $\gamma^{Trace}((isEven\ isOdd)^\omega)$ $= \{\}$.

Example 24

$\tau_B =$ 

Here, $wft(\tau_B) = {\Sigma_A}^\omega$, and $[\![\phi]\!]^A\emptyset$ defines all possible patterns of abstract traces that satisfy ϕ.

Example 25

$\tau_C =$ 

$wft(\tau_C)$ does not cover $wft(\tau)$, meaning that some traces (e.g., $s_0 s_1 s_2 \cdots$) cannot be studied by means of $wft(\tau_C)$. But the abstract model has the property that, for all $\kappa \in wft(\tau_C)$, there exists some $\pi \in wft(\tau)$ such that $\kappa\ \mathcal{R}\ \tau$.

Example 26

$\tau_D =$

$wft(\tau_D) = \{\}$, and soundness vacuously holds, because no $\pi \in wft(\tau)$ is approximated by a trace in $wft(\tau_D)$.

Although Corollary 20, the soundness property, is insensitive to the selection of abstract transition system, the computation of universal properties *is* affected. Recall that a universal property assertion takes the format, $s \models^{\mathcal{M}} \forall\phi$, for $s \in \Sigma$, and asserts that $\pi \models^{\mathcal{M}} \phi$ for all $\pi \in \mathcal{M} \downarrow s$ (that is, for all π such that $\pi(0) = s$). *Universal soundness* asserts, for all $s \in \Sigma, a \in \Sigma_A$, such that $s \mathcal{R} a$ (equivalently, $s \in \gamma(a)$)) and $a \models^{\mathcal{A}} \forall\phi$ implies $s \models^{\mathcal{M}} \forall\phi$.

Clearly, the abstract transition systems in Examples 25 and 26 violate universal soundness. (The latter Example makes $a \models^{wft(\tau_D)} \forall\phi$ hold for all a and ϕ.) We can repair the situation with the following definitions.

Definition 27 *For abstraction relation, $\mathcal{R} \subseteq \Sigma \times \Sigma_A$, a model, $\mathcal{M} \subseteq \Sigma^\omega$,* is $\mathcal{R}$-simulated by $\mathcal{A} \subseteq \Sigma_A^\omega$ *iff for all $s \in \Sigma$, $a \in \Sigma_A$, $\pi \in \mathcal{M}$,*

$$s \mathcal{R} a \text{ and } \pi(0) = s \text{ imply there exists } \kappa \in \mathcal{A}$$
$$\text{such that } \kappa(0) = a \text{ and } \pi \mathcal{R}^\omega \kappa$$

The notion of simulation is the usual one [32]: when $s \mathcal{R} a$, then every trace in $\mathcal{M}$ starting at s is reproduced by a corresponding trace in $\mathcal{A}$ that starts at a. This property implies universal soundness:

Theorem 28 *If $\mathcal{R} \subseteq \Sigma \times \Sigma_A$ reflects atomic properties (that is, $s \mathcal{R} a$ implies $\mathcal{I}(s) \supseteq \mathcal{I}_A(a)$) and $\mathcal{M} \subseteq \Sigma^\omega$ is $\mathcal{R}$-simulated by $\mathcal{A} \subseteq \Sigma_A^\omega$, then for all $a \in \Sigma_A, s \in \Sigma$ such that $s \mathcal{R} a$,*

$$a \models^{\mathcal{A}} \forall\phi \text{ implies } s \models^{\mathcal{M}} \forall\phi$$

We say that $\mathcal{A}$ is a universally sound model of $\mathcal{M}$.

PROOF. To show $s \models^{\mathcal{M}} \forall\phi$ from the assumptions, consider an arbitrary $\pi \in \mathcal{M}$ such that $\pi(0) = s$. By definition of simulation, there exists $\kappa \in \mathcal{A}$ such that $\pi \mathcal{R}^\omega \kappa$ and $\kappa(0) = a$. By assumption, $\kappa \models^{\mathcal{A}} \phi$; by Corollary 20 (soundness), $\pi \models^{\mathcal{M}} \phi$.

When models $\mathcal{M}$ and $\mathcal{A}$ are characterized by transition relations, τ and τ_A, respectively, we can employ a definition of simulation that is stated in terms of the transition relations:

Definition 29 *For abstraction relation, $\mathcal{R} \subseteq \Sigma \times \Sigma_A$, and transition relations, $\tau \subseteq \Sigma \times \Sigma$ and $\tau_A \subseteq \Sigma_A \times \Sigma_A$, relation τ* is $\mathcal{R}$-simulated by *τ_A iff for all $s \in \Sigma$, $a \in \Sigma_A$,*

$$s \mathcal{R} a \text{ and } s \rightarrow s' \in \tau \text{ imply}$$
$$\text{there exists } a' \in \Sigma_A \text{ such that } a \rightarrow a' \in \tau_A \text{ and } s' \mathcal{R} a'$$

This notion of $\mathcal{R}$-simulated is due to Park [33,36]. The following consequence is well known:

Theorem 30 *If $\mathcal{R} \subseteq \Sigma \times \Sigma_A$ reflects atomic properties and τ is $\mathcal{R}$-simulated by τ_A, then $wft(\tau)$ is $\mathcal{R}$-simulated by $wft(\tau_A)$; hence, when $s \mathcal{R} a$,*

$$a \models^{wft(\tau_A)} \forall\phi \text{ implies } s \models^{wft(\tau)} \forall\phi$$

We say that τ_A is a universally sound transition relation of *τ.*

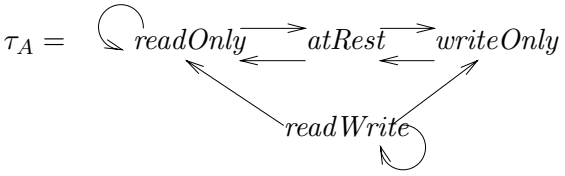
When there is a Galois connection between $\mathcal{P}(\Sigma)$ and Σ_A, we can use the definition of τ to directly formulate this universally sound transition relation, $\rightarrow_\beta$:

$$s \mathcal{R} a \text{ and } s \rightarrow s' \in \tau \text{ imply } a \rightarrow_\beta \beta_{\mathcal{R}}(s'), \text{ where } \beta_{\mathcal{R}}(s') = \sqcap\{a \mid s \mathcal{R} a\}$$

This definition lets τ be $\mathcal{R}$-simulated by $\rightarrow_\beta$, while preserving as much precision as possible in the target states.[15] When a Galois connection is unavailable, a less precise but universally sound transition relation is merely

$$a \rightarrow a' \in \tau_A \text{ if there exist } s, s' \in \Sigma \text{ such that } s \mathcal{R} a, s' \mathcal{R} a', \text{ and } s \rightarrow s' \in \tau$$

Example 31 *Recall the system of readers and writer processes in Example 4 and its abstract state set from Example 10, $\{readOnly, writeOnly, atRest, readWrite\}$. Based on the transition-relation scheme in Example 4, we define this universally sound abstract transition relation, τ_A:*



[15] Dams, et al. [14] prove this transition relation to be the most expressive abstract relation with respect to Σ_A—any universal property that can be proved with a sound abstract transition relation can be proved with $\rightarrow_\beta$ also. Dams's result was proved with respect to branching-time logic, which we study in the next section.

(Note that $\mathcal{I}_A(readOnly) = \{read, \neg write\}$*, etc.) We can validate* $atRest \models^{\tau_A} \forall\mathsf{G}(\neg(read \wedge write))$ *(that is,* $\forall\mathsf{G}(\neg read \vee \neg write)$*) holds, hence all paths from the concrete system's initial state,* $\langle sleep, \cdots, sleep, wait\rangle$*, make* $\mathsf{G}(\neg(read \wedge write))$ *hold.*

The dual notion to universal soundness is *existential soundness*: For all $s \in \Sigma, a \in \Sigma_A$, $s \mathcal{R} a$ (equivalently, $s \in \gamma(a)$) and $a \models^{\mathcal{A}} \exists\phi$ imply $s \models^{\mathcal{M}} \exists\phi$. One example of an existentially sound abstract transition relation is Example 25, where the only trace defined by the abstract relation is $isHalted^\omega$.

Universal soundness does not imply existential soundness: Reconsider Example 23 which satisfies the hypotheses of Theorem 28 and is universally sound. Within that example, we have that $(isEven\ isOdd)^\omega \models^{wft(\tau_A)} \mathsf{GF}odd$, that is, the path $(isEven\ isOdd)^\omega$ makes *odd* hold true infinitely often. But the path is spurious— $\gamma^{Trace}((isEven\ isOdd)^\omega) = \{\}$— so $isEven \models^{wft(\tau_A)} \exists\mathsf{GF}odd$ does not imply $s_0 \models^{wft(\tau)} \exists\mathsf{GF}odd$, even though $s_0 \mathcal{R}\ isEven$.

Indeed, the *dual* result holds when $\mathcal{R}$ *preserves* atomic properties:

Proposition 32 *If* $\mathcal{R} \subseteq \Sigma \times \Sigma_A$ *preserves atomic properties (that is,* $s \mathcal{R} a$ *implies* $\mathcal{I}(s) \subseteq \mathcal{I}_A(a)$*) and* $\mathcal{M} \subseteq \Sigma^\omega$ *is* $\mathcal{R}$*-simulated by* $\mathcal{A} \subseteq \Sigma_A^\omega$*, then for all* $a \in \Sigma_A, s \in \Sigma$ *such that* $s \mathcal{R} a$*,*

$$s \models^{\mathcal{M}} \exists\phi \text{ implies } a \models^{\mathcal{A}} \exists\phi$$

Thus, checking $a \models^{\mathcal{A}} \exists\phi$ "overapproximates" $\exists\phi$ on the concrete model's states. This situation can be used to *refute* existential properties: If $a \not\models^{\mathcal{A}} \exists\phi$, then $s \not\models^{\mathcal{M}} \exists\phi$ as well. But this is not existential soundness.

As hinted by the previous Proposition, to obtain existential soundness, we must dualize the simulation itself: Given $\mathcal{M} \subseteq \Sigma^\omega$ and $\mathcal{A} \subseteq \Sigma_A^\omega$, we require that $\mathcal{A}$ is $\mathcal{R}^{-1}$-simulated by $\mathcal{M}$.[16] That is, $s \mathcal{R} a$ implies that every trace in $\mathcal{A}$ starting at a is reproduced by a trace in $\mathcal{M}$ starting at s— no trace in $\mathcal{A}$ is spurious with respect to a and s.

The reverse simulation gives existential soundness:

Theorem 33 *If* $\mathcal{R}^{-1} \subseteq \Sigma_A \times \Sigma$ *reflects atomic properties and* $\mathcal{A}$ *is* $\mathcal{R}^{-1}$*-simulated by* $\mathcal{M}$*, then for all* $s \in \Sigma$ *and* $a \in \Sigma_A$ *such that* $s \mathcal{R} a$*,*

$$a \models^{\mathcal{A}} \exists\phi \text{ implies } s \models^{\mathcal{M}} \exists\phi$$

We say that $\mathcal{A}$ *is an* existentially sound model of $\mathcal{M}$*.*

[16] Define $\mathcal{R}^{-1} \subseteq \Sigma_A \times \Sigma$ as $(a, s) \in \mathcal{R}^{-1}$ iff $(s, a) \in \mathcal{R}$.

As before, this result applies to the transition relations: For $\tau \subseteq \Sigma \times \Sigma$, $\tau_A \subseteq \Sigma_A \times \Sigma_A$, *and* $\mathcal{R} \subseteq \Sigma \times \Sigma_A$, *if* $\mathcal{R}^{-1}$ *reflects atomic properties and* τ_A *is* $\mathcal{R}^{-1}$*-simulated by* τ, *then we have*

$$s \; \mathcal{R} \; a \text{ and } a \models^{wft(\tau_A)} \exists\phi \text{ imply } s \models^{wft(\tau)} \exists\phi$$

and we say that τ_A *is an* existentially sound transition relation for τ.

When there is a Galois connection available, we can define from τ this most precise existentially sound transition relation, $\rightarrow_\alpha$ [14]:

$a \rightarrow_\alpha \alpha(S)$ iff $AE(a, S)$ and $Minimal(a, S)$,

where $AE(a, S)$ iff for all $s \; \mathcal{R} \; a$, there exists $s' \in S$ such that $s \rightarrow s' \in \tau$

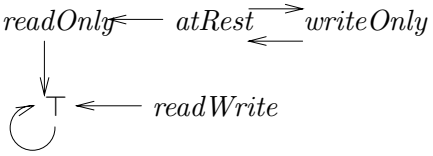
and $Minimal(a, S)$ iff for all $S' \neq S, AE(a, S')$ implies $S' \not\subseteq S$

When a Galois connection is unavailable, the following less precise relation guarantees existential soundness:

$$a \rightarrow a' \in \tau_A \text{ if } AE(a, \gamma(a'))$$

where, as always, $\gamma(a') = \{s \in \Sigma \mid s \; \mathcal{R} \; a'\}$.

Example 34 *Returning again to the readers-writer system in Example 4, we extend its abstract state set in Example 10 with an extra state,* $\top$, *such that* $s \; \mathcal{R} \; \top$, *for all* $s \in \Sigma$; *the new state represents complete loss of read-write information:* $\mathcal{I}_A(\top) = \{\}$. *Here is the resulting existentially sound transition system,* τ_E*:*



The $\top$ *state ensures that all traces in* $wft(\tau_E)$ *have infinite length. The transition,* $readOnly \rightarrow \top$, *is included in* τ_E *because it is the only transition that is guaranteed to be simulated by all concrete states,* s, *such that* $s \; \mathcal{R} \; readOnly$.

The previous example should be contrasted with its universally sound counterpart in Example 31— *different* transition systems are used. [17] Indeed, when

[17] We should add the $\top$ state to Example 31 as well; this can be done with little effort.

one and the same abstract transition relation ensures both universal and existential soundness, then the abstract transition system is said to be *(strongly) bisimular* to its corresponding concrete transition system [33]. Bisimularity is a strong property that is rarely encountered in abstraction examples. For this reason, we wish to develop a single abstraction framework that can accommodate both a universally sound abstract transition system along with an existentially sound abstract transition system; we develop this framework in the remainder of the paper.

5 Abstracting trace checking to state checking

Although the theory of linear-time logic is simple and even elegant, many practitioners do not use LTL or LMC to validate properties of execution traces. As noted by Vardi [50], one reason is the time complexity of LTL model checking: Given an LTL proposition of size m and a transition relation of cardinality n, to validate a universal property, $\forall\phi$, takes time complexity $n \cdot 2^{\mathcal{O}(m)}$ and is PSPACE-complete [30,42].

Given this result and the strong culture based on iterative data-flow analysis, where properties are calculated based on information flow from state to state in a transition relation, there is the natural urge to simplify the validation process by computing only on states—not traces. That is, we wish to abstract further the inductive definition of LMC model checking in Figure 3 so that the valuation function, $[\![\phi]\!]^{wft(\tau_A)} \in AbsEnv \to \mathcal{P}(wft(\tau_A))$, which computes sets of traces, is abstracted to $[\![\phi]\!]^{\tau_A} \in AbsEnv' \to \mathcal{P}(\Sigma_A)$, which computes sets of states. Then we concretize the calculated set of states to determine the set of traces for which ϕ holds.

Of course, precision might be lost, but under reasonable assumptions, there *is* an abstraction that loses no precision when universal properties are checked: $a \models^{wft(\tau_A)} \forall\phi$ iff $a \in [\![\phi]\!]^{\tau_A}\emptyset$. There is a dual result for existential-property checking. These results let us do linear-time model checking with so-called *branching-time* techniques.

The results that follow apply to *any* transition relation, $\tau \subseteq \Sigma \times \Sigma$, and its generated model, $wft(\tau)$, regardless of whether τ defines a set of "concrete" or "abstract" transitions— our goal is to perform state checking with τ that is as precise as trace checking with $wft(\tau)$.

How do we abstract an infinite trace to a single state? Crude as it may seem, we use the trace's initial state as its abstraction. As noted by Cousot and Cousot [12], we retain some precision if we abstract *sets* of traces to sets of states: Here is the first Galois connection that formalizes such an abstraction:

Definition 35 *[12] For model $\mathcal{A} = wft(\tau)$, where $\tau \subseteq \Sigma \times \Sigma$, the model's* universal trace abstraction *is defined as this Galois connection,*

$$\langle \alpha^{\forall} : \langle \mathcal{P}(\mathcal{A}), \supseteq \rangle \to \langle \mathcal{P}(\Sigma), \supseteq \rangle,\ \gamma : \langle \mathcal{P}(\Sigma), \supseteq \rangle \to \langle \mathcal{P}(\mathcal{A}), \supseteq \rangle \rangle$$
$$\alpha^{\forall}(T) = \{a \in \mathcal{A} \mid (\mathcal{A} \downarrow a) \subseteq T\}$$
$$\gamma(S) = \{\kappa \in \mathcal{A} \mid \kappa(0) \in S\}$$

(Recall that $\mathcal{A} \downarrow a = \{\kappa \in \mathcal{A} \mid \kappa(0) = a\}$.) The abstraction is "universal" in the way it abstracts a set of paths— a state, a, appears in T's abstraction only if *all* traces that begin with state a are contained in T. This implies $\gamma(\alpha^{\forall}(T)) \subseteq T$ and $\alpha^{\forall}(\gamma(S)) = S$, giving a Galois connection (when superset inclusion is used for ordering the two powersets).

We employ the universal trace abstraction to derive a semantics for LMC that does its checking directly on states. Recall, for operation, $op : \mathcal{P}(\mathcal{A}) \to \mathcal{P}(\mathcal{A})$, its most precise abstraction must be $op_{\forall} = \alpha^{\forall} \circ op \circ \gamma : \mathcal{P}(\Sigma) \to \mathcal{P}(\Sigma)$. We have these pleasant results:

Proposition 36 *For $\mathcal{A} = wft(\tau)$ and the three operations from Definition 21, the most precise abstract operations with respect to the universal trace abstraction are*

$$next_{\forall}(S) = \{a' \in \Sigma \mid \textit{for all } a \in \Sigma, a' \to a \in \tau \textit{ implies } a \in S\}$$
$$and_{\forall}(S_1, S_2) = S_1 \cap S_2$$
$$or_{\forall}(S_1, S_2) = S_1 \cup S_2$$

Using these operations, we obtain the *universal state-checking semantics* in Figure 5. A semantics like the one in the Figure is called a *branching-time semantics* [18,27], because the semantics of a next-state transition, $\mathsf{X}\phi$, checks the "branches" defined by τ:

$$a \models^{\forall\tau} \mathsf{X}\phi \text{ iff for all } a' \in \Sigma, a \to a' \in \tau \text{ implies } a' \models^{\forall\tau} \phi$$

$\rho \in AbsEnv = Var \to \mathcal{P}(\Sigma)$
$[\![\phi]\!]^{\forall\tau} \in AbsEnv \to \mathcal{P}(\Sigma)$

$$[\![p]\!]^{\forall\tau}\rho = \{a \in \Sigma \mid p \in \mathcal{I}^A(a)\}$$
$$[\![\neg p]\!]^{\forall\tau}\rho = \{a \in \Sigma \mid \neg p \in \mathcal{I}^A(a)\}$$
$$[\![\phi_1 \wedge \phi_2]\!]^{\forall\tau}\rho = and_\forall([\![\phi_1]\!]^{\forall\tau}\rho, [\![\phi_2]\!]^{\forall\tau}\rho)$$
where $and_\forall(S_1, S_2) = S_1 \cap S_2$
$$[\![\phi_1 \vee \phi_2]\!]^{\forall\tau}\rho = or_\forall([\![\phi_1]\!]^{\forall\tau}\rho, [\![\phi_2]\!]^{\forall\tau}\rho)$$
where $or_\forall(S_1, S_2) = S_1 \cup S_2$
$$[\![\mathsf{X}\phi]\!]^{\forall\tau}\rho = next_\forall([\![\phi]\!]^{\forall\tau}\rho)$$
where $next_\forall = \{a \in \Sigma \mid \text{for all } a' \in \Sigma, a \to a' \in \tau \text{ implies } a' \in S\}$
$$[\![\mu Z.\phi]\!]^{\forall\tau}\rho = \bigcap\{S \subseteq \Sigma \mid F(S) \subseteq S\}$$
$$[\![\nu Z.\phi]\!]^{\forall\tau}\rho = \bigcup\{S \subseteq \Sigma \mid S \subseteq F(S)\},$$
where $F(S) = [\![\phi]\!]^{\forall\tau}(\rho + [Z \mapsto S])$
$$[\![Z]\!]^{\forall\tau}\rho = \rho(Z)$$

Fig. 5. Universally abstracted LMC semantics

The semantics of X rebuilds the next state(s) in the path(s) abstracted by state a. Indeed, to reflect this, the modality's syntax is often written as $\forall\mathsf{X}\phi$— the logic in the Figure is the branching-time logic, ACTL [7] with recursion.

5.2 Implementing the state-checking semantics

To compute $[\![\phi]\!]^{\forall\tau}\rho$, we can use the semantics definition in Figure 5 as an algorithm that inductively calculates the sets of states, $[\![\phi']\!]^{\forall\tau}\rho$, for all subformulas, ϕ' of ϕ [19].

This straightforward approach can be done with time complexity, $\mathcal{O}(m \cdot n)$, where ϕ has size m and τ has cardinality n [4]. To obtain this complexity bound, we must assume that ϕ contains no *alternating fixed points*, that is, no subformulas of the form, $\mu Z. \cdots (\nu Z'. \cdots Z \cdots) \cdots$ (or similarly, with μ and ν swapped) [34].

When τ has finite cardinality, then the semantics of the two fixed-point forms

can be calculated as

$$[\![\mu Z.\phi]\!]^{\forall\tau}\rho = \bigcup_{i\geq 0} S_i, \text{ where } \begin{cases} S_0 = \emptyset \\ S_{i+1} = [\![\phi]\!]^{\forall\tau}([Z \mapsto S_i]\rho) \end{cases}$$

$$[\![\nu Z.\phi]\!]^{\forall\tau}\rho = \bigcap_{i\geq 0} S_i, \text{ where } \begin{cases} S_0 = \Sigma \\ S_{i+1} = [\![\phi]\!]^{\forall\tau}([Z \mapsto S_i]\rho) \end{cases}$$

meaning that the traditional iterative technique calculates the state sets that satisfy the fixed points. This iterative approach resembles the one taken by iterative data-flow analysis [9,23,26], and an overt connection will be made later in the paper [39,41,43,44].

When ϕ contains alternating fixed points, a tableau method [34,46] can be used to decide specific goals of form, $a \models^{\forall\tau} \phi$.

5.3 Soundness and completeness of universal state checking

Because the universal trace abstraction, $\langle \alpha^{\forall}, \gamma \rangle$, is a Galois connection, the standardized soundness result of Cousot and Cousot [12] applies to the semantics of Figure 5:

Theorem 37 *[12]: For $\mathcal{A} = wft(\tau)$ and $\rho : Var \to \mathcal{P}(\mathcal{A})$, define $\alpha^{\forall}\rho : Var \to \mathcal{P}(\Sigma)$ as $(\alpha^{\forall}\rho)(Z) = \alpha^{\forall}(\rho(Z))$. Then, for all ρ and ϕ, soundness holds:*

$$[\![\phi]\!]^{\forall\tau}(\alpha^{\forall}\rho) \subseteq \alpha^{\forall}([\![\phi]\!]^{\mathcal{A}}\rho)$$

This result is important, because

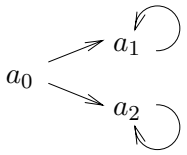
$$a \models^{\mathcal{A}} \forall\phi \text{ iff } a \in \alpha^{\forall}([\![\phi]\!]^{\mathcal{A}}\emptyset)$$

meaning that the semantics of Figure 5 does universal soundness checking directly on states.

Alas, the subset inclusion in Theorem 37 is not an equality, so precision can be lost by checking states rather than traces, as can be seen in this counterexample:

Example 38 *Let $\Sigma = \{a_0, a_1, a_2\}$ such that $\mathcal{I}_A(a_0) = \mathcal{I}_A(a_1) = \{p\}$ and*

$\mathcal{I}_A(a_2) = \{q\}$, *and define* τ *as*

$\tau =$ 

We can easily check that $a_0 \models^{wft(\tau)} \forall(\mathsf{X}p \vee \mathsf{X}q)$, *but* $[\![\mathsf{X}p \vee \mathsf{X}q]\!]^{\forall\tau}\emptyset = \{\}$, *because* $[\![\mathsf{X}p]\!]^{\forall\tau}\emptyset = \{\} = [\![\mathsf{X}q]\!]^{\forall\tau}\emptyset$— *universal quantification does not distribute across disjunction.*

Since the next modality, X, is implicitly universally quantified, we can obtain completeness by limiting the syntax of LMC so that modalities do not appear in both operands of a disjunction:

Theorem 39 *[12] : For this restricted syntax of linear-time logic,* $\forall$LMC,

$$p, \neg p \in \mathit{AtomProp} \qquad Z \in \mathit{Var} \qquad \psi \in \mathit{StateProp} \qquad \phi \in \mathit{PathProp}$$

$$\begin{array}{rcl} \phi & ::= & \psi \mid \psi \vee \phi \mid \phi \vee \psi \mid \phi_1 \wedge \phi_2 \mid \mathsf{X}\phi \mid \mu Z.\phi \mid \nu Z.\phi \mid Z \\ \psi & ::= & p \mid \neg p \mid \psi_1 \vee \psi_2 \mid \psi_1 \wedge \psi_2 \end{array}$$

the universal state-checking semantics is complete: $[\![\phi]\!]^{\forall\tau}(\alpha^{\forall}\rho) = \alpha^{\forall}([\![\phi]\!]^{\mathcal{A}}\rho)$

For $\forall$LMC, checking states for universal properties is exactly as precise as checking traces. The new syntactic category, $\mathit{StateProp}$, merely defines propositional properties of states. (There is no need for recursion in $\mathit{StateProp}$, since X is disallowed there.)

Example 40 *We can employ the universal state-checking semantics to validate mutual exclusion for the readers-writer system in Example 31. For the transition relation,* τ_A, *given in the Example, we wish to validate that*

$$\mathit{atRest} \models^{wft(\tau_A)} \forall\mathsf{G}\neg(\mathit{read} \wedge \mathit{write})$$

that is, $\forall(\nu Z.(\neg \mathit{read} \vee \neg \mathit{write}) \wedge \mathsf{X}Z)$. *This proposition fits the syntax of* $\forall$*LMC, and we can apply the universal state-checking semantics and indeed validate the result.*

5.4 Existential abstraction of traces to states

The sequence of steps that derived the universal state-checking semantics can be repeated for an existential state-checking semantics. The key ingredient is this Galois connection:

Definition 41 *[12] For model $\mathcal{A} = wft(\tau)$, where $\tau \subseteq \Sigma \times \Sigma$, the model's* exitential trace abstraction *is this Galois connection,*

$$\langle \alpha^{\exists} : \langle \mathcal{P}(\mathcal{A}), \subseteq \rangle \to \langle \mathcal{P}(\Sigma), \subseteq \rangle,\ \gamma : \langle \mathcal{P}(\Sigma), \subseteq \rangle \to \langle \mathcal{P}(\mathcal{A}), \subseteq \rangle \rangle$$
$$\alpha^{\exists}(T) = \{a \in \mathcal{A} \mid \kappa(0) = a, \kappa \in T\}$$
$$\gamma(S) = \{\kappa \in \mathcal{A} \mid \kappa(0) \in S\}$$

The operations used for the existential state-checking semantics are defined as $op_{\exists} = \alpha^{\exists} \circ op \circ \gamma : \mathcal{P}(\Sigma) \to \mathcal{P}(\Sigma)$:

Proposition 42 *For $\mathcal{A} = wft(\tau)$ and the three operations from Definition 21, the most precise abstract operations with respect to the universal trace abstraction are*

$$next_{\exists}(S) = \{a' \in \Sigma \mid \text{there exists } a \in \Sigma \text{ such that } a' \to a \in \tau \text{ and } a \in S\}$$
$$and_{\exists}(S_1, S_2) = S_1 \cap S_2$$
$$or_{\exists}(S_1, S_2) = S_1 \cup S_2$$

These operations define the *existential state-checking semantics* in Figure 6. The operator, X, is often written as $\exists \mathsf{X} \phi$— the logic in the Figure is the branching-time logic, ECTL with recursion.

5.5 Co-soundness and completeness of existential state checking

The (co)soundness theorem for existential abstraction gives a reverse inclusion:

Theorem 43 *[12]: For $\mathcal{A} = wft(\tau)$ and $\rho : Var \to \mathcal{P}(\mathcal{A})$, define $\alpha^{\exists}\rho : Var \to \mathcal{P}(\Sigma)$ as $(\alpha^{\exists}\rho)(Z) = \alpha^{\exists}(\rho(Z))$. For all ρ and ϕ,*

$$[\![\phi]\!]^{\exists\tau}(\alpha^{\exists}\rho) \supseteq \alpha^{\exists}([\![\phi]\!]^{\mathcal{A}}\rho)$$

$\rho \in AbsEnv = Var \to \mathcal{P}(\Sigma)$
$[\![\phi]\!]^{\exists\tau} \in AbsEnv \to \mathcal{P}(\Sigma)$

$$[\![p]\!]^{\exists\tau}\rho = \{a \in \Sigma \mid p \in \mathcal{I}^A(a)\}$$
$$[\![\neg p]\!]^{\exists\tau}\rho = \{a \in \Sigma \mid \neg p \in \mathcal{I}^A(a)\}$$
$$[\![\phi_1 \wedge \phi_2]\!]^{\exists\tau}\rho = and_\exists([\![\phi_1]\!]^{\exists\tau}\rho, [\![\phi_2]\!]^{\exists\tau}\rho)$$
where $and_\exists(S_1, S_2) = S_1 \cap S_2$
$$[\![\phi_1 \vee \phi_2]\!]^{\exists\tau}\rho = or_\exists([\![\phi_1]\!]^{\exists\tau}\rho, [\![\phi_2]\!]^{\exists\tau}\rho)$$
where $or_\exists(S_1, S_2) = S_1 \cup S_2$
$$[\![\mathsf{X}\phi]\!]^{\exists\tau}\rho = next_\exists([\![\phi]\!]^{\exists\tau}\rho)$$
where $next_\exists = \{a \in \Sigma \mid \text{exists } a' \in \Sigma, a \to a' \in \tau \text{ and } a' \in S\}$
$$[\![\mu Z.\phi]\!]^{\exists\tau}\rho = \bigcap\{S \subseteq \Sigma \mid F(S) \subseteq S\}$$
$$[\![\nu Z.\phi]\!]^{\exists\tau}\rho = \bigcup\{S \subseteq \Sigma \mid S \subseteq F(S)\},$$
where $F(S) = [\![\phi]\!]^{\exists\tau}(\rho + [Z \mapsto S])$
$$[\![Z]\!]^{\exists\tau}\rho = \rho(Z)$$

Fig. 6. Existentially abstracted LMC semantics

Co-soundness overestimates the states that possess a trace with the desired property. Like before, we have that

$$a \models^{\mathcal{A}} \exists\phi \text{ iff } a \in \alpha^\exists([\![\phi]\!]^{\mathcal{A}}\emptyset)$$

and as before, the inclusion in Theorem 43 is not an equality. Consider again the transition relation in Example 38: We have that $a_0 \not\models^{wft(\tau)} \exists(\mathsf{X}p \wedge \mathsf{X}q)$, but $[\![\mathsf{X}p \wedge \mathsf{X}q]\!]^{\exists\tau}\emptyset = \{a_0\}$, because existential quantification does not distribute across conjunction.

Theorem 44 *[12] : For this restricted syntax of linear-time logic,* ∃LMC*,*

$$p, \neg p \in AtomProp \qquad Z \in Var \qquad \psi \in StateProp \qquad \phi \in PathProp$$
$$\phi ::= \psi \mid \psi \wedge \phi \mid \phi \wedge \psi \mid \phi_1 \vee \phi_2 \mid \mathsf{X}\phi \mid \mu Z.\phi \mid \nu Z.\phi \mid Z$$
$$\psi ::= p \mid \neg p \mid \psi_1 \vee \psi_2 \mid \psi_1 \wedge \psi_2$$

the existential state-checking semantics is complete: $[\![\phi]\!]^{\exists\tau}(\alpha^\exists\rho) = \alpha^\exists([\![\phi]\!]^{\mathcal{A}}\rho)$

For ∃LMC, checking states for existential properties is exactly as precise as checking traces.

The development of linear-time logic into LMC and its state-checkable variants, ∀LMC and ∃LMC, has an important precedent in the data-flow analysis community: Classic program analyses are intuitively understood and formulated as *meet-over-all-paths* (that is, linear-time) properties but are implemented as *iterative, maximal fixed point* (branching-time) state-checking algorithms. When an analysis problem is *distributive*, then the meet-over-all-paths semantics equals the maximal-fixed-point semantics (the state-checking semantics is complete) [26]. [18]

We present one well-told example of an existential data-flow property: live-variables analysis [1,12,39]. Given a sequential program, a variable, `x`, is *live* at a program point, p, if some trace proceeding from p references `x`'s value before overwriting it. A compiler can use the results of live-variable analysis to improve data locality— values of live variables are retained in registers when possible.

Figure 7 displays a program in flow-graph format, where the program's "states" are the statements, p_i. The flow graph defines an abstract transition relation τ_A, where $wft(\tau_A)$ are considered legal executions. This is an inexact approximation— $p_0p_1p_2p_4p_5p_6^\omega \in wft(\tau_A)$, but this is not an actual execution. (Indeed, the statement at p_5 is dead code.) The atomic properties, use_v and $kill_v$, denote whether a state references `v`'s value or overwrites it, respectively.

When the definition of "live variable" is applied to the graph at state p_1, `x` is live (it is used in `x = 1`) and `y` is live, because the path, $p_1p_3p_4p_5\cdots$, leads to a reference. Live variables at other states can be determined similarly.

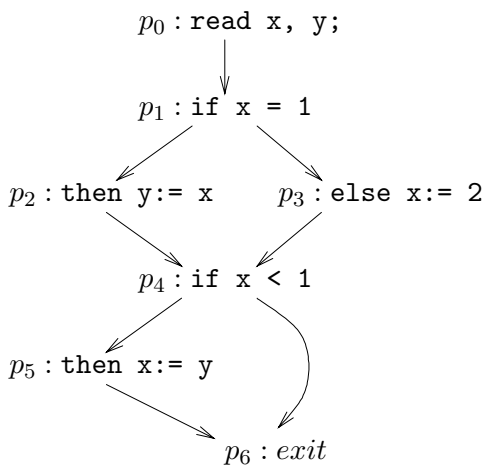
The definition of "live variable" is a meet-over-all-paths specification, because all execution traces (paths) from state p are considered when determining a variable's liveness at p. But a compiler implements the analysis not by generating traces, but by propagating information from program point to program point by means of flow equations, one per program point. The flow-equation schema that calculates the set of variables that are live at point p, $LV(p)$, reads as follows:

$$LV(p) = \mathit{Used}(p) \cup (\mathit{notModified}(p) \cap (\bigcup_{p' \in succ\, p} LV(p')))$$

$$\text{where } \mathit{Used}(p) = \{v \mid use_v \in \mathcal{I}(p)\}$$

$$\text{and } \mathit{notModified}(p) = \{v \mid \neg kill_v \in \mathcal{I}(p)\}$$

[18] Note that lattice distributivity states, in order-theoretic terms, that universal/existential abstraction distributes over the disjunction/conjunction connective used to join/meet flow information.



$$\mathcal{I}(p_0) = \{kill_x, kill_y, \neg use_x, \neg use_y\},\ \mathcal{I}(p_1) = \{\neg kill_x, \neg kill_y, use_x, \neg use_y\}$$
$$\mathcal{I}(p_2) = \{\neg kill_x, kill_y, use_x, \neg use_y\},\ \mathcal{I}(p_3) = \{kill_x, \neg kill_y, \neg use_x, \neg use_y\}$$
$$\mathcal{I}(p_4) = \{\neg kill_x, \neg kill_y, use_x, \neg use_y\},\ \mathcal{I}(p_5) = \{kill_x, \neg kill_y, \neg use_x, use_y\}$$
$$\mathcal{I}(p_6) = \{\neg kill_x, \neg kill_y, \neg use_x, \neg use_y\}$$

Fig. 7. Program flow graph and atomic properties

The variables that are live at entry to p are those variables that are used in p's statement or are live at one of p's immediate-successor states and are not modified within p's statement.

A flow equation is generated for each state in the graph, and a least-fixed point solution is calculated. The flow equations and their solution for the example are listed in Figure 8. Since the solution to the live-variables analysis is computed on the program's states and not upon its traces, we might question if the result is sensible. There is a standard result that shows that live-variables analysis is a distributive analysis [26], meaning that the solution calculated by the flow equations equals the solution calculated from checking the execution traces.

Live-variables analysis can be elegantly formulated as a model-checking exercise: Figure 5 defines an abstract transition relation, τ_A, whose traces define an abstract model. The assertion that variable v is live at state p is this existential LTL property:

$$p \models^{wft(\tau_A)} \exists(\neg kill_v \,\mathsf{U}\, use_v)$$

The LTL formula translates to the LMC coding, $\mu Z.\ use_v \vee (\neg kill_v \wedge \mathsf{X}Z)$. Compare this coding to that of $LV(p)$, which has virtually identical logical structure [39]:

$$v \in LV(p) \text{ iff } p \models^{wft(\tau_A)} \mu Z.\ use_v \vee (\neg kill_v \wedge \mathsf{X}Z)$$

Equations:

$$
\begin{aligned}
LV(p_0) &= \mathit{Used}(p_0) \cup (\mathit{notModified}(p_0) \cap LV(p_1)) \\
LV(p_1) &= \mathit{Used}(p_1) \cup (\mathit{notModified}(p_1) \cap (LV(p_2) \cup LV(p_3)) \\
LV(p_2) &= \mathit{Used}(p_2) \cup (\mathit{notModified}(p_2) \cap LV(p_4)) \\
LV(p_3) &= \mathit{Used}(p_3) \cup (\mathit{notModified}(p_3) \cap LV(p_4)) \\
LV(p_4) &= \mathit{Used}(p_4) \cup (\mathit{notModified}(p_4) \cap (LV(p_5) \cup LV(p_6)) \\
LV(p_5) &= \mathit{Used}(p_5) \cup (\mathit{notModified}(p_5) \cap LV(p_6)) \\
LV(p_6) &= \mathit{Used}(p_6) \cup (\mathit{notModified}(p_6) \cap \{\})
\end{aligned}
$$

Solutions:

$$
\begin{aligned}
&LV(p_0) = \{\} \qquad LV(p_1) = \{\mathtt{x}, \mathtt{y}\} \\
&LV(p_2) = \{\mathtt{x}\} \quad LV(p_3) = \{\mathtt{y}\} \\
&LV(p_4) = \{\mathtt{x}, \mathtt{y}\} \; LV(p_5) = \{\mathtt{y}\} \\
&LV(p_6) = \{\}
\end{aligned}
$$

Fig. 8. Live-variables data-flow analysis of program flow graph

The formula falls within ∃LMC (where existential abstraction distributes across the restricted form of conjunction), and by completeness of ∃LMC, the existential state-checking semantics can be used to compute exactly the same solution as that calculated by the trace semantics. This foundational development justifies using data-flow equations to compute live-variables analysis.

One background issue remains: The program's concrete model, $\mathcal{M}$, contains exactly the legal executions built from states of format, $\langle p, x, y\rangle$, consisting of an instruction counter and values of the variables. For example, $\langle p_0, ?, ?\rangle\langle p_1, 1, 4\rangle$ $\langle p_2, 1, 1\rangle\langle p_4, 1, 1\rangle\langle p_6, 1, 1\rangle^\omega \in \mathcal{M}$.

Model $\mathcal{M}$ is $\mathcal{R}$-simulated by $wft(\tau_A)$, where $\langle p, x, y\rangle \; \mathcal{R} \; p$— every concrete execution in $\mathcal{M}$ is reproduced by a trace in $wft(\tau_A)$. (The converse fails to hold.) This makes τ_A suitable for soundly validating universal properties and for *refuting*— not validating— existential ones. Proposition 32 notes that the result of validating an existential property on such a model can yield "false positives," and this is indeed the case for the example, which decides that variable $\mathtt{y}$ is live at state p_1, which is not the case for any execution in $\mathcal{M}$.

Here, the live-variable false positive might cause an unnecessary program transformation to be performed, but fortunately it does not harm the program's semantics. But false positives should never be accepted when program validation is the goal.

$\rho \in Env = Var \to \mathcal{P}(\mathcal{M})$
$[\![\cdot]\!]^{\mathcal{M}} \in Env \to \mathcal{P}(\mathcal{M})$

$$[\![\forall\phi]\!]^{\mathcal{M}}\rho = \{\pi \in \mathcal{M} \mid (\mathcal{M} \downarrow \pi(0)) \subseteq [\![\phi]\!]^{\mathcal{M}}\rho\}$$
$$[\![\exists\phi]\!]^{\mathcal{M}}\rho = \{\pi \in \mathcal{M} \mid (\mathcal{M} \downarrow \pi(0)) \cap [\![\phi]\!]^{\mathcal{M}}\rho \neq \{\}\}$$

See Figure 2 for the semantics of the remaining syntax constructions.

Fig. 9. mu-CTL* semantics

6 A logic that mixes universal and existential properties

At this point, we have comprehensive understanding of checking properties that are prefixed by a single universal or existential quantifier. Now we study how to mix quantifiers within a single proposition. Here is the syntax of LMC extended by quantification:

$$p, \neg p \in AtomProp \qquad Z \in Var \qquad \phi \in PathProp$$
$$\phi \ ::= \ p \mid \neg p \mid \forall\phi \mid \exists\phi \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \mathsf{X}\phi$$
$$\mid \mu Z.\phi \mid \nu Z.\phi \mid Z$$

We call this logic, *mu-CTL**, because it is the CTL* logic extended by recursion [6]. Given a trace-set model, $\mathcal{M}$, we define the semantics of mu-CTL* as exactly the semantics of LMC in Figure 2 extended by the semantics of the two new constructions, seen in Figure 9 [12]. The Figure states that a trace, $\pi \in \mathcal{M}$, has property $\forall\phi$ iff it and all of its "sibling traces" (those traces in $\mathcal{M}$ that also start with $\pi(0)$), have property ϕ also. Trace π has $\exists\phi$ iff it or a sibling trace has ϕ. The quantifiers define *state properties* about $\pi(0)$, similar to the state properties defined by the atomic properties, p and $\neg p$. In this sense, mu-CTL* is a mix of linear-time time and branching-time logic [6,27].

We can use mu-CTL* to define universal-existential properties. For example, consider again the readers-writer system in Example 4 and this property:

$$\forall(write \supset (\mathsf{X}(\neg write \wedge \exists \mathsf{F} read)))$$

The property asserts, for all traces that begin by writing to the shared file, the next transition is to a state that is not writing and has the possibility of reading sometime in the finite future. (Recall that $\mathsf{F}read$ abbreviates $\mu Z.read \vee \mathsf{X}Z$.) The embedded existential quantifier does *not* demand that reading will occur—only that there is a potential ("sibling") path that reads.

6.1 Applying mu-CTL to an abstract transition relation*

Say that a concrete system, as represented by a transition relation, $\tau \subseteq \Sigma \times \Sigma$, has been abstracted to a transition relation, $\tau_A \subseteq \Sigma_A \times \Sigma_A$, such that τ is $\mathcal{R}$-simulated by τ_A and $\mathcal{R}$ reflects atomic properties, as usual. From the previous sections, we know that we can soundly check $\forall\phi$ propositions on $wft(\tau_A)$, but we cannot do the same for propositions of form $\exists\phi$.

For the sake of discussion, say that τ_A is $\mathcal{R}^{-1}$-simulated by τ also—that is, $\mathcal{R}$ is a bisimulation between τ and τ_A. Now we can soundly check all of mu-CTL* on $wft(\tau_A)$:

Theorem 45 *For $\tau \subseteq \Sigma \times \Sigma$, $\tau_A \subseteq \Sigma_A \times \Sigma_A$, assume that $\mathcal{R} \subseteq \Sigma \times \Sigma_A$ is a bisimulation and reflects atomic properties. Then, for all $\pi \in wft(\tau)$, $\kappa \in wft(\tau_A)$, and closed $\phi \in$ mu-CTL*,*

$$\pi\ \mathcal{R}^{\omega}\ \kappa \text{ and } \kappa \in [\![\phi]\!]^{wft(\tau_A)}\emptyset \text{ imply } \pi \in [\![\phi]\!]^{wft(\tau)}\emptyset$$

PROOF. The proof is the usual induction on the structure of ϕ, augmenting the proof of Theorem 22 by those of Theorems 30 and 33 for the two new quantified constructions.

6.2 Abstracting mu-CTL trace checking into state checking*

The next step is to abstract the trace checking defined by Figure 9 into a semantics that does state checking of mu-CTL* properties. Let τ be a transition relation such that we wish to use for state checking. We can apply the results from Section 5 to devise a semantics that is the union of $[\![\cdot]\!]^{\forall\tau}$ (Figure 5) and $[\![\cdot]\!]^{\exists\tau}$ (Figure 6). But we must take care to formulate a syntax that preserves completeness of state checking for both semantics. Here is one result, which unions the sublogics, $\forall LMC$ and $\exists LMC$:

$$\begin{array}{lll}
& p, \neg p \in \mathit{AtomProp} \qquad Z \in \mathit{UnivVar} \qquad Z' \in \mathit{ExistVar} & \\
& \psi \in \mathit{StateProp} \qquad \phi \in \mathit{UnivPathProp} \qquad \phi' \in \mathit{ExistPathProp} & \\
\psi & ::= & p \mid \neg p \mid \psi_1 \vee \psi_2 \mid \psi_1 \wedge \psi_2 \mid \forall\phi \mid \exists\phi' \\
\phi & ::= & \psi \mid \psi \vee \phi \mid \phi \vee \psi \mid \phi_1 \wedge \phi_2 \mid \mathsf{X}\phi \mid \mu Z.\phi \mid \nu Z.\phi \mid Z \\
\phi' & ::= & \psi \mid \psi \wedge \phi' \mid \phi' \wedge \psi \mid \phi'_1 \vee \phi'_2 \mid \mathsf{X}\phi' \mid \mu Z'.\phi' \mid \nu Z'.\phi' \mid Z'
\end{array}$$

The semantics of formulas, $\psi \in StateProp$, is $[\![\psi]\!]^\tau\rho = [\![\psi]\!]^{\forall\tau}\rho = [\![\psi]\!]^{\exists\tau}\rho$. For the quantified constructions, we use

$$[\![\forall\phi]\!]^\tau\rho = [\![\phi]\!]^{\forall\tau}\rho$$
$$[\![\exists\phi']\!]^\tau\rho = [\![\phi']\!]^{\exists\tau}\rho$$

Finally, we use $[\![\phi]\!]^{\forall\tau}\rho$ and $[\![\phi']\!]^{\exists\tau}\rho$ for the semantics of universal and existential path propositions, respectively.

The logic we have just derived is clumsy; its cumbersome syntax merely portrays those propositions, $\forall\phi$, that can be equivalently rewritten so that the outermost universal quantifier can be shifted inwards until it comes to rest against ϕ's X modalities, e.g.,

$$\forall((\exists(p \wedge \mathsf{X}q) \vee \mathsf{X}r) \wedge \forall p)$$
$$\text{iff } (\forall\exists(p \wedge \mathsf{X}q) \vee \forall\mathsf{X}r) \wedge \forall\forall p$$
$$\text{iff } (\exists(p \wedge \mathsf{X}q) \vee \forall\mathsf{X}r) \wedge p$$

The same holds for propositions, $\exists\phi'$: $(\exists(p \wedge \mathsf{X}q) \vee \forall\mathsf{X}r) \wedge p$

$$\text{iff } ((\exists p \wedge \exists\mathsf{X}q) \vee \forall\mathsf{X}r) \wedge p \text{ iff } ((p \wedge \exists\mathsf{X}q) \vee \forall\mathsf{X}r) \wedge p$$

This observation lets us simplify the subset of mu-CTL* that we use for sound and complete state checking to the syntax of the *modal-mu-calculus*:

$$p, \neg p \in AtomProp \qquad Z \in Var \qquad \phi \in Prop$$
$$\phi ::= p \mid \neg p \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \forall\mathsf{X}\phi \mid \exists\mathsf{X}\phi$$
$$\mid \mu Z.\phi \mid \nu Z.\phi \mid Z$$

For documentation, the trace semantics of the modal-mu calculus appears in Figure 10 and its corresponding, sound-and-complete state-checking semantics follows in Figure 11.

Using the semantics definitions in those two Figures, we have this result:

Theorem 46 *For all $\pi \in wft(\tau)$, closed $\phi \in modalMuCalculus$,*

$$\pi(0) \in [\![\phi]\!]^\tau\emptyset \text{ iff } \pi \in [\![\phi]\!]^{wft(\tau)}\emptyset$$

$\rho \in Env = Var \to \mathcal{P}(\mathcal{M})$
$[\![\cdot]\!]^{\mathcal{M}} \in Env \to \mathcal{P}(\mathcal{M})$

$$[\![\forall \mathsf{X}\phi]\!]^{\mathcal{M}}\rho = \{\pi \in \mathcal{M} \mid (\mathcal{M} \downarrow \pi(0)) \subseteq [\![\mathsf{X}\phi]\!]^{\mathcal{M}}\rho\}$$
$$= \{\pi \in \mathcal{M} \mid \text{for all } \pi' \in \mathcal{M} \text{ such that } \pi'(0) = \pi(0), \pi'^1 \in [\![\phi]\!]^{\mathcal{M}}\rho\}$$
$$[\![\exists \mathsf{X}\phi]\!]^{\mathcal{M}}\rho = \{\pi \in \mathcal{M} \mid (\mathcal{M} \downarrow \pi(0)) \cap [\![\mathsf{X}\phi]\!]^{\mathcal{M}}\rho \neq \{\}\}$$
$$= \{\pi \in \mathcal{M} \mid \text{exists } \pi' \in \mathcal{M} \text{ such that } \pi'(0) = \pi(0) \text{ and } \pi'^1 \in [\![\phi]\!]^{\mathcal{M}}\rho\}$$

See Figure 2 for the semantics of the remaining syntax constructions .

Fig. 10. trace semantics of the modal-mu calculus

$\rho \in Env = Var \to \mathcal{P}(\Sigma)$
$[\![\phi]\!]^{\tau} \in Env \to \mathcal{P}(\Sigma)$

$$[\![p]\!]^{\tau}\rho = \{a \in \Sigma \mid p \in \mathcal{I}(a)\}$$
$$[\![\neg p]\!]^{\tau}\rho = \{a \in \Sigma \mid \neg p \in \mathcal{I}(a)\}$$
$$[\![\phi_1 \wedge \phi_2]\!]^{\tau}\rho = and([\![\phi_1]\!]^{\tau}\rho, [\![\phi_2]\!]^{\tau}\rho)$$
where $and(S_1, S_2) = S_1 \cap S_2$
$$[\![\phi_1 \vee \phi_2]\!]^{\tau}\rho = or([\![\phi_1]\!]^{\tau}\rho, [\![\phi_2]\!]^{\tau}\rho)$$
where $or(S_1, S_2) = S_1 \cup S_2$
$$[\![\forall \mathsf{X}\phi]\!]^{\tau}\rho = next_{\forall}([\![\phi]\!]^{\tau}\rho)$$
where $next_{\forall}(S) = \{a \in \Sigma \mid \text{for all } a' \in \Sigma, a \to a' \in \tau \text{ implies } a' \in S\}$
$$[\![\exists \mathsf{X}\phi]\!]^{\tau}\rho = next_{\exists}([\![\phi]\!]^{\tau}\rho)$$
where $next_{\exists}(S) = \{a \in \Sigma \mid \text{exists } a' \in \Sigma, a \to a' \in \tau \text{ and } a' \in S\}$
$$[\![\mu Z.\phi]\!]^{\tau}\rho = \bigcap\{S \subseteq \Sigma \mid F(S) \subseteq S\}$$
$$[\![\nu Z.\phi]\!]^{\tau}\rho = \bigcup\{S \subseteq \Sigma \mid S \subseteq F(S)\},$$
where $F(S) = [\![\phi]\!]^{\tau}(\rho + [Z \mapsto S])$
$$[\![Z]\!]^{\tau}\rho = \rho(Z)$$

Fig. 11. state-checking semantics for modal-mu calculus

Implicit in the proof of the above Theorem is the Galois connection, $\langle \alpha^{\bullet} : \mathcal{P}(wft(\tau)) \to \mathcal{P}(\Sigma), \gamma : \mathcal{P}(\Sigma) \to \mathcal{P}(wft(\tau))\rangle$, where $\alpha^{\bullet}(T) = \{s \mid \pi(0) = s, \pi \in T\}$ and $\gamma(S) = \{\pi \mid \pi(0) \in S\}$. That is, $\pi \mathcal{R} s$ iff $\pi(0) = s$.

A close inspection of the trace semantics of the modal-mu calculus shows that it is a logic of purely state checking, because each X-modality must be prefixed by a quantifier. Also, the limitations on expressivity mean that simple

quantified LTL formulas, such as $\forall(\mathsf{FG}p)$ ("for all paths, sometime in the finite future, p generally holds") have no natural translation into the modal-mu calculus— the best one might do is $\forall\mathsf{F}(\forall\mathsf{G}p)$, where $\forall\mathsf{G}p = \nu Z.p \wedge \forall\mathsf{X}Z$ and $\forall\mathsf{F}\phi = \mu Z'.\phi \vee \forall\mathsf{X}Z'$. This formula underapproximates the states that have the desired property.

7 Mixed Transition Systems

The logics mu-CTL* and the modal mu-calculus let us mix universal and existential quantifiers in the same proposition. But if we use these logics to check properties of an abstract model, $wft(\tau_A)$, and we want the validated properties to be sound for the corresponding concrete model, $wft(\tau)$, then we appear to be limited to bisimulations between τ and τ_A (that is, there must exist $\mathcal{R} \subseteq \Sigma \times \Sigma_A$ such that τ is $\mathcal{R}$-simulated by τ_A and τ_A is $\mathcal{R}^{-1}$-simulated by τ).

Dams noted that the bisimulation criterion between τ and τ_A can be weakened so that τ is related to *two* transition relations, τ_{may} and τ_{must}, such that τ is $\mathcal{R}$-simulated by τ_{may} and τ_{must} is $\mathcal{R}^{-1}$-simulated by τ [13]; the resulting abstract system is a *mixed transition system*:

Definition 47 *[13,14]: A* (Kripke) mixed transition system *is a tuple,* $\mathcal{X} = \langle \Sigma_A, \tau_{must}, \tau_{may}, \mathcal{I}_A \rangle$*, where*

- Σ_A *is a set of states and* $\mathcal{I}_A : \Sigma_A \rightarrow \mathcal{P}(AtomProp)$ *is an interpretation mapping, as before;*
- $\tau_{must} \subseteq \Sigma_A \times \Sigma_A$ *and* $\tau_{may} \subseteq \Sigma_A \times \Sigma_A$ *are transition relations.*

The set of traces generated by a mixed transition system, $\mathcal{X}$*, is* $wft(\mathcal{X}) = wft(\tau_{must}) \uplus wft(\tau_{may})$*, where* $\uplus$ *is disjoint union.*

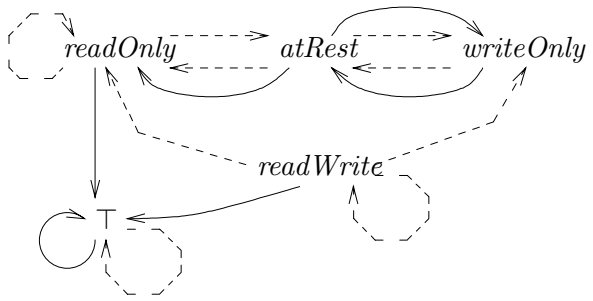
A mixed transition system is always intended to be an abstraction of a concrete transition system. τ_{must} generates traces that *must* appear in the concrete model, and τ_{may} generates a cover of those traces that *may* appear. To be useful, a mixed transition system must be soundly related to a concrete transition system:

Definition 48 *Given a transition system,* $\mathcal{C} = \langle \Sigma, \tau, \mathcal{I} \rangle$ *(see Definition 6), a mixed transition system,* $\mathcal{X} = \langle \Sigma_A, \tau_{must}, \tau_{may}, \mathcal{I}_A \rangle$*, is a* sound abstraction *of* $\mathcal{C}$ *iff there is an abstraction relation,* $\mathcal{R} \subseteq \Sigma \times \Sigma_A$ *such that*

(1) $\mathcal{R}$ *reflects the atomic properties of* $\mathcal{I}$ *in* $\mathcal{I}_A$*;*
(2) τ *is* $\mathcal{R}$*-simulated by* τ_{may}*;*
(3) τ_{must} *is* $\mathcal{R}^{-1}$*-simulated by* τ*.*

When $\mathcal{X}$ is a sound abstraction of $\mathcal{C}$, then all traces generated by τ_{must} are guaranteed nonspurious. Similarly, every trace generated by τ is abstracted by one generated by τ_{may}. Theorem 30 states that we can use τ_{may} to soundly validate universal properties of $\mathcal{C}$, and Theorem 33 states that we can use τ_{must} to soundly validate existential properties of $\mathcal{C}$.

Example 49 *We can assemble a sound mixed transition system for the readers-writer example in Example 4 from the universally sound transition system in Example 31 and the existentially sound transition system in Example 34. Here is the system, where transitions in τ_{must} are drawn as solid lines, and transitions in τ_{may} are drawn as dotted ones:*



Recall that the $\top$ element was added to ensure that all traces generated by τ_{must} were infinite (where $s\ \mathcal{R}\ \top$, for all $s \in \Sigma$), hence, $(\top, \top) \in \tau_{may}$ is added to preserve soundness.

Not only can we compare a mixed transition system to a traditional transition system, we can compare two mixed transition systems for relative precision:

Definition 50 *Mixed transition system $\mathcal{X}_1 = \langle \Sigma_1, \tau_{must1}, \tau_{may_1}, \mathcal{I}_1 \rangle$* refines *$\mathcal{X}_2 = \langle \Sigma_2, \tau_{must2}, \tau_{may_2}, \mathcal{I}_2 \rangle$ iff there is an abstraction relation, $\mathcal{R} \subseteq \Sigma_1 \times \Sigma_2$, such that*

(1) $\mathcal{R}$ reflects the atomic properties of $\mathcal{I}_1$ in $\mathcal{I}_2$;
(2) τ_{may_1} is $\mathcal{R}$-simulated by τ_{may_2};
(3) τ_{must2} is $\mathcal{R}^{-1}$-simulated by τ_{must1}.

If $\mathcal{X}_1$ refines $\mathcal{X}_2$, then the former is more precise than the latter about certainties (more of them) and possibilities (less of them) in traces. A most precise mixed transition system is *concrete*:

Definition 51 $\langle \Sigma, \tau_{must}, \tau_{may}, \mathcal{I} \rangle$ *is* concrete *when* $\tau_{must} = \tau_{may}$.

When $\mathcal{X}_1$ is a concrete MTS, then it is a traditional transition system (Definition 6), and $\mathcal{X}_1$ refines $\mathcal{X}_2$ means that $\mathcal{X}_2$ soundly abstracts $\mathcal{X}_1$ (Definition 48). When τ_{must2} of $\mathcal{X}_2$ is the empty set, then refinement degenerates to the simulation used for proving universal soundness of ordinary abstract transition

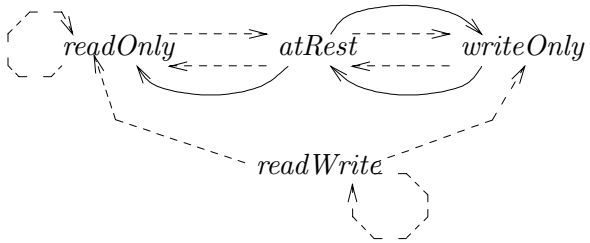
systems.

The notion of refinement of a mixed transition system was originally proposed by Larsen and Thomson for loose system specification [28,29], where stepwise refinement is used to transform an imprecise mixed transition-system specification into a concrete one. Larsen also proposed a restriction to mixed transition systems that ensures internal consistency:

Definition 52 *When $\tau_{must} \subseteq \tau_{may}$, a mixed transition system, $\langle \Sigma, \tau_{must}, \tau_{may}, \mathcal{I} \rangle$, is a* modal transition system.

It is easy to prove that every modal transition system has a refinement to a concrete system, whereas this is not the case for all mixed transition systems.

The mixed transition system in Example 49 can be made into a useful model transition system by deleting those transitions in τ_{must} that are not already present in τ_{may}:



This makes the $\top$ state unnecessary, and we delete it. Because $\tau_{must} \subseteq \tau_{may}$, the system's model can be defined as $wft(\mathcal{X}) = wft(\tau_{may} \cup \tau_{must})$ and this equals $wft(\tau_{may})$. This might suggest that τ_{must} is unimportant, but this is hardly the case, because τ_{must} assigns a degree of extra confidence to transitions in a trace: A trace, $s_0 s_1 s_2 \cdots s_i s_{i+1} \cdots$, can be annotated with the certainty of knowledge about its transitions, e.g.,

$$s_0 \xrightarrow{must} s_1 \xrightarrow{must} s_2 \cdots s_i \xrightarrow{may} s_{i+1} \cdots$$

We exploit this idea later.

7.1 Checking properties of mixed transition systems

We can adapt the mu-CTL* logic to checking properties of mixed transition systems. For mixed transition system, $\mathcal{X} = \langle \Sigma_A, \tau_{must}, \tau_{may}, \mathcal{I}_A \rangle$, recall that its model is $wft(\mathcal{X}) = wft(\tau_{may}) \uplus wft(\tau_{must})$. The trace semantics for mu-CTL* appears in Figure 12. The semantics for the quantifiers are a bit awkward, but

Let $\mathcal{M} = wft(\tau_{may}) \uplus wft(\tau_{must})$.
Define $\rho \in Env = Var \to \mathcal{P}(\mathcal{M})$
and $[\![\cdot]\!]^{\mathcal{M}} \in Env \to \mathcal{P}(\mathcal{M})$:

$$[\![\forall\phi]\!]^{\mathcal{M}}\rho = \{\pi \in \mathcal{M} \mid (wft(\tau_{may}) \downarrow \pi(0)) \subseteq [\![\phi]\!]^{\mathcal{M}}\rho\}$$
$$[\![\exists\phi]\!]^{\mathcal{M}}\rho = \{\pi \in \mathcal{M} \mid (wft(\tau_{must}) \downarrow \pi(0)) \cap [\![\phi]\!]^{\mathcal{M}}\rho \neq \{\}\}$$

See Figure 2 for the semantics of the remaining syntax constructions .

Fig. 12. mu-CTL* semantics for mixed transition systems

$\rho \in Env = Var \to \mathcal{P}(\Sigma)$
$[\![\phi]\!]^{\tau_{may},\tau_{must}} \in Env \to \mathcal{P}(\Sigma)$

$$[\![\forall \mathsf{X}\phi]\!]^{\tau_{may},\tau_{must}}\rho = next_{\tau_{may}}([\![\phi]\!]^{\tau_{may},\tau_{must}}\rho)$$
where $next_{\tau_{may}}(S) = \{a \in \Sigma \mid \text{for all } a' \in \Sigma, a \to a' \in \tau_{may} \text{ implies } a' \in S\}$
$$[\![\exists \mathsf{X}\phi]\!]^{\tau_{may},\tau_{must}}\rho = next_{\tau_{must}}([\![\phi]\!]^{\tau_{may},\tau_{must}}\rho)$$
where $next_{\tau_{must}}(S) = \{a \in \Sigma \mid \text{exists } a' \in \Sigma, a \to a' \in \tau_{must} \text{ and } a' \in S\}$

The remaining semantics clauses for $[\![\phi]\!]^{\tau_{may},\tau_{must}}$ are taken from Figure 11.

Fig. 13. state-checking semantics for modal-mu calculus on mixed transition systems

they expose that a quantified proposition must be checked on the appropriate subsystem of the mixed system [8].

The two simulations of the mixed transition system do the same work as a bisimulation, meaning that the proof of Theorem 45 applies here as well:

Theorem 53 *For transition system, $\mathcal{C} = \langle \Sigma, \tau, \mathcal{I} \rangle$ and mixed transition system, $\mathcal{X} = \langle \Sigma_A, \tau_{must}, \tau_{may}, \mathcal{I}_A \rangle$, assume $\mathcal{C}$ is soundly abstracted by $\mathcal{X}$ by means of abstraction relation, $\mathcal{R} \subseteq \Sigma \times \Sigma_A$. Then, for all $\pi \in wft(\tau)$, $\kappa \in \mathcal{M} = wft(\tau_{must}) \uplus wft(\tau_{may})$, and closed $\phi \in$ mu-CTL*,*

$$\pi \; \mathcal{R}^{\omega} \; \kappa \text{ and } \kappa \in [\![\phi]\!]^{\mathcal{M}}\emptyset \text{ imply } \pi \in [\![\phi]\!]^{wft(\tau)}\emptyset$$

We can continue the development and obtain a completeness result for state checking, analogous to Theorem 46. The semantics for state checking modal mu-calculus properties on mixed transition systems is defined in Figure 13. This semantics, which is a simple adaptation of the one in Figure 11, gives us the completeness result:

Let $\mathcal{M} = wft(\tau_{may} \cup \tau_{must})$. Define $\rho \in Env = Var \to \mathcal{P}(\mathcal{M})$ and $[\![\cdot]\!]^{\mathcal{M}}_m \in Env \to \mathcal{P}(\mathcal{M})$, for $m \in \{must, may\}$:

$$[\![\mathsf{X}\phi]\!]^{\mathcal{M}}_{may}\rho = next_{may}([\![\phi]\!]^{\mathcal{M}}\rho)$$
$$[\![\mathsf{X}\phi]\!]^{\mathcal{M}}_{must}\rho = next_{must}([\![\phi]\!]^{\mathcal{M}}\rho)$$

where $next_m : \mathcal{P}(\mathcal{M}) \to \mathcal{P}(\mathcal{M})$ is defined as

$$next_m(S) = \{\pi \in \mathcal{M} \mid \pi(0) \to \pi(1) \in \tau_m \text{ and } \pi^1 \in S\}$$

$$[\![\forall\phi]\!]^{\mathcal{M}}_m\rho = \{\pi \in \mathcal{M} \mid (\mathcal{M} \downarrow \pi(0)) \subseteq [\![\phi]\!]^{\mathcal{M}}_{may}\rho\}$$
$$[\![\exists\phi]\!]^{\mathcal{M}}_m\rho = \{\pi \in \mathcal{M} \mid (\mathcal{M} \downarrow \pi(0)) \cap [\![\phi]\!]^{\mathcal{M}}_{must}\rho \neq \{\}\}$$

See Figure 2 for the semantics of the remaining syntax constructions, where $[\![\phi]\!]^{\mathcal{M}}_m = [\![\phi]\!]^{\mathcal{M}}$, for $m \in \{may, must\}$

Fig. 14. mu-CTL* semantics for modal transition systems

Theorem 54 *For all* $\pi \in \mathcal{M} = wft(\tau_{must}) \uplus wft(\tau_{may})$*, closed* $\phi \in modalMuCalc$*,*

$$\pi(0) \in [\![\phi]\!]^{\tau_{may},\tau_{must}}\emptyset \text{ iff } \pi \in [\![\phi]\!]^{\mathcal{M}}\emptyset$$

In the case that the mixed transition system is a modal transition system, then the trace semantics of mu-CTL* should be changed to reflect that a trace is a mix of must- and may-transitions. A valuation, $[\![\phi]\!]^{\mathcal{M}}_m\rho$ calculates those traces that possess property ϕ to a degree of confidence, $m \in \{may, must\}$; see Figure 14.

The usual soundness theorem holds for this semantics, and the restriction of the semantics to the modal-mu calculus gives the usual completeness result.

7.2 A logic tailored to modal-transition systems

We can better exploit the structure of modal-transition systems by slightly altering the syntax of mu-CTL*: We annotate the X-modality so that it indicates the degree of confidence we have about the next transition in a trace. $\mathsf{X}_{must}\phi$ asserts that the next transition *must* make ϕ hold, whereas $\mathsf{X}_{may}\phi$ asserts that the transition *may* make ϕ hold. The annotated X-modality lets us write assertions like this example:

$$\forall(\mathsf{X}_{may}(unstable \supset \mathsf{F}_{must}stable)), \text{ where } \mathsf{F}_{must}\phi = \mu Z.\phi \vee \mathsf{X}_{must}Z$$

That is, for all paths starting from a state, if the path possibly leads to an unstable state at the end of the first transition, then that same path must

Let $\mathcal{M} = wft(\tau_{may} \cup \tau_{must})$. Define $\rho \in Env = Var \to \mathcal{P}(\mathcal{M})$ and $[\![\cdot]\!]^{\mathcal{M}} \in Env \to \mathcal{P}(\mathcal{M})$:

$$[\![\mathsf{X}_m \phi]\!]^{\mathcal{M}} \rho = next_m([\![\phi]\!]^{\mathcal{M}} \rho), \text{ for } m \in \{may, must\}$$
$$\text{where } next_m(S) = \{\pi \in \mathcal{M} \mid \pi(0) \to \pi(1) \in \tau_m \text{ and } \pi^1 \in S\}$$

The remainder of mu-XCTL* matches mu-CTL*: See Figure 9 for the semantics of $\forall\phi$ and $\exists\phi$, and see Figure 2 for the semantics of the remaining constructions.

Fig. 15. mu-XCTL* semantics for modal-transition systems

continue to a stable state. The annotated X-modality provides an ease of description that goes beyond mu-CTL*, and we call the resulting logic, *mu-XCTL**.

Figure 15 displays the semantics of mu-XCTL*, which is merely mu-CTL* plus the annotated X. For a concrete transition system, that is, when $\tau_{may} = \tau_{must}$, $[\![\mathsf{X}_{must}\phi]\!]^{\mathcal{M}} = [\![\mathsf{X}_{may}\phi]\!]^{\mathcal{M}}$, and mu-XCTL* becomes mu-CTL*.

The example mu-XCTL* formula seen a few lines earlier included an implication operator. If we understand $\phi \supset \phi'$ as $\neg\phi \vee \phi'$, then we must devise translation rules for shifting negations across annotated X-modalities; the results are striking:

$$\neg\mathsf{X}_{may}\phi \Rightarrow \mathsf{X}_{must}\neg\phi$$
$$\neg\mathsf{X}_{must}\phi \Rightarrow \mathsf{X}_{may}\neg\phi$$

For the semantics in Figure 15, we can extend Proposition 5 to show that negation shifting is sound for the above rules. Theorem 53 can be reproved also, but the free-wheeling use of existential quantification with X_{may} means that Theorem 54 must be rephrased.

Mu-XCTL* appears to be a useful linear-time logic for modal-transition systems, but it has not been intensively studied at the time of this writing.

8 Conclusion

Thanks to the application of stepwise abstract interpretation, the gulf between linear-time and branching-time logic has been bridged. Also, the duality uncovered when formulating abstractions for sound universal and existential property checking has motivated mixed- and modal-transition systems, where two abstract models, rather than one, support analyses of interest.

Several outstanding issues remain, however:

- Galois connections between concrete state sets, Σ, and abstract state sets, Σ_A, play the key role in synthesizing sound and complete abstract semantics of linear-time and branching-time logics. But the formalization of $\mathcal{R}$- and $\mathcal{R}^{-1}$-simulations, which induce the may- and must-transition relations for a mixed transition system, do not have an obvious formulation as a Galois connection. What is the correct domain of transition systems so that there are appropriate Galois connections that define simulations between transition systems?
- The restriction that $\tau_{must} \subseteq \tau_{may}$ of a modal transition system gives strong logical properties that go beyond those of a mixed transition system [24]. These properties have been expressed in terms of branching-time logic, but it is likely that they have stronger expression in linear-time logic and mu-XCTL*; this should be investigated.
- The modal-transition system framework should, in principle, support intuitionistic logic rather than the three-valued logic given by a mixed transition system [24]. (This would provide a range of possible worlds, m, for annotating X_m in mu-XCTL*.) Huth has proposed an intuitionistic version of modal transition system, based on a framework of Fitting [20], but there is no obvious notion of refinement for his proposal.

These problems appear to be related, and a solution to one might well yield a solution to one or both of the others.

9 Acknowledgements

Yoshiki Kinoshita, John Power, and the other organizers of the Workshop on Structure-Preserving Relations are thanked for their encouragement. Michael Huth has participated in research related to the material presented in this paper and is thanked for his interest.

References

[1] A. Aho, R. Sethi, and J. Ullman. *Compilers: Principles, Techniques, and Tools.* Addison Wesley, 1986.

[2] B. Banieqbal and H. Barringer. Temporal logic with fixed points. In B. Banieqbal, H. Barringer, and A. Pnueli, editors, *Temporal logic in specifications*, number 398 in Lecture Notes in Computer Science, pages 62–74. Springer-Verlag, 1897.

[3] E. Clarke and I.A. Draghicescu. Expressibility results for linear-time and branching-time logics. In J.W. deBakker, W.P. deRoever, and G. Rozenberg, editors, *Proc. Workshop on Linear Time, Branching Time, and Order in Logics and Models for Concurrency*, pages 257–268. Springer LNCS 354, 1988.

[4] E. Clarke, E. Emerson, and A. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8:244–263, 1986.

[5] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *Proc. Computer-Aided Verification 2000*, Lecture Notes in Computer Science. Springer, 2000.

[6] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[7] E.M. Clarke, O. Grumberg, and D.E. Long. Verification tools for finite-state concurrent systems. In J.W. deBakker, W.-P. deRoever, and G. Rozenberg, editors, *A Decade of Concurrency: Reflections and Perspectives*, number 803 in Lecture Notes in Computer Science, pages 124–175. Springer, 1993.

[8] R. Cleaveland, P. Iyer, and D. Yankelevich. Optimality in abstractions of model checking. In *SAS'95: Proc. 2d. Static Analysis Symposium*, Lecture Notes in Computer Science 983, pages 51–63. Springer, 1995.

[9] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs. In *Proc. 4th ACM Symp. on Principles of Programming Languages*, pages 238–252. ACM Press, 1977.

[10] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Proc. 6th ACM Symp. on Principles of Programming Languages*, pages 269–282. ACM Press, 1979.

[11] P. Cousot and R. Cousot. Abstract interpretation frameworks. *Journal of Logic and Computation*, 2(4):511–547, 1992.

[12] P. Cousot and R. Cousot. Temporal abstract interpretation. In *Proc. 27th ACM Symp. on Principles of Programming Languages*, pages 12–25. ACM Press, 2000.

[13] D. Dams. *Abstract interpretation and partition refinement for model checking*. PhD thesis, Technische Universiteit Eindhoven, The Netherlands, 1996.

[14] D. Dams, R. Gerth, and O. Grumberg. Abstract interpretation of reactive systems. *ACM TOPLAS*, 19:253–291, 1997.

[15] P. Diembiński, W. Penczek, and A. Pólrola. Automated verification of infinite state concurrent systems: an improvement in model generation. In *Proc. PPAM'01*. Springer LNCS, 2001.

[16] A. Emerson. Temporal and modal logic. In J. vanLeeuwen, editor, *Handbook of Theoretical Computer Science: Volume B*, pages 995–1072. Elsevier, 1990.

[17] A. Emerson and J. Halpern. Sometimes and not never revisited: on branching versus linear time. *J. ACM*, 33:151–178, 1986.

[18] E.A. Emerson and C.L. Lei. Modalities for model checking: Branching time logic strikes back. In *Proc. 20th ACM Symp. Principles of Prog. Lang.*, pages 84–96. ACM Press, 1985.

[19] E.A. Emerson and C.L. Lei. Efficient model checking in fragments of the propositional mu-calculus. In *First Annual Symposium on Logic in Computer Science*, pages 267–278. IEEE, 1986.

[20] M. Fitting. Tableaus for many-valued modal logics. *Studia Logica*, 55:63–87, 1992.

[21] R. Giacobazzi and E. Quintarelli. Incompleteness, counterexamples and refinements in abstract model-checking. In P. Cousot, editor, *Proc. Static Analysis Symposium*. Springer LNCS, 2001.

[22] S. Graf and H. Saidi. Verifying invariants using theorem proving. In *Conference on Computer Aided Verification CAV'96*, LNCS 1102, Springer Verlag, 1996.

[23] M. Hecht. *Flow Analysis of Computer Programs*. Elsevier, 1977.

[24] M. Huth, R. Jadadeesan, and D. Schmidt. Modal transition systems: a foundation for three-valued program analysis. In *Proc. European Symposium on Programming 2001*, Lecture Notes in Computer Science. Springer, 2001.

[25] N. Jones and F. Nielson. Abstract interpretation: a semantics-based tool for program analysis. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science, Vol. 4*, pages 527–636. Oxford Univ. Press, 1995.

[26] J. Kam and J. Ullman. Global data flow analysis and iterative algorithms. *J. ACM*, 23:158–171, 1976.

[27] L. Lamport. Sometimes is sometimes "not never"–on the temporal logic of programs. In *Proc. 7th ACM Symp. Principles of Prog. Lang.*, pages 174–185. ACM Press, 1980.

[28] K. Larsen. Modal specifications. In J. Sifakis, editor, *CAV'89*, number 407 in Lecture Notes in Computer Science, pages 232–246. Springer-Verlag, 1989.

[29] K. Larsen and B. Thomsen. A modal process logic. In *Third IEEE Symp. Logic in Computer Science*, pages 203–210. IEEE Press, 1988.

[30] O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their specifications. In *12th ACM Symp. on Principles of Programming Languages*, pages 97–107, New Orleans, 1985.

[31] A. Melton, G. Strecker, and D. Schmidt. Galois connections and computer science applications. In *Category Theory and Computer Programming*, pages 299–312. Lecture Notes in Computer Science 240, Springer-Verlag, 1985.

[32] R. Milner. An algebraic definition of simulation between programs. In *Proc. 2nd International Joint Conf. on Artificial Intelligence*. British Computer Society, 1971.

[33] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[34] M. Müller-Olm, D.A. Schmidt, and B. Steffen. Model checking: A tutorial introduction. In G. Filé and A. Cortesi, editors, *Proc. 6th Static Analysis Symposium*. Springer LNCS, 1999.

[35] F. Nielson. Expected forms of data flow analysis. In N. D. Jones, editor, *Programs as Data Objects*, pages 172–191. Lecture Notes in Computer Science 217, Springer-Verlag, 1986.

[36] D. Park. Concurrency and automata in infinite strings. Lecture Notes in Computer Science 104, pages 167–183. Springer, 1981.

[37] A. Pnueli. The temporal logic of programs. In *Proc. 18th IEEE Symp. on Foundations of Computer Science*, pages 46–57, 1977.

[38] D.A. Schmidt. Natural-semantics-based abstract interpretation. In A. Mycroft, editor, *Static Analysis Symposium*, number 983 in Lecture Notes in Computer Science, pages 1–18. Springer-Verlag, 1995.

[39] D.A. Schmidt. Data-flow analysis is model checking of abstract interpretations. In *Proc. 25th ACM Symp. on Principles of Prog. Languages*. ACM Press, 1998.

[40] D.A. Schmidt. Binary relations for abstraction and refinement. *Workshop on Refinement and Abstraction, Amagasaki, Japan, Nov. 1999. Elsevier Electronic Notes in Computer Science*, to appear.

[41] D.A. Schmidt and B. Steffen. Data-flow analysis as model checking of abstract interpretations. In G. Levi, editor, *Proc. 5th Static Analysis Symposium*. Springer LNCS 1503, 1998.

[42] A.P. Sistla and E.M. Clarke. The complexity of propositional linear temporal logic. *J. ACM*, 32:733–749, 1985.

[43] B. Steffen. Data flow analysis as model checking. In A. Meyer, editor, *Theoretical Aspects of Computer Software: TACS'91*, volume 526 of *Lecture Notes in Computer Science*. Springer-Verlag, 1991.

[44] B. Steffen. Generating data-flow analysis algorithms for modal specifications. *Science of Computer Programming*, 21:115–139, 1993.

[45] C. Stirling. Modal and temporal logics. In S. Abramsky, D. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, pages 477–563. Oxford University Press, 1992.

[46] C. Stirling and D. Walker. Local model checking in the modal mu-calculus. In J. Diaz and F. Orejas, editors, *Proc. TAPSOFT '89*, volume 351 of *Lecture Notes in Computer Science*, pages 369–383. Springer-Verlag, 1989.

[47] W. Thomas. A combinatorial approach to the theory of ω-automata. *Information and Computation*, 48:261–283, 1981.

[48] W. Thomas. Automata on infinite objects. In *Handbook of Theoretical Computer Science, Vol. 2*, pages 165–191. MIT Press, 1990.

[49] M. Vardi. A temporal fixpoint calculus. In *Proc. 15th ACM Symp. Principles Prog. Lang.*, pages 250–259, 1988.

[50] M. Vardi. Branching vs. linear time: final showdown. In *ETAPS'01*. Springer-Verlag, Lecture Notes in Computer Science, `http://www.cs.rice.edu/~vardi/papers/index.html`, 2001.

[51] M. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. 1st Symp. Logic in Computer Science*, pages 332–344. IEEE, 1986.

[52] P. Wolper. Temporal logic can be more expressive. *Information and Control*, 56:72–99, 1983.