Modal Transition Systems: A Foundation for Three-Valued Program Analysis

Michael Huth[1], Radha Jagadeesan[⋆2], and David Schmidt[⋆⋆1]

[1] Computing and Information Sciences, Kansas State University,
{huth,schmidt}@cis.ksu.edu, WWW home page:
http://www.cis.ksu.edu/{~huth,~schmidt}
[2] Department of Mathematics and Computer Science, Loyola University of Chicago,
radha@cs.luc.edu, WWW home page: http://www.cs.luc.edu/~radha

Abstract. We present *Kripke modal transition systems* (Kripke MTSs), a generalization of modal transition systems [27,26], as a foundation for three-valued program analysis. The semantics of Kripke MTSs are presented by means of a mixed power domain of states; soundness and consistency are proved. Two major applications, model checking partial state spaces and three-valued program shape analysis, are presented as evidence of the suitability of Kripke MTSs as a foundation for three-valued analyses.

1 Introduction

A *modal transition system* (MTS) [27,26] labels each of its state transitions with a modality — *may* or *must* — expressing transition behaviors that *(i) necessarily* occur (*must* modality), *(ii) possibly* occur (*may* modality), and *(iii) not possibly* occur (*absence* of a transition). Figure 1 shows an example MTS — a specification of a slot machine, where some behaviors of the final implementation are fixed (the *must*-transitions) and some are uncertain (the *may*-transitions).

Conventional state-transition modellings are over-approximations made by adding more computation paths [8,7], thereby limiting validation to safety properties ("nothing bad will happen"). MTSs, however, perform both over- and under-approximation, admitting both safety *and* liveness properties ("something good will happen") to be deduced. As a bonus, the outcomes of analyses of MTSs are three-valued, meaning that validation, refutation, and conditional reasoning can be undertaken in the framework. Abstractions of both control *and* data can be modelled with MTSs.

The paper proceeds as follows: Section 2 reviews doubly labeled transition systems [12] and characterizes their behaviors by logics of "liveness" and "safety." Section 3 introduces *Kripke MTSs*, their refinement relation, and their semantics for the modal mu-calculus [25]. Sections 4 and 5 show how to apply Kripke MTSs to two analyses that rely on three-valued logic: *(i)* Bruns and Godefroid's partial

⋆ Supported by NSF CCR-9901071.
⋆⋆ Supported by NSF CCR-9970679 and INT-9981558.

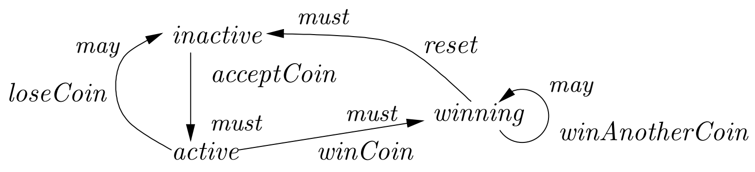


Fig. 1. Slot machine specification in modal transition format

Kripke structures [3] and extended transition systems [29,39] and *(ii)* the pointer shape-graph analysis of Sagiv, Reps, and Wilhelm [35]. Section 6 concludes.

2 Doubly Labeled Transition Systems

We begin with the definition of a *doubly labeled transition system* [12]:

Definition 1 (Doubly labeled transition systems). *A* doubly labeled transition system *(DLTS), $\mathcal{K}$, is a tuple $(\Sigma_K, \mathtt{Act}, \mathtt{AP}, \longrightarrow, L)$, where Σ_K is a set of states, $\mathtt{Act}$ is a (countable) set of action symbols, $\mathtt{AP}$ is a (countable) set of atomic propositions, $\longrightarrow$ is a transition relation that is a subset of $\Sigma_K \times \mathtt{Act} \times \Sigma_K$, and L is a labeling function $L\colon \Sigma_K \to \mathcal{P}(\mathtt{AP})$. We call $\mathcal{K}$* finitely-branching *if for each $s \in \Sigma_K$ and $a \in \mathtt{Act}$, the sets $L(s)$ and $\{s' \in \Sigma_K \mid s \to^a s'\}$ are finite.* [1]

As Figure 2 shows, each state is annotated with the set of primitive properties that hold for it. Behaviors are compared by means of *simulations*:

Definition 2 (Simulation). *Let $\mathcal{C}$ and $\mathcal{A}$ be doubly labeled transition systems, where for simplicity $\mathtt{Act}_C = \mathtt{Act}_A$. A relation, $Q \subseteq \Sigma_C \times \Sigma_A$, is a* simulation *if, for all $s \in \Sigma_C, t \in \Sigma_A$, $Q(s,t)$ holds iff: for all $a \in \mathtt{Act}_C$ and $s' \in \Sigma_C$ with $s \to^a s'$, there is some $t' \in \Sigma_A$ with $t \to^a t'$ and $Q(s', t')$.*

Given $\mathcal{C}$ and $\mathcal{A}$, we can compute the greatest simulation, $\prec$, on $\Sigma_C \times \Sigma_A$, a preorder, by a standard fixed-point argument. The intuition behind a simulation is that a transition made by $\mathcal{C}$ can be "mimicked" by one in $\mathcal{A}$. In practice, one of $\mathcal{C}$ or $\mathcal{A}$ is an "implementation" and the other is its "abstraction" or "specification" or "model," which must be analyzed for correctness properties. As noted in [36], there are natural connections between simulations and Galois connections [8] on such transition systems.

Live Simulations. Simulations should respect atomic properties. A simulation, $Q \subseteq \Sigma_C \times \Sigma_A$, is a *live simulation* if $Q(s,t)$ implies $L(s) \subseteq L(t)$, for all $s \in \Sigma_C$ and $t \in \Sigma_A$. It is easy to prove that there is a greatest live simulation, $\prec_{live}$, on $\Sigma_C \times \Sigma_A$. There is a crucial connection between live simulations and modal logic:

[1] Making $L(s)$ finite as well prevents inconsistencies if we convert state propositions into action labels.

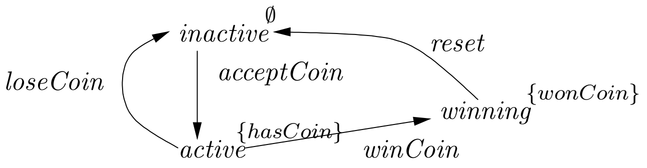


Fig. 2. Slot machine implementation as a doubly labeled transition system

Consider the following modal logic, $L_{\mathtt{pos}}$, which expresses liveness or "possibility" properties [33], where $p \in \mathtt{AP}$ and $a \in \mathtt{Act}$:

$$\phi ::= \top \mid p \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \langle a \rangle \, \phi \tag{1}$$

The diamond modality denotes the possibility of an a-transition. For a DLTS, $\mathcal{K}$, we define $\llbracket \phi \rrbracket \subseteq \Sigma_K$ by induction on the grammar for ϕ:

$\llbracket \top \rrbracket \stackrel{\text{def}}{=} \Sigma_K$,
$\llbracket p \rrbracket \stackrel{\text{def}}{=} \{s \in \Sigma_K \mid p \in L(s)\}$,
$\llbracket \phi_1 \wedge \phi_2 \rrbracket \stackrel{\text{def}}{=} \llbracket \phi_1 \rrbracket \cap \llbracket \phi_2 \rrbracket$,
$\llbracket \phi_1 \vee \phi_2 \rrbracket \stackrel{\text{def}}{=} \llbracket \phi_1 \rrbracket \cup \llbracket \phi_2 \rrbracket$,
$\llbracket \langle a \rangle \, \phi \rrbracket \stackrel{\text{def}}{=} \{s \in \Sigma_K \mid \text{for some } s',\ s \rightarrow^a s' \text{ and } s' \in \llbracket \phi \rrbracket\}$.

Proposition 1 (Logical characterization). *[18] Let $\mathcal{C}$ and $\mathcal{A}$ be finitely-branching DLTSs and $s \in \Sigma_C$, $t \in \Sigma_A$. Then $s \prec_{live} t$ iff for all $\phi \in L_{\mathtt{pos}}$, $[s \in \llbracket \phi \rrbracket \Rightarrow t \in \llbracket \phi \rrbracket]$.*

Thus, to calculate liveness properties of an implementation, $\mathcal{A}$, we construct a model, $\mathcal{C}$, and calculate the greatest live simulation. Then, liveness properties that are deduced to hold for $\mathcal{C}$'s states will hold for the corresponding states in $\mathcal{A}$. Dually, we might model an implementation, $\mathcal{C}$, by an abstract model, $\mathcal{A}$, and use the latter to *refute* liveness properties of $\mathcal{C}$.

Safe simulations. The dual of a live simulation is a *safe* one: a simulation, $Q \subseteq \Sigma_C \times \Sigma_A$, is *safe* if $Q(s,t)$ implies $L(s) \supseteq L(t)$, for all $s \in \Sigma_C$ and $t \in \Sigma_A$. There is a greatest safe simulation, $\prec_{safe}$, on $\Sigma_C \times \Sigma_A$. The logic $L_{\mathtt{nec}}$ expresses safety or "necessarily" properties [33], where $p \in \mathtt{AP}$ and $a \in \mathtt{Act}$:

$$\phi ::= \top \mid p \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid [a]\phi \tag{2}$$

We define $\llbracket \phi \rrbracket \subseteq \Sigma_K$ for the first four clauses in the same manner as for the logic $L_{\mathtt{pos}}$, and we define $\llbracket [a]\phi \rrbracket \stackrel{\text{def}}{=} \{s \in \Sigma_K \mid \text{for all } s',\ s \rightarrow^a s' \text{ implies } s' \in \llbracket \phi \rrbracket\}$ as the meaning of the box modality.

Proposition 2 (Logical characterization). *Let $\mathcal{C}$ and $\mathcal{A}$ be finitely-branching DLTSs and $s \in \Sigma_C$, $t \in \Sigma_A$. Then $s \prec_{safe} t$ iff for all $\phi \in L_{\mathtt{nec}}$, $[t \in \llbracket \phi \rrbracket \Rightarrow s \in \llbracket \phi \rrbracket]$.*

Thus, to calculate safety properties of an implementation, $\mathcal{C}$, we construct $\mathcal{A}$ and calculate a safe simulation. Then, safety properties that hold for $\mathcal{A}$'s states will hold for the corresponding states in $\mathcal{C}$. (This is the standard approach in *abstract interpretation* studies [8,36].) Dually, we might model an implementation, $\mathcal{A}$, by an abstract model, $\mathcal{C}$, and use the latter to *refute* safety properties of $\mathcal{A}$.

3 Modal Transition Systems

Kripke MTSs allow us to freely combine safety *and* liveness properties in property validation *and* refutation. An MTS's "loose" transitions — that is, transitions that may or may not be present in the final implementation — are labeled as *may-transitions*, and "tight" transitions, which must be preserved in the final implementation, are labeled as *must-transitions* [27]. Review Figure 1. With this intuition, every must-transition is by definition a may-transition. These ideas also apply to the atomic properties that label an MTS's states, giving us the modal version of DLTSs, which we call a *Kripke MTS*:

Definition 3 (Kripke MTS). *A* Kripke MTS *is a tuple* $\mathcal{K} = \langle \Sigma_K, \mathtt{Act}, \mathtt{AP}, \overset{must}{\longrightarrow}, \overset{may}{\longrightarrow}, L^{must}, L^{may}\rangle$*, where both* $\langle \Sigma_K, \mathtt{Act}, \mathtt{AP}, \overset{must}{\longrightarrow}, L^{must}\rangle$ *and* $\langle \Sigma_K, \mathtt{Act}, \mathtt{AP}, \overset{may}{\longrightarrow}, L^{may}\rangle$ *are DLTSs with* $\overset{must}{\longrightarrow} \subseteq \overset{may}{\longrightarrow}$ *and* $L^{must}(s) \subseteq L^{may}(s)$*, for all* $s \in \Sigma_K$*.*

Note the pairings: $\overset{must}{\longrightarrow}$ and L^{must} are paired, because they define a system of transitions and properties that *must* be preserved in any implementation of the Kripke MTS; $\overset{may}{\longrightarrow}$ and L^{may} are paired, because they define a system of transitions and properties that *may* be preserved in any implementation. For a Kripke MTS $\mathcal{C}$ to be a refinement of a Kripke MTS $\mathcal{A}$, it must preserve all must-aspects of $\mathcal{A}$ and it may selectively discard $\mathcal{A}$'s may-aspects:

Definition 4 (Refinement). *A* refinement *between Kripke MTSs* $\mathcal{C}$ *and* $\mathcal{A}$ *is a relation* $Q \subseteq \Sigma_C \times \Sigma_A$ *such that, for all* $s \in \Sigma_C$ *and* $t \in \Sigma_A$*, if* $Q(s,t)$*, then*

1. *if* $t \rightarrow_a^{must} t'$*, then for some* $s' \in \Sigma_C$*,* $s \rightarrow_a^{must} s'$ *and* $Q(s',t')$*;*
2. *if* $s \rightarrow_a^{may} s'$*, then for some* $t' \in \Sigma_A$*,* $t \rightarrow_a^{may} t'$ *and* $Q(s',t')$*;*
3. $L^{must}(t) \subseteq L^{must}(s)$*; and*
4. $L^{may}(s) \subseteq L^{may}(t)$*.*

A Kripke MTS such that $\overset{must}{\longrightarrow} = \overset{may}{\longrightarrow}$ and $L^{must} = L^{may}$ is *concrete*, that is, it is a doubly labeled transition system [12], a "final implementation." As usual, for Kripke MTSs $\mathcal{C}$ and $\mathcal{A}$, there is a greatest refinement relation $\prec_r$.

Next, consider the logic $\mathcal{L}$:

$$\phi ::= \top \mid p \mid \phi_1 \wedge \phi_2 \mid \langle a \rangle \phi \mid [a]\phi \tag{3}$$

where $p \in \mathtt{AP}$ and $a \in \mathtt{Act}$.

Definition 5 (Semantics of modal logic). *For a Kripke MTS $\mathcal{K}$ and any $\phi \in \mathcal{L}$, we define a semantics $\llbracket \phi \rrbracket \in \mathcal{P}(\Sigma_K) \times \mathcal{P}(\Sigma_K)$, where $\mathcal{P}(\Sigma_K)$ is the powerset of Σ_K, ordered by set inclusion, and $\llbracket \phi \rrbracket^{\mathrm{nec}}$ and $\llbracket \phi \rrbracket^{\mathrm{pos}}$ are the projection of $\llbracket \phi \rrbracket$ to its first and second component, respectively:*

1. $\llbracket \top \rrbracket \stackrel{\mathrm{def}}{=} \langle \Sigma_K,\ \Sigma_K \rangle$;
2. $\llbracket p \rrbracket \stackrel{\mathrm{def}}{=} \langle \{s \in \Sigma_K \mid p \in L^{must}(s)\},\ \{s \in \Sigma_K \mid p \in L^{may}(s)\} \rangle$;
3. $\llbracket \phi_1 \wedge \phi_2 \rrbracket \stackrel{\mathrm{def}}{=} \langle \llbracket \phi_1 \rrbracket^{\mathrm{nec}} \cap \llbracket \phi_2 \rrbracket^{\mathrm{nec}},\ \llbracket \phi_1 \rrbracket^{\mathrm{pos}} \cap \llbracket \phi_2 \rrbracket^{\mathrm{pos}} \rangle$;
4. $\llbracket \langle a \rangle \phi \rrbracket \stackrel{\mathrm{def}}{=} \langle \{s \in \Sigma_K \mid \textit{for some } s',\ s \rightarrow^{a}_{must} s' \textit{ and } s' \in \llbracket \phi \rrbracket^{\mathrm{nec}}\},$
   $\{s \in \Sigma_K \mid \textit{for some } s',\ s \rightarrow^{a}_{may} s' \textit{ and } s' \in \llbracket \phi \rrbracket^{\mathrm{pos}}\} \rangle$
5. $\llbracket [a]\phi \rrbracket \stackrel{\mathrm{def}}{=} \langle \{s \in S \mid \textit{for all } s',\ s \rightarrow^{a}_{may} s' \textit{ implies } s' \in \llbracket \phi \rrbracket^{must}\},$
   $\{s \in S \mid \textit{for all } s',\ s \rightarrow^{a}_{must} s' \textit{ implies } s' \in \llbracket \phi \rrbracket^{may}\} \rangle,$

The "necessarily" interpretation, $\llbracket \phi \rrbracket^{\mathrm{nec}}$, is an under-approximation of those states for which a proposition necessarily holds true (that is, the states for which the proposition holds for all future refinements/implementations). Dually, the "possibly" interpretation, $\llbracket \phi \rrbracket^{\mathrm{pos}}$, is an over-approximation of those states for which there is some refinement for which the proposition holds. The semantics $\llbracket \phi \rrbracket^{\mathrm{nec}}$ is the one given by Larsen [26]; it produces this result:

Proposition 3 (Logical characterization). *[26]*[2] *Let $\mathcal{C}$ and $\mathcal{A}$ be finitely-branching*[3] *Kripke MTSs and $s \in \Sigma_C$, $t \in \Sigma_A$. Then $s \prec_r t$ iff for all $\phi \in \mathcal{L}$, $[t \in \llbracket \phi \rrbracket^{\mathrm{nec}} \Rightarrow s \in \llbracket \phi \rrbracket^{\mathrm{nec}}]$.*

This result tells us to build an MTS, $\mathcal{A}$, that abstracts an implementation, $\mathcal{C}$. Both safety *and* liveness properties can be validated on $\mathcal{A}$, and they carry over to $\mathcal{C}$. Using the "possibly" interpretation, a new logical characterization follows, allowing us to refute safety and liveness properties of an implementation, $\mathcal{C}$, by refuting them on $\mathcal{A}$:

Proposition 4 (Logical characterization). *Let $\mathcal{C}$ and $\mathcal{A}$ be finitely-branching Kripke MTSs and $s \in \Sigma_C$, $t \in \Sigma_A$. Then $s \prec_r t$ iff for all $\phi \in \mathcal{L}$, $[s \in \llbracket \phi \rrbracket^{\mathrm{pos}} \Rightarrow t \in \llbracket \phi \rrbracket^{\mathrm{pos}}]$.*

Negation and Invariants. We can retain both validation and refutation on Kripke MTSs if we add negation and recursive definition to our logic, giving the modal-mu calculus [25,2], `ActMu`:

$$\phi ::= \top \mid p \mid Z \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \langle a \rangle \phi \mid [a]\phi \mid \mu Z.\phi \tag{4}$$

where p ranges over `AP`, Z over a (countable) set of variables, $a \in$ `Act`, and the bodies ϕ in $\mu Z.\phi$ are formally monotone. Disjunction ($\vee$) and implication ($\rightarrow$) are derived as $\neg(\neg\phi \wedge \neg\psi)$ and $\neg(\phi \wedge \neg\psi)$, respectively. This logic is very expressive,

[2] Larsen's results were proved for MTSs.
[3] For all $s \in \Sigma_K$ and $a \in$ `Act`, the sets $\{s' \in \Sigma_K \mid s \rightarrow^{may}_{a} s'\}$ and $L^{may}(s)$ are finite.

and important specification logics like CTL* can be embedded into it [10]. We require environments, ρ, mapping variables Z to elements of $\mathcal{P}(\Sigma_K) \times \mathcal{P}(\Sigma_K)$. The semantics, $\| \phi \|_\rho \in \mathcal{P}(\Sigma_K) \times \mathcal{P}(\Sigma_K)$, is defined as for $\mathcal{L}$, but parametric in an environment ρ; this semantics is essentially the one in [19]. Given the must- and may-aspects of an MTS, the semantics of negation is delicate, and we follow Kelb [24] and Levi [28]. The semantics of the remaining clauses, Z and $\mu Z.\phi$, are dealt with in a standard manner:

1. $\| Z \|_\rho \overset{\text{def}}{=} \rho(Z)$;
2. $\| \neg\phi \|_\rho \overset{\text{def}}{=} \langle \Sigma_K \setminus \| \phi \|_\rho^{\text{pos}},\ \Sigma_K \setminus \| \phi \|_\rho^{\text{nec}} \rangle$;
3. $\| \mu Z.\phi \|_\rho$ is the least fixed point of the monotone functional
$$d \mapsto \| \phi \|_{\rho[Z \mapsto d]} \colon \mathcal{P}(\Sigma_K) \times \mathcal{P}(\Sigma_K) \to \mathcal{P}(\Sigma_K) \times \mathcal{P}(\Sigma_K).$$

Note the semantics of negation: "*necessarily* $\neg\phi$" is "not *possibly* ϕ," and "*possibly* $\neg\phi$" is "not *necessarily* ϕ".

Theorem 1 (Soundness and consistency of semantics). *For any Kripke MTS $\mathcal{K}$, $\phi, \psi \in$ `ActMu`, and environment ρ:*

1. $\| \phi \|_\rho^{\text{nec}} \subseteq \| \phi \|_\rho^{\text{pos}}$*;*
2. $\| \phi \wedge \neg\phi \|_\rho^{\text{nec}} = \emptyset$*; and* $\| \phi \vee \neg\phi \|_\rho^{\text{pos}} = \Sigma_K$*. That is, the semantics is consistent for* $\| \ \|^{\text{nec}}$ *and "complete" for* $\| \ \|^{\text{pos}}$*;*
3. *if* $s \prec_r t$*, then* $t \in \| \phi \|_\rho^{\text{nec}}$ *implies* $s \in \| \phi \|_\rho^{\text{nec}}$*; and* $s \in \| \phi \|_\rho^{\text{pos}}$ *implies* $t \in \| \phi \|_\rho^{\text{pos}}$*. That is, the semantics is sound;*
4. *if* $\mathcal{K}$ *is concrete, then* $\| \phi \|_\rho^{\text{nec}} = \| \phi \|_\rho^{\text{pos}}$ *and corresponds to the standard semantics for doubly labeled transition systems (as given for CTL* in [12]).*

The semantics of negation behaves classically, that is, $\| \neg\neg\phi \| = \| \phi \|$ and $\| \neg\langle a \rangle \phi \| = \| [a]\neg\phi \|$ hold. The underlying interpretations $\| \phi \|^{\text{nec}}$ and $\| \phi \|^{\text{pos}}$, however, are not classical, but *three valued*, in the sense that a state, s, can possess a property, ϕ, in only three possible ways:

1. $s \in \| \phi \|^{\text{nec}}$ (and hence, $s \in \| \phi \|^{\text{pos}}$): "$\phi$ necessarily holds for s."
2. $s \in \| \phi \|^{\text{pos}}$ and $s \notin \| \phi \|^{\text{nec}}$ ($s \in \| \phi \wedge \neg\phi \|^{\text{pos}}$): "$\phi$ possibly holds for s."
3. $s \notin \| \phi \|^{\text{pos}}$ (hence, $s \notin \| \phi \|^{\text{nec}}$): "$\phi$ does not possibly hold for s."

Note the loss of precision in the second case above: $s \in \| \phi \wedge \neg\phi \|^{\text{pos}}$; there cannot be a final implementation which satisfies $\phi \wedge \neg\phi$. For the partial Kripke structures of Section 4, a more precise analysis is possible [4].

To finish the technical development, we note that the Kripke MTS semantics and its properties adapt to the scenario where Σ_K is no longer flat. For finite-state models, meanings $\| \phi \|_\rho$ are elements of the mixed power domain $\mathsf{M}[\Sigma_K]$ [15,17,20].[4] This lets us adapt standard abstract interpretation studies

[4] Elements of $\mathsf{M}[\Sigma_K]$ are pairs $\langle H,\ S \rangle$, where H is a Scott-closed lower subset and S a Scott-compact upper subset of Σ_K such that H equals $\{s \in \Sigma_K \mid \exists s' \in H \cap S \colon s \leq s'\}$. This consistency condition replaces the inclusion requirement ($H \subseteq S$) of the discrete case.

and even lets us define and manipulate a fully abstract domain of MTSs by the isomorphism, $D \cong \prod_{a \in \mathtt{Act}} \mathsf{M}[D]$ [20].

Abstract Kripke MTSs. Let $\mathcal{C} = (\Sigma_K, \mathtt{Act}, \mathtt{AP}, \longrightarrow, L)$ be a possibly infinite-state DLTS. Given a finite set of boolean predicates $\{p_1, p_2, \ldots, p_n\}$, we can derive a finite-state abstract Kripke MTS $\mathcal{A}$: abstract states, a, are equivalence classes of states that satisfy exactly the same predicates p_i in $\mathcal{C}$; abstract transitions are defined, as in [11], by (i) $[s] \rightarrow^a_{must} [s']$ iff for all $s_* \in [s]$, there exists $s'_* \in [s']$ such that $s_* \rightarrow^a s'_*$ and (ii) $[s] \rightarrow^a_{may} [s']$ iff there exist $s_* \in [s]$ and $s'_* \in [s']$ such that $s_* \rightarrow^a s'_*$; finally, propositions are defined as $L^{must}([s]) \stackrel{\text{def}}{=} \bigcap_{s' \in [s]} L(s')$ and $L^{may}([s]) \stackrel{\text{def}}{=} \bigcup_{s' \in [s]} L(s')$.

This refinement relationship is well behaved and induces *two* Galois connections [8], one between the concrete model $\mathcal{C}$ and the may-component of $\mathcal{A}$, and one between $\mathcal{C}$ and $\mathcal{A}$'s must-component. This phemonemon is foreshadowed by the universal ($\alpha^{\forall}$) and existential ($\alpha^{\exists}$) abstraction maps of Cousot and Cousot [9], which extract may- and must-behaviors from linear-time models.

It is immediate that $\mathcal{C}$ is a refinement of $\mathcal{A}$, giving us a sound tool for verifying *and* refuting *all* properties expressed in the modal mu-calculus. With some effort, this approach applies to refinements to non-concrete Kripke MTSs as well.

4 Abstracting Control and Data

Partial Kripke Structures. For model checking partial state spaces, Bruns and Godefroid devised *partial Kripke structures* [3]:

Definition 6. *A* partial Kripke structure[5] *is a 4-tuple, $\mathcal{K} = (\Sigma_K, \mathtt{AP}, \longrightarrow, L)$, where Σ_K is a set of states, $\mathtt{AP}$ is a set of atomic propositions, $\longrightarrow \subseteq \Sigma_K \times \Sigma_K$ is a set of transitions, and $L\colon \Sigma_K \times \mathtt{AP} \rightarrow \mathbf{3}$ is a labeling function, where $\mathbf{3}$ is the set $\{\bot, \mathtt{F}, \mathtt{T}\}$ endowed with the* information ordering $\bot \leq \mathtt{F}$ *and* $\bot \leq \mathtt{T}$.

Figure 3 depicts three partial Kripke structures with initial states s_1, s_2, and s_3, respectively [3]. We write $p = v$ at a state s to denote $L(s,p) = v$. These systems have different information regarding the truth of p at their initial states and their rightmost successor states. At their leftmost successor states, all three systems leave the truth status of p unresolved. Below, we show that these systems, their abstraction notion and temporal logic semantics are special instances of the corresponding notions for Kripke MTSs. Partial Kripke structures are related in the following fashion:

Definition 7 (Completeness order). *A* completeness order *[3] on two partial Kripke structures is a binary relation $Q \subseteq \Sigma_C \times \Sigma_A$ such that $Q(s,t)$ implies*

1. *for all $p \in \mathtt{AP}$, $L(s,p) \leq L(t,p)$ in the information ordering,*
2. *if $s \longrightarrow s'$, then there is some $t' \in \Sigma_A$ with $t \longrightarrow t'$ and $Q(s',t')$, and*
3. *if $t \longrightarrow t'$, then there is some $s' \in \Sigma_C$ with $s \longrightarrow s'$ and $Q(s',t')$.*

[5] We assume that these structures are finitely branching.

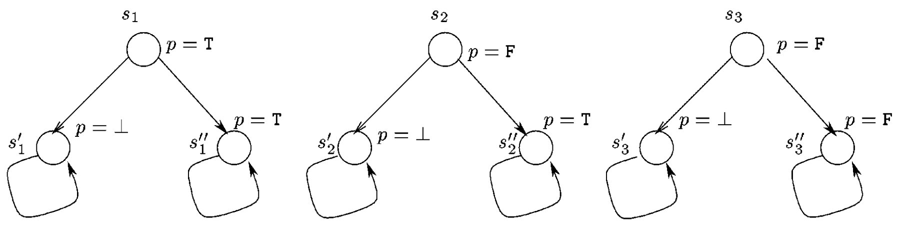


Fig. 3. Three partial Kripke structures [3]

In usual fashion, we write $s \lhd t$ if there is a completeness order Q in which $Q(s,t)$ holds. Properties of states of partial Kripke structures are expressed in the logic PML: $\phi ::= p \mid \neg\phi \mid \phi \wedge \phi \mid \Diamond\phi$. Bruns and Godefroid require a second, *truth ordering* on the set $\{\bot, \mathtt{F}, \mathtt{T}\}$ for defining a three-valued semantics for this logic: $\mathtt{F} < \bot < \mathtt{T}$. Denotations $[s \models \phi]$ are elements of $\{\mathtt{F} < \bot < \mathtt{T}\}$:

$$\begin{aligned}
[s \models p] &\stackrel{\text{def}}{=} L(s,p) \qquad (5)\\
[s \models \neg\phi] &\stackrel{\text{def}}{=} \text{neg}[s \models \phi]\\
[s \models \phi_1 \wedge \phi_2] &\stackrel{\text{def}}{=} \min([s \models \phi_1], [s \models \phi_2])\\
[s \models \Diamond\phi] &\stackrel{\text{def}}{=} \max\{[s' \models \phi] \mid s \longrightarrow s'\},
\end{aligned}$$

where neg is strict logical complement and min and max are meet and join in the truth ordering, respectively. Bruns and Godefroid logically characterize the completeness preorder: For partial Kripke structures $\mathcal{C}$ and $\mathcal{A}$, $s \in \Sigma_C$, $t \in \Sigma_A$, $s \lhd t$ iff for all ϕ in PML, $[s \models \phi] \leq [t \models \phi]$ in the truth ordering.

The embedding of partial Kripke structures into Kripke MTS rests on the order-isomorphism $\Psi\colon (E \to \mathbf{3}) \to \mathsf{M}[E]$ between the mixed power domain $\mathsf{M}[E]$ and the set of all functions $E \to \mathbf{3}$, ordered pointwise in the information ordering — if E is discrete; identify a function $f\colon E \to \mathbf{3}$ with the pair $\Psi f \stackrel{\text{def}}{=} \langle f^{-1}\{\mathtt{T}\},\ f^{-1}\{\bot, \mathtt{T}\}\rangle$. We then translate a partial Kripke structure, $\mathcal{K}$, into the Kripke MTS, $\mathcal{K}' = (\Sigma_K, \{*\}, \mathtt{AP}, \longrightarrow, \longrightarrow, L^{must}, L^{may})$,[6] where the propositional component $(L^{must}(s), L^{may}(s))$ is defined as ΨL.

Proposition 5 (Correspondence of semantics). *For partial Kripke structure, $\mathcal{K}$, and its Kripke MTS translation, $\mathcal{K}'$, for all $s \in \Sigma_K$ and ϕ in PML, $\llbracket \phi \rrbracket = \Psi(\lambda s.[s \models \phi])$. The inverse $\lhd^{-1}$ of the greatest completeness order $\lhd$ on $\mathcal{K}$ is the greatest refinement of its Kripke MTS translation $\mathcal{K}'$.*

The full MTS formulation, unlike partial Kripke structures, allows for modalities on transitions and remains well defined when the domain of states is nonflat, making it applicable to conventional abstraction frameworks [8,7]. To illustrate

[6] We identify $\Sigma_K \times \Sigma_K$ with $\Sigma_K \times \{*\} \times \Sigma_K$.

model checks on such systems, consider $\phi \stackrel{\text{def}}{=} \mu Z.p \vee ([*]Z \wedge \langle * \rangle \top)$, saying "$p$ holds eventually on all paths" [2], on the systems of Figure 3.[7] Our semantics computes $s_1 \in \llbracket \phi \rrbracket^{\text{nec}}$, since p is true at s_1; $s_2 \in \llbracket \phi \rrbracket^{\text{pos}}$, since p may be true at s'_2; and $s_3 \notin \llbracket \phi \rrbracket^{\text{pos}}$, since there is a path on which p is never true.

Partial Bisimulations. Partial Kripke structures abstract propositional information only. Bruns and Godefroid also studied systems that abstract state transitions, the so-called extended transition systems [3], and their partial bisimulation [29,39]:

Definition 8. *An* extended transition system (ETS)[8] *[3] is a 4-tuple* $\mathcal{E} = (\Sigma_E, \mathtt{Act}, \longrightarrow, \uparrow)$*, where* Σ_E *is a set of states,* $\longrightarrow \; \subseteq \Sigma_E \times \mathtt{Act} \times \Sigma_E$ *is a set of transitions, and* $\uparrow \; \subseteq \Sigma_E \times \mathtt{Act}$ *is a* divergence relation*.*

Read $s \uparrow a$ as "some of the a-transitions from s in the full model may be missing at s in the ETS" [3]. We write $s \downarrow a$ when $s \uparrow a$ fails to hold, meaning that all a-transitions from s in the full state space are present at s in the ETS.

Definition 9. *Given ETSs,* $\mathcal{E}$ *and* $\mathcal{F}$*, a* partial bisimulation *[29,39] (*divergence preorder *[3]) is a subset,* Q*, of* $\Sigma_E \times \Sigma_F$ *such that* $Q(s,t)$ *implies*

1. *whenever* $s \rightarrow_a s'$*, there exists some* $t' \in \Sigma_F$ *such that* $t \rightarrow_a t'$ *and* $Q(s',t')$*;*
2. *if* $s \downarrow a$*, then (i)* $t \downarrow a$*, and (ii) whenever* $t \rightarrow_a t'$*, there exists some* $s' \in \Sigma_E$ *such that* $s \rightarrow_a s'$ *and* $Q(s',t')$*.*

Every ETS has a greatest partial bisimulation, $\sqsubseteq$. We embed $\mathcal{E}$ into a Kripke MTS $\mathbf{T}[\mathcal{E}] \stackrel{\text{def}}{=} (\Sigma_E, \mathtt{Act}, \emptyset, \stackrel{must}{\longrightarrow}, \stackrel{may}{\longrightarrow}, \emptyset, \emptyset)$ by (i) $s \rightarrow_a^{must} s'$ iff $s \rightarrow_a s'$ and (ii) $s \rightarrow_a^{may} s'$ iff ($s \rightarrow_a s'$ or $s \uparrow a$). Note how $\uparrow$ makes $\mathcal{E}$ three-valued in may-transitions.

Theorem 2. *Given an ETS* $\mathcal{E}$*,* $\mathbf{T}[\mathcal{E}]$ *is an MTS satisfying, for all* $s \in \Sigma_E$*,* $(\exists s' \in \Sigma_E \colon s \rightarrow_{may}^a s' \wedge s \not\rightarrow_{must}^a s') \rightarrow (\forall s'' \in \Sigma_E \colon s \rightarrow_{may}^a s'')$*. The inverse* $\sqsubseteq^{-1}$ *of the greatest partial bisimulation* $\sqsubseteq$ *on* $\mathcal{E}$ *is* $\prec_r$*, the greatest refinement on* $\mathbf{T}[\mathcal{E}]$*. The intuitionistic semantics for Hennessy-Milner logic in [29] corresponds to the semantics* $\llbracket \cdot \rrbracket^{\text{nec}}$ *of that fragment of* $\mathtt{ActMu}$ *on* $\mathbf{T}[\mathcal{E}]$*.*

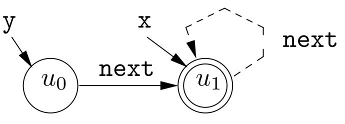
5 Abstracting Data: Shape-Based Pointer Analysis

An important form of pointer analysis is *shape analysis* [6,14,22,35,40], where the contents of heap storage is approximated by a graph whose nodes denote objects and whose arcs denote the values of the objects' fields. Local ("stack") variables that point into the heap are drawn as arcs pointing to the nodes.

Figure 4 displays the syntax of such *shape graphs.* The example in the Figure depicts an approximation to a singly linked list of length at least two: Objects are circles; a double-circled object is a "summary node," meaning that it possibly

[7] We use $*$ in ϕ to denote the sole action type.

[8] We assume that all such structures are finitely branching.

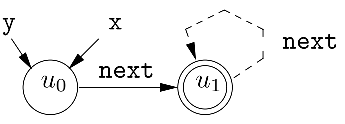


$x(u_0) = 0$	$next(u_0, u_0) = 0$
$x(u_1) = 1$	$next(u_0, u_1) = 1$
$y(u_0) = 1$	$next(u_1, u_0) = 0$
$y(u_1) = 0$	$next(u_1, u_1) = 1/2$
$sm(u_0) = 0$	
$sm(u_1) = 1$	

Fig. 4. Shape graph and its coding as predicates

T[x = y] : $x'(v) = y(v)$; all other predicates $p' = p$
T[x.next = y] : $next'(v_1, v_2) = next(v_1, v_2) \wedge (sm(v_1) \vee \neg x(v_1)) \vee (x(v_1) \wedge y(v_2)$;
all other $p' = p$
T[x = y.next] : $x'(v) = \exists v_1.y(v_1) \wedge next(v_1, v)$; all other $p' = p$
T[x = new Node()] : let v_{new} be a fresh node, in $x'(v) = (v = v_{new})$;
all other $p'(v) = (p(v) \wedge (v \neq v_{new}))$

Effect of x = y on Figure 4:



Effect of x.next = y on Figure 4:

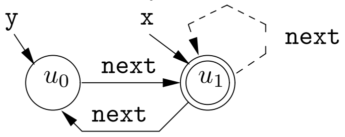


Fig. 5. Transfer functions on shape graphs

represents more than one concrete object. Since the objects were constructed from a class/struct that owns a next field, objects have next-labeled arcs. For discussion, the objects are named u_0 and u_1, and local variables x and y point to the objects. A solid arc denotes that a field definitely points to an object; a dotted arc means the field possibly points to it. Thus, the self-arc on u_1 must be dotted because u_1 possibly denotes multiple nodes, meaning that a next dereference possibly points to one of the concrete objects denoted by the node.

Shape graphs can be encoded in various ways; in Figure 4, we display a coding due to Sagiv, Reps, and Wilhelm [35], who define local-variable points-to information with unary predicates and field points-to information with binary ones. The predicates produce the answers "necessarily points to" (1), "possibly points to" (1/2), and "not points to" (0), where the values are ordered $0 \leq 1/2 \leq 1$. The predicate, sm, notes which nodes are summary nodes.

Shape graphs can be used as data values for a data-flow analysis, where a program's transfer functions transform an input shape graph to an output one. The transfer functions for assignment and object construction appear in Figure 5, where p' denotes predicate p updated by the transfer function, T[C], for command C. The transfer functions are written as predicate-logic formulas, where conjunction is interpreted as meet; disjunction as join; and negation as strict complement. A data-flow analysis is assembled in the usual way [8,16,23, 31,32]:

1. a control-flow graph is extracted from the source program, where transfer functions annotate the arcs of the graph;
2. a finite-height sup-semilattice of data-flow values (here, based on shape graphs) is defined, where values from the semilattice will be collected at the nodes (program points) of the control-flow graph.
3. a least fixed-point calculation is undertaken on the flow equations induced by the control-flow graph.

Step 2 is the most interesting, in that a *single* shape graph might be collected at each node ("independent attribute" analysis, e.g., [40]) or a set of shape graphs might be collected ("relational" analysis, e.g., [35]). In the former case, the join operation weakens solid arcs into dotted ones when dissimilar graphs are joined; in the latter case, a bounded set union (*widening* [8]) operation is employed so that no infinite set of graphs is constructed.

Modal Shape Graphs. The dotted and solid arcs of shape graphs strongly suggest that modal transition systems lurk in the foundations, and so they do:

Definition 10 (Modal shape graph). *A* modal shape graph (MSG) *is a Kripke MTS* $\mathcal{M} \stackrel{\text{def}}{=} (\Sigma_M, \mathtt{Act}, \mathtt{AP}, \xrightarrow{must}, \xrightarrow{may}, L^{must}, L^{may})$*, where* AP *is a set of local variables along with the distinguished predicate, sm,* Act *is a set of field names, and* Σ_M *is a set of heap objects.*

Our modelling sets $\mathtt{x} \in L^{must}(s)$ when a solid arc shows that x points to object s. Similarly, $L^{may}(s)$ collects the names of local variables that possibly point to it. When s is a summary node, then $sm \in L^{must}(s)$. Of course, $\xrightarrow{must}$ and $\xrightarrow{may}$ model the solid and dotted field-labeled arcs, respectively. It is easy to translate shape graphs, like that in Figure 4, into MTS format:

$$\begin{array}{lll}
\Sigma_M = \{u_0, u_1\} & \mathtt{Act} = \{next\} & \mathtt{AP} = \{x, y, sm\} \\
\xrightarrow{must} = \{(u_0, next, u_1)\} & \xrightarrow{may} = \{(u_1, next, u_1)\} & \\
L^{must}(u_0) = \{y\}, \quad L^{must}(u_1) = \{x, sm\} & & \\
L^{may}(u_0) = \emptyset, \quad L^{may}(u_1) = \emptyset & &
\end{array}$$

A concrete, run-time execution state is coded as a concrete modal shape graph.[9]

Transfer Functions. There is little challenge in writing the transfer functions for modal shape graphs: Given a modal shape graph $\mathcal{M}$, let $\mathcal{M}[\mathtt{C}]$ be the graph obtained from $\mathcal{M}$ by executing command C. We only specify those aspects of $\mathcal{M}[\mathtt{C}]$ that are different from $\mathcal{M}$:

1. `x = y`: For all $s \in \Sigma_M$, $\mathtt{x} \in L[\mathtt{C}]^{must}(s)$ iff $\mathtt{y} \in L^{must}(s)$; and $\mathtt{x} \in L[\mathtt{C}]^{may}(s)$ iff $\mathtt{y} \in L^{may}(s)$.
2. `x = y.n`: For all $s \in \Sigma_M$, $\mathtt{x} \in L[\mathtt{C}]^{must}(s)$ iff $(\exists s' \in S)\, y \in L^{must}(s')$ and $s' \xrightarrow{must}_{\mathtt{n}} s$; and $\mathtt{x} \in L[\mathtt{C}]^{may}(s)$ iff $(\exists s' \in S)\, y \in L^{may}(s')$ and $s' \xrightarrow{may}_{\mathtt{n}} s$.

[9] Where every local variable, x, belongs to at most one $L^{must}(s)$.

3. `x.n = y`: For all $s \in \Sigma_M$, (i) if $x \in L^{must}(s)$ and there is some s'' with $s \overset{must}{\longrightarrow}_{\mathtt{n}} s''$, then, for all $s' \in S$, $\mathtt{y} \in L[\mathtt{C}]^{must}(s')$ implies $s \overset{must[C]}{\longrightarrow}_{\mathtt{n}} s'$; and (ii) if $x \in L^{may}(s)$ and there is some s'' with $s \overset{may}{\longrightarrow}_{\mathtt{n}} s''$, then, for all $s' \in S$, $\mathtt{y} \in L[\mathtt{C}]^{may}(s')$ implies $s \overset{may[C]}{\longrightarrow}_{\mathtt{n}} s'$. If $sm \in L^{must}(y)$, then all transitions from y are preserved.
4. `x = new Node()`: Creates a fresh object that is labeled with x. Set $\Sigma_{M[C]} \overset{\mathrm{def}}{=} \Sigma_M \cup \{s_{\mathrm{new}}\}$ ($s_{\mathrm{new}} \notin \Sigma_M$); and for all $s \in \Sigma_M$, $x \notin L[\mathtt{C}]^{may}(s)$, and $x \in L[\mathtt{C}]^{must}(s_{\mathrm{new}})$.

Properties as Temporal Formulas. Once a data-flow analysis builds a shape graph, we check the graph for correctness properties that are expressible in the CTL-subset [5,10] of the modal mu-calculus. In [35], such properties are encoded in predicate logic augmented with a transitive closure operator.

Here are examples: The direction relationship [14], stating that an access path exists from the object named by x to an object named by y, is written $\mathtt{D(x,y)} \overset{\mathrm{def}}{=} \mathtt{EF}_{\mathtt{next}}(\mathtt{x} \wedge \mathtt{EF}_{\mathtt{next}}\mathtt{y})$ — an object, s, has atomic property x iff x points to s. Recall that $\mathtt{EF}_a\phi$ states, "there exists a path of a-labeled transitions such that, at some state in the future, ϕ holds true." To validate that there is *necessarily* (*possibly*) a path from s, we check if $s \in [\![\mathtt{D(x,y)}]\!]^{\mathrm{nec}}$ ($s \in [\![\mathtt{D(x,y)}]\!]^{\mathrm{pos}}$); to refute existence of a path, we check $s \in [\![\neg\mathtt{D(x,y)}]\!]^{\mathrm{nec}}$.

The interference relationship [14], saying that pointers x and y have access paths to a common heap node, is written with *inverse* transition relationships of $\overset{must}{\longrightarrow}$: $\mathtt{I(x,y)} \overset{\mathrm{def}}{=} (\mathtt{EF}_{\mathtt{next}^{-1}}\mathtt{x}) \wedge (\mathtt{EF}_{\mathtt{next}^{-1}}\mathtt{y})$. We check $s \in [\![\mathtt{I(x,y)}]\!]^{\mathrm{pos}}$ to see if aliasing of object s by x and y is possible.

Aliasing of pointers can be expressed: For $\mathtt{aliasing} \overset{\mathrm{def}}{=} \mathtt{EF}_{\mathtt{next}}(\bigvee_{\mathtt{x}\neq\mathtt{y}} \mathtt{x} \wedge \mathtt{y})$, the formulas (a) $\mathtt{AG}_{\mathtt{next}}\neg\mathtt{aliasing}$, (b) $\mathtt{AG}_{\mathtt{next}}\neg(\mathtt{x}\wedge\bigvee_{\mathtt{x}\neq\mathtt{y}} \mathtt{y})$, and (c) $\mathtt{AG}_{\mathtt{next}}\neg(\mathtt{x}\wedge\mathtt{y})$ can then be used to check: (a) the absense of any kind of aliasing; (b) that x has no alias; and that (c) x and y never point to the same heap node. (Recall that $\mathtt{AG}_a\phi$ states, "for all a-paths, it is globally true for all states along the path, that ϕ holds.")

We can check for possibly cyclic data structures. The predicate $\mathtt{cyclic} \overset{\mathrm{def}}{=} \bigvee_{\mathtt{x}\in\mathtt{AP}} \mathtt{x} \wedge \mathtt{EX}_{\mathtt{next}}\mathtt{EF}_{\mathtt{next}}\mathtt{x}$ states that a heap node is pointed to by some x that has an access path to, presumably the same, heap node pointed to by x. (Recall that $\mathtt{EX}_a\phi$ says, "there exists an a-transition to a next state where ϕ holds.")

Improving Precision: The Embedding Theorem and *Focus* Operation. For improving the analysis's precision (e.g., "strong updates"), Sagiv, Wilhelm, and Reps employ a *focus* operation [35], which refines a shape graph into a set of shape-graph variants, such that a predicate that formerly evaluated to 1/2 ("possibly holds") in the original graph now evaluates to 0 or to 1 in every variant. The set of variants must be consistent and complete with regards to the original graph, and an Embedding Theorem is used to make this argument. A more precise, relational, data-flow analysis is the result.

Within the representation of modal shape graphs, the hypotheses of the Embedding Theorem ensure a refinement relation, and the consequences of the Theorem follow from the soundness of refinement (Theorem 1).

The *focus* operation itself defines a *cover*: A set of MTSs, $\mathcal{S}_\mathcal{A}$, *covers* an MTS, $\mathcal{A}$, iff *(i)* for all $\mathcal{K} \in \mathcal{S}_\mathcal{A}$, $\mathcal{K} \prec_r \mathcal{A}$; *(ii)* for every concrete MTS, $\mathcal{C}$, such that $\mathcal{C} \prec_r \mathcal{A}$, there exists some $\mathcal{K} \in \mathcal{S}_\mathcal{A}$ such that $\mathcal{C} \prec_r \mathcal{K}$. Any property that necessarily holds true for all the MTSs in $\mathcal{S}_\mathcal{A}$ holds true for all concrete refinements of $\mathcal{A}$; dually, any property that possibly holds for any MTS in $\mathcal{S}_\mathcal{A}$ holds true for $\mathcal{A}$; thus, $\mathcal{S}_\mathcal{A}$ is consistent and complete regarding the concrete MTSs represented by $\mathcal{A}$.

The *focus* operation in [35] generates one particular shape-graph cover by examining a may-transition reachable from a program variable and refining it to a must-transition in one graph variant, removing it in another graph variant, and splitting the transition's source or destination summary node in a third variant. Of course, there can exist other forms of *focus* operations; in all cases, a cover must be defined.

6 Conclusions

The case studies demonstrate how modal transition systems provide a foundation for development of analyses whose outcomes take the form, "necessarily" (yes), "possibly" (maybe), and "not possibly" (no). These applications go beyond the traditional use for MTSs (proving properties of loose specifications). In addition to the two examples in this paper, there are other program analyses that are neatly expressed via Kripke MTSs; two noteworthy ones are

- Whaley and Rinard's *points-to escape analysis* [40], where multi-threaded programs are analyzed for object sharing. Shape graphs are generated, where solid arcs represent assignments made by the thread being analyzed, and dotted arcs represent assignments made by other threads executing in parallel. Jackson's Z-like Alloy logic [21] is used to model check the graphs.
- Interprocedural data-flow analysis [34,36], where graphs are used to denote control flow. Those program transitions that must occur (e.g., intraprocedural transitions) are denoted by solid arcs; transitions that might occur (e.g., procedure call- and return-arcs, where the exact procedure invoked or the exact invocation point is uncertain) are denoted by may-arcs.

Other applications await discovery (e.g., cartesian abstraction in the SLAM project [1]), and the relationship of our semantic framework to earlier studies of 3-valued modal logic [13,30,37] and intuitionistic modal logic [38] deserves examination as well.

References

1. T. Ball, A. Podelski, and S. K. Rajamani. Boolean and Cartesian Abstraction for Model Checking C Programs. Personal communication, December 2000.
2. J. C. Bradfield. *Verifying Temporal Properties Of Systems.* Birkhäuser, Boston, Mass., 1991.

3. G. Bruns and P. Godefroid. Model Checking Partial State Spaces with 3-Valued Temporal Logics. In *Proceedings of the 11th Conference on Computer Aided Verification*, volume 1633 of *Lecture Notes in Computer Science*, pages 274–287. Springer Verlag, July 1999.
4. G. Bruns and P. Godefroid. Gernalized Model Checking: Reasoning about Partial State Spaces. In *Proceedings of CONCUR'2000 (11th International Conference on Concurrency Theory)*, volume 1877 of *Lecture Notes in Computer Science*, pages 168–182. Springer Verlag, August 2000.
5. J. R. Burch, E. M. Clarke, D. L. Dill K. L. McMillan, and J. Hwang. Symbolic model checking: 10^{20} states and beyond. Proceedings of the Fifth Annual Symposium on Logic in Computer Science, June 1990.
6. D. Chase, M. Wegman, and F. Zadeck. Analysis of pointers and structures. In *SIGPLAN Conf. on Prog. Lang. Design and Implementation*, pages 296–310. ACM Press, 1990.
7. E.M. Clarke, O. Grumberg, and D.E. Long. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems*, 16(5):1512–1542, 1994.
8. 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.
9. P. Cousot and R. Cousot. Temporal abstract interpretation. In *Conference Record of the Twentyseventh Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 12–25, Boston, Mass., January 2000. ACM Press, New York, NY.
10. M. Dam. CTL* and ECTL* as Fragments of the Modal mu-Calculus. *Theoretical Computer Science*, 126:77–96, 1994.
11. D. Dams. *Abstract interpretation and partition refinement for model chec king.* PhD thesis, Technische Universiteit Eindhoven, The Netherlands, 1996.
12. R. de Nicola and F. Vaandrager. Three Logics for Branching Bisimulation. *Journal of the Association of Computing Machinery*, 42(2):458–487, March 1995.
13. M. Fitting. Many-valued modal logics. *Fundamenta Informaticae*, 17:55–73, 1992.
14. R. Ghiya and L. J. Hendren. Is it a Tree, a DAG, or a Cyclic Graph? A Shape Analysis for Heap-Directed Pointers in C. In *Proceedings of the 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 1–15, 1996.
15. C. Gunter. The mixed power domain. *Theoretical Computer Science*, 103:311–334, 1992.
16. M. Hecht. *Flow Analysis of Computer Programs.* Elsevier, 1977.
17. R. Heckmann. Power domains and second order predicates. *Theoretical Computer Science*, 111:59–88, 1993.
18. M. C. B. Hennessy and Robin Milner. Algebraic laws for non-determinism and concurrency. *JACM*, 32:137–161, 1985.
19. M. Huth. A Unifying Framework for Model Checking Labeled Kripke Structures, Modal Transition Systems, and Interval Transition Systems. In *Proceedings of the 19th International Conference on the Foundations of Software Technology & Theoretical Computer Science*, Lecture Notes in Computer Science, pages 369–380, IIT Chennai, India, December 1999. Springer Verlag.
20. M. Huth, R. Jagadeesan, and D. Schmidt. Modal transition systems: new foundations and new applications. To appear as a KSU-CIS Techreport, August 2000.
21. D. Jackson, I. Schechter, and I. Shlyakhter. Alcoa: the alloy constraint analyzer. In *Proc. International Conference on Software Engineering*, Limerick, Ireland, 2000.

22. N.D. Jones and S. Muchnick. Flow analysis and optimization of LISP-like structures. In *Proc. 6th. ACM Symp. Principles of Programming Languages*, pages 244–256, 1979.
23. J. Kam and J. Ullman. Global data flow analysis and iterative algorithms. *J. ACM*, 23:158–171, 1976.
24. P. Kelb. Model checking and abstraction: a framework preserving both truth and failure information. Technical Report Technical report, OFFIS, University of Oldenburg, Germany, 1994.
25. D. Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science*, 27:333–354, 1983.
26. K. G. Larsen. Modal Specifications. In J. Sifakis, editor, *Automatic Verification Methods for Finite State Systems*, number 407 in Lecture Notes in Computer Science, pages 232–246. Springer Verlag, June 12–14 1989. International Workshop, Grenoble, France.
27. K. G. Larsen and B. Thomsen. A Modal Process Logic. In *Third Annual Symposium on Logic in Computer Science*, pages 203–210. IEEE Computer Society Press, 1988.
28. F. Levi. A symbolic semantics for abstract model checking. In *Static Analysis Symposium: SAS'98*, volume 1503 of *Lecture Notes in Computer Science*. Springer Verlag, 1998.
29. R. Milner. A modal characterisation of observable machine behaviours. In G. Astesiano and C. Böhm, editors, *CAAP '81*, volume 112 of *Lecture Notes in Computer Science*, pages 25–34. Springer Verlag, 1981.
30. O. Morikawa. Some modal logics based on a three-valued logic. *Notre Dame J. of Formal Logic*, 30:130–137, 1989.
31. S. Muchnick and N.D. Jones, editors. *Program Flow Analysis: Theory and Applications*. Prentice-Hall, 1981.
32. F. Nielson, H. R. Nielson, and C. Hankin. *Principles of Program Analysis*. Springer Verlag, 1999.
33. A. Pnueli. Applications of temporal logic to the specification and verification of reactive systems: a survey of current trends. In J.W. de Bakker, editor, *Current Trends in Concurrency*, volume 224 of *Lecture Notes in Computer Science*, pages 510–584. Springer-Verlag, 1985.
34. T. Reps. Program analysis via graph reachability. In J. Maluszynski, editor, *Proc. Int'l. Logic Prog. Symp.'97*, pages 5–19. MIT Press, 1997.
35. M. Sagiv, T. Reps, and R. Wilhelm. Parametric Shape Analysis via 3-Valued Logic. In *Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of programming languages*, pages 105–118, January 20-22, San Antonio, Texas 1999.
36. D. A. Schmidt. Binary relations for abstraction and refinement. *Elsevier Electronic Notes in Computer Science*, November 1999. Workshop on Refinement and Abstraction, Osaka, Japan. To appear.
37. K. Segerberg. Some modal logics based on a three-valued logic. *Theoria*, 33:53–71, 1967.
38. C. Stirling. Modal logics for communicating systems. *Theoretical Computer Science*, 39:331–347, 1987.
39. D. J. Walker. Bisimulation and divergence. *Information and Computation*, 85(2):202–241, 1990.
40. J. Whaley and M. Rinard. Compositional pointer and escape analysis for Java programs. In *Proc. OOPSLA'99*, pages 187–206. ACM, 1999.