May/Must Abstraction-Based Software Model Checking For Sound Verification and Falsification

Patrice GODEFROID
Microsoft Research

Abstract.

Three-valued models, in which properties of a system are either true, false or unknown, have recently been advocated as a better representation for reactive program abstractions generated by automatic techniques such as predicate abstraction. Indeed, for the same cost, model checking three-valued abstractions, also called *may/must abstractions*, can be used to both prove and disprove any temporal-logic property, whereas traditional conservative abstractions can only prove universal properties. Also, verification results can be more precise with *generalized model checking*, which checks whether there exists a concretization of an abstraction satisfying a temporal-logic formula. Generalized model checking generalizes both model checking (when the model is complete) and satisfiability (when everything in the model is unknown), probably the two most studied problems related to temporal logic and verification.

This paper presents an introduction to the main ideas behind this framework, namely models for three-valued abstractions, completeness preorders to measure the level of completeness of such models, three-valued temporal logics and generalized model checking. It also discusses algorithms and complexity bounds for three-valued model checking and generalized model-checking for various temporal logics. Finally, it discusses applications to program verification via automatic abstraction.

1. Introduction

How to broaden the scope of model checking to software is currently one of the most challenging problems related to computer-aided verification. Essentially two approaches have been proposed and are still actively being investigated. The first approach consists of adapting model checking into a form of systematic testing that simulates the effect of model checking while being applicable to operating-system processes executing arbitrary code [17,23]; although counter-examples reported with this approach are sound, it is inherently incomplete for large systems. The second approach consists of automatically extracting a model out of a software application by statically analyzing its code, and then of analyzing this model using traditional model-checking algorithms (e.g., [4,8,49,41, 29]); although automatic abstraction may be able to prove correctness, counter-examples are generally unsound since abstraction usually introduces unrealistic behaviors that may yield to spurious errors being reported when analyzing the model.

In this paper, we present an overview of a series of articles [5,6,20,21,22,18,14,19,25] discussing how automatic abstraction can be performed to verify arbitrary formulas of the propositional μ-calculus [35] in such a way that both correctness proofs and counter-examples are guaranteed to be sound.

The key to make this possible is to represent abstract systems using richer models that distinguish properties that are *true*, *false* and *unknown* of the concrete system. Examples of such richer modeling formalisms are partial Kripke structures [5] and Modal Transition Systems [36,20]. Reasoning about such systems requires 3-valued temporal logics [5], i.e., temporal logics whose formulas may evaluate to *true*, *false* or $\perp$ ("unknown") on a given model. Then, by using an automatic abstraction process that generates by construction an abstract model which is less complete than the concrete system with respect to a completeness preorder logically characterized by 3-valued temporal logic, every temporal property that evaluates to *true* (resp. *false*) on the abstract model automatically holds (resp. does not hold) of the concrete system, hence guaranteeing soundness of both proofs and counter-examples. In case a property evaluates to $\perp$ on the model, a more complete (i.e., less abstract) model is then necessary to provide a definite answer concerning this property of the concrete system. This approach is applicable to check arbitrary formulas of the propositional μ-calculus (thus including negation and arbitrarily nested path quantifiers), not just universal properties as with a traditional "conservative" abstraction that merely simulates the concrete system.

2. Three-Valued Modeling Formalisms

Examples of 3-valued modeling formalisms for representing partially defined systems are *partial Kripke structures* (PKS) [5], *Modal Transition Systems* (MTS) [36,20] or *Kripke Modal Transition Systems* (KMTS) [30].

Definition 1 *A* KMTS *M is a tuple $(S, P, \xrightarrow{must}, \xrightarrow{may}, L)$, where S is a nonempty finite set of states, P is a finite set of atomic propositions, $\xrightarrow{may} \subseteq S \times S$ and $\xrightarrow{must} \subseteq S \times S$ are transition relations such that $\xrightarrow{must} \subseteq \xrightarrow{may}$, and $L : S \times P \to \{true, \perp, false\}$ is an* interpretation *that associates a truth value in $\{true, \perp, false\}$ with each atomic proposition in P for each state in S. An MTS is a KMTS where $P = \emptyset$. A PKS is a KMTS where $\xrightarrow{must} = \xrightarrow{may}$.*

The third value $\perp$ (read "unknown") and *may*-transitions that are not *must*-transitions are used to model explicitly a loss of information due to abstraction concerning, respectively, state or transition properties of the concrete system being modeled. A standard, *complete* Kripke structure is a special case of KMTS where $\xrightarrow{must} = \xrightarrow{may}$ and $L : S \times P \to \{true, false\}$, i.e., no proposition takes value $\perp$ in any state.

It can be shown [22] that PKSs, MTSs, KMTSs and variants of KMTSs where transitions are labeled and/or two interpretation functions L^{may} and L^{must} are used [30], are all equally expressive (i.e., one can translate any formalism into any other). In this paper, we will use KMTSs since they conveniently generalize models with *may*-transitions only, which are used with traditional conservative abstractions. Obviously, our results also hold for other equivalent formalisms (exactly as traditional model-checking algorithms and complexity bounds apply equally to systems modeled as Kripke structures or Labeled Transition Systems, for instance).

3. Three-Valued Temporal Logics

When evaluating a temporal-logic formula on a 3-valued model, there are three possible outcomes: the formula can evaluate to *true*, *false* or $\perp$ (unknown). Formally, we define 3-valued (temporal) logics as follows.

In interpreting propositional operators on KMTSs, we use Kleene's strong 3-valued propositional logic [34], which generalizes the standard 2-valued semantics. Conjunction $\wedge$ in this logic is defined as the function that returns *true* if both of its arguments are *true*, *false* if either argument is *false*, and $\perp$ otherwise. We define negation $\neg$ using the function 'comp' that maps *true* to *false*, *false* to *true*, and $\perp$ to $\perp$. Disjunction $\vee$ is defined as usual using De Morgan's laws: $p \vee q = \neg(\neg p \wedge \neg q)$. Note that these functions give the usual meaning of the propositional operators when applied to values *true* and *false*.

Propositional modal logic (PML) is propositional logic extended with the modal operator AX (which is read "for all immediate successors"). Formulas of PML have the following abstract syntax: $\phi ::= p \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid AX\phi$, where p ranges over P. The following 3-valued semantics generalizes the traditional 2-valued semantics for PML.

Definition 2 *The value of a formula ϕ of 3-valued PML in a state s of a KMTS $M = (S, P, \xrightarrow{must}, \xrightarrow{may}, L)$, written $[(M,s) \models \phi]$, is defined inductively as follows:*

$$
\begin{aligned}
[(M,s) \models p] &= L(s,p) \\
[(M,s) \models \neg\phi] &= comp([(M,s) \models \phi]) \\
[(M,s) \models \phi_1 \wedge \phi_2] &= [(M,s) \models \phi_1] \wedge [(M,s) \models \phi_2] \\
[(M,s) \models AX\phi] &= \begin{cases} true \;\; if\, \forall s' : s \xrightarrow{may} s' \Rightarrow [(M,s') \models \phi] = true \\ false \; if\, \exists s' : s \xrightarrow{must} s' \wedge [(M,s') \models \phi] = false \\ \perp \quad\; otherwise \end{cases}
\end{aligned}
$$

This 3-valued logic can be used to define a preorder on KMTSs that reflects their degree of completeness. Let $\leq$ be the *information ordering* on truth values, in which $\perp \leq true$, $\perp \leq false$, $x \leq x$ (for all $x \in \{true, \perp, false\}$), and $x \not\leq y$ otherwise. Note that the operators comp, min and max are monotonic with respect to the information ordering $\leq$: if $x \leq x'$ and $y \leq y'$, we have $\mathrm{comp}(x) \leq \mathrm{comp}(x')$, $\min(x,y) \leq \min(x',y')$, and $\max(x,y) \leq \max(x',y')$. This property is important to prove the results that follow.

Definition 3 *Let $M_A = (S_A, P, \xrightarrow{must}_A, \xrightarrow{may}_A, L_A)$ and $M_C = (S_C, P, \xrightarrow{must}_C, \xrightarrow{may}_C, L_C)$ be KMTSs. The* completeness preorder *$\preceq$ is the greatest relation $\mathscr{B} \subseteq S_A \times S_C$ such that $(s_a, s_c) \in \mathscr{B}$ implies the following:*

- $\forall p \in P : L_A(s_a, p) \leq L_C(s_c, p)$,
- *if $s_a \xrightarrow{must}_A s'_a$, there is some $s'_c \in S_C$ such that $s_c \xrightarrow{must}_C s'_c$ and $(s'_a, s'_c) \in \mathscr{B}$,*
- *if $s_c \xrightarrow{may}_C s'_c$, there is some $s'_a \in S_A$ such that $s_a \xrightarrow{may}_A s'_a$ and $(s'_a, s'_c) \in \mathscr{B}$.*

This definition allows to abstract M_C by M_A by letting truth values of propositions become $\perp$ and by letting *must*-transitions become *may*-transitions, but all *may*-transitions of M_C must be preserved in M_A. We then say that M_A is *more abstract*, or *less com-*

plete, than M_C. The inverse of the completeness preorder is also called *refinement preorder* in [36,30,20]. Note that relation $\mathscr{B}$ reduces to a simulation relation when applied to MTSs with *may*-transitions only. Also note that relation $\mathscr{B}$ reduces to bisimulation when applied to MTSs with *must*-transitions only and where all atomic propositions in P are either *true* or *false*.

It can be shown that 3-valued PML logically characterizes the completeness preorder [5,30,20].

Theorem 4 *[5] Let* $M_A = (S_A, P, \xrightarrow{must}_A, \xrightarrow{may}_A, L_A)$ *and* $M_C = (S_C, P, \xrightarrow{must}_C, \xrightarrow{may}_C, L_C)$ *be KMTSs such that* $s_a \in S_A$ *and* $s_c \in S_C$*, and let* Φ *be the set of all formulas of 3-valued PML. Then,*

$$s_a \preceq s_c \textit{ iff } (\forall \phi \in \Phi : [(M_A, s_a) \models \phi] \leq [(M_C, s_c) \models \phi]).$$

In other words, KMTSs that are "more complete" with respect to $\preceq$ have more definite properties with respect to $\leq$, i.e., have more properties that are either *true* or *false*. Moreover, any formula ϕ of 3-valued PML that evaluates to *true* or *false* on a KMTS has the same truth value when evaluated on any more complete structure. This result also holds for PML extended with fixpoint operators, i.e., the propositional μ-calculus [5].

The following theorem states that 3-valued propositional modal logic logically characterizes the equivalence relation induced by the completeness preorder $\preceq$.

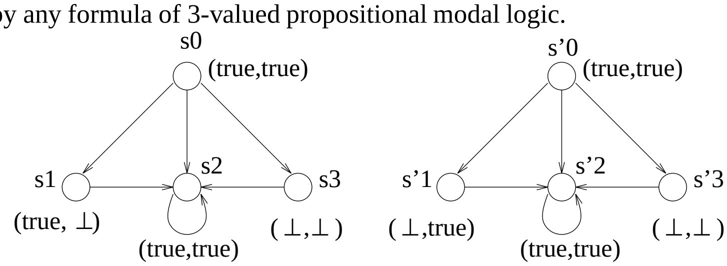
Theorem 5 *[5] Let* $M_1 = (S_1, P, \xrightarrow{must}_1, \xrightarrow{may}_1, L_1)$ *and* $M_2 = (S_2, P, \xrightarrow{must}_2, \xrightarrow{may}_2, L_2)$ *be KMTSs such that* $s_1 \in S_1$ *and* $s_2 \in S_2$*, and let* Φ *denote the set of all formulas of 3-valued propositional modal logic. Then*

$$(\forall \phi \in \Phi : [(M_1, s_1) \models \phi] = [(M_2, s_2) \models \phi]) \textit{ iff } (s_1 \preceq s_2 \textit{ and } s_2 \preceq s_1).$$

Note that if two states s_1 and s_2 are *bisimilar*, denoted $s_1 \sim s_2$, this implies both $s_1 \preceq s_2$ and $s_2 \preceq s_1$. This means that 3-valued propositional modal logic cannot distinguish between bisimilar states.

However, the converse is not true: $s_1 \preceq s_2$ and $s_2 \preceq s_1$ does not imply $s_1 \sim s_2$. This is illustrated by the example below. The existence of such an example proves that, in contrast with 2-valued propositional modal logic, 3-valued propositional modal logic is *not* a logical characterization of bisimulation.

Example 6 [5] Here is an example of two non-bisimilar states that cannot be distinguished by any formula of 3-valued propositional modal logic.



These two partial Kripke structures have two atomic propositions p and q, whose truth value is defined in each state as indicated in the figure by a pair of the form (p,q). We have the following relations:

- $s_2 \preceq s'_2$ and $s'_2 \preceq s_2$,
- $s_3 \preceq s'_3$ and $s'_3 \preceq s_3$,
- $s_1 \preceq s'_2$ and $s'_3 \preceq s_1$, $s'_1 \preceq s_2$ and $s_3 \preceq s'_1$,
- $s_0 \preceq s'_0$ and $s'_0 \preceq s_0$.

We have that $s_0 \preceq s'_0$ and $s'_0 \preceq s_0$, but $s_0 \not\sim s'_0$ since s_1 is not bisimilar to any state in the second partial Kripke structure. ■

4. Three-Valued Model Checking

Given a state s of a 3-valued model M and a formula ϕ, how to compute the value $[(M,s) \models \phi]$?

This is the *3-valued model checking* problem. In [6], it is shown that computing $[(M_A,s) \models \phi]$ can be reduced to two traditional (2-valued) model-checking problems on complete systems (such as Kripke structures or Labeled Transition Systems).

Theorem 7 *[6] The model-checking problem for a 3-valued temporal logic can be reduced to two model-checking problems for the corresponding 2-valued logic.*

The reduction can be performed in 3 steps as follows.

Step 1. Complete M into two "extreme" complete Kripke structures, called the **optimistic** M_o and **pessimistic** M_p completions, defined as follows:

- Extend P to P' such that, for every $p \in P$ there exists a $\overline{p} \in P'$ such that $L(s,p) = \text{comp}(L(s,\overline{p}))$ for all s in S.
- $M_o = (S, L_o, \xrightarrow{must})$ with
$$L_o(s,p) \stackrel{\text{def}}{=} \begin{cases} true & \text{if } L(s,p) = \perp \\ L(s,p) & \text{otherwise} \end{cases}$$
- $M_p = (S, L_p, \xrightarrow{may})$ with
$$L_p(s,p) \stackrel{\text{def}}{=} \begin{cases} false & \text{if } L(s,p) = \perp \\ L(s,p) & \text{otherwise} \end{cases}$$

Step 2. Transform the formula ϕ to its positive form $T(\phi)$ by pushing negation inwards using De Morgan's laws, and replacing remaining negations $\neg p$ at the propositional level by (p).

Step 3. Evaluate $T(\phi)$ on M_o and M_p using traditional 2-valued model checking, and combine the results as follows:

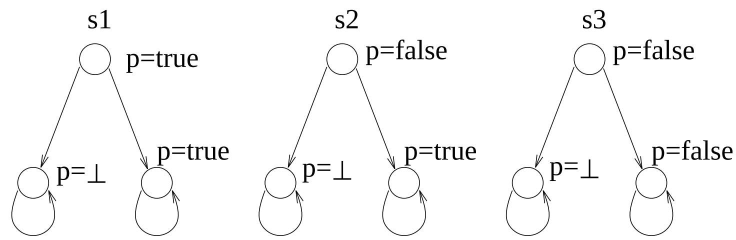
$$[(M,s) \models \phi] = \begin{cases} true & \text{if } (M_p,s) \models T(\phi) \\ false & \text{if } (M_o,s) \not\models T(\phi) \\ \perp & \text{otherwise} \end{cases}$$

This can be done using existing model-checking tools! The formula is *true* at s if it is *true* under the pessimistic interpretation, is *false* at s if it is *false* under the optimistic interpretation, and is $\perp$ otherwise.

It can be proved [6] that the above procedure computes the correct value for $[(M,s) \models \phi]$ according to the 3-valued semantics defined in the previous section.

An immediate corollary from this result is that *3-valued model checking has the same (time and space) complexity as traditional 2-valued model checking*. Indeed, the transformations of M into M_o and M_p, and of ϕ into $T(\phi)$ can be done in linear time and logarithmic space in the size of M and ϕ, respectively.

Example 8 [5] Consider the three following partial Kripke structures with a single atomic proposition p, whose truth value is defined in each state as indicated in the figure.



The formula $A(true\,\mathscr{U}\,p)$ of 3-valued CTL is read "for all paths, does p eventually hold?". It has a different truth value in each of the top states of these partial Kripke structures: $[s_1 \models A(true\,\mathscr{U}\,p)] = true$, $[s_2 \models A(true\,\mathscr{U}\,p)] = \perp$, and $[s_3 \models A(true\,\mathscr{U}\,p)] = false$. ■

5. Generalized Model Checking

However, as argued in [6], the semantics of $[(M,s) \models \phi]$ returns $\perp$ more often than it should. Consider a KMTS M consisting of a single state s such that the value of proposition p at s is $\perp$ and the value of q at s is *true*. The formulas $p \vee \neg p$ and $q \wedge (p \vee \neg p)$ are $\perp$ at s, although in all complete Kripke structures more complete than (M,s) both formulas evaluate to *true*. This problem is not confined to formulas containing subformulas that are tautological or unsatisfiable. Consider a KMTS M' with two states s_0 and s_1 such that $p = q = true$ in s_0 and $p = q = false$ in s_1, and with a *may*-transition from s_0 to s_1. The formula $AX\,p \wedge \neg AX\,q$ (which is neither a tautology nor unsatisfiable) is $\perp$ at s_0, yet in all complete structures more complete than (M', s_0) the formula is *false*.

This observation is used in [6] to define an alternative 3-valued semantics for modal logics called the *thorough* semantics since it does more than the other semantics to discover whether enough information is present in a KMTS to give a definite answer. Let the *completions* $\mathscr{C}(M,s)$ of a state s of a KMTS M be the set of all states s' of complete Kripke structures M' such that $s \preceq s'$.

Definition 9 *Let ϕ be a formula of any two-valued logic for which a satisfaction relation $\models$ is defined on complete Kripke structures. The truth value of ϕ in a state s of a KMTS M under the* thorough *interpretation, written $[(M,s) \models \phi]_t$, is defined as follows:*

$$[(M,s) \models \phi]_t = \begin{cases} true \;\; if\ (M',s') \models \phi \ for\ all\ (M',s')\ in\ \mathscr{C}(M,s) \\ false\ if\ (M',s') \not\models \phi \ for\ all\ (M',s')\ in\ \mathscr{C}(M,s) \\ \perp \quad\ otherwise \end{cases}$$

It is easy to see that, by definition, we always have $[(M,s) \models \phi] \leq [(M,s) \models \phi]_t$. In general, interpreting a formula according to the thorough three-valued semantics is equivalent to solving two instances of the generalized model-checking problem [6].

Definition 10 (Generalized Model-Checking Problem) *Given a state s of a KMTS M and a formula ϕ of a (two-valued) temporal logic L, does there exist a state s' of a complete Kripke structure M' such that $s \preceq s'$ and $(M',s') \models \phi$?*

This problem is called *generalized model-checking* since it generalizes both model checking and satisfiability checking. At one extreme, where $M = (\{s_0\}, P, \xrightarrow{must} = \xrightarrow{may} = \{(s_0,s_0)\}, L)$ with $L(s_0,p) = \perp$ for all $p \in P$, all complete Kripke structures are more complete than M and the problem reduces to the satisfiability problem. At the other extreme, where M is complete, only a single structure needs to be checked and the problem reduces to model checking.

Therefore, the worst-case complexity for the generalized model-checking problem will never be better than the worst-case complexities for the model-checking and satisfiability problems for the corresponding logic. The following theorem formally states that the generalized model-checking problem is at least as hard as the satisfiability problem.

Theorem 11 *[6] Let L denote the propositional μ-calculus or any of its fragments (propositional logic, PML, LTL, CTL, CTL*, etc.). Then the satisfiability problem for L is reducible (in linear-time and logarithmic space) to the generalized model-checking problem for L.*

Is generalized model checking harder than satisfiability? It depends.

For *branching-time* temporal logics, it can be shown [6] that generalized model checking has the same complexity as satisfiability.

Theorem 12 *[6] Let L denote propositional logic, PML, CTL, or any branching-time logic including CTL (such as CTL* or the propositional μ-calculus). The generalized model-checking problem for the logic L has the same complexity as the satisfiability problem for L.*

In contrast, for *linear-time* temporal logic (LTL), generalized model checking can be harder than satisfiability [25]. We have the following.

Theorem 13 *[25] Given a state s_0 of partial Kripke structure $M = (S, L, \mathscr{R})$ and an LTL formula ϕ, one can construct an alternating parity word automaton $A_{(M,s_0),\phi}$ over a 1-letter alphabet with at most $O(|S| \cdot 2^{2^{|\phi| log(|\phi|)}})$ states and $2^{O(|\phi|)}$ priorities such that*

$$(\exists (M', s'_0) : s_0 \preceq s'_0 \textit{ and } (M', s'_0) \models \phi) \textit{ iff } \mathscr{L}(A_{(M,s_0),\phi}) \neq \emptyset.$$

Theorem 14 *[25] The generalized model-checking problem for linear-time temporal logic is 2EXPTIME-complete.*

For LTL, generalized model checking is thus *harder* than satisfiability and model checking, since both of these problems are PSPACE-complete for LTL. Algorithms for LTL generalized model checking use alternating/tree automata [25]. Other problems of

Logic	MC	SAT	GMC
Propositional Logic	Linear	NP-complete	NP-complete
PML	Linear	PSPACE-complete	PSPACE-complete
CTL	Linear	EXPTIME-complete	EXPTIME-complete
μ-calculus	NP$\cap$co-NP	EXPTIME-complete	EXPTIME-complete
LTL	PSPACE-complete	PSPACE-complete	2EXPTIME-complete

Figure 1. Known results on the complexity in the size of the formula for (2-valued and 3-valued) model checking (MC), satisfiability (SAT) and generalized model checking (GMC).

that flavor include the *realizability* [1] and *synthesis* [42,43] problems for linear-time temporal logic specifications.

Figure 1 summarizes the previous complexity results. These results show that the complexity in the size of the formula of computing $[(M,s) \models \phi]_t$ (GMC) is always higher than that of computing $[(M,s) \models \phi]$ (MC).

Regarding the complexity in the size of the model $|M|$, it is shown in [6] that generalized model checking for CTL can be solved in time quadratic in $|M|$. For LTL, generalized model checking can be solved in time polynomial in $|M|$ [25]. More precisely, the complexity in $|M|$ is

- *linear* for safety ($\Box\, p$) and weak (i.e., recognizable by Deterministic Weak Word automata) properties;
- *quadratic* for response ($\Box(p \rightarrow \Diamond q)$), persistence ($\Diamond \Box\, p$) and generalized reactivity[1] properties [32].

Note that for CTL and LTL, generalized model checking is PTIME-hard in $|M|$ while model checking is NLOGSPACE-complete in $|M|$ [18].

6. How to Generate 3-Valued Abstractions

In [20], it is shown how to adapt the abstraction mappings of [9] to construct abstractions that are less complete than a given concrete program with respect to the completeness preorder.

Definition 15 *Let $M_C = (S_C, P, \xrightarrow{must}_C, \xrightarrow{may}_C, L_C)$ be a (concrete) KMTS. Given a set S_A of abstract states and a total[1] abstraction relation on states $\rho \subseteq S_C \times S_A$, we define the (abstract) KMTS $M_A = (S_A, P, \xrightarrow{must}_A, \xrightarrow{may}_A, L_A)$ as follows:*

- $a \xrightarrow{must}_A a'$ *if* $\forall c \in S_C : c\rho a \Rightarrow (\exists c' \in S_C : c'\rho a' \wedge c \xrightarrow{must}_C c')$*;*
- $a \xrightarrow{may}_A a'$ *if* $\exists c, c' \in S_C : c\rho a \wedge c'\rho a' \wedge c \xrightarrow{may}_C c'$*;*
- $L_A(a,p) = \begin{cases} \mathit{true} \ \ \mathit{if}\ \forall c : c\rho a \Rightarrow L_C(c,p) = \mathit{true} \\ \mathit{false}\ \mathit{if}\ \forall c : c\rho a \Rightarrow L_C(c,p) = \mathit{false} \\ \perp \quad \mathit{otherwise} \end{cases}$

The previous definition can be used to build abstract KMTSs.

[1] That is, $(\forall c \in S_C : \exists a \in S_A : c\rho a)$ and $(\forall a \in S_A : \exists c \in S_C : c\rho a)$.

Theorem 16 *Given a KMTS M_C, any KMTS M_A obtained by applying Definition 15 is such that $M_A \preceq M_C$.*

Given a KMTS M_C, any abstraction M_A less complete than M_C with respect to the completeness preorder $\preceq$ can be constructed using Definition 15 by choosing the inverse of ρ as $\mathscr{B}$ [20]. When applied to MTSs with *may*-transitions only, the above definition coincides with traditional "conservative" abstraction that is a *simulation* of the concrete system. Building a 3-valued abstraction can be done using existing abstraction techniques at the *same computational cost* as building a conservative abstraction [20].

7. Application to Software Model Checking

The usual procedure for performing program verification via predicate abstraction and iterative abstraction refinement is the following (e.g., see [3,12]).

1. Abstract: compute an abstraction M_A that simulates the concrete program M_C.
2. Check: given a universal property ϕ, decide whether $M_A \models \phi$.
    - if $M_A \models \phi$: stop (the property is proved: $M_C \models \phi$).
    - if $M_A \not\models \phi$: go to Step 3.
3. Refine: refine M_A (possibly using a counter-example found in Step 2). Then go to Step 1.

Using predicate abstraction [26,13,50], the abstraction computed in Step 1 is defined with respect to a set of predicates $\Psi = \{\psi_1, \ldots, \psi_n\}$, which are typically quantifier-free formulas of first-order logic (for instance, $(x == y+1) \vee (x < y-5)$). An abstract state is defined as a vector of n bits induced by n-ary conjunctions, with each predicate ψ_i contributing either ψ_i or $\neg\psi_i$, which identifies all concrete states that satisfy the same set of predicates in Ψ. Thus, a concrete state c is abstracted by an abstract state $[c] = (b_1, \ldots, b_n)$ such that $\forall 1 \leq i \leq n : b_i = \psi_i(c)$. A transition is defined between abstract states $[c_1]$ and $[c_2]$ if there exists a transition from some concrete state in $[c_1]$ to some concrete state in $[c_2]$. The resulting abstract transition system M_A is guaranteed by construction to simulate the concrete transition system M_C.

Since M_A simulates M_C, one can only prove the correctness of universal properties (i.e., properties over all paths) of M_C by analyzing M_A in Step 2. In particular, the violation of a universal property (or equivalently, the satisfaction of an existential property) cannot be established by analyzing such abstractions in general. Step 3 typically involves the addition of new predicates to refine the current abstraction. Note that the three steps above can also be interleaved and performed in a strict demand-driven fashion as described in [28].

Thanks to the framework described in the previous sections, we can now present the following new procedure for automatic abstraction [21].

1. Abstract: compute an abstraction M_A using Def. 15 such that $M_A \preceq M_C$.
2. Check: given *any* property ϕ,

   (a) (3-valued model checking) compute $[M_A \models \phi]$.

   - if $[M_A \models \phi] = \mathit{true}$ or *false*: stop (the property is proved (resp. disproved) on M_C).
   - if $[M_A \models \phi] = \perp$, continue.

   (b) (generalized model checking) compute $[M_A \models \phi]_t$.

   - if $[M_A \models \phi]_t = \mathit{true}$ or *false*: stop (the property is proved (resp. disproved) on M_C).
   - if $[M_A \models \phi] = \perp$, go to Step 3.

3. Refine: refine M_A (possibly using a counter-example found in Step 2). Then go to Step 1.

This new procedure strictly generalizes the traditional one in several ways. First, any temporal logic formula can be checked (not just universal properties). Second, all correctness proofs and counter-examples obtained by analyzing any abstraction M_A such that $M_A \preceq M_C$ are guaranteed to be sound (i.e., hold on M_C) for any property (by Theorem 4). Third, verification results can be more precise than with the traditional procedure: the new procedure will not only return *true* whenever the traditional one returns *true* (trivially, since the former includes the latter), but it can also return *true* more often thanks to a more thorough check using generalized model-checking, and it can also return *false*. The new procedure can thus terminate sooner and more often than the traditional procedure — the new procedure will never loop through its 3 steps more often than the traditional one.

Remarkably, each of the 3 steps of the new procedure can be performed at roughly the same cost as the corresponding step of the traditional procedure: as shown in [20], building a 3-valued abstraction using Definition 15 (Step 1 of new procedure) can be done at the same computational cost as building a conservative abstraction (Step 1 of traditional procedure); computing $[M_A \models \phi]$ in Step 2.a can be done at the same cost at traditional (2-valued) model checking [6]; following the results of Section 5, computing $[M_A \models \phi]_t$ in Step 2.b can be more expensive than Step 2.a, but is still polynomial (typically linear or quadratic) in the size of M_A; Step 3 of the new procedure is similar to Step 3 of the traditional one (in the case of LTL properties for instance, refinement can be guided by error traces found in Step 2 as in the traditional procedure). Finally note that the new procedure could also be adapted so that the different steps are performed in a demand-driven basis following the work of [28].

8. Examples

We now give examples of programs, models and properties, all taken from [21], where computing $[(M,s) \models \phi]_t$ returns a more precise answer than $[(M,s) \models \phi]$.

Consider the three programs shown in Figure 2, where x and y denote variables, and f denotes some unknown function. The notation “x,y = 1,0” means variables x and y are simultaneously assigned to values 1 and 0, respectively. Consider the two predicates

```
program C1() {
  x,y = 1,0;
  x,y = f(x),f(y);
  x,y = 1,0;
}
```

```
program C2() {
  x,y = 1,0;
  x,y = 2*f(x),f(y);
  x,y = 1,0;
}
```

```
program C3() {
  x = 1;
  x = f(x);
}
```

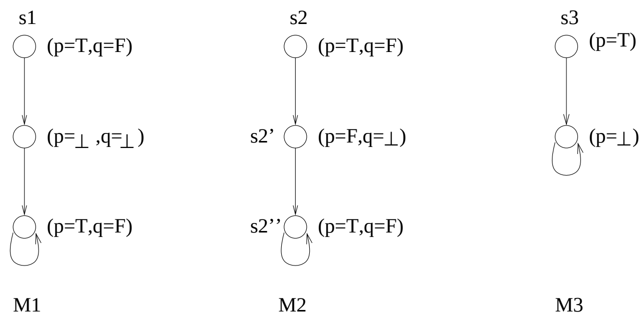


Figure 2. Examples of programs and models

p : "is x odd?" and q : "is y odd?". Figure 2 shows an example of KMTS model for each of the three programs. These models can be computed automatically using Definition 15, predicate abstraction techniques and predicates p and q, so that by construction they satisfy Theorem 16. Each model is a KMTS with *must*-transitions only and with atomic propositions p and q whose truth value is defined in each state as indicated in the figure.

Consider the LTL formula $\phi_1 = \Diamond q \Rightarrow \Box(p \vee q)$ (where $\Diamond$ is read "eventually" and $\Box$ is read "always" [38]). While $[(M_1,s_1) \models \phi_1] = \perp$, $[(M_1,s_1) \models \phi_1]_t = \mathit{true}$. In other words, using the thorough interpretation yields a more definite answer in this case. Note that the gain in precision obtained in this case is somewhat similar to the gain in precision that can be obtained using an optimization called *focusing* [3] aimed at recovering some of the imprecision introduced when using *cartesian abstraction* (see [3,20]).

Consider now the formula $\phi_2 = \Diamond q \wedge \Box(p \vee \neg q)$ evaluated on (M_2,s_2). In this case, we have $[(M_2,s_2) \models \phi_2] = \perp$, while $[(M_2,s_2) \models \phi_2]_t = \mathit{false}$. Again, using the thorough interpretation yields a more definite answer, although solving a generalized model-checking problem is necessary to return a negative answer. Indeed, one needs to prove in this case that there exists a computation of (M_2,s_2) (namely $s_2 s_2' s_2''^{\omega}$ – there is only one computation in this simple example) that does not have any completion satisfying ϕ_2, which itself requires using alternating automata and can thus be more expensive as discussed in Section 5. Another example of formula is $\phi_2' = \bigcirc q \wedge \Box(p \vee \neg q)$ (where $\bigcirc$ is read "next" [38]). Again we have that $[(M_2,s_2) \models \phi_2'] = \perp$, while $[(M_2,s_2) \models \phi_2']_t = \mathit{false}$. Note that, although ϕ_2' is an LTL safety formula and hence is within the scope of analysis of existing tools ([4], [8], etc.), none of these tools can prove that ϕ_2' does not hold: this result can only be obtained using generalized model checking.

Last, consider (M_3,s_3) and formula $\phi_3 = \Box p$. In this case, we have both $[(M_3,s_3) \models \phi_3] = [(M_3,s_3) \models \phi_3]_t = \perp$, and the thorough interpretation cannot produce a more definite answer than the standard 3-valued interpretation.

9. Precision of GMC Vs. MC

How often is generalized model checking (GMC) more precise than model checking (MC)? This question is addressed in [19]. Specifically, [19] studies when it is possible to reduce GMC(M, ϕ) to MC(M, ϕ'). Such a transformed formula ϕ' is called a *semantic minimization* of ϕ. [19] shows that propositional logic, PML and the propositional μ-calculus are *closed under semantic minimization*, i.e., that a reduction from GMC(M, ϕ) to MC(M, ϕ') is always possible for ϕ and ϕ' in propositional logic, PML or the μ-calculus. But in contrast, the temporal logics LTL, CTL and CTL* are *not* closed under semantic minimization.

[19] also identifies *self-minimizing* formulas, i.e., formulas ϕ for which GMC(M, ϕ) and MC(M, ϕ) are equivalent. By definition, GMC and MC have thus the same precision for any self-minimizing formula. Self-minimizing formulas can be defined both semantically using automata-theoretic techniques (for instance, this is EXPTIME-hard in $|\phi|$ for the μ-calculus) and syntactically by providing syntactic *sufficient* criteria which are linear in $|\phi|$. For instance, [19] shows that *any formula that does not contain any atomic proposition in mixed polarity (in its negation normal form) is self-minimizing*.

Fortunately, in practice, many frequent formulas are self-minimizing, and MC is as precise as GMC for those.

10. Other Related Work

The framework presented in the previous sections has also been extended to *open systems* [18] (i.e., systems which interacts with their environment), and to *games* in general [14]. For instance, [14] studies *abstractions of games* where moves of each player can be abstracted while preserving winning strategies of *both* players. An abstraction of a game is now a game where each player has both may and must moves, yielding may/must strategies. In this context, the completeness preorder becomes an *alternating refinement* relation, logically characterized by 3-valued alternating μ-calculus [2].

Another interesting topic is *semantic completeness*: given an infinite-state system C and property ϕ, if C satisfies ϕ, does there exist a finite-state abstraction A of C such that A satisfies ϕ?

For arbitrary formulas ϕ of LTL, the existence of such finite abstractions A can be guaranteed provided that abstractions A are extended to include the modeling of *fairness* constraints [33], which are used to model termination in loops. For arbitrary formulas ϕ of the propositional μ-calculus (hence including existential properties), the existence of such finite abstractions can again be guaranteed but now provided that abstractions A may include *nondeterministic* must transitions [37], also called *hyper-must* transitions [40,11,14]. When using hyper-must transitions, abstraction refinement with predicate abstraction becomes *monotonic* with respect to the completeness preorder, i.e., adding a predicate p now generates an abstraction which is always more complete than the previous one (see [20,47,14]).

11. Concluding Remarks

This paper presents an introduction to 3-valued "may/must" abstraction-based software model checking for sound property verification and falsification. The results presented here previously appeared in a series of papers [5,6,20,21,22,18,14,19,25]. These results shed light on the techniques used in abstraction-based software model checking tools like SLAM [4], BLAST [28], YASM [27], TERMINATOR [7] and YOGI [24]. In particular, YASM [27] uses 3-valued models as described in this paper, while YOGI [24] uses (compositional) may/must abstractions that share transitions instead of states.

The reader interested in the topic of this paper should consult the references listed above, as well as the related work discussed in those references. We mention below only a few other main pointers to related work.

The study of abstraction for model checking of both universal and existential program properties was pioneered in [9,10]. This work defines a general abstraction framework where abstractions are *mixed transition systems*. Intuitively, a mixed transition system is a modal transition system without the constraint $\xrightarrow{must} \subseteq \xrightarrow{may}$. Mixed transition systems are more expressive and, in full generality, allow for a 4-valued world where some mixed transition systems cannot be refined by any complete (2-valued) system [31]. Nevertheless, the goal and some of the results of this prior work are very similar to our own work with 3-valued models and logics. The use of "conservative" abstractions for proving properties of the full μ-calculus is also discussed in [45].

Extended transition systems [39] are Labeled Transition Systems extended with a *divergence predicate*, and can be viewed as a particular class of 3-valued models [5,30]. In [5], it is shown that Hennessy-Milner Logic with a 3-valued interpretation provides an alternative characterization of the divergence preorder in addition to the intuitionistic interpretation of Plotkin [48]. Further work on divergence preorders and logics to characterize them can be found in [48,51]. In all this work, logic formulas are interpreted normally in the 2-valued sense. The close correspondence between 3-valued logic and Plotkin's intuitionistic modal logic inspired the reduction procedure from 3-valued model checking to 2-valued model checking of [6] (see Section 4).

Prior to the work reported here, most work on 3-valued modal logic focused on its proof theory (e.g., [46,15]). Our definition of partial Kripke structure is closest to [16], where two interpretations of modal logic are presented: a many-valued version and another version based on obtaining 2-valued interpretations from each of a set of experts. [16] shows that such a multi-expert interpretation corresponds in a precise way to a multi-valued interpretation, similarly to how we show that a 3-valued interpretation can be obtained by separate optimistic and pessimistic interpretations. However, [16] does not define a completeness preorder over models or characterization results.

In [44], a 3-valued logic is used for program shape analysis. The state of a program store is represented as a 3-valued structure of first-order logic. The possible values of the program store as the program executes are conservatively computed by a traditional "may-only" abstract interpretation of the concrete program with such a structure as the abstract domain. The main technical result is an embedding theorem showing that, for a certain class of abstraction functions on the domain of such structures, the interpretation of a first-order formula on the abstract structure is less definite than its interpretation on the structure itself. Despite a common use of 3-valued logic, our goals and results are fairly different from [44] since our focus is on 3-valued abstractions of *reactive* (transition) systems and the sound verification (and falsification) of *temporal* properties of such systems.

Acknowledgements. This paper covers one of the lectures (Lecture 4) which I gave at the 2013 Marktoberdorf Summer School. I thank the organizers of the Summer School for encouraging me to write this paper. I also thank my co-authors (in chronological order) Glenn Bruns, Radha Jagadeesan, Michael Huth, Luca de Alfaro, and Nir Piterman for their insights and without whom this work would not exist.

References

[1] M. Abadi, L. Lamport, and P. Wolper. Realizable and unrealizable concurrent program specifications. In *Proc. 16th Int. Colloquium on Automata, Languages and Programming*, volume 372 of *Lecture Notes in Computer Science*, pages 1–17. Springer-Verlag, July 1989.

[2] R. Alur, T.A. Henzinger, and O. Kupferman. Alternating time temporal logic. *Journal of the ACM*, 49:672–713, 2002.

[3] T. Ball, A. Podelski, and S. K. Rajamani. Boolean and Cartesian Abstraction for Model Checking C Programs. In *Proceedings of TACAS'2001 (Tools and Algorithms for the Construction and Analysis of Systems)*, volume 2031 of *Lecture Notes in Computer Science*. Springer-Verlag, April 2001.

[4] T. Ball and S. Rajamani. The SLAM Toolkit. In *Proceedings of CAV'2001 (13th Conference on Computer Aided Verification)*, volume 2102 of *Lecture Notes in Computer Science*, pages 260–264, Paris, July 2001. Springer-Verlag.

[5] G. Bruns and P. Godefroid. Model Checking Partial State Spaces with 3-Valued Temporal Logics. In *Proceedings of CAV'99 (11th Conference on Computer Aided Verification)*, volume 1633 of *Lecture Notes in Computer Science*, pages 274–287, Trento, July 1999. Springer-Verlag.

[6] G. Bruns and P. Godefroid. Generalized 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, University Park, August 2000. Springer-Verlag.

[7] B. Cook, A. Podelski, and A. Rybalchenko. Termination Proofs for Systems Code. In *Proceedings of PLDI'2006 (ACM SIGPLAN 2006 Conference on Programming Language Design and Implementation)*, pages 415–426, Ottawa, June 2006.

[8] J. C. Corbett, M. B. Dwyer, J. Hatcliff, S. Laubach, C. S. Pasareanu, Robby, and H. Zheng. Bandera: Extracting Finite-State Models from Java Source Code. In *Proceedings of the 22nd International Conference on Software Engineering*, 2000.

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

[10] D. Dams, R. Gerth, and O. Grumberg. Abstract interpretation of reactive systems. *ACM Transactions on Programming Languages and Systems*, 19(2):253–291, 1997.

[11] D. Dams and K. Namjoshi. The existence of finite abstractions for branching time model checking. In *Proceedings of LICS'2004 (19th IEEE conference on Logic in Computer Science)*, Turku, July 2004.

[12] S. Das and D. L. Dill. Successive Approximation of Abstract Transition Relations. In *Proceedings of LICS'2001 (16th IEEE Symposium on Logic in Computer Science)*, pages 51–58, Boston, June 2001.

[13] S. Das, D. L. Dill, and S. Park. Experience with Predicate Astraction. In *Proc. of the 11th International Conference on Computer-Aided Verification*, Lecture Notes in Computer Science, pages 160–172, Trento, July 1999. Springer Verlag.

[14] L. de Alfaro, P. Godefroid, and R. Jagadeesan. Three-Valued Abstractions of Games: Uncertainty, but with Precision. In *Proceedings of LICS'2004 (19th IEEE Symposium on Logic in Computer Science)*, pages 170–179, Turku, July 2004.

[15] M. Fitting. Many-Valued Modal Logics I. *Fundamenta Informaticae*, 15:235–254, 1992.

[16] M. Fitting. Many-Valued Modal Logics II. *Fundamenta Informaticae*, 17:55–73, 1992.

[17] P. Godefroid. Model Checking for Programming Languages using VeriSoft. In *Proceedings of POPL'97 (24th ACM Symposium on Principles of Programming Languages)*, pages 174–186, Paris, January 1997.

[18] P. Godefroid. Reasoning about Abstract Open Systems with Generalized Module Checking. In *Proceedings of EMSOFT'2003 (3rd Conference on Embedded Software)*, volume 2855 of *Lecture Notes in Computer Science*, pages 223–240, Philadelphia, October 2003. Springer-Verlag.

[19] P. Godefroid and M. Huth. Model Checking Vs. Generalized Model Checking: Semantic Minimizations for Temporal Logics. In *Proceedings of LICS'2005 (20th IEEE Symposium on Logic in Computer Science)*, pages 158–167, Chicago, June 2005.

[20] P. Godefroid, M. Huth, and R. Jagadeesan. Abstraction-based Model Checking using Modal Transition Systems. In *Proceedings of CONCUR'2001 (12th International Conference on Concurrency Theory)*, volume 2154 of *Lecture Notes in Computer Science*, pages 426–440, Aalborg, August 2001. Springer-Verlag.

[21] P. Godefroid and R. Jagadeesan. Automatic Abstraction Using Generalized Model Checking. In *Proceedings of CAV'2002 (14th Conference on Computer Aided Verification)*, volume 2404 of *Lecture Notes in Computer Science*, pages 137–150, Copenhagen, July 2002. Springer-Verlag.

[22] P. Godefroid and R. Jagadeesan. On the Expressiveness of 3-Valued Models. In *Proceedings of VMCAI'2003 (4th Conference on Verification, Model Checking and Abstract Interpretation)*, volume 2575 of *Lecture Notes in Computer Science*, pages 206–222, New York, January 2003. Springer-Verlag.

[23] P. Godefroid, N. Klarlund, and K. Sen. DART: Directed Automated Random Testing. In *Proceedings of PLDI'2005 (ACM SIGPLAN 2005 Conference on Programming Language Design and Implementation)*, pages 213–223, Chicago, June 2005.

[24] P. Godefroid, A.V. Nori, S.K. Rajamani, and S.D. Tetali. Compositional May-Must Program Analysis: Unleashing The Power of Alternation. In *Proceedings of POPL'2010 (37th ACM Symposium on Principles of Programming Languages)*, pages 43–55, Madrid, January 2010.

[25] P. Godefroid and N. Piterman. LTL Generalized Model Checking Revisited. *International Journal on Software Tools for Technology Transfer (STTT)*, 13(6):571–584, 2011.

[26] S. Graf and H. Saidi. Construction of Abstract State Graphs with PVS. In *Proceedings of the 9th International Conference on Computer Aided Verification*, volume 1254 of *Lecture Notes in Computer Science*, pages 72–83, Haifa, June 1997. Springer-Verlag.

[27] A. Gurfinkel, O. Wei, and M. Chechik. Yasm: A Software Model Checker for Verification and Refutation. In *Proceedings of CAV'2006 (18th Conference on Computer Aided Verification)*, volume 4144 of *Lecture Notes in Computer Science*, pages 170–174, Seattle, August 2006. Springer-Verlag.

[28] T. Henzinger, R. Jhala, R. Majumdar, and G. Sutre. Lazy Abstraction. In *Proceedings of the 29th ACM Symposium on Principles of Programming Languages*, pages 58–70, Portland, January 2002.

[29] G. J. Holzmann and M. H. Smith. A Practical Method for Verifying Event-Driven Software. In *Proceedings of the 21st International Conference on Software Engineering*, pages 597–607, 1999.

[30] M. Huth, R. Jagadeesan, and D. Schmidt. Modal Transition Systems: a Foundation for Three-Valued Program Analysis. In *Proceedings of the European Symposium on Programming (ESOP'2001)*, volume 2028 of *Lecture Notes in Computer Science*. Springer-Verlag, April 2001.

[31] M. Huth, R. Jagadeesan, and D. Schmidt. A Domain Equation for Refinement of Partial Systems. Submitted to Mathematical Structures in Computer Science, 2002.

[32] Y. Kesten, N. Piterman, and A. Pnueli. Bridging the gap between fair simulation and trace containment. In *15th Computer Aided Verification*, volume 2725 of *Lecture Notes in Computer Science*, pages 381–393. Springer-Verlag, 2003.

[33] Y. Kesten and A. Pnueli. Verification by Augmented Finitary Abstraction. *Information and Computation*, 163(1), 2000.

[34] S. C. Kleene. *Introduction to Metamathematics*. North Holland, 1987.

[35] D. Kozen. Results on the Propositional Mu-Calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[36] K. G. Larsen and B. Thomsen. A Modal Process Logic. In *Proceedings of Third Annual Symposium on Logic in Computer Science*, pages 203–210. IEEE Computer Society Press, 1988.

[37] K. G. Larsen and Liu Xinxin. Equation solving using modal transition systems. In *Proceedings of the 5th IEEE conference on Logic in Computer Science*, pages 108–117. IEEE, 1990.

[38] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, 1992.

[39] R. Milner. A Modal Characterization of Observable Machine Behavior. In *Proc. CAAP'81*, volume 112 of *Lecture Notes in Computer Science*, pages 25–34. Springer-Verlag, 1981.

[40] K. S. Namjoshi. Abstraction for branching time properties. In *Proceedings of CAV'2003 (15th International Conference on Computer Aided Verification)*, volume 2725 of *Lecture Notes in Computer Science*, pages 288–300. Springer, 2003.

[41] K. S. Namjoshi and R. K. Kurshan. Syntactic Program Transformations for Automatic Abstraction. In *Proceedings of the 12th Conference on Computer Aided Verification*, volume 1855 of *Lecture Notes in*

Computer Science, pages 435–449, Chicago, July 2000. Springer-Verlag.

[42] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. of the Sixteenth Symposim on Principles of Programming Languages*, Austin, January 1989.

[43] A. Pnueli and R. Rosner. On the synthesis of an asynchronous reactive module. In *Proceedings of ICALP'89*, Stresa, July 1989.

[44] M. Sagiv, T. Reps, and R. Wilhelm. Parametric Shape Analysis Via 3-Valued Logic. In *Proceedings of the 26th ACM Symposium on Principles of Programming Languages*, January 1999.

[45] H. Saidi and N. Shankar. Abstract and model check while you prove. In *Proc. of the 11th Conference on Computer-Aided Verification*, number 1633 in Lecture Notes in Computer Science, pages 443–454. Springer, 1999.

[46] K. Segerberg. Some Modal Logics Based on a Three-Valued Logic. *Theoria*, 33:53–71, 1967.

[47] S. Shoham and O. Grumberg. Monotonic Abstraction-Refinement for CTL. In *Tools and Algorithms for the Construction and Analysis of Systems: 10th International Conference*, number 2988 in Lecture Notes in Computer Science, pages 546–560. Springer Verlag, 2004.

[48] C. Stirling. Modal Logics for Communicating Systems. *Theoretical Computer Science*, 49:331–347, 1987.

[49] W. Visser, K. Havelund, G. Brat, and S. Park. Model Checking Programs. In *Proceedings of ASE'2000 (15th International Conference on Automated Software Engineering)*, Grenoble, September 2000.

[50] W. Visser, S. J. Park, and J. Penix. Using Predicate Abstraction to Reduce Object-oriented Programs for Model Checking. In *Proceedings of FMSP'00 (Formal methods in Software Practice)*, pages 3–12, Portland, August 2000.

[51] D. Walker. Bisimulation and Divergence. *Information and Computation*, 85(2):202–241, 1990.