Automatic Abstraction Using Generalized Model Checking

Patrice Godefroid[1] and Radha Jagadeesan[⋆2]

[1] Bell Laboratories, Lucent Technologies, god@bell-labs.com
[2] Department of Computer Science, Loyola University of Chicago, radha@cs.luc.edu

Abstract. Generalized model checking is a framework for reasoning about partial state spaces of concurrent reactive systems. The state space of a system is only "partial" (partially known) when a full state-space exploration is not computationally tractable, or when abstraction techniques are used to simplify the system's representation. In the context of automatic abstraction, generalized model checking means checking whether there exists a concretization of an abstraction that satisfies a temporal logic formula. In this paper, we show how generalized model checking can extend existing automatic abstraction techniques (such as predicate abstraction) for model checking concurrent/reactive programs and yield the three following improvements: (1) any temporal logic formula can be checked (not just universal properties as with traditional conservative abstractions), (2) correctness proofs and counter-examples are both guaranteed to be sound, and (3) verification results can be more precise. We study the cost needed to improve precision by presenting new upper and lower bounds for the complexity of generalized model checking in the size of the abstraction.

1 Introduction

How to broaden the scope of model checking to reactive software is currently one of the most challenging open 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 (Unix-like) processes executing arbitrary code [10]; 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., [2, 6, 25, 21, 14]); 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.

⋆ Supported in part by NSF.

Recently [11], we showed how automatic abstraction can be performed to verify arbitrary formulas of the propositional μ-calculus [17] 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 [3] and Modal Transition Systems [19, 11]. Reasoning about such systems requires 3-valued temporal logics [3], 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. It is shown in [11] that building a 3-valued abstraction can be done using existing abstraction techniques at the same computational cost as building a conservative abstraction.

In this paper, we build upon this previous work and study the use of *generalized model checking* in the context of automatic abstraction. Generalized model checking was introduced in [4] as a way to improve precision when reasoning about partially defined systems. Specifically, given a model M and a temporal-logic formula ϕ, the generalized model-checking problem is to decide whether there exists a complete system M' that is more complete than M and that satisfies the formula ϕ. The model M can thus be viewed as a partial solution to the satisfiability problem for ϕ which reduces the solution space to complete systems that are more complete than M with respect to a completeness preorder. Generalized model checking is thus a generalization of both satisfiability and model checking. Algorithms and complexity bounds for the generalized model-checking problem for various temporal logics were presented in [4].

We present here several new results. First, we study the complexity of generalized model checking in the size of the abstraction $|M|$, and provide new upper and lower bounds for various temporal logics. We show that the worst-case runtime complexity of generalized model checking for the temporal logics LTL and CTL can be quadratic in $|M|$, but that generalized model checking can be solved in time linear in $|M|$ in the case of *persistence* properties, i.e., properties recognizable by *co-Büchi automata*. Complexity in the size of the abstraction is important in practice since the abstraction can be large and hence is often the main limiting factor that prevents obtaining verification results.

Second, we show how generalized model checking can help improve precision of verification via automatic abstraction. We present a new process for iterative abstraction refinement that takes advantage of the techniques introduced here. Iterative abstraction refinement [1, 9, 13] in the context of predicate abstraction [12] is a process for automatically refining an abstraction that is guided by spurious counter-examples found at higher levels of abstraction. In contrast with abstractions used in traditional program analysis, iterative abstraction refinement using predicate abstraction has thus the advantage of making it possible to adapt the level of abstraction dynamically on a demand-driven basis guided by the verification needs. Unfortunately, refining an abstraction can be an expensive operation since successive abstraction refinements can generate exponentially larger abstractions. Better precision when analyzing an abstraction is therefore critical to avoid unnecessary refinements of this abstraction. We believe generalized model checking is a useful addition to existing techniques for automatic abstraction since it can help an iterative "abstract-check-refine" verification process terminate sooner and more often by providing better analysis precision for a cost polynomial (quadratic or linear) in the size of the abstraction.

2 Background: Generalized Model Checking

In this section, we recall the main ideas and key notions behind the framework of [3, 4, 11, 15] for reasoning about partially defined systems. Examples of modeling formalisms for representing such systems are *partial Kripke structures* (PKS) [3], *Modal Transition Systems* (MTS) [19, 11] or *Kripke Modal Transition Systems* (KMTS) [15].

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* $L : S \times P \to \{true, \bot, false\}$ *is an* interpretation *that associates a truth value in* $\{true, \bot, false\}$ *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 $\bot$ (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 $\bot$ in any state. It can be shown that PKSs, MTSs, KMTSs and variants of KMTSs where transitions are labeled and/or two interpretation functions L^{may} and L^{must} are used [15], 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).

In interpreting propositional operators on KMTSs, we use Kleene's strong 3-valued propositional logic [16]. 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:*

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

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.

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 $\mathcal{B} \subseteq S_A \times S_C$ such that $(s_a, s_c) \in \mathcal{B}$ 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 $(s'_a, s'_c) \in \mathcal{B}$,*
- *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 $(s'_a, s'_c) \in \mathcal{B}$.*

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 complete*, than M_C. The inverse of the completeness preorder is also called *refinement preorder* in [19, 15, 11]. Note that relation $\mathcal{B}$ reduces to a simulation relation when applied to MTSs with *may*-transitions only.

It can be shown that 3-valued PML logically characterizes the completeness preorder [3, 15, 11].

Theorem 1. *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,*

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

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 [3].

In [11], we showed how to adapt the abstraction mappings of [8] to construct abstractions that are less complete than a given concrete program with respect to the completeness preorder.

Definition 4. *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[3] 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* $\exists c, c' \in S_C : c\rho a \wedge c'\rho a' \wedge c \xrightarrow{may}_C c'$;
- $L_A(a, p) = \begin{cases} \mathit{true} \text{ if } \forall c : c\rho a \Rightarrow L_C(c, p) = \mathit{true} \\ \mathit{false} \text{ if } \forall c : c\rho a \Rightarrow L_C(c, p) = \mathit{false} \\ \bot \quad \mathit{otherwise} \end{cases}$

The previous definition can be used to build abstract KMTSs.

Theorem 2. *Given a KMTS M_C, any KMTS M_A obtained by applying Definition 4 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 4 by choosing the inverse of ρ as $\mathcal{B}$ [11]. When applied to MTSs with *may*-transitions only, the above definition coincides with traditional "conservative" abstraction. Building a 3-valued abstraction can be done using existing abstraction techniques at the same computational cost as building a conservative abstraction [11].

Since by construction $M_A \preceq M_C$, any temporal-logic formula ϕ that evaluates to *true* (resp. *false*) on M_A automatically holds (resp. does not hold) on M_C. It is shown in [4] that computing $[(M_A, s) \models \phi]$ can be reduced to two traditional (2-valued) model-checking problems on regular fully-defined systems (such as Kripke structures or Labeled Transition Systems), and hence that 3-valued model-checking for any temporal logic L has the same time and space complexity as 2-valued model checking for the logic L.

However, as argued in [4], the semantics of $[(M, s) \models \phi]$ returns $\bot$ more often than it should. Consider a KMTS M consisting of a single state s such that the

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

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 $p = q = \mathit{true}$ in s_0 and $p = q = \mathit{false}$ in s_1, and with a *may*-transition from s_0 to s_1. The formula $AXp \wedge \neg AXq$ (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 [4] 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* $\mathcal{C}(M, s)$ 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 5. *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:*

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

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 [4].

Definition 6 (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.

Algorithms and complexity bounds for the generalized model-checking problem for various temporal logics were presented in [4]. In the case of branching-time temporal logics, generalized model checking has the same complexity in the size of the formula as satisfiability. In the case of linear-time temporal logic, generalized model checking is EXPTIME-complete in the size of the formula, i.e., harder than both satisfiability and model checking, which are both PSPACE-complete in the size of the formula for LTL. Figure 1 summarizes the complexity results of [4]. 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).

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	EXPTIME-complete

Fig. 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).

Regarding the complexity in the size of the model $|M|$, it is only shown in [4] that generalized model checking can be solved in time quadratic in $|M|$. In the next two sections, we refine this result by presenting new upper and lower bounds for the complexity of generalized model checking in the size of the model for various classes of temporal properties. Our algorithms and constructions make use of automata-theoretic techniques (e.g., see [18]). For basic notions of automata theory (including definitions of nondeterministic/alternating/weak Büchi automata on words and trees), please refer to [18]. Let us simply recall that a co-Büchi acceptance condition is the dual of a Büchi acceptance condition: an infinite execution w satisfies a co-Büchi acceptance condition F if it does not intersect the set F of (rejecting) states infinitely often (i.e., $Inf(w) \cap F = \emptyset$), while an infinite execution w satisfies a Büchi acceptance condition F if it intersects the set F of (accepting) states infinitely often (i.e., $Inf(w) \cap F \neq \emptyset$).

3 Generalized Model Checking for LTL

We first consider the case of properties expressed as linear-time temporal-logic (LTL) formulas [20]. To begin, we recall the following property of LTL: if ϕ is an LTL formula and (M, s) is a complete Kripke structure, $(M, s) \not\models \phi$ is *not* logically equivalent to $(M, s) \models \neg\phi$. Indeed, if $\mathcal{L}(M, s)$ denotes the language of ω-words represented by (M, s), the former statement is equivalent to $\exists w \in \mathcal{L}(M, s) : w \not\models \phi$ while the latter is equivalent to $\forall w \in \mathcal{L}(M, s) : w \models \neg\phi$, which in turn is equivalent to $\forall w \in \mathcal{L}(M, s) : w \not\models \phi$.

Therefore, computing $[(M, s) \models \phi]_t$ in the LTL case reduces to only one generalized model-checking problem, namely "does there exists a completion (M', s') of (M, s) such that $(M', s') \models \phi$?", plus a second problem of the form "does there exists a completion (M', s') of (M, s) such that $(M', s') \not\models \phi$?". This second problem is easier to solve than generalized model checking.

Theorem 3. *Given a state s of a KMTS M and a LTL formula ϕ, checking whether there exists a state s' of a complete Kripke structure M' such that $s \preceq s'$ and $(M', s') \not\models \phi$ can be done in time linear in $|M|$ and is PSPACE-complete in $|\phi|$.*

Proof. (Sketch)[4] For any two infinite sequences $w = a_1 a_2 \ldots$ and $w' = a'_1 a'_2 \ldots$, let $w \leq w'$ denote that $\forall i > 0 : \forall p \in P : L(a_i, p) \leq L(a'_i, p)$. It is easy to show that

[4] Complete proofs are omitted in this extended abstract due to space limitations.

$\exists(M', s') : s \preceq s' \wedge (M', s') \not\models \phi$ iff $\exists w \in \mathcal{L}(M, s) : \exists w' : w \leq w' \wedge w' \not\models \phi$. The latter condition can be reduced to checking nonemptiness of a nondeterministic Büchi word automaton A defined by a product construction of the KMTS (M, s) and of a nondeterministic Büchi automaton $A_{\neg\phi}$ accepting the set of ω-words violating the property ϕ. This product construction does not distinguish *may* and *must* transitions of M and attempts to match all of these to transitions of $A_{\neg\phi}$; occurrences of value $\perp$ in M are matched to both values *true* and *false* in $A_{\neg\phi}$; the size of A is therefore linear in $|M|$. Checking nonemptiness of the resulting Büchi automaton A can be done in time linear in $|A|$ (hence linear in $|M|$). By analogy with traditional model checking, it is easy to show that the problem is also PSPACE-complete in $|\phi|$.

Computing $[(M, s) \models \phi]_t$ for an LTL formula ϕ can thus be done using the following procedure:

1. Check whether $(M, s) \times A_{\neg\phi} = \emptyset$. By the previous theorem, this can be done in time linear in $|M|$ and is PSPACE-complete in $|\phi|$. If the outcome of this check is positive (i.e., the product is empty), then all completions of (M, s) satisfy ϕ, and $[(M, s) \models \phi]_t = \mathit{true}$; otherwise, continue.
2. Check whether $\exists(M', s') : s \preceq s' \wedge (M', s') \models \phi$ (generalized model checking). As recalled in the previous section, this can be done in time quadratic in $|M|$ and is EXPTIME-complete in $|\phi|$. (Intuitively, this check is more expensive since it requires checking that $\forall w \in \mathcal{L}(M, s) : \exists w' : w \leq w' \wedge w' \models \phi$, which includes an alternation of $\forall$ and $\exists$.) If the outcome of this check is positive, we have $[(M, s) \models \phi]_t = \perp$; otherwise, all completions of (M, s) violate ϕ, and $[(M, s) \models \phi]_t = \mathit{false}$.

Because Step 2 of the above procedure requires solving an instance of the generalized model-checking problem, the time needed to compute $[(M, s) \models \phi]_t$ using the generalized model-checking algorithm of [4] can be quadratic in $|M|$. Unfortunately, the lower bound provided by the following theorem shows that it is unlikely this quadratic complexity can be reduced in the general case.

Theorem 4. *The problem of checking emptiness of nondeterministic Büchi tree automata is reducible in linear time and logarithmic space to the generalized model-checking problem for LTL properties represented by nondeterministic Büchi word automata.*

Since the worst-case run-time complexity of the best known algorithm for checking emptiness of nondeterministic Büchi tree automata is quadratic in the size of the automaton [24], it is therefore unlikely that the generalized model-checking problem for LTL can be solved using better than quadratic time in $|M|$ in the worst case.

However, we now identify an important class of LTL formulas for which the generalized model-checking problem can be solved in time linear in $|M|$. This class is the class of *persistence properties* [20]. Persistence properties can be represented by LTL formulas of the form $\Diamond \Box p$ (where $\Diamond$ is read "eventually" and $\Box$ is read "always" [20]). Persistence properties also correspond to languages of ω-words recognizable by *co-Büchi automata*.

Theorem 5. *The generalized model-checking problem for LTL persistence properties can be solved in time linear in the size of the model.*

Proof. (Sketch) Every persistence property can be represented by a co-Büchi automaton A_ϕ. A_ϕ can then easily be transformed into a *weak* (Büchi or co-Büchi) nondeterministic automaton A'_ϕ accepting the same language and of size linear in $|A_\phi|$. Generalized model checking can then be reduced to checking emptiness of an alternating Büchi word automaton $A_{(M,s_0),\phi}$ over a 1-letter alphabet defined using a product construction of the KMTS (M, s_0) and the weak Büchi automaton A'_ϕ. If A'_ϕ is weak, $A_{(M,s_0),\phi}$ is also weak. Moreover, the size of $A_{(M,s_0),\phi}$ is linear in $|M|$. Since checking emptiness of a weak alternating Büchi word automaton over a 1-letter alphabet can be done in linear time [18], we obtain a decision procedure for the generalized model-checking problem of LTL persistence properties that is linear in $|M|$.

Of practical interest, persistence properties include all safety properties (LTL formulas of the form $\Box\, p$), as well as guarantee properties (LTL formulas of the form $\Diamond\, p$) and obligation properties (boolean combinations of safety and guarantee properties) [20]. Examples of LTL properties that are not persistence properties are response properties (LTL formulas of the form $\Box \Diamond\, p$).

4 Generalized Model Checking for BTL

We now consider the case of branching-time temporal logics (BTL) such as propositional modal logic (e.g., see [23]), CTL [5] or the propositional μ-calculus [17]. In the case of a BTL formula ϕ, computing $[(M, s) \models \phi]_t$ reduces to two generalized model-checking problems, namely "does there exist two completions (M', s') and (M'', s'') of (M, s) such that $(M', s') \models \phi$ and $(M'', s'') \not\models \phi$?", the latter statement being equivalent to $(M'', s'') \models \neg\phi$.

Given a CTL formula ϕ, the worst-case run-time complexity of computing $[(M, s) \models \phi]_t$ for a CTL formula ϕ using the generalized model-checking algorithm of [4] is quadratic in $|M|$. The next theorem provides a lower bound similar to the one given in Theorem 4 in the LTL case.

Theorem 6. *The problem of checking emptiness of nondeterministic Büchi tree automata is reducible in linear time and logarithmic space to the generalized model-checking problem for CTL properties represented by nondeterministic Büchi tree automata.*

As in the LTL case, we can identify classes of properties for which generalized model checking can be done in time linear in $|M|$.

Theorem 7. *The generalized model-checking problem for BTL properties recognizable by nondeterministic co-Büchi tree automata can be solved in time linear in the size of the model.*

Proof. (Sketch) BTL properties recognizable by nondeterministic co-Büchi tree automata are also recognizable by weak tree automata. One can then use a product construction of such a weak tree automaton with a KMTS M to define a weak alternating Büchi word automaton A over a 1-letter alphabet and of size linear in $|M|$ such that generalized model checking can be reduced to checking emptiness of this alternating automaton. A key observation to prove the linear bound on $|M|$ is that generalized model checking on a KMTS M can be reduced to generalized model checking on a PKS M' of size linear in M, hence showing that not all (exponentially many) subsets of *may*-transitions of M need be considered when checking branching properties of the set of all possible completions of M. Since checking emptiness of a weak alternating Büchi word automaton over a 1-letter alphabet can be done in linear time [18], we obtain a decision procedure for generalized model-checking that is linear in $|M|$.

The previous theorem generalizes Theorem 5 since tree automata are generalizations of word automata. Examples of CTL properties that are recognizable by nondeterministic co-Büchi tree automata are AGp and EFp. In contrast, CTL formulas such as $AGAFp$ and $AGEFp$ are not recognizable by co-Büchi tree automata. As a corollary to the previous theorem, it is easy to show that the generalized model checking for any PML formula can be solved in linear time in $|M|$. Finally note that in order to compute $[(M, s) \models \phi]_t$ in time linear in $|M|$, both ϕ and $\neg\phi$ need be recognizable by co-Büchi tree automata.

5 Application to Automatic Abstraction

The usual procedure for performing verification via predicate abstraction and iterative abstraction refinement is the following (e.g., see [1, 9]).

1. Abstract: compute an abstraction M_A that simulates the concrete prgm 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.

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. Note that the three steps above can also be interleaved and performed in a strict demand-driven fashion as described in [13].

The purpose of this paper is thus to advocate a new procedure for automatic abstraction.

1. Abstract: compute an abstraction M_A using Def. 4 such that $M_A \preceq M_C$.
2. Check: given *any* property ϕ,
   (a) (3-valued model checking) compute $[M_A \models \phi]$.
      - if $[M_A \models \phi]$ = *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 $[M_A \models \phi]_t$ = *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 1 of Section 2). 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 [11], building a 3-valued abstraction using Definition 4 (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 [4]; following the results of Sections 3 and 4, computing $[M_A \models \phi]_t$ in Step 2.b can be more expensive than Step 2.a, but is still polynomial (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 [13].

6 Examples

We now give examples of programs, models and properties 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 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 4, predicate abstraction techniques and predicates p and q, so that by construction they satisfy Theorem 2. 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.

```
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);
}
```

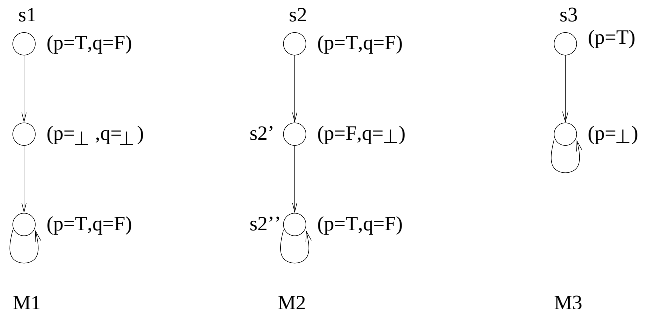

Fig. 2. Examples of programs and models

Consider the LTL formula $\phi_1 = \Diamond q \Rightarrow \Box(p \vee q)$. While $[(M_1, s_1) \models \phi_1] = \perp$, $[(M_1, s_1) \models \phi_1]_t = true$. 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* [1] aimed at recovering some of the imprecision introduced when using *cartesian abstraction* (see [1, 11]).

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 $[(M_2, s_2) \models \phi_2]_t = false$. 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 3. Another example of formula is $\phi_2' = \bigcirc q \wedge \Box(p \vee \neg q)$ (where $\bigcirc$ is read "next" [20]). Again we have that $[(M_2, s_2) \models \phi_2'] = \perp$, while $[(M_2, s_2) \models \phi_2']_t = false$. Note that, although ϕ_2' is an LTL safety formula and hence is within the scope of analysis of existing tools ([2], [6], 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.

7 Conclusions

We have introduced generalized model checking as a way to improve precision of automatic abstraction for the verification of temporal properties of programs.

In this context, generalized model checking means checking whether there exists a concretization of an abstraction M that satisfies a temporal logic formula ϕ.

We believe generalized model checking is quite practical despite its seemingly higher complexity than that of model checking. Indeed, a higher worst-case complexity in the size of the formula (for instance, EXPTIME-complete instead of PSPACE-complete for LTL formulas) may not be too troublesome since formulas are usually quite short and checking algorithms typically behave better than the worst case in practice. Perhaps more importantly, we showed in this paper that generalized model checking may require in general quadratic time in the size of the abstraction, which may be a more severe limitation, but that it can be solved in linear time for important classes of properties including safety properties. In any case, generalized model checking can help an iterative "abstract-check-refine" verification process terminate sooner and more often by providing better analysis precision for a cost only polynomial in the size of the abstraction, which in turn may prevent the unnecessary generation and analysis of possibly exponentially larger refinements of that abstraction.

References

1. 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.
2. T. Ball and S. Rajamani. The SLAM Toolkit. In *Proceedings of CAV'2001 (13th Conference on Computer Aided Verification)*, Paris, July 2001.
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, Trento, July 1999. Springer-Verlag.
4. 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.
5. E. M. Clarke and E. A. Emerson. Design and Synthesis of Synchronization Skeletons using Branching-Time Temporal Logic. In D. Kozen, editor, *Proceedings of the Workshop on Logic of Programs,* Yorktown Heights, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer-Verlag, 1981.
6. 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.
7. P. Cousot and R. Cousot. Temporal Abstract Interpretation. In *Proceedings of the 27th ACM Symposium on Principles of Programming Languages*, pages 12–25, Boston, January 2000.
8. D. Dams. *Abstract interpretation and partition refinement for model checking*. PhD thesis, Technische Universiteit Eindhoven, The Netherlands, 1996.
9. 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.

10. P. Godefroid. Model Checking for Programming Languages using VeriSoft. In *Proceedings of the 24th ACM Symposium on Principles of Programming Languages*, pages 174–186, Paris, January 1997.
11. 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.
12. 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.
13. T. Henzinger, R. Jhala, R. Majumdar, and G. Sutre. Lazy Abstraction. In *Proceedings of the 29th ACM Symposium on Principles of Programming Languages*, Portland, January 2002.
14. 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.
15. 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.
16. S. C. Kleene. *Introduction to Metamathematics*. North Holland, 1987.
17. D. Kozen. Results on the Propositional Mu-Calculus. *Theoretical Computer Science*, 27:333–354, 1983.
18. O. Kupferman, M. Y. Vardi, and P. Wolper. An Automata-Theoretic Approach to Branching-Time Model Checking. *Journal of the ACM*, 47(2):312–360, March 2000.
19. 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.
20. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, 1992.
21. 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.
22. F. Nielson, H. R. Nielson, and C. Hankin. *Principles of Program Analysis*. Springer-Verlag, 1999.
23. M.Y. Vardi. Why is Modal Logic So Robustly Decidable? In *Proceedings of DIMACS Workshop on Descriptive Complexity and Finite Models*. AMS, 1997.
24. M.Y. Vardi and P. Wolper. Automata-Theoretic Techniques for Modal Logics of Programs. *Journal of Computer and System Science*, 32(2):183–221, April 1986.
25. 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.