Model-checking by abstraction: Modal Transition Systems

Ernesto Posse

April 24, 2002

Abstract

This paper reviews a framework given in [3] for verifying complex systems by abstracting them as *modal transition systems* ([8]), or MTS for short, in order to reduce the cost of performing model-checking. MTS's are interesting in their own right, as a means to provide partial specifications of dynamic systems.

1 Introduction

Model-checking has as objective the verification of properties of dynamic systems. Many dynamic systems can be modelled by *transition systems*, which describe, at least, the possible states of the system, and the possible transitions between states. The complexity of model-checking algorithms depends on the size of such transition systems. Therefore, applying these algorithms might prove forbidding for systems with a large state-space, or even impossible, in the case of systems with an infinite number of states.

A closer examination of a given transition system might reveal certain characteristics as irrelevant from the point of view of the analysis of interest. For instance, consider a transition system generated from a large distributed program. We might be interested in verifying safety or liveness properties of the communications protocols, or security properties of some cryptographic protocols. For such analysis, the behaviour of parts of the system, such as the graphics library, is likely to be irrelevant. Nonetheless, a naive software-to-transition-system translator would generate states and transitions for all parts of the system. It would be in our best interest to ignore the parts of the generated transition system which have nothing to do with the properties of interest.

This paper presents a framework for dealing with this problem, as introduced in [3] and [8]. It can be applied in particular to infinite-state systems. The approach can be summarized as follows: Assume that we have a "concrete" *labelled-transition system*. Build an "abstract" transition system from the concrete system, and then apply some model-checking algorithm on this abstract system.

There are two basic underlying assumptions for this approach to make sense: 1) we expect an abstraction to have the same behaviour of the concrete system, with respect to the properties of interest (e.g. safety or liveness), and 2) the abstraction and the concrete system must satisfy the same properties of interest.

To guarantee that these assumptions hold, we must address the following questions. What do we mean by an "abstract" system? What relation must hold between the abstract and the concrete systems? What is an appropriate representation for the abstract system? How to construct the abstraction? What logic can be used to describe the properties of such abstract systems? Can we reuse existing model-checking algorithms for the abstract system, or if not, what must be done to perform model-checking?

Before discussing what do we mean by abstraction in this context, we will address the issues of the logic used to describe properties of a system, and the relation between a concrete system and an abstraction.

In this framework, the logic used to describe the properties of interest is the μ-calculus ([7], [1], [13], [6]), which guarantees a great degree of generality by subsuming several temporal logics such as LTL, CTL and CTL*.

The two underlying assumptions immediately suggest a candidate for the relation that must hold between the concrete system and the abstract system: bisimilarity ([11], [9]). Bisimilarity satisfies the two criteria: if the abstract system and the concrete system are bisimilar, then 1) the abstract system immitates the behaviour of the concrete system, and 2) they satisfy exactly the same formulas of the μ-calculus.

There is a problem with bisimilarity as the main relation between the concrete and abstract systems. It is too strong. On one hand, it requires the concrete system to immitate the behaviour of the abstraction, but this is too restrictive, since it does not let us consider weaker abstractions (i.e. the set of possible abstractions is restricted to those which can be simulated by the concrete system). On the other hand, the abstraction itself must match *all* behaviours of the concrete system. This could hardly be called an abstraction in the first place.

A second candidate is suggested: similarity ([9], [10]). In this case we only require the abstract system to simulate the concrete system, but not the other way around. Unfortunately this is also inadequate. Even though it is a weaker relation than bisimilarity, it still requires the abstraction to simulate all behaviours of the concrete system. Even worse, we loose the main property that we are after: Satisfability is reflected by the abstraction process, but it is not preserved. This is: if the abstract system simulates the concrete system, then whenever the abstract system satisfies a formula, so will the concrete system, but not the other way around. This implies that we would be able to check only positive properties. If the model-checker tells us that a formula is not satisfied by the abstract system, we cannot conclude anything about the concrete system.

A closer look at the desired property of abstract and concrete systems satisfying the same formulas forces us to reconsider the approach. The problem is that two systems satisfy the same properties *if and only if* they are bisimilar. Therefore the only possible relation is bisimilarity, yet we already argued that it is too strong. It turns out that we can actually relax the constraint on satisfability in the following sense: we will be able to prove that whenever a property is *necessarily* satisfied by the abstract system, it will be necessarily satisfied by the concrete system, and dually, whenever a property is *impossible* in the abstract system, it will be impossible in the concrete system.

This is a weaker constraint in the following sense. The original constraint of having the abstract and concrete systems satisfying exactly the same formulas implies that we could always decide satisfability for the concrete system by deciding satisfability of the abstract system. In the relaxed constraint, we have three cases: a property may be either necessary, possible, or impossible. The new constraint allows us to decide satisfability of the concrete system for the cases where the property is either necessary or impossible, but will be left undecided if the property is only possible, but not necessary.

Two questions arise: 1) what is meant by "necessary" satisfability and "possible" satisfability? and 2) which relation has this "relaxed" satisfability property? In order to answer them, we return to the original question: what is an "abstract" system? In this context, we expect an abstraction to have the same behaviour of the concrete system, but only with respect to the properties of interest, while hiding other behaviours. We also expect that behaviours specified in an abstraction as necessary, must be present in the corresponding concrete system, while behaviours specified in an abstraction as possible, may or may not be present in the corresponding concrete system. This motivates the introduction of *modal transition systems* and the corresponding *refinement* relation.

2 Modal Transition Systems

Modal Transition Systems, or MTS for short, were introduced in [8] as a generalization of Labelled-Transition Systems (LTS) that allows the definition of partially specified systems. The intuition behind this concept is that the designer of a system should not have to commit to specific behaviours in the early stages of the specification and design process[1]. The designer should be able to develop the system incrementally in the sense that a partial specification can be refined, possibly several times, yielding more concrete partial implementations, and ultimately a total (completely specified) implementation.

First we recall the definition of LTS's.

Definition 1 (LTS) *A* ***labelled-transition system*** *is a triple* $(S, \mathcal{A}, \rightarrow)$ *where* S *is a set of states,* $\mathcal{A}$ *is a set of labels, and* $\rightarrow \subseteq S \times \mathcal{A} \times S$ *is called the transition relation.*

[1] Note that this intuition represents the opposite direction of the approach presented in this framework, that is, going from abstract to concrete rather than the concrete to abstract. Nonetheless this is not an issue to be worried about. It is only a motivation for the definition of MTS.

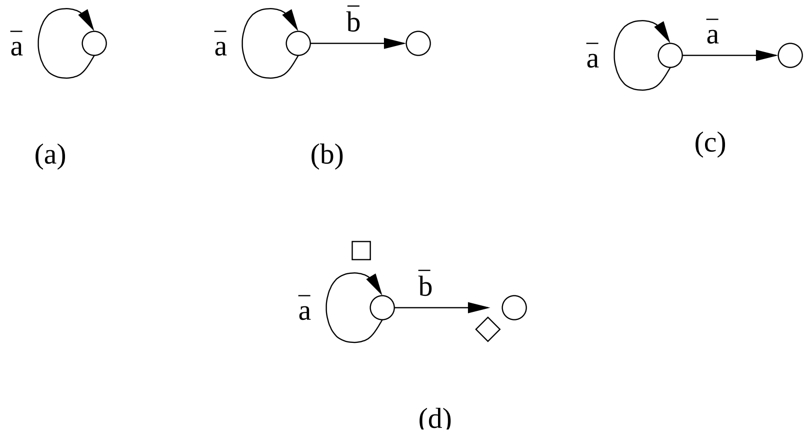

Figure 1: a-senders

MTS's are a generalization of LTS's, where we assign a modality to each transition.

Definition 2 (MTS, [8]) *A* ***modal-transition system*** *is a tuple* $(S, \mathcal{A}, \rightarrow_\Box, \rightarrow_\Diamond)$ *where* S *is a set of states,* $\mathcal{A}$ *is a set of labels,* $\rightarrow_\Box \subseteq S \times \mathcal{A} \times S$ *is called the* ***necessary transition relation****, and* $\rightarrow_\Diamond \subseteq S \times \mathcal{A} \times S$ *is called the* ***possible transition relation****. It must hold that* $\rightarrow_\Box \subseteq \rightarrow_\Diamond$*. Given* $p, q \in S$*, we write* $p \rightarrow q$ *whenever there is an* $\alpha \in \mathcal{A}$ *such that either* $p \xrightarrow{\alpha}_\Box q$ *or* $p \xrightarrow{\alpha}_\Diamond q$*.*

Necessary transitions of an MTS $\mathcal{M}$ are transitions that must be present in any implementation (refinement) of $\mathcal{M}$, while possible transitions are allowed, but not required, in any implementation. Absence of transitions represent the impossibility of having such transitions in any refinement. The condition that $\rightarrow_\Box \subseteq \rightarrow_\Diamond$ simply states that necessary transitions are also possible.

LTS's are a particular case of MTS's. Namely, when $\rightarrow_\Box = \rightarrow_\Diamond$ the MTS is an LTS.

As a simple example consider a process called an *a-sender*. An *a-sender* is a process that sends a message a as long as a receiver is asking for an a. If the receiver is asking for something else, the behaviour of the a-sender is not specified. In figure 1, the LTS's (a) and (b) are a-senders, while (c) is not (because if the receiver asks for an a the system might not follow the loop and end up in a state where it will be unable to provide further a's). The MTS (d) abstracts both (a) and (b), by specifying one transition as necessary to be present in its implementations (the loop), and one transition which is optional in the implementations. Notice that we do not argue that neither (d) is the only abstraction nor (a) and (b) the only implementations.

MTS's represent abstractions in the sense that a system specified as an MTS, while describing what behaviours are essential, does not commit all the implementations to have particular behaviours which are not essential. Thus, an MTS leaves partially specified the precence of non-essential features.

3 Refinement

We have given an intuition of what is the refinement relation between an abstract MTS and a concrete LTS. The formalization of this notion is more general. We define refinement between two MTS.

Definition 3 (Refinement) *Let $\mathcal{M} = (S, \mathcal{A}, \rightarrow_{\Box}, \rightarrow_{\Diamond})$ be an MTS. A binary relation $\rho \subseteq S \times S$ is said to be a* ***refinement*** *iff for any states $p, q \in S$, $(p, q) \in \rho$ implies that:*

(i) Whenever $p \stackrel{\alpha}{\rightarrow}_{\Diamond} p'$ for some $p' \in S$ and $\alpha \in \mathcal{A}$ then there is a $q' \in S$ such that $q \stackrel{\alpha}{\rightarrow}_{\Diamond} q'$ and $(p', q') \in \rho$, and

(ii) Whenever $q \stackrel{\alpha}{\rightarrow}_{\Box} q'$ for some $q' \in S$ and $\alpha \in \mathcal{A}$ then there is a $p' \in S$ such that $p \stackrel{\alpha}{\rightarrow}_{\Box} p'$ and $(p', q') \in \rho$.

We say that p is ***a refinement*** *of q, or equivalently that q is* ***an abstraction*** *of p, written $p \triangleleft q$ iff there is some refinement ρ with $(p, q) \in \rho$.*

The intuition behind this relation is as follows: if p is a refinement of q, and q has a necessary transition α, then p *must* have this transtion (clause *(ii)*), and furthermore any refinement of p as well. On the other hand, if p has a transition marked as possible, it is because it was specified in the abstraction q as possible (clause *(i)*). In any case, p needs not have transitions specified as possible in q.

Refinement does not introduce new transitions, it does the following: 1) forces the preservation of necessary transitions (clause *(i)*, 2) allows the preservation of possible transitions (clause *(ii)*), and 3) allows to change the modality of a transition from possible to neccessary. Evidently, if p is a refinement of q, and p has a necessary transition, by definition it is a possible transition, so by clause *(i)* the abstraction q must have it as a possible transition.

Fact 1 *$\triangleleft$ is the largest refinement.*

Refinement for two separate MTS can be easily defined by constructing the disjoint union of the two MTS.

Both simulation and bisimulation are specific cases of refinement. When the MTS is an LTS ($\rightarrow_{\Box} = \rightarrow_{\Diamond}$) refinement coincides with bisimulation. When $\rightarrow_{\Box} = \emptyset$, refinement coincides with simulation.

4 Semantics of the μ-calculus w.r.t. MTS

As we mentioned in the introduction, the relation between abstract and concrete system could be such that whenever a property is *necessarily* satisfied by the abstract system, it will be necessarily satisfied by the concrete system, and dually, whenever a property is *impossible* in the abstract system, it will be impossible in the concrete system.

The modal transition systems described above allow us to define systems where some behaviours are necessarily present in their refinements, possibly present, or not present at all. This suggests that the satisfability of formulas depends on the necessary or the possible presence of behaviours.

The key new ingredient here is the separation of the notion of satisfability into two related notions: *necessary satisfability* and *possible satisfability*.

Intuitively, a system will *necessarily* satisfy a formula whenever any refinement of the system satisfies the formula, whereas a system will *possibly* satisfy a formula when some abstraction satisfies it, but only some of its refinements satisfy the formula. This intuition is made precise by theorem 1, and it rests on the definition of satisfability of μ-calculus formulas with respect to MTS's.

4.1 The μ-calculus: syntax

The syntax for the μ-calculus is the following ([7], [13])[2]: assume that $\mathcal{A}$ is a set of actions, and Var is a set of variables.

$$\phi \quad ::= \quad \top \mid X \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \exists\alpha.\phi \mid \mu X.\phi$$

where $\alpha \in \mathcal{A}$ and $X \in \mathsf{Var}$. The operator $\mu X.\phi$ is the least fixed-point operator, which binds the variable X in ϕ. Sometimes we write $\phi(X)$ to emphasise that X is a free variable in ϕ.

4.2 The μ-calculus: satisfability

Now we define what it means for a formula ϕ to be necessarily satisfiable in a state s with respect to a variable environment σ, written $s, \sigma \models_\Box \phi$, and dually what it means for ϕ to be possibly satisfiable in s w.r.t. σ, written $s, \sigma \models_\Diamond \phi$. The definition must be simultaneous beacuse of the nature of negation.

Definition 4 (Satisfability, [3]) *Let $\mathcal{M} = (S, \mathcal{A}, \rightarrow_\Box, \rightarrow_\Diamond)$ be an MTS and ϕ a μ-calculus formula. Given an environment $\sigma : \mathsf{Var} \rightarrow 2^S \times 2^S$ we define inductively the semantics $[\![\phi]\!]_\sigma \in 2^S \times 2^S$, with left and right projections $[\![\phi]\!]^\Box_\sigma$ and $[\![\phi]\!]^\Diamond_\sigma$ (i.e. $[\![\phi]\!]_\sigma = ([\![\phi]\!]^\Box_\sigma, [\![\phi]\!]^\Diamond_\sigma)$) as shown in table 1. The satisfability relations $\models_\Box$ and $\models_\Diamond$ are defined for a given state $s \in S$ as:*

$$s, \sigma \models_\Box \phi \qquad \textit{iff } s \in [\![\phi]\!]^\Box_\sigma \qquad \textit{and} \qquad s, \sigma \models_\Diamond \phi \qquad \textit{iff } s \in [\![\phi]\!]^\Diamond_\sigma$$

[2]We do not include propositional atoms since we are not dealing with Kripke structures. However in [5] Kripke MTS's are introduced, and propositional atoms are included in the μ-calculus syntax. Also, the syntax $\exists\alpha.\phi$ is equivalent to $\langle\alpha\rangle\phi$ found in the literature.

$$
\begin{aligned}
[\![\top]\!]_\sigma &\stackrel{def}{=} (S, S) \\
[\![X]\!]_\sigma &\stackrel{def}{=} \sigma(X) \\
[\![\neg\phi]\!]_\sigma &\stackrel{def}{=} (S \setminus [\![\phi]\!]^\Diamond_\sigma, S \setminus [\![\phi]\!]^\Box_\sigma) \\
[\![\phi_1 \wedge \phi_2]\!]_\sigma &\stackrel{def}{=} ([\![\phi_1]\!]^\Box_\sigma \cap [\![\phi_2]\!]^\Box_\sigma, [\![\phi_1]\!]^\Diamond_\sigma \cap [\![\phi_2]\!]^\Diamond_\sigma) \\
[\![\exists\alpha.\phi]\!]_\sigma &\stackrel{def}{=} (\{s \in S \mid \text{for some } s' \in S,\ s \xrightarrow{\alpha}_\Box s',\ \text{and } s' \in [\![\phi]\!]^\Box_\sigma\}, \\
&\qquad \{s \in S \mid \text{for some } s' \in S,\ s \xrightarrow{\alpha}_\Diamond s',\ \text{and } s' \in [\![\phi]\!]^\Diamond_\sigma\}) \\
[\![\mu X.\phi]\!]_\sigma &\stackrel{def}{=} \mathrm{lfp}(d \mapsto [\![\phi]\!]_{\sigma[X \mapsto d]}) \\
&\qquad \text{where } \sigma[X \mapsto d](X) \stackrel{def}{=} d \text{ and } \sigma[X \mapsto d](Y) \stackrel{def}{=} \sigma(Y) \text{ for } Y \neq X
\end{aligned}
$$

Table 1: μ-calculus MTS semantics.

$$
\begin{aligned}
\bot &\stackrel{def}{=} \neg\top \\
\phi_1 \vee \phi_2 &\stackrel{def}{=} \neg(\neg\phi_1 \wedge \neg\phi_2) \\
\phi_1 \Rightarrow \phi_2 &\stackrel{def}{=} \neg\phi_1 \vee \phi_2 \\
\phi_1 \Leftrightarrow \phi_1 &\stackrel{def}{=} (\phi_1 \Rightarrow \phi_2) \wedge (\phi_2 \Rightarrow \phi_1) \\
\forall\alpha.\phi &\stackrel{def}{=} \neg\exists\alpha.\neg\phi \\
\nu X.\phi(X) &\stackrel{def}{=} \neg\mu X.\neg\phi(\neg X)
\end{aligned}
$$

Table 2: Additional operators of the full μ-calculus

Notice the semantics of negation. We have that for a state s and environment σ, $s, \sigma \models_\Box \neg\phi$ iff $s, \sigma \not\models_\Diamond \phi$, this is, $\neg\phi$ is necessary iff ϕ is impossible. Dually, $s, \sigma \models_\Diamond \neg\phi$ iff $s, \sigma \not\models_\Box \phi$, this is, $\neg\phi$ is possible iff ϕ is not necessary.

We can enrich this basic core of the μ-calculus to include other propositional operators ($\vee, \Rightarrow, \Leftrightarrow$), modal universal quantification ($\forall\alpha$) and the greatest fixed-point operator (νX), in the usual way. However we can simply define them in terms of the given core, since it is an adequate set of operators, satisfying the usual equations ([13]), shown in table 2.

4.3 Soundness

We summarize the basic properties of the semantics given in the previous section.

Proposition 1 ([3]) *For any MTS $\mathcal{M} = (S, \mathcal{A}, \rightarrow_\Box, \rightarrow_\Diamond)$, μ-calculus formula ϕ and environment σ, the following holds:*

(i) $[\![\phi]\!]_\sigma^\Box \subseteq [\![\phi]\!]_\sigma^\Diamond$

(ii) $[\![\phi \wedge \neg\phi]\!]_\sigma^\Box = \emptyset$

(iii) $[\![\phi \vee \neg\phi]\!]_\sigma^\Diamond = S$

(iv) *If* $\rightarrow_\Box = \rightarrow_\Diamond$ *then* $[\![\phi]\!]_\sigma^\Box = [\![\phi]\!]_\sigma^\Diamond$ *and it corresponds to the standard semantics for LTS.*

Clause *(i)* of this proposition simply states that if a formula is necessarily satisfied then it is possibly satisfied. Clause *(ii)* says that the semantics are consistent for necessary satisfability, i.e. no state will necessarily satisfy a formula and its negation at the same time. Clause *(iii)* states that the semantics are complete for possible satisfability, i.e. for every state, either a formula or its negation is possibly satisfiable. Finally clause *(iv)* states that for the particular case of LTS, both satisfability relations coincide.

The main result that allows us to use the abstraction approach is stated by the following theorem.

Theorem 1 ([3]) *Let* $\mathcal{M} = (S, \mathcal{A}, \rightarrow_\Box, \rightarrow_\Diamond)$ *be any MTS,* ϕ *any* μ*-calculus formula and* σ *any environment. Then, for any states* $c, a \in S$*, if* $c \triangleleft a$ *then:*

(i) $a, \sigma \models_\Box \phi$ *implies* $c, \sigma \models_\Box \phi$*, and*

(ii) $c, \sigma \models_\Diamond \phi$ *implies* $a, \sigma \models_\Diamond \phi$

The theorem expresses precisely the intended property. If c is a refinement of a, then any property necessarily satisfied by a is also necessarily satisfied by c, and any property impossible in a, is also impossible in c. The theorem leaves open the case when a only possibly satisfies the property. In such case we cannot conclude anything about c.

5 Constructing abstractions

5.1 Relational abstraction

Theorem 1 asserts the property required for verification and refutation of formulas, but it does not describe which systems can be an abstraction of a given system. The following definition specifies an abstract MTS $\mathcal{M}'$ for a concrete MTS $\mathcal{M}$, given a set of abstract states S' and some relation ρ between the concrete and the abstract states.

Definition 5 (Relational abstraction, [3]) *Given an MTS* $\mathcal{M} = (S, \mathcal{A}, \rightarrow_\Box, \rightarrow_\Diamond)$*, a set of states* S'*, and a total relation* $\rho \subseteq S \times S'$ *then the* ***relational abstraction*** *of* $\mathcal{M}$ *with respect to* S' *and* ρ *is the tuple* $\mathcal{M}^\rho_{S'} \stackrel{def}{=} (S', \mathcal{A}, \rightarrow'_\Box, \rightarrow'_\Diamond)$ *where for each* $p', q' \in S'$

(i) $p' \xrightarrow{\alpha}_\Box{}' q' \stackrel{def}{\iff}$ *for all* $p \in S$ *such that* $(p, p') \in \rho$*, there is a* $q \in S$ *such that* $p \xrightarrow{\alpha}_\Box q$ *and* $(q, q') \in \rho$*.*

(ii) $p' \overset{\alpha}{\rightarrow}_{\diamond}{}' q' \overset{def}{\iff}$ *there are* $p, q \in S$ *such that* $p \overset{\alpha}{\rightarrow}_{\diamond} q$*,* $(p, p') \in \rho$ *and* $(q, q') \in \rho$*.*

Lemma 1 ([3]) *Let* $\mathcal{M} = (S, \mathcal{A}, \rightarrow_{\Box}, \rightarrow_{\diamond})$ *be an MTS,* S' *any set, and* $\rho \subseteq S \times S'$ *a total relation, then the relational abstraction* $\mathcal{M}^{\rho}_{S'}$ *is an MTS and* ρ *is a refinement relation for the disjoint union of* $\mathcal{M}$ *and* $\mathcal{M}^{\rho}_{S'}$*.*

Clause *(i)* of the construction creates a necessary transition for an abstract state whenever all the concrete states to which it is related, have a similar necessary transition. Clause *(ii)* creates a possible transition whenever there is at least one possible transition for some corresponding concrete state.

5.2 Computing the abstraction

The relational abstraction described by definition 5 assumes that you are given the abstract set of states and their relation to the concrete states, in order to obtain an abstract MTS. Thus, we need to specify the abstract set of states and the relation. The framework presented by [3] proposes two approaches: 1) *predicate abstraction*, and 2) *cartesian abstraction*.

5.2.1 Predicate abstraction

Predicate abstraction was introduced in [4], in a different context. An informal example of this approach, in the context of program abstraction, is as follows ([12]). Suppose that you have a program whose state is defined by two variables x and y. Furthermore, suppose that you are only interested in observing the equality of these variables, rather than their actual value. Then you can define a boolean variable B to represent the predicate $x = y$. Replace in the program all the checks for this equality with B, and replace any operation modifying x or y with a modification to B accordingly. The resulting program is an abstraction of the original, with respect to this predicate.

In our context predicate abstraction is used to obtain the abstract MTS by defining the abstract set of states and their relation to the concrete states. To obtain the abstract set of states we start with a *finite* set of predicates of interest $\Phi \overset{def}{=} \{\phi_1, \phi_2, \ldots, \phi_n\}$, which are assumed to be quantifier-free first-order formulas. The concrete state-space is partitioned with respect to this set Φ, according to how the concrete states agree on the given predicates. We consider two states to be equivalent if they agree on every formula in this set. Each equivalence class forms an abstract state, and it is represented by a boolean combination of the predicates, called a *monomial*. Formally:

Definition 6 (Abstract states) *Given an LTS* $(S, \mathcal{A}, \rightarrow)$ *and a set of predicates* $\Phi \overset{def}{=} \{\phi_1, \phi_2, \ldots, \phi_n\}$*, the* ***abstract set*** *of states* S'*, is the set of equivalence classes of* S *with respect to satisfability of* Φ*, i.e.* $S' \overset{def}{=} S/\cong_{\Phi}$ *where*

$\cong_\Phi \stackrel{def}{=} \{(p,q) \in S \times S \mid \textit{for all } \phi \in \Phi,\ p \models \phi \textit{ iff } q \models \phi\}$.[3]

Definition 7 (Monomials) *Given a set of predicates $\Phi \stackrel{def}{=} \{\phi_1, \phi_2, \ldots, \phi_n\}$, a **monomial** is a predicate $\eta = \eta_1 \wedge \eta_2 \wedge \cdots \wedge \eta_n$ where $\eta_i = \phi_i$ or $\eta_i = \neg\phi_i$ for each $i \in \{1, 2, \ldots, n\}$. Given a set of abstract states S and an abstract state $a \in S$, η^a denotes the formula representing a, i.e. $[\![\eta^a]\!] = a$. We can represent a monomial $\eta = \eta_1 \wedge \eta_2 \wedge \cdots \wedge \eta_n$ as a "bit-vector" $b \in \{0,1\}^n$, where $b_i = 1$ if $\eta_i = \phi_i$, and $b_i = 0$ if $\eta_i = \neg\phi_i$. For a bit vector b, η^b is its corresponding monomial.*

Since each monomial represents an equivalence class, for n predicates, there will be 2^n abstract states. Also, every concrete state c in the abstract state a satisfies η^a.

With the monomials as abstract states we only need a total relation to be able to use relational abstraction and obtain our abstract MTS.

Definition 8 (Predicate abstraction, [3]) *Let $\mathcal{L} = (S, \mathcal{A}, \rightarrow)$ be an LTS (the concrete system), and $\Phi = \{\phi_1, \phi_2, \ldots, \phi_n\}$ a finite set of first-order logic quantifier-free formulas. The **predicate abstraction** of $\mathcal{L}$ with respect to Φ is the relational abstraction $\mathcal{L}^p_\Phi = (\Theta, \mathcal{A}, \rightarrow_\Box, \rightarrow_\Diamond)$ of $\mathcal{L}$ w.r.t. Θ and ρ where:*

- $\Theta \stackrel{def}{=} \{b \in \{0,1\}^n \mid s \models \eta^b \textit{ for some } s \in S\}$
- $\rho \stackrel{def}{=} \{(s,b) \in S \times \Theta \mid s \models \eta^b\}$

The set Θ contains all the bit-vector representations of the monomials that are satisfied by some concrete state, thus it is essentially the same as the set of abstract states as described in definition 6.[4]

Since the relation ρ is total, by lemma 1, $\mathcal{L}^p_\Phi$ an MTS and ρ a refinement relation.

The relational abstraction of definition 5 tells us *what* are the successors of a state, but it does not tell us *how* to compute them. In order to do that we need some preliminary definitions. Given a formula η, the set of *concrete* "predecessor" states satisfying η is defined by the pre operator, and the set of *concrete* "successor" states satisfying η is given by the post operator, defined as follows.

Definition 9 ([3]) *Let $(S, \mathcal{A}, \rightarrow)$ be an LTS, and $\alpha \in \mathcal{A}$.*

$$\mathsf{pre}_\alpha(\eta) \stackrel{def}{=} \{p \in S \mid \textit{for all } p' \in S \textit{ if } p \stackrel{\alpha}{\rightarrow} p',\ \textit{then } p' \models \eta\}$$
$$\mathsf{post}_\alpha(\eta) \stackrel{def}{=} \{p' \in S \mid \textit{there is a } p \in S \textit{ such that } p \stackrel{\alpha}{\rightarrow} p',\ \textit{and } p \models \eta\}$$

[3] Since it is an LTS, by proposition 1, the satisfability relation $\models$ does not need to have a modality $\Box$ or $\Diamond$.

[4] By "essentially the same" we mean that it is easy to establish a bijection between them, namely the function $f : S/\cong_\Phi \rightarrow \Theta$ defined as $f(a) \stackrel{def}{=} b$ iff $a = [x]_{\cong_\Phi}$, and $\forall c \in a.\ c \models \eta^b$. An alternative would have been to define Θ to be $S/\cong_\Phi$ and ρ to be the inclusion of concrete states in their equivalence classes.

Now we can define the necessary and possible successors. Given an abstract state a, an abstract state b (satisfying a monomial η^b) is a necessary successor of a if there is no transition from any concrete state c in a to some state incompatible with b (i.e. satisfying $\neg\eta^b$). In other words, b is a neccessary successor of a if the concrete states of a are not compatible with the concrete states of the predecessors of those states not satisfying η^b, this is, if $\eta^a \wedge \mathsf{pre}(\neg\eta^b)$ is unsatisfiable. Similarly, for possible successors, we say that b is a possible successor of a if there is some transition from a concrete state c in a to a state compatible with b, i.e. $\mathsf{post}(\eta^a) \wedge \eta^b$ is satisfiable.

Definition 10 *Let S be a set of concrete states and ψ be a quantifier-free first-order formula representing a subset of S. The set of* ***necessary successors*** *of ψ for α is defined as:*

$$\mathsf{next}^{\Box}_{\alpha}(\psi) \stackrel{def}{=} \{s \in S \mid [\![\psi]\!]^{\Box} \cap \mathsf{pre}_{\alpha}(\neg\eta^s) = \emptyset\}$$

and the set of ***possible successors*** *of ψ for α is defined as:*

$$\mathsf{next}^{\Diamond}_{\alpha}(\psi) \stackrel{def}{=} \{s \in S \mid \mathsf{post}_{\alpha}(\psi) \cap [\![\eta^s]\!]^{\Diamond} \neq \emptyset\}$$

These sets can be computed using standard BDD operations and a model-checker for quantifier-free first-order formulas. The BDD for $\mathsf{next}^{\Box}_{\alpha}$ is obtained by computing $H^{\Box}_{\alpha}(\psi, \top, 1)$ where the function $H^{\Box}_{\alpha}$ is defined as follows using $x_1, x_2, \ldots, x_n$ as the BDD's variables:

$$H^{\Box}_{\alpha}(\psi,\eta,i) \stackrel{def}{=} \begin{cases} (x_i \wedge H^{\Box}_{\alpha}(\psi, \eta \wedge \phi_i, i+1)) & \\ \vee(\neg x_i \wedge H^{\Box}_{\alpha}(\psi, \eta \wedge \neg\phi_i, i+1)) & \text{if } 0 < i \leq n \\ 1 & \text{if } i = n+1 \text{ and} \\ & \psi \wedge \mathsf{pre}_{\alpha}(\neg\eta) \text{ is satisfiable} \\ 0 & \text{if } i = n+1 \text{ and} \\ & \psi \wedge \mathsf{pre}_{\alpha}(\neg\eta) \text{ is unsatisfiable} \end{cases}$$

The BDD for $\mathsf{next}^{\Diamond}_{\alpha}$ is computed similarly by $H^{\Diamond}_{\alpha}(\psi, \top, 1)$ where:

$$H^{\Diamond}_{\alpha}(\psi,\eta,i) \stackrel{def}{=} \begin{cases} (x_i \wedge H^{\Diamond}_{\alpha}(\psi, \eta \wedge \phi_i, i+1)) & \\ \vee(\neg x_i \wedge H^{\Diamond}_{\alpha}(\psi, \eta \wedge \neg\phi_i, i+1)) & \text{if } 0 < i \leq n \\ 1 & \text{if } i = n+1 \text{ and} \\ & \mathsf{post}_{\alpha}(\psi) \wedge \eta \text{ is satisfiable} \\ 0 & \text{if } i = n+1 \text{ and} \\ & \mathsf{post}_{\alpha}(\psi) \wedge \eta \text{ is unsatisfiable} \end{cases}$$

In both constructions the logical operators are computed by the standard BDD apply operation, and the base case of the recursion (the leaves of the BDD) call an external model-checking algorithm for the concrete LTS. By unwinding the recursion it is possible to see how all the possible monomials are considered, and thus for n predicates there will be 2^n calls to the model-checker.

The following theorem gives us the soundness and completeness of the construction of the predicate abstraction with respect to the sets of necessary and possible successors. This basically establishes that the sets $\mathsf{next}^{\Box}_{\alpha}$ and $\mathsf{next}^{\Diamond}_{\alpha}$, computed by $H^{\Box}$ and $H^{\Diamond}$ respectively, define indeed the transitions for the relational abstraction.

Theorem 2 ([3]) *Let $\mathcal{L}$ be an LTS, and $\mathcal{L}^p_{\Phi} = (\Theta, \mathcal{A}, \rightarrow_{\Box}, \rightarrow_{\Diamond})$ be its predicate abstraction w.r.t. Φ, then for every $a, a' \in \Theta$, $\alpha \in \mathcal{A}$:*

(i) $a \xrightarrow{\alpha}_{\Box} a'$ in $\mathcal{L}^p_{\Phi}$ if and only if $a' \in \mathsf{next}^{\Box}_{\alpha}(\eta^a)$

(ii) $a \xrightarrow{\alpha}_{\Diamond} a'$ in $\mathcal{L}^p_{\Phi}$ if and only if $a' \in \mathsf{next}^{\Diamond}_{\alpha}(\eta^a)$

5.2.2 Cartesian abstraction

A desired property for an abstraction method is *incrementality*, i.e. if you add more predicates to define the abstract states, the resulting relational abstraction should be a refinement of the original. Predicate abstraction unfortunately is not incremental. This means that whenever you add predicates you must recompute the entire abstraction.

Cartesian abstraction is an approach that solves this problem. According to [3] the idea is "[...]to approximate sets of tuples with tuples of sets". We used a bit-vector representation for monomials as abstract states in predicate abstraction. Now we extend this notion by using *tri-vectors*, i.e. bit-vectors with "wildcards". For instance, the set of states $\{1001, 1011\}$ can be represented by 10*1.

Definition 11 *Given a set of predicates $\Phi = \{\phi_1, \phi_2, \ldots, \phi_n\}$ and $t \in \{0, 1, *\}^n$, called a **tri-vector**, its corresponding monomial η^t is $\eta^t = \eta_1 \wedge \eta_2 \wedge \cdots \wedge \eta_n$, where $\eta_i = \phi_i$ if $t_i = 1$, $\eta_i = \neg\phi_i$ if $t_i = 0$, or $\eta_i = \top$ if $t_i = *$. We say that a bit vector $b \in \{0, 1\}^n$ **is represented by** a tri-vector t, written $b \prec t$ iff for all $i \in \{1, 2, \ldots, n\}$, whenever $t_i \neq *$, we have that $b_i = t_i$.*

Definition 12 (Cartesian abstraction, [3]) *Let $\mathcal{L} = (S, \mathcal{A}, \rightarrow)$ be an LTS (the concrete system), and $\Phi = \{\phi_1, \phi_2, \ldots, \phi_n\}$ a finite set of first-order logic quantifier-free formulas. The **cartesian abstraction** of $\mathcal{L}$ with respect to Φ is the relational abstraction $\mathcal{L}^c_{\Phi} = (\Theta, \mathcal{A}, \rightarrow_{\Box}, \rightarrow_{\Diamond})$ of $\mathcal{L}$ w.r.t. Θ and ρ where:*

$$\Theta \stackrel{def}{=} \{t \in \{0,1,*\}^n \mid \text{there is a } b \in \{0,1\}^n \text{ such that } s \vDash \eta^b \text{ for some } s \in S, \text{ and } b \prec t\}$$

$$\rho \stackrel{def}{=} \{(s,t) \in S \times \Theta \mid \text{there is a } b \in \{0,1\}^n \text{ such that } s \vDash \eta^b \text{ and } b \prec t\}$$

The following fact shows that the abstract states of cartesian abstraction coincide with the monomials of the set of predicates Φ.

Proposition 2 *Let $\mathcal{L}$ be an LTS with state-space S and $\mathcal{L}^c_\Phi$ be its cartesian abstraction w.r.t. Φ. Then for all $s \in S$ and $t \in \Theta$, $s \vDash \eta^t$ if and only if $(s,t) \in \rho$.*

The new property of cartesian abstraction is that it is incremental.

Theorem 3 *Let Φ and Φ' be two finite set of predicates. Let $\mathcal{L}^c_\Phi$ be the cartesian abstraction of an LTS $\mathcal{L}$. Then the cartesian abstraction $\mathcal{L}^c_{\Phi\cup\Phi'}$ is a refinement of $\mathcal{L}^c_\Phi$.*

Cartesian abstraction is built in a similar manner to predicate abstraction, but instead of using BDD's to represent $\mathsf{next}^\Box$ and $\mathsf{next}^\Diamond$, TDD's (ternary decision diagrams) are used. Their computation is very similar to the one shown for BDD's. Each node x_i in the TDD has three branches, one for $x_i = 1$, one for $x_i = 0$ and one for $x_i = *$. The complexity of constructing the TDD's is $O(3^n)$ calls to the model-checker. The computed sets are also correct with respect to the definition of relational abstraction, with a theorem similar to 2. This is stated in the following theorem.

Theorem 4 ([3]) *Let $\mathcal{L}$ be an LTS, and $\mathcal{L}^c_\Phi = (\Theta, \mathcal{A}, \rightarrow_\Box, \rightarrow_\Diamond)$ be its cartesian abstraction w.r.t. Φ, then for every $a, a' \in \Theta$, $\alpha \in \mathcal{A}$:*

(i) $a \xrightarrow{\alpha}_\Box a'$ in $\mathcal{L}^c_\Phi$ if and only if $a' \in \mathsf{next}^\Box_\alpha(\eta^a)$

(ii) $a \xrightarrow{\alpha}_\Diamond a'$ in $\mathcal{L}^c_\Phi$ if and only if $a' \in \mathsf{next}^\Diamond_\alpha(\eta^a)$

6 Model checking MTS's

The previous sections defined how to construct abstractions that have the same behaviour of the concrete system with respect to the properties of interest, and that satisfy the same formulas in the logic. The last part of the framework is model-checking itself.

Theorem 1 is what allows us to perform model-checking on the abstraction and have sound results. However model-checking the abstraction has three possible results for a formula ϕ, either: 1) ϕ is necessarily satisfiable in the abstraction, and this in the concrete system, 2) ϕ is only possibly satisfiable in the abstraction, in which case we cannot conclude anything about the concrete system, or 3) ϕ is not possibly satisfiable in the abstraction, and thus it is impossible in the concrete system. This is refered in [3] as a *three-value model-checking problem.*

The approach to solve this problem given in [3], barrowed from [2], transforms this problem into two classical model-checking problems for LTS. This approach is now described.

First, rewrite the formula in positive normal form by "pushing in" all the negation symbols, using classical De Morgan laws. Then replace all occurrences of $\forall\alpha.\phi$ by $\forall\alpha_\forall$, and all occurrences of $\exists\alpha$ by $\exists\alpha_\exists$. Call this new formula $\bar{\phi}$.

Second, construct two LTS's $\mathcal{L}^{pess}$ and $\mathcal{L}^{opt}$ as follows.

Definition 13 *Let $\mathcal{M} = (S, \mathcal{A}, \rightarrow_\Box, \rightarrow_\Diamond)$ be an abstract MTS, and $\mathcal{A}' \stackrel{def}{=} \{\alpha_\exists \mid \alpha \in \mathcal{A}\} \cup \{\alpha_\forall \mid \alpha \in \mathcal{A}\}$. Define $\mathcal{L}^{pess} \stackrel{def}{=} (S, \mathcal{A}', \rightarrow_{pess})$ to be the* ***pessimistic*** *LTS where*

(i) $s \stackrel{\alpha_\exists}{\rightarrow}_{pess} s'$ iff $s \stackrel{\alpha}{\rightarrow}_\Box s'$ in $\mathcal{M}$

(ii) $s \stackrel{\alpha_\forall}{\rightarrow}_{pess} s'$ iff $s \stackrel{\alpha}{\rightarrow}_\Diamond s'$ in $\mathcal{M}$

and define $\mathcal{L}^{opt} \stackrel{def}{=} (S, \mathcal{A}', \rightarrow_{opt})$ to be the ***optimistic*** *LTS where*

(i) $s \stackrel{\alpha_\exists}{\rightarrow}_{pess} s'$ iff $s \stackrel{\alpha}{\rightarrow}_\Diamond s'$ in $\mathcal{M}$

(ii) $s \stackrel{\alpha_\forall}{\rightarrow}_{pess} s'$ iff $s \stackrel{\alpha}{\rightarrow}_\Box s'$ in $\mathcal{M}$

With this construction we might now perform LTS model-checking. This is guaranteed by the following theorem.

Theorem 5 ([3]) *Let ϕ be a μ-calculus formula, $\mathcal{M} = (S, \mathcal{A}, \rightarrow_\Box, \rightarrow_\Diamond)$ be an abstract MTS, and $\mathcal{M}^{pess}$ and $\mathcal{M}^{opt}$ be the pessimistic and the optimistic LTS's of $\mathcal{M}$ respectively. Then for every state $s \in S$,*

(i) $s \models_\Box \phi$ if and only if $\mathcal{M}^{pess}, s \models \bar{\phi}$

(ii) $s \models_\Diamond \phi$ if and only if $\mathcal{M}^{opt}, s \models \bar{\phi}$

Hence, checking for necessary satisfability is done by doing classical model-checking on the pessimistic LTS, and checking for possible satisfability is done by doing classical model-checking on the optimistic LTS.

7 Conclusions

The framework presented is promising, and might allow a great reduction in the size of the transition system to be verified. However this is at the expense of a possibly expensive abstraction algorithm, which furthermore, does not seem to guarantee a reduction in neither the size of the state-space nor the number of transitions. Consider for instance an LTS with n states. Then, with only $\log_2 n$ predicates, n abstract states will be introduced by predicate abstraction.

This is not as bad as it appears at first sight. Firstly, the framework allows the reduction of an *infinite* state LTS into a *finite* state LTS. Secondly, there is a trade-off between abstraction and precision. If few predicates are used, there will be few abstract states, and therefore both abstraction construction and model-checking will be fast, but the behaviours will be coarse-grained and thus the properties might not be too interesting. If more predicates are used, abstraction construction and model-checking will be more expensive (exponentially), but fine-grained behaviours will be obtained and there will be more precision on the checking of properties of interest. Also, cartesian abstraction, though more expensive than predicate abstraction, produces more abstract MTS's.

References

[1] Julian Charles Bradfield. *Verifying Temporal Properties of Systems.* Brikhäuser, Boston, 1991.

[2] G. Burns and P. Godefroid. Generalized model-checking: Reasoning about partial state spaces. In *Proceedings of the 12th International Conference on Concurrency Theory, CONCUR 2000*, volume 1877 of *Lecture Notes in Computer Science*, pages 168–182. Springer-Verlag, 2000.

[3] Patrice Godefroid, Michael Huth, and Radha Jagadeesan. Abstraction-based model checking using modal transition systems. In *Proceedings of the 12th International Conference on Concurrency Theory, CONCUR 2001*, volume 2154 of *Lecture Notes in Computer Science*, pages 426–?? Springer-Verlag, 2001.

[4] S. Graf and H. Saidi. Construction of abstract state graphs with PVS. In *Proceedings of the 9th International Computer Aided Verification Conference*, pages 72–83, 1997.

[5] Michael Huth, Radha Jagadeesan, and David Schmidt. Modal transition systems: A foundation for three-valued program analysis. In *Proceedings of the European Symposium on Programming Languages ESOP-2001*, volume 2028 of *Lecture Notes in Computer Science*, pages 155–169. Springer, Verlag, 2001.

[6] Michael Huth and Mark Ryan. *Logic in Computer Science: Modelling and reasoning about systems.* Cambridge University Press, 2000.

[7] Dexter Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[8] Kim G. Larsen and Bent Thomsen. A modal process logic. In *Proceedings of the 3th Annual Symposium on Logic in Computer Science*, pages 203–210, Edinburgh, Scotland, 1988. IEEE Computer Society.

[9] Robin Milner. An algebraic definition of simulation between programs. In D. C. Cooper, editor, *Proceedings of the 2nd International Joint Conference on Artificial Intelligence*, pages 481–489, London, UK, 1971. William Kaufmann.

[10] Robin Milner. *A Calculus for Communicating Systems*, volume 92 of *Lecture Notes in Computer Science.* Springer-Verlag, 1980.

[11] David Park. Concurrency and automata on infinite sequences. In *Theoretical Computer Science, 5th GI-Conf.*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer-Verlag, 1981.

[12] Willern Visser, Seung Joon Park, and John Penix. Using predicate abstraction to reduce object-oriented programs for model checking. In Mats P. E. Heimdahl, editor, *Proceedings of the 3rd Workshop on Formal Methods in Software Practice FMSP-2000*, pages 3–12, New York, 2000. ACM Press.

[13] Igor Walukiewicz. Notes on the propositional μ-calculus: Completeness and related results. Notes Series NS-95-1, BRICS, 1995.