Comparing Abstract Semantics for Model Checking

María del Mar Gallardo, Pedro Merino, Ernesto Pimentel

Dpto. de Lenguajes y Ciencias de la Computacion
University of Malaga, 29071 Malaga, Spain

Abstract. Model checking is a well-known automatic verification method based on the exhaustive exploration of the state space produced by a system searching for traces satisfying (not satisfying) a given property. Different techniques have been explored to manage the state explosion problem that occurs when analyzing complex concurrent systems. One of them is the reduction by abstraction of the model and the properties to be checked. The aim of this paper is to compare two semantics for abstracting models when using data type abstractions. One of these semantics is the classic approach based on abstracting transitions. The other one abstracts models by means of a collecting semantics as in the typical applications of abstract interpretation. The framework developed shows that the degree of abstraction and the potential state space generated by both methods are strongly influenced by the underlying abstract domain utilized to simplify the model. We clearly characterize the natural tradeoff that exists in abstract model checking between the number of transitions and the state space produced by an abstract model. However, the partial order reduction technique may substantially modify this tradeoff when implementing both approaches in abstract model checkers. We finally discuss the refinement of models guided by properties. ¹

1 Introduction

The integration of formal techniques, assuring software reliability, into the software development process is only possible if automatic verification tools are constructed. This may be why model checking [1] has become one of the most exciting and studied methods for analyzing complex systems. Applying model checking involves three steps: a) modelling the system $M$ to be analyzed; b) specifying the property $f$ that $M$ should verify; and finally c) using some model checking algorithm “$|$” to automatically check if $M$ is a model for the property $f$, i. e., $M | f$. Model checking algorithms work by exhaustively exploring the state space produced by the model, searching for traces satisfying $f$.

Clearly, the main restriction for model checking is the need of completely constructing the state space to analyze a given system. Therefore, the technique is only applicable when handling medium-sized finite state systems due to the

¹ Work supported by projects TIC99-1083-C02-01 and TIC2001-2705-C03-02
state explosion problem. Different methods have been proposed to improve the size of the systems to be analyzed. Abstract model checking [2, 17, 7] is one of these proposals.

Abstract model checking introduces abstract interpretation [5] into model checking for reducing the size of the models to be analyzed. Abstract interpretation (AI) is a semantic-based automatic analysis method, mainly used to extract information during compilation to improve program execution. Thus AI transforms by abstraction a given (concrete) semantics of a program into an abstract semantics that is computable and safe by construction and that provides approximated information about the program data. In abstract model checking, this loss of information should affect as little as possible the preservation of temporal properties from the abstract to the concrete model. However, strong preservation (that is, the preservation of both universal and existential properties) is only achieved when these two models are bisimilar [17].

There exist two main methods for reducing models in abstract model checking. The collecting semantics which works as in the typical applications of AI, and the non-collecting semantics which directly modifies the original transition system. Abstract model checkers used to implement this last approach.

In this paper we present a generic framework to describe and compare these two abstract semantics. The framework is inspired in the works [13–15] which make use of closure operators to guide the abstraction and refinement of models. The main contributions of the paper are the following ones:

- The presentation based on closure operators allows us to easily show that there exist two natural ways of abstracting properties. One abstraction is appropriate for analyzing universal temporal properties and the other one for refuting existential ones. Although, in general, proving universal properties and refuting existential properties are only equivalent for concrete models [10, 11], we prove that when using the non-collecting semantics, these two methods are equivalent for many practical cases.
- We relate the abstract models generated by both semantics with respect to the precision obtained when analyzing temporal properties. In general, there exists a tradeoff between the number of different states produced by the abstraction (given by the abstract domain) and the number of different transitions in the abstract computation. We discuss the effect of applying some partial order reduction technique [16] when constructing abstract models using the collecting semantics.
- We prove that the refinement of abstract models, necessary to eliminate spurious traces, based on directly modifying the abstract model [14, 15] is applicable in more cases than the refinement approach based on refining properties studied in [11]. However this last method involves less computational cost.

The paper is organized as follows. Section 2 introduces the upper closure operators. Section 3 defines the concrete framework, that is, the transition systems and the Kripke structures. Abstract semantics are presented in Section 4,
and the discussion about its implementation is given in Section 5. Sections 6 and 7 present the extension of both semantics to Kripke structures, giving some preservation results and relating different methods to refine models. Section 8 gives the conclusions.

2 Preliminaries

An upper closure operator on a poset \((P, \leq)\) is an operator \(\rho : P \to P\) monotone 
\((\forall x, y \in P. x \leq y \Rightarrow \rho(x) \leq \rho(y))\), idempotent \((\forall x \in P. \rho(\rho(x)) = \rho(x))\) and extensive \((\forall x \in P. x \leq \rho(x))\). The set of all closure operators (closures) on a poset \((P, \leq)\) is denoted by \(uco(P)\).

When \((P, \leq, \lor, \land, \top, \bot)\) is a complete lattice, an important property of closure operators is that each \(\rho \in uco(P)\) is uniquely determined by the set of its fixed points \(\rho(P)\), as follows: \(X \subseteq P\) is the set of fixed points of a closure iff \(X\) is a Moore-family, that is, if \(X = M(X)\), where \(M(X) = \{\land S | S \subseteq X\}\), defining \(\land \emptyset = \top\). Thus, if \(X \subseteq P\) is a Moore-family, operator \(\rho_X = \lambda y. \land \{x \in X | y \leq x\}\) is the closure on \((P, \leq)\) determined by \(X\).

In addition, when \((P, \leq, \lor, \land, \top, \bot)\) is a complete lattice, the pointwise order relation also defines a complete lattice structure on \(uco(P)\), denoted as \((uco(P), \subseteq, \lor, \land, \top, \bot)\) where if \(\rho, \mu \in uco(P)\) and \(\forall i \in I, \rho_i \in uco(P)\)

- \(\rho \subseteq \mu\) iff \(\forall x \in P. \rho(x) \leq \mu(x)\) iff \(\mu(P) \subseteq \rho(P)\).
- \((\lor_{\rho_i}(x)) = \vee_{\rho_i}(\rho_i(x)).\)
- \((\land_{\rho_i}(x)) = \wedge_{\rho_i}(\rho_i(x)).\)

In the context of abstract interpretation, closure operators are important because abstract domains can be equivalently defined by using them or by Galois insertions [6]. Let \(\Sigma\) be the concrete domain to be abstracted. Consider the complete lattice \((\wp(\Sigma), \subseteq, \lor, \land, \top, \emptyset)\) and a closure \(\rho \in uco(\wp(\Sigma))\). Construct an abstract domain \(A \cong \rho(\wp(\Sigma)), \nu : \rho(\wp(\Sigma)) \to A\) being the isomorphism. Then \((\wp(\Sigma), \nu \circ \rho, \nu^{-1}, A)\) is a Galois insertion. Inversely, if \((\wp(\Sigma), \alpha, \gamma, A)\) is a Galois insertion, then \(\gamma \circ \alpha \circ \nu(\wp(\Sigma)) \to \wp(\Sigma)\) is a closure operator on \((\wp(\Sigma), \subseteq)\).

Basically, this equivalence means that the underlying Galois insertion that defines an abstract interpretation is uniquely determined by an upper closure operator on the concrete domain. Therefore, \(uco(\wp(\Sigma))\) defines the lattice of all the abstract interpretations over \(\wp(\Sigma)\). In addition, given \(\rho_1, \rho_2 \in uco(\wp(\Sigma))\), the relation \(\rho_1 \subseteq \rho_2\) may be interpreted as domain \(\rho_1\) being a refinement of \(\rho_2\) or, from the opposite point of view, as \(\rho_2\) being more abstract than \(\rho_1\).

3 The concrete domain

3.1 Transition Systems

Execution of a concurrent program may be defined by means of a labeled transition system (LTS) such as \(M = (\Sigma, \mathcal{L}, \xrightarrow{\cdot}, s_0)\) where \(\Sigma\) is the set of states, \(\mathcal{L}\)
is a finite set of labels, and $\rightarrow \subseteq \Sigma \times L \times \Sigma$ is the transition relation, and $s_0$ is the initial state. We write $s \xrightarrow{l} s'$ for $(s, l, s') \in \rightarrow$.

A trace $x$ of $M$ is a sequence $x = s_0 \xrightarrow{l_0} s_1 \xrightarrow{l_1} s_2 \ldots$ of states, and it represents a (possibly infinite) computation from state $s_0$. In addition, $x^j$ refers to the suffix path $s_j \xrightarrow{l_j} s_{j+1} \ldots$ of a trace $x = s_0 \xrightarrow{l_0} s_1 \xrightarrow{l_1} s_2 \ldots$. A full-trace $x = s_0 \xrightarrow{l_0} s_1 \xrightarrow{l_1} s_2 \ldots$ is an infinite trace. We assume that terminating traces have a final state which is repeated forever. The set $\mathcal{O}(M) = \{x | x \text{ is a full trace}\}$ defines the trace semantics determined by the transition system $M$.

Consider the relation $\rightarrow$, and $l \in L$, we define the predicate transformer function $\text{post}[\downarrow l] : \wp(\Sigma) \rightarrow \wp(\Sigma)$ as $\text{post}[\downarrow l](X) = \lambda X. \{s' \in \Sigma | \exists s \in X . s \xrightarrow{l} s'\}$

That is, $\text{post}[\downarrow l](X)$ represents the set of successors of $X$ using relation $\xrightarrow{l}$.

### 3.2 Temporal Logic

Kripke structures are used to evaluate temporal formulas against models. In this section, we define Kripke structures over transition systems, and in the following ones, we extend this notion to abstract transition systems. In order to make this extension easily, we consider weak Kripke structures where the negation $\neg$ is not dealt with as a connective, but as a way of constructing an atomic proposition.

Given $\text{Prop}$ a set of propositions, we construct the set $\mathcal{P} = \text{Prop} \cup \neg \text{Prop}$, where $\neg \text{Prop} = \{\neg p : p \in \text{Prop}\}$. Let $\mathcal{F}$ be the set of LTL temporal formulas built inductively using the elements of $\mathcal{P}$, the standard Boolean operators, except the negation $\neg$, and the temporal operators: next “$\square$”, always “$\lozenge$”, eventually “$\diamond$”, and until “$U$”.

A LTS $M = (\Sigma, L, \rightarrow, s_0)$ may be extended to weak Kripke structure $\kappa = \langle M, \tau \rangle$, by using a function $\tau : \Sigma \rightarrow 2^\mathcal{P}$ that assigns truth values to the propositions of $\mathcal{P}$ in each state.

$\kappa = \langle M, \tau \rangle$ is a Kripke structure iff $\forall s \in \Sigma, \forall p \in \text{Prop}$ the following conditions hold:

- **Principle of Excluded Middle**: $p \in \tau(s) \lor \neg p \in \tau(s)$
- **Principle of Non-Contradiction**: $p \notin \tau(s) \lor \neg p \notin \tau(s)$

Note that negated propositions are explicitly included in $\mathcal{P}$. In the sequel, $p \in \mathcal{P}$ denotes both positive and negative atomic propositions. In addition, we assume that $\neg \neg p = p$.

Structure $\kappa = \langle M, \tau \rangle$ defines an interpretation of atomic propositions, which may be rewritten as a satisfiability relation $\models^\tau$ over traces like $x = s \xrightarrow{l} \ldots$ using the initial state $s$ to evaluate the propositions as $x \models^\tau p \iff p \in \tau(s)$.
Now we can extend $\models^\tau$ to temporal formulas by defining the meaning of the temporal operators as follows.

**Definition 1.** Let $\mathcal{K} = (M, \tau)$ be a weak Kripke structure. Given a trace $x$, and $f, g \in \mathcal{F}$, we define relation $\models^\tau$ inductively as follows:

- $x \models^\tau f \lor g$ iff $x \models^\tau f$ or $x \models^\tau g$.
- $x \models^\tau f \land g$ iff $x \models^\tau f$ and $x \models^\tau g$.
- $x \models^\tau f \rightarrow g$ iff $x \models^\tau f$ implies $x \models^\tau g$.
- $x \models^\tau \bigcirc f$ iff $x^1 \models^\tau f$.
- $x \models^\tau \bigdiamond f$ iff $\forall k \geq 0.x^k \models^\tau f$.
- $x \models^\tau f. U g$ iff $\exists k \geq 0.(x^k \models^\tau g$ and $\forall j < k.(x^j \models^\tau f))$.

Finally, we extend $\models^\tau$ to Universal and Existential formulae:

$$M \models^\tau \forall f \iff \forall x \in \mathcal{O}(M).x \models^\tau f$$
$$M \models^\tau \exists f \iff \exists x \in \mathcal{O}(M).x \models^\tau f$$

### 4 Abstract Semantics of LTS

In this section, we present and compare different abstractions of a model $M$. The first one works on the original transition system $M$. The second one is based on constructing and abstracting the collecting semantics of $M$. Both methods allow us to relate the precision of the abstract model constructed depending on the precision of the abstract interpretation used. We prove that the first method is the most accurate when proving temporal formulas; however, in contrast, it produces a bigger set of abstract traces.

#### 4.1 Notation

States of the abstract models built in the sequel are elements of $\wp(\Sigma)$. To properly relate different abstractions of a given model, we define the set $\mathcal{R}(\wp(\Sigma)) = \{\Rightarrow \subseteq \wp(\Sigma) \times \mathcal{L} \times \wp(\Sigma)\}$ of all relations over $\wp(\Sigma)$.

Given $\Rightarrow_1, \Rightarrow_2 \in \mathcal{R}(\wp(\Sigma))$ and two traces $x_1 = s_{s01} \overset{l_0}{\rightarrow} s_{s11} \overset{l_1}{\rightarrow} \cdots$ and $x_2 = s_{s02} \overset{l_0}{\rightarrow} s_{s12} \overset{l_2}{\rightarrow} \cdots$, we say that $x_1$ is more precise than $x_2$ and denote it as $x_1 \preceq x_2$, iff $\forall i \geq 0.s_{s_{ij}} \subseteq s_{s_{ij}'}. \cdots$

In the sequel, we will say that the abstract model $M$ is a sound abstraction of $M$ iff $\forall x = s_0 \overset{l_0}{\rightarrow} s_1 \overset{l_1}{\rightarrow} \cdots \in \mathcal{O}(M)$ there exists $x' \in \mathcal{O}(M)$ such that $\iota(x) \leq x'$, where $\iota(x) = \{s_0\} \overset{l_0}{\rightarrow} \{s_1\} \overset{l_1}{\rightarrow} \{s_2\} \cdots$.

In addition, two abstract LTS $M_1$ and $M_2$, we say that $M_2$ is an abstraction of $M_1$ and write $M_1 \preceq M_2$ when $\forall x \in \mathcal{O}(M_1).\exists x' \in \mathcal{O}(M_2)$ such that $x \leq x'$.

---

Note that symbols $\forall$ and $\exists$ are not LTL connectives. We use them to simplify the notation.
4.2 Non-Collecting Semantics

Given a model \( M = (\Sigma, L, \rightarrow, s_0) \) and a closure \( \rho : \wp(\Sigma) \rightarrow \wp(\Sigma) \), we may construct the abstract transition system \( M^\rho_{nc} = (\wp(\Sigma), L, \rightarrow_\rho, \rho(\{s_0\})) \), where the abstract transition relation \( \rightarrow_\rho \in R(\wp(\Sigma)) \) is defined as the smallest relation satisfying

\[
\forall s \in s_0 \quad \rho(\{s\}) \rightarrow_\rho \rho(\{s'\})
\]

Note that the abstract transition relation extends the original one in that it relates sets of states.

Let \( O(M^\rho_{nc}) \) denote the trace-based semantics generated by the transition relation \( \rightarrow_\rho \). When closure \( \rho \) is the identity \( id : \wp(\Sigma) \rightarrow \wp(\Sigma) \), the abstract model constructed is simply the original one where each state \( s \) has been converted into the set \( \{s\} \). The following proposition relates \( O(M^\rho_{nc}) \) and \( O(M^id_{nc}) \).

**Proposition 1.** Consider \( M \) and \( M^id_{nc} \), then

\[
x \in O(M) \Rightarrow \exists x' \in O(M^id_{nc}) \text{ such that } \iota(x) \leq x'
\]

\[
x' \in O(M^id_{nc}) \Rightarrow \exists x \in O(M) \text{ such that } \iota(x) \leq x'
\]

**Example 1.** Consider \( M = (\Sigma, L, \rightarrow, s_0) \) where \( L = \{inc1, dec1, skip\} \), \( \Sigma = \{s_0, s_1, \ldots, s_n\} \) and relation \( \rightarrow \) is defined as:

(a) \( s_i \xrightarrow{inc1} s_{i+1}, \forall i \neq n \)

(b) \( s_i \xrightarrow{dec1} s_{i-1}, \forall i \neq 0 \)

(c) \( s_i \xrightarrow{skip} s_i, \forall 0 \leq i \leq n \)

The first trace of Fig. 1 shows a possible execution trace \( t \) produced by \( M \). The second trace of this same figure shows the same trace \( t \) in \( M^id_{nc} \).

The following proposition proves that more abstract closures produce more abstract models.

**Proposition 2.** Given \( \rho, \mu \in uco(\wp(\Sigma)) \), \( \rho \subseteq \mu \Rightarrow M^\rho_{nc} \leq M^\mu_{nc} \).
Since for all \( \rho \in uco(\wp(\Sigma)) \) we have that \( \text{id} \subseteq \rho \), the previous result gives us the soundness of any \( M_{nc}^\rho \) wrt \( M_{nc}^\text{id} \), and by Proposition 1, as well wrt \( M \).

**Example 2.** Following Example 1, consider the closures \( \rho_1, \rho_2, \rho_3 : \wp(\Sigma) \rightarrow \wp(\Sigma) \) determined by the fixed point sets

\[
\begin{align*}
\rho_1 &= \{ \emptyset, \{ s_0 \}, \{ s_1, \ldots, s_{n-1} \}, \{ s_n \} \}, \\
\rho_2 &= \{ \emptyset, \{ s_0 \}, \{ s_1 \}, \{ s_2, \ldots, s_{n-1} \}, \{ s_n \} \}, \\
\rho_3 &= \{ \emptyset, \{ s_0 \}, \{ s_1, \ldots, s_{n-1} \}, \{ s_0, s_n \}, \{ s_0, \ldots, s_{n-1} \} \} \}
\end{align*}
\]

Fig. 2 shows the corresponding lattices. The set of traces of Fig. 3 shows the traces of \( M_{nc}^\rho \) approximating the traces in Fig. 1.

Note that the number of states to be represented in the abstract model has decreased with respect to the original model. However, in general, the abstraction process is not complete. The abstract model usually has some abstract traces that do not correspond to any original one. These added false traces are called spurious in the literature. See for instance [3, 11, 14].

**Definition 2.** We say that two closures \( \rho, \mu \in uco(\wp(\Sigma)) \) are equivalent wrt the non-collecting semantics, and we denote it as \( \rho \approx \mu \), iff \( \forall s \in \Sigma, \rho(\{ s \}) = \mu(\{ s \}) \).
The following proposition justifies this definition.

**Proposition 3.** Given $\rho, \mu \in uco(\varphi(\Sigma))$, $\rho \approx \mu$ iff $\mathcal{O}(M^\rho_{nc}) = \mathcal{O}(M^\mu_{nc})$.

That is, given a closure $\rho \in uco(\varphi(\Sigma))$, the non-collecting semantics only takes into account the set $\{\rho(\{s\})|s \in \Sigma\}$ to construct the $\mathcal{O}(M^\rho_{nc})$. For instance, considering the closures in Fig. 2, since $\rho_1$ and $\rho_2$ are equivalent wrt the non-collecting semantics (note that the first level of the lattices coincide), we have that $\mathcal{O}(M^\rho_{nc}) = \mathcal{O}(M^\rho_{nc})$. However, since closures $\rho_1$ and $\rho_2$ are not equivalent ($\rho_1(\{s_1\}) \neq \rho_2(\{s_1\})$), the corresponding abstract models are different.

### 4.3 Collecting Semantics

Given a LTS $M = (\Sigma, \mathcal{L}, \rightarrow, s_0)$ and a closure $\rho : \varphi(\Sigma) \rightarrow \varphi(\Sigma)$, we may construct the (abstract) collecting transition system $M^\rho_c = (\varphi(\Sigma), \mathcal{L}, \Rightarrow_{\rho}, \rho(\{s_0\}))$, where the transition relation $\Rightarrow_{\rho}$ is defined as $ss \Rightarrow_{\rho} \rho(\text{post}[l](ss))$.

As before, the trace-based semantics given by $M^\rho_c$ is $\mathcal{O}(M^\rho_c)$. The next proposition relates different abstractions of a given transition system $M$.

**Proposition 4.** Given $\rho, \mu \in uco(\varphi(\Sigma))$ then

(a) $M^\rho_{nc} \leq M^\rho_c$

(b) If $\rho \subseteq \mu$, then $M^\rho_c \leq M^\mu_c$

That is, the collecting semantics produces more abstract models than the non-collecting one (a). In addition, we have a result (b) similar to Proposition 2 regarding the collecting semantics. In particular, when $\rho$ is the identity function, point (b) gives us the soundness of any abstraction wrt the collecting semantics $\mathcal{O}(M^\rho_{cd})$. Furthermore, using the previous propositions, we obtain the soundness of any abstraction $\mathcal{O}(M^\rho_c)$ wrt $\mathcal{O}(M)$.

\[
\begin{array}{c}
\{s_0\} \xrightarrow{inc_1} \{s_1, \ldots, s_{n-1}\} \xrightarrow{inc_1} \{s_1, \ldots, s_n\} \xrightarrow{dec_1} \{s_n, \ldots, s_{n-1}\} \xrightarrow{dec_1} \{s_n, \ldots, s_1\} \xrightarrow{skip} \cdots \\
\{s_0\} \xrightarrow{inc_1} \{s_1, \ldots, s_{n-1}\} \xrightarrow{inc_1} \{s_1, \ldots, s_n\} \xrightarrow{dec_1} \{s_n, \ldots, s_{n-1}\} \xrightarrow{dec_1} \{s_n, \ldots, s_1\} \xrightarrow{skip} \cdots
\end{array}
\]

**Fig. 4.** Abstract trace in $M^\rho_{c3}$ and $M^\rho_{c1}$

**Example 3.** Considering closure $\rho_3$ shown in Fig. 2, we obtain the first trace of Fig. 4. Comparing the set of abstract traces in Fig. 3 and this trace, we may observe that the number of transitions has decreased.

Note that the notion of equivalence wrt the abstract semantics is not applicable now. Collecting semantics takes into account all the elements of the lattice, and therefore although closures $\rho_1$ and $\rho_3$ given in Fig. 2 are equivalent wrt the non-collecting semantics, abstract models $\mathcal{O}(M^\rho_c)$ and $\mathcal{O}(M^\rho_{c})$ are different. In fact, since $\rho_3 \subseteq \rho_1$, $\mathcal{O}(M^\rho_{c3})$ is a refinement of $\mathcal{O}(M^\rho_{c})$. For instance, the second trace in Fig. 4 is the abstraction in $\mathcal{O}(M^\rho_{c})$ of the first trace in this same figure.

The following definition is used to prove the completeness of $M^\rho_c$ wrt $M^\rho_{nc}$. 


Consider the abstract models $M_1 = (\wp(\Sigma), \mathcal{L}, \rightarrow_1, ss_0^1)$ and $M_2 = (\wp(\Sigma), \mathcal{L}, \rightarrow_2, ss_0^2)$ such that $M_1 \leq M_2$. We say that trace $x = ss_0^1 \xrightarrow{l} ss_1^1 \xrightarrow{l} ss_2^1 \cdots \in \mathcal{O}(M_2)$ may be unfolded in the traces $x_1, \ldots, x_n \in \mathcal{O}(M_1)$, and we denote it as $x = x_1 \lor \cdots \lor x_n$ iff $\forall i = 1, \ldots, n$ trace $x_i = ss_0^1 \xrightarrow{l} ss_1^1 \xrightarrow{l} ss_2^1 \cdots$ and $\forall j \geq 0 ss_j^2 = \bigcup_{i=1}^n ss_{ij}^1$.

That is, each set $ss_j^2$ may be unfolded in the sets $ss_{1j}^1, \ldots, ss_{nj}^1$. For instance, the trace of $M^n_{nc}$ in Fig. 4 may be unfolded in the set of traces of $M^n_{nc}$ in Fig. 3.

**Proposition 5.** Given $\rho \in \, uco(\wp(\Sigma))$ and $x \in \mathcal{O}(M^n_{nc})$, there exists a set $\{x_1, \ldots, x_n\} \subseteq \mathcal{O}(M^n_{nc})$ such that $x = x_1 \lor \cdots \lor x_n$.

Therefore, $M^n_{nc}$ is a more abstract model than $M^n_{nc}$, but in general, it has less traces than $M^n_{nc}$.

**Example 4 (Disjunction of Closures).** This example is devoted to showing the use of disjunction of closures to represent successive abstractions of a model.

Given $\rho, \mu \in \, uco(\wp(\Sigma))$, closure $\rho \lor \mu$ is clearly an abstraction of both. In practice, these joint closures used to be constructed when applying successive abstractions to a model. In this section, we give an example of this. We consider a slight extension of the model $M$ given in Example 1. The new model represents a system with two integer variables $i$ and $j$ which range in the set of states $\Sigma = \{s_0, \ldots, s_n\}$. Thus, we define $M = (\Sigma \times \Sigma, \mathcal{L}, \rightarrow, (s_0, s_0))$, where $\mathcal{L} = \{inc_i, inc_j, dec_i, dec_j, skip\}$, and the transition relation is defined as:

(a) $(s_i, s_j) \xrightarrow{inc_i} (s_{i+1}, s_j), \forall i \neq n$

(b) $(s_i, s_j) \xrightarrow{inc_j} (s_i, s_{j+1}), \forall j \neq n$

(c) $(s_i, s_j) \xrightarrow{dec_i} (s_{i-1}, s_j), \forall i > 0$

(d) $(s_i, s_j) \xrightarrow{dec_j} (s_i, s_{j-1}), \forall j > 0$

(e) $(s_i, s_j) \xrightarrow{skip} (s_i, s_j)$

Consider operators $\rho_{30}, \rho_{33} : \wp(\Sigma \times \Sigma) \rightarrow \wp(\Sigma \times \Sigma)$ defined using the closure $\rho_3$ given in Figure 3 as:

$$\rho_{30}(ss) = \{(s_i', s_j') | s_i' \in \rho_3\{s_i | (s_i, s_j) \in ss\}\}$$

$$\rho_{33}(ss) = \{(s_i', s_j') | s_i' \in \rho_3\{s_j | (s_i, s_j) \in ss\}\}$$

Since $\rho_3$ is a closure, we may easily prove that $\rho_{30}, \rho_{33} \in \, uco(\wp(\Sigma \times \Sigma))$. Each of these two closures abstract a model variable $i$ or $j$ using $\rho_3$. To simultaneously abstract both variables, we should consider closure $\rho_{30} \lor \rho_{33}$.
5 The role of determinism in the Implementation

In this section we discuss how the semantics selected could influence the reduction of the state space in the abstract model. There are two important learned lessons from the use of αspin [8, 9, 12] with real examples. The first one is that we cannot ensure a priori which method produces the smallest state space. The second one is that partial order reductions can help us to implement abstraction. We now discuss these problems and propose some future research.

The reduction depends on the model. From the implementation point of view, the use of the non-collecting semantics implies utilizing non-deterministic assignments when updating abstracted variables, which provokes the direct creation of several branches (see, for instance, the tree shown in Fig. 3). In contrast, the use of the collecting semantics eliminates the non-deterministic assignments and allows us to delay the creation of branches until a test over the updated variable is found. The state space produced depends on the model, and not only on the lattice or on the way of implementing the abstraction. For example,

```
/* define GLOBAL or LOCAL for c[] */
#define P 1 /* copies of the process type */
#define N 49 /*limits the values for c[] */
#ifndef GLOBAL
#define id _pid
#else
#endif
byte c[P]
active [P] proctype update() {
  if
    :: c[id] = 0
    :: c[id] = 1
    :: c[id] = N
  fi;
}
```

Fig. 5. PROMELA model to update a variable

the abstraction of the trace in Fig. 1 produces several abstract traces using the non-collecting semantics (Fig. 3). In contrast, Fig. 4 shows the corresponding abstract trace generated following the collecting semantics.

However, in other cases, the use of more values in the abstract domain (seven for ρ3) than in the non-collecting method (three for ρ1, or four if 0 is also considered) produces worse results for the first method. For example, consider the PROMELA model in Fig. 5. The code in this figure could be considered as a model for a process containing three typical manipulations of a variable which
are critical when the variable is abstracted: initialization to precise values, updating (with a potential loss of precision) and testing with a required precision. Table 1 contains a measure of the abstract states for the abstraction \( \rho_3 \) (Fig. 2) using the collecting and non-collecting implementations, respectively. We are assuming that each state \( s_i \) represents the value \( i \in \{1, \ldots, N\} \) for the model variable, and that labels \( \text{inc} \) and \( \text{dec} \) represent the increment and decrement operations. Note that the abstract model constructed in both cases is smaller than the original one (denoted as concrete). In addition, for this example, the non-collecting implementation produces better results than the collecting choice when the variable \( c[] \) is declared as global. It is also worth noting that in both abstract methods the number of states does not depend on parameter \( N \) in the model, although the number of transitions varies with this parameter.

<table>
<thead>
<tr>
<th>P</th>
<th>Model</th>
<th>States</th>
<th>Transitions</th>
<th>Depth</th>
<th>States</th>
<th>Transitions</th>
<th>Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Concrete</td>
<td>255</td>
<td>2753</td>
<td>104</td>
<td>156</td>
<td>2655</td>
<td>55</td>
</tr>
<tr>
<td></td>
<td>Collecting</td>
<td>36</td>
<td>139</td>
<td>12</td>
<td>25</td>
<td>131</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>Non-Collecting</td>
<td>20</td>
<td>123</td>
<td>10</td>
<td>19</td>
<td>123</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>Concrete</td>
<td>64503</td>
<td>1.3755e+06</td>
<td>28435</td>
<td>466</td>
<td>7963</td>
<td>110</td>
</tr>
<tr>
<td></td>
<td>Collecting</td>
<td>1104</td>
<td>7081</td>
<td>79</td>
<td>121</td>
<td>651</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>Non-Collecting</td>
<td>348</td>
<td>3633</td>
<td>65</td>
<td>325</td>
<td>1811</td>
<td>53</td>
</tr>
<tr>
<td>3</td>
<td>Concrete</td>
<td>out-of-memory</td>
<td></td>
<td></td>
<td>1086</td>
<td>18579</td>
<td>165</td>
</tr>
<tr>
<td></td>
<td>Collecting</td>
<td>32064</td>
<td>262177</td>
<td>471</td>
<td>505</td>
<td>2731</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>Non-Collecting</td>
<td>5792</td>
<td>78217</td>
<td>575</td>
<td>5527</td>
<td>28250</td>
<td>273</td>
</tr>
<tr>
<td>4</td>
<td>Concrete</td>
<td>out-of-memory</td>
<td></td>
<td></td>
<td>2326</td>
<td>39811</td>
<td>220</td>
</tr>
<tr>
<td></td>
<td>Collecting</td>
<td>902400</td>
<td>8.51367e+06</td>
<td>3192</td>
<td>2041</td>
<td>11051</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>Non-Collecting</td>
<td>93904</td>
<td>1.47904e+06</td>
<td>5259</td>
<td>93961</td>
<td>471642</td>
<td>1373</td>
</tr>
</tbody>
</table>

Table 1. State space for the model UPDATE

**Abstracting local variables.** Our experience with \( \alpha \text{SPIN} \) has allows us to find a very interesting point in favor of using the collecting semantic based implementation for some systems. In particular, preventing non-deterministic sentences in the abstract model permits using the “statement merging” optimization described in [16]. This method is a kind of partial order reduction that can be safely applied to local variables with only deterministic manipulations. Column “Local” in Table 1 shows how both the concrete model and the collecting abstract model can take advantages of this optimization. Note that this is incompatible with the non-collecting semantics when it utilizes non-deterministic assignments.

This kind of optimization is clearly valuable, because many real examples require abstracting local variables as a side effect of abstracting global variables. Usually, local variables to be abstracted are found with dependency analysis.

11
Some conclusions. One conclusion of our work is that more research has to be done in order to have a criteria to a priori choose the best implementation approach. As both implementation approaches produce sound over-approximations of the model, we have included the two methods as options in aSPiN, allowing the user to experiment both for a given model and for different optimization options. We are currently working on semiautomatic methods to move from one implementation to another with mechanisms like folding and unfolding particular elements of the lattice used for the abstraction.

6 Abstracting Kripke Structures

In this section, we study the effect of abstraction on the evaluation of formulae. Given $K = (M, \tau)$ a (weak) Kripke structure, and any sound abstraction $\mathbf{M}$ of $M$, we construct the Kripke structure $\mathbf{AK} = (\mathbf{M}, \tau_u, \tau_o)$ where functions $\tau_u, \tau_o : \wp(\Sigma) \rightarrow \mathcal{P}$ are defined as $\tau_u(ss) = \cap_{s \in ss} \tau(s)$ and $\tau_o(ss) = \cup_{s \in ss} \tau(s)$.

That is, for a given set of states $ss \in \wp(\Sigma)$, $\tau_u(\tau_o)$ defines as true those propositions that are satisfied (using $\tau$) by all (some of) the states in $ss$. We use subindexes $u$ ($o$) because $\tau_u$ ($\tau_o$) under (over) approximates $\tau$.

Extending Definition 1 to the weak Kripke structure $\mathbf{AK}$ using both $\tau_o$ and $\tau_u$, we obtain the following results regarding the preservation of universal formulae and the refutation of existential formulae. In order to simplify the notation, we write $|=, |=_u$ and $|=o$ instead of $|=\tau$, $|=\tau_u$ and $|=\tau_o$.

Proposition 6. Given $f \in \mathcal{F}$ and $\mathbf{M}$ a sound abstraction of $M$ then $\mathbf{M} |=_u \forall f \Rightarrow M |= \forall f$ and $\mathbf{M} \not|=o \exists f \Rightarrow M \not|= \exists f$

That is, we use function $\tau_u$ to verify universal properties over any abstract model, and function $\tau_o$ to refute existential properties. We only obtain the strong preservation of temporal formulas when the two transition systems $M$ and $\mathbf{M}$ are bisimilar [17]. However, in general there is an implicit loss of information during the abstract process that influences the analysis of temporal formulae.

Proposition 7. Given $f \in \mathcal{F}$, and $\rho, \mu \in uco(\wp(\Sigma))$ such that $\rho \subseteq \mu$

1) $M^{\mu_{nc}}_c |=_u \forall f \Rightarrow M^{\mu_{nc}}_c |=_u \forall f$ and $M^{\mu_{nc}}_c \not|=o \exists f \Rightarrow M^{\mu_{nc}}_c \not|=o \exists f$
2) $M^{\mu}_{c} |=_u \forall f \Rightarrow M^{\mu}_{c} \not|=o \exists f$ and $M^{\mu}_{c} \not|=o \exists f \Rightarrow M^{\mu}_{c} \not|=o \exists f$
3) $M^{\rho}_{c} |=_u \forall f \Rightarrow M^{\rho}_{c} \not|=o \exists f$ and $M^{\rho}_{c} \not|=o \exists f \Rightarrow M^{\rho}_{c} \not|=o \exists f$

In [11], we saw that in general $M \not|=u \exists f$ does not imply that $M \not|=o \exists f$, and inversely, $M \not|=o \forall f$ does not imply that $M \not|=u \forall f$, which means that we cannot use $\tau_u$ to refute properties nor $\tau_o$ to verify them. However, both methods may be interchanged when the conditions given by the following definition hold.

Definition 4. Given $\rho \in uco(\wp(\Sigma))$, proposition $p \in \mathcal{P}$ is compatible with $\rho$ iff $\forall s \in \Sigma$, either $p \in \tau(s'), \forall s' \in \rho(\{s\})$ or $p \notin \tau(s'), \forall s' \in \rho(\{s\})$. Formula $f \in \mathcal{F}$ is compatible with $\rho$ iff all the propositions in $f$ are compatible with $\rho$. 

12
Proposition 8. Given $\rho \in \mathrm{uco}(\varphi(\Sigma))$ and $f \in \mathcal{F}$ compatible with $\rho$, then

$$M^\rho_{nc} \models_u \forall f \Leftrightarrow M^\rho_{nc} \models_o \forall f$$

$$M^\rho_{nc} \not\models_o \exists f \Leftrightarrow M^\rho_{nc} \not\models_u \exists f$$

This proposition holds because the loss of information in $\mathcal{O}(M^\rho_{nc})$ is only due to the application of function $\rho$ to individual states. Thus if the formula is compatible with the abstraction, the over and under-approximation methods coincide for the non-collecting semantics.

7 Refinement of Models guided by Properties

Diverse techniques have been proposed to eliminate spurious traces in abstract models, for instance see [3, 13]. An alternative proposal studied in [11] is the use of properties to automatically refine models, exploiting the model checking power. In this section, we rewrite this last approach with the notation introduced in this paper and relate the model refinement based on properties with the domain refinement presented above. The following results are also applied to the non-collecting semantics. In [11], we proved the following result.

Proposition 9. Given $\rho, \mu \in \mathrm{uco}(\varphi(\Sigma))$ two closures, and $f, g \in \mathcal{F}$ then

$$M^\mu_c \models_u \forall g, M^\rho_c \not\models_o \exists (g \land \neg f) \Rightarrow M \models \forall f$$

$$M^\mu_c \not\models_o \exists g, M^\rho_c \not\models_o \exists (\neg g \land f) \Rightarrow M \not\models \exists f$$

In the previous proposition, we assume that $f$ cannot be proved (or discarded) on $M^\rho_c$ because of some spurious traces. Formula $g$ is the refiner formula which allows for eliminating those traces in $M^\rho_c$ satisfying (or not satisfying) $g$.

The following proposition proves that the model refinement based on properties is stronger than refinement of abstract domains. That is, the second one is, in general, more effective but, in contrast, it implies more computational cost because it involves the construction of a new refined abstract model $M^{\mu \land \rho}_c$.

Proposition 10. Given $\rho, \mu \in \mathrm{uco}(\varphi(\Sigma))$ two closures, and $f$ and $g \in \mathcal{F}$, then

$$M^\mu_c \models_u \forall g, M^\rho_c \not\models_o \exists (g \land \neg f) \Rightarrow M^{\mu \land \rho}_c \models_u \forall f$$

$$M^\mu_c \not\models_o \exists g, M^\rho_c \not\models_o \exists (\neg g \land f) \Rightarrow M^{\mu \land \rho}_c \not\models_o \exists f$$

8 Conclusions

Diverse tools implement the abstract model checking technique. We have developed the tool $\alpha$-spin [8, 9, 12, 18] which integrates the two methods for abstracting models and also the two dual methods of abstracting properties. Bandera [4] is another well-known abstract model checker which mainly utilizes the non-collecting abstraction method. Our experience suggests that in some applications it is better to use this way of abstracting models, while in others, the collecting method is preferable. Future work is focused on using the representation developed here to eliminate spurious traces of abstract models following [11].
References


