Abstract Interpretation of Reactive Systems

DENNIS DAMS and ROB GERTH
Eindhoven University of Technology
and
ORNA GRUMBERG
Technion, Haifa

The advent of ever more complex reactive systems in increasingly critical areas calls for the development of automated verification techniques. Model checking is one such technique, which has proven quite successful. However, the state-explosion problem remains a major stumbling block. Recent experience indicates that solutions are to be found in the application of techniques for property-preserving abstraction and successive approximation of models. Most such applications have so far been based solely on the property-preserving characteristics of *simulation relations*. A major drawback of all these results is that they do not offer a satisfactory formalisation of the notion of *precision* of abstractions.

The theory of Abstract Interpretation offers a framework for the definition and justification of property-preserving abstractions. Furthermore, it provides a method for the effective computation of abstract models directly from the text of a program, thereby avoiding the need for intermediate storage of a full-blown model. Finally, it formalises the notion of optimality, while allowing to trade precision for speed by computing sub-optimal *approximations*.

For a long time, applications of Abstract Interpretation have mainly focussed on the analysis of universal safety properties, i.e. properties that hold in all states along every possible execution path. In this paper, we extend Abstract Interpretation to the analysis of both existential and universal reactive properties, as expressible in the *modal μ-calculus*. It is shown how abstract models may be constructed by symbolic execution of programs. A notion of approximation between abstract models is defined while conditions are given under which optimal models can be constructed. Examples are given to illustrate this. We indicate conditions under which also falsehood of formulae is preserved. Finally, we compare our approach to those based on simulation relations.

Categories and Subject Descriptors: F.3.1 [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs—*Mechanical verification*; *Logics of programs*; D.2.4 [**Software Engineering**]: Program Verification—*Validation*

General Terms: Reliability, Theory, Verification

Additional Key Words and Phrases: Abstract interpretation, formal methods, model checking, mu-calculus, reactive systems

Correspondence address: D. Dams, Dept. of Math. and Computing Science, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands. `wsindd@win.tue.nl`

1. INTRODUCTION

In the *model-checking* approach [Queille and Sifakis 1982; Clarke et al. 1986; Vardi and Wolper 1986; Kurshan 1994; Lichtenstein and Pnueli 1985] to program verification, a model of the program is constructed over which formulae are checked for satisfaction. The model reflects the possible behaviours of the program, while the formulae express certain required properties of such behaviours. Obviously, the size of the model is a limiting factor to the feasibility of the model-checking approach. In the worst case, it doubles with every extra bit of memory that the program may access. This problem is referred to as the *state-explosion problem*. One solution to it is the application of abstraction techniques, which aim to abstract the model to a smaller one, in such a way that if some property holds for the abstracted model, it also holds for the original model.

Such abstraction techniques are formalised in the framework of *Abstract Interpretation* [Cousot and Cousot 1977], which was originally conceived as a unifying theory of compile-time (data-flow) analyses. For a long time, applications of Abstract Interpretation have been focussed on the analysis of *universal safety* properties, that hold in all states (safety) along all possible executions (universality) of the program.[1]

With the advent of *reactive systems*, interest has broadened to a larger class of properties. Reactive systems are systems whose main role is to maintain an ongoing interaction with the environment, rather than to produce some final result on termination. Usually, such systems consist of several concurrent processes, and display a non-deterministic behaviour. Typical examples are flight reservation systems, industrial plant controllers, embedded systems and operating systems. In the presence of non-determinism, one may be interested to know whether some property holds along *some* possible execution path. Such properties will be called *existential*. Besides safety, another kind of property that is often considered is *liveness*, meaning that something should hold eventually (given an execution). Thus, we have classified properties into four kinds by the criteria universal/existential and safety/liveness. A typical combination of universal safety and existential liveness properties is "along every possible execution path, in every state there is a possible continuation that will eventually reach a reset state".

The semantic models and abstraction techniques used in the analysis of universal safety properties cannot be used for properties that involve aspects of existentiality and eventuality. The reason is that these techniques abstract away from information about the choices that a program encounters during execution. The analysis of existentiality and eventuality properties of behaviours, however, requires models that, in addition to information about single states, also provide the transitions between states. For this reason, in model checking reactive systems, *transition systems* are used to model the behaviour of programs. Being directed graphs over program states, such transition systems give detailed information about program executions, including the possible choices in every state. Our aim is to find notions of abstraction of such transition systems that *preserve* certain combined forms of

[1] The notions of universality and safety of a property are not always distinguished as explicitly as we do in this paper. What we call "universal safety" is often just termed "safety" or "invariance" elsewhere.

universal/existential safety/liveness properties. This means that in order to know that such a property holds in the original system, it suffices to know that it holds in the abstracted system.

The properties may be formalised by expressing them in a logic whose formulae can be interpreted over transition systems. One such logic is L_μ, the modal μ-calculus [Kozen 1983]. Besides a basic set of propositions stating local properties about states, and the usual boolean operations, it contains modalities that express that something holds in some next state or in all next states[2]. Furthermore, fixpoint operators allow to combine such next-state properties into formulae expressing existential and universal properties about execution paths. Although properties specified in the μ-calculus are often less comprehensible than when expressed in temporal logics like CTL* (*computation tree logic*, see Emerson and Halpern [1986]), L_μ is preferable for our purposes, for the following reasons. Firstly, the basic temporal modalities correspond directly to two types of abstract transition relations to be defined, which improves understanding and facilitates proofs. Furthermore, L_μ allows a clean identification of universal and existential properties — see Section 2.2 where this point is discussed. Finally, L_μ's expressivity exceeds that of many other temporal logics, including CTL* [Dam 1994]. Hence, the results of this paper immediately transfer to those logics as well. Together with the fact that the main focus of this paper is on preservation results, and not on the practice of specifying in L_μ, these reasons should justify our choice for the μ-calculus.

The structure of this article is as follows. The next section introduces the formal machinery to be used. In Section 3, a notion of abstract transition system is developed that preserves properties from L_μ. A canonical abstraction is chosen so as to satisfy the maximum number of L_μ properties. This choice is then justified in Section 4, where an approximation order between such systems is defined. This approximation order is shown to coincide with the L_μ-property ordering, and the canonical abstraction of Section 3 turns out to be optimal, i.e. it is the least element with regard to this ordering which is still safe. Section 5 shows how abstract transition systems may be computed directly from a program text by "lifting" the operations of a programming language to a domain of data *descriptions*. Conditions are given under which the constructed models are optimal. Furthermore, it is shown that sub-optimal models are constructed when computing approximations to the lifted operations. An elaborate example is presented in Section 6. Section 7 briefly indicates the consequences of insisting on *strong preservation*, meaning that not only truth, but also falsehood of formulae is preserved. Section 8 compares ours to related work, in particular to simulation-based approaches, and Section 9 concludes.

On first reading, it may be helpful to skip the more technical results of Subsections 4.1, 5.1, 5.2 and 5.3, to get to the example in Section 6 first.

[2]As models we consider Kripke structures, without action labelling. Hence, the modalities do not refer to actions.

2. PRELIMINARIES

2.1 Temporal logic

Given is a set *Prop* of *propositions*. We choose to define L_μ in its negation normal form, i.e. negations only appear in front of propositions. This facilitates the definition of the fragments $\Box L_\mu$ and $\Diamond L_\mu$ below. The set of *literals* is defined by $Lit = Prop \cup \{\neg p \mid p \in Prop\}$.

Definition 2.1.1. Let *Var* be a set of *propositional variables*. Moreover, let $p \in Lit$ and $x \in Var$. The logic L_μ is the set of formulae that is defined by the following grammar:

$$\varphi ::= p \mid x \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \Box\varphi \mid \Diamond\varphi \mid \mu x.\varphi \mid \nu x.\varphi$$

For $\varphi \in L_\mu$, the formula $\neg\varphi$ is considered to be an abbreviation of the equivalent L_μ formula in negation normal form (obtained in the usual way). The abbreviations *true*, *false* and $\rightarrow$ can then be defined as usual.

The *universal* and *existential* fragments $\Box L_\mu$ and $\Diamond L_\mu$ are subsets of L_μ in which the only allowed next-state operators are $\Box$ and $\Diamond$ respectively. Likewise, a formula that is (equivalent to a formula) in $\Box L_\mu$ ($\Diamond L_\mu$) is called universal (existential).

A formula $\Box\varphi$ expresses that φ is true for every (immediate) successor while $\Diamond\varphi$ expresses that there exists at least one successor for which φ is true. A propositional variable from *Var* can be seen as a formula whose meaning (i.e. the set of states in which it holds) depends on some environment that binds variables to sets of states. $\mu x.\varphi$ and $\nu x.\varphi$ are the least resp. greatest fixpoint operators. Their meaning is the smallest (resp. greatest) set x of states in which φ holds — where φ typically depends on x. Thus, e.g., $\nu x.(p \wedge \Box x)$ expresses invariance of the truth of proposition p, while $\mu x.(p \vee \Diamond x)$ expresses the possibility of establishing p. Formal definitions and further examples are given in the next subsection.

2.2 Transition systems

L_μ formulae are interpreted over *transition systems* $\mathcal{T} = (\Sigma, I, R)$ where Σ is a set of *states*, $I \subseteq \Sigma$ is a set of *initial* states, and $R \subseteq \Sigma \times \Sigma$ is a *transition relation* over Σ. By an *execution*, or *path*, we mean any sequence of pairwise related states that is maximal, i.e. it is infinite or its last state has no successors in $\mathcal{T}$. State $s \in \Sigma$ is *reachable* iff it lies on a path that starts from a state in I.

Associated to $\mathcal{T}$, we assume a function $||\cdot||_{Lit} : Lit \rightarrow \mathcal{P}(\Sigma)$, satisfying $||p||_{Lit} \cap ||\neg p||_{Lit} = \emptyset$ for every proposition $p \in Prop$, that specifies the interpretation of literals over states. Intuitively, $||p||_{Lit}$ is the set of states where p holds. Transition systems thus defined are closely related to Kripke structures [Kripke 1963]. The main difference is that we have the function $||\cdot||_{Lit}$ instead of a labelling function from Σ to sets of literals. The reason for not requiring $||p||_{Lit} \cup ||\neg p||_{Lit} = \Sigma$ will become clear in Section 3.1.

The following definition gives the meaning of L_μ formulae relative to a given transition system by specifying a function $||\cdot||$ that maps a formula to the set of states in which it holds. $||\cdot||$ has an additional argument, written behind it, which is a function specifying the interpretation of propositional variables. The role of this "environment" becomes clear in the cases of the fixpoint formulae.

Definition 2.2.1. The function $||\cdot||\cdot : L_\mu \times (Var \rightarrow \mathcal{P}(\Sigma)) \rightarrow \mathcal{P}(\Sigma)$ is defined as follows. Let $p \in Lit$, $x \in Var$, $\varphi, \varphi_1, \varphi_2 \in L_\mu$ and $e : Var \rightarrow \mathcal{P}(\Sigma)$.

$$\begin{array}{ll}
||p||e & = ||p||_{Lit} \\
||x||e & = e(x) \\
||\varphi_1 \vee \varphi_2||e & = ||\varphi_1||e \cup ||\varphi_2||e \\
||\varphi_1 \wedge \varphi_2||e & = ||\varphi_1||e \cap ||\varphi_2||e \\
||\Box\varphi||e & = \{s \in \Sigma \mid \forall_{s' \in \Sigma}\ R(s, s') \Rightarrow s' \in ||\varphi||e\} \\
||\Diamond\varphi||e & = \{s \in \Sigma \mid \exists_{s' \in \Sigma}\ R(s, s') \wedge s' \in ||\varphi||e\} \\
||\mu x.\varphi||e & = \bigcap\{S \subseteq \Sigma \mid ||\varphi||e[x \mapsto S] \subseteq S\} \\
||\nu x.\varphi||e & = \bigcup\{S \subseteq \Sigma \mid S \subseteq ||\varphi||e[x \mapsto S]\}
\end{array}$$

$e[x \mapsto S]$ is the mapping that is the same as e except in x, which is mapped to S. $\perp_{env}$ is the environment that maps every $x \in Var$ to $\emptyset$. For a closed[3] formula φ, $||\varphi||$ abbreviates $||\varphi||\perp_{env}$. We write $s \models \varphi$ for $s \in ||\varphi||$.

For a set S of states, the notation $S \models \varphi$ abbreviates $\forall_{s \in S}\ s \models \varphi$. When there may be confusion between different systems, we write $(\mathcal{T}, s) \models \varphi$ to denote that $s \models \varphi$ in $\mathcal{T}$, and similar for $(\mathcal{T}, S) \models \varphi$. $\mathcal{T} \models \varphi$ abbreviates $(\mathcal{T}, I) \models \varphi$.

L_μ formulae can express a variety of properties of transition systems. The distinction between universal and existential properties is captured by the division into $\Box L_\mu$ and $\Diamond L_\mu$. Similarly, safety properties correspond to greatest fixpoints while liveness is expressed through least fixpoints. For example, $s \models \nu x.(p \wedge \Box x)$ expresses the universal safety property that p is true in all states that are reachable from s. In the branching-time temporal logic CTL* this would be expressed as $s \models \forall Gp$. The CTL* formula $\exists Gp$, which says that there exists a path along which p holds in all states, is expressed in L_μ by $\nu x.(p \wedge \Diamond x)$ — this is an existential safety formula. An existential liveness property like "there exists a path along which p eventually holds" ($\exists Fp$ in CTL*) is similarly expressed as $\mu x.(p \vee \Diamond x)$. Only universal liveness properties require a slightly more involved formulation. The reason is that the $\Box$, used to express universal properties, does not require the existence of a successor state — i.e. a property $\Box\varphi$ is satisfied by any state that has no successors. On the other hand, liveness *does* require the existence of successors, at least until the point where the eventuality is fulfilled. Thus, "along all paths, p eventually holds" is $\mu x.(p \vee (\Diamond true \wedge \Box x))$. Note that this formula is neither universal nor existential: the $\Box$ is essential to express the "along all paths" part, while the $\Diamond$ is needed to reflect correctly the meaning of "eventually". One could say that the notion of liveness has an existential character[4]. On the other hand, it is not difficult to see that for *deadlock free* transition systems, the $\Diamond true$ may be dropped from formulae.

LEMMA 2.2.2. *Let $\mathcal{T} = (\Sigma, I, R)$ be a transition system such that every state that is reachable from an initial state has at least one successor. Let $\varphi \in L_\mu$ and denote by φ' the formula obtained from φ by replacing all subformulae of the form $\Diamond true$ by true. Then $\mathcal{T} \models \varphi$ iff $\mathcal{T} \models \varphi'$.*

---

[3] A formula is closed if every propositional variable that occurs in it is bound by a fixpoint operator.
[4] It is mainly this observation that has led us to present the preservation results for fragments of the μ-calculus, rather than for the fragments $\forall$CTL* and $\exists$CTL* of CTL* in which the existential character of liveness remains hidden and would complicate the results (cf. Dams et al. [1995]).

PROOF. Because every reachable state has at least one successor, $\Diamond true$ is equivalent to $true$ in those states. As the truth of $\mathcal{T} \models \varphi$ only depends on reachable states and furthermore equivalence of formulae is a congruence, $\Diamond true$ may be replaced by $true$ in φ. □

We fix a transition system $\mathcal{C} = (\Sigma, I, R)$ called the *concrete model*. This will play the role of the original, large model that we need to abstract in order to be able to verify its L_μ properties.

2.3 Abstract Interpretation

A transition system forms the *interpretation* of a program: it models the possible behaviours. Formally, this is captured by an interpretation function $\mathcal{I}$ from programs[5] to transition systems. Properties of a program P's behaviour may be analysed by studying $\mathcal{I}(P)$. As this model may be too complex to handle (because of the state explosion), we look for abstractions of it that can provide partial information. Two points are of quintessential importance in the definition of such abstracted transition systems. Firstly, the abstraction should *preserve* the information that we are interested in: any (L_μ) property that holds for the abstract model, should hold for the original concrete model as well. Sections 3 and 4 focus on this aspect. Secondly, such abstractions are to be constructed directly from the program, and not by first building a full model and then abstracting it. That is, we are looking for an *abstract interpretation function* ${}_\alpha\mathcal{I}$ that maps each program P to an abstraction of $\mathcal{I}(P)$. Section 5 defines such a function, fixing a simple notion of program for this purpose.

In the rest of this section we review the framework of *Abstract Interpretation*, which offers ways to formalise the notion of abstraction and provides means to design "good" abstract interpretation functions through the notions of *optimality* and *approximation*. Briefly, the idea is as follows. As the concrete object that we are interested in, in this case the concrete model, is too large to handle, we abstract from certain aspects of the states. Thereby, states that were different now become identified. This is formally captured by introducing a set ${}_\alpha\Sigma$ of *abstract states* and a concretisation function γ mapping each abstract state a to a set of concrete states that are described by a. For example, we may wish to abstract from everything but the fact whether variable x is greater than 5 or not, in which case we introduce two abstract states named, e.g., $\mathsf{grt\_5}$ and $\mathsf{leq\_5}$, acting as descriptions of sets of concrete states as specified by the function γ with $\gamma(\mathsf{grt\_5}) = \{s \in \Sigma \mid s(x) > 5\}$ and $\gamma(\mathsf{leq\_5}) = \{s \in \Sigma \mid s(x) \leq 5\}$ (we view a state as a valuation function on variables here). The goal is then to construct transition systems over abstract states by interpreting the operations occurring in the program over the data descriptions $\mathsf{grt\_5}$ and $\mathsf{leq\_5}$. For example, the execution of the assignment $x := x + 1$ in the abstract state $\mathsf{grt\_5}$ results in the state $\mathsf{grt\_5}$ again. On the other hand, executing it in the state $\mathsf{leq\_5}$, we do not know whether the result is $\mathsf{grt\_5}$ or $\mathsf{leq\_5}$; both are possible. Thus, information is lost. If this is unwanted, we could introduce a new abstract state, $\mathsf{leq\_6}$, with the obvious meaning, in order to capture this more precisely. Similarly, we might introduce a state $\mathsf{leq\_7}$ to be able to capture the effect of $x := x + 1$ on $\mathsf{leq\_6}$, etc. In order for each set of concrete states to have a

[5] In Section 5 we will become more specific about the syntax of programs.

description, the set of abstract states should be such that for every subset C of Σ, there exists an abstract state a with $\gamma(a) \supseteq C$. Note that this does *not* imply that there is a different abstract state for each subset of concrete states. For example, the abstract domain may consist of only the state $\top$ with $\gamma(\top) = \Sigma$. On the other hand, the requirement does imply that there must always be such a "top" state in the abstract domain.

Abstract states may now be ordered according to their information content or *precision*. Define $a \preceq a' \Leftrightarrow \gamma(a) \subseteq \gamma(a')$; in this case we say that a is more precise than a', or that a' approximates a. So, e.g., $\mathsf{leq\_5} \preceq \mathsf{leq\_6}$ and $\mathsf{leq\_6} \preceq \top$. If for each set C of concrete states there exists a *unique* most precise description, then an *abstraction function* $\alpha : \mathcal{P}(\Sigma) \rightarrow {}_{\alpha}\Sigma$ may be introduced that maps each C to this description. Minimality with regard to $\preceq$ is also referred to as *optimality*. Correctness of the abstract interpretation of the assignment $x := x + 1$ can be stated by requiring that for each abstract state a, a" $+ 1$" $\succeq \alpha(\gamma(a) + 1)$ (where on the left-hand-side of the $\succeq$, the quotes indicate that the function $+1$ has to be interpreted abstractly, over the domain of abstract values; on the right-hand-side, $+1$ is pointwise extended to sets).

The formalisation of these ideas as offered by the theory of Abstract Interpretation [Cousot and Cousot 1977] generalises the concrete and abstract domains — $(\mathcal{P}(\Sigma), \subseteq)$ and $({}_{\alpha}\Sigma, \preceq)$ resp. in the example above — to arbitrary partially ordered sets $(C, \sqsubseteq)$ and $(A, \preceq)$. The abstraction function $\alpha : C \rightarrow A$ and concretisation function $\gamma : A \rightarrow C$ are usually required to form a Galois connection from $(C, \sqsubseteq)$ to $(A, \preceq)$:

Definition 2.3.1. $(\alpha : C \rightarrow A, \gamma : A \rightarrow C)$ is a *Galois connection* from $(C, \sqsubseteq)$ to $(A, \preceq)$ iff (1) α and γ are total and monotonic, (2) for all $c \in C$, $\gamma \circ \alpha(c) \sqsupseteq c$, and (3) for all $a \in A$, $\alpha \circ \gamma(a) \preceq a$.

Under these requirements, α by definition yields the optimal, that is $\preceq$-least, abstraction of every $c \in C$ and γ yields for every $a \in A$ the most general, that is $\sqsubseteq$-greatest, element that is abstracted by a. Note that this framework does not necessarily imply that $a \preceq a' \Leftrightarrow \gamma(a) \sqsubseteq \gamma(a')$. If in addition that condition *does* hold, like in our example above, then (and only then) we have $\alpha \circ \gamma(a) = a$ for all $a \in A$; (α, γ) is then called a Galois *insertion* from $(C, \sqsubseteq)$ to $(A, \preceq)$.

Under the assumptions of the Galois connection framework, $a \in A$ is an abstraction of $c \in C$ iff $\alpha(c) \preceq a$ (or, equivalently, $c \sqsubseteq \gamma(a)$). Given a program P, the goal is to compute, in an efficient manner, an abstraction ${}_{\alpha}\mathcal{I}(P)$ of $\mathcal{I}(P)$. Usually, ${}_{\alpha}\mathcal{I}$ is constructed by providing an abstract "counterpart" for each operation used in the definition of $\mathcal{I}$. For example, in this paper, where a transition system forms the interpretation of a program, the function $\mathcal{I}$ is defined in terms of predicates c_i and t_i that correspond to tests and transformations in programs; see the last point of Definition 5.0.1. The abstract interpretation of programs is defined in a similar way in terms of abstractions c_i^F, t_i^F, c_i^C and t_i^C; see the last points of Definition 5.0.2.

A number of weaker frameworks than the Galois-connection framework have been proposed in which the abstraction and/or concretisation functions are replaced by relations (see Marriott [1993] or Dams [1996] for an overview). These cater for situations where most precise abstractions and/or most general concretisations do not exist, for example because concrete elements have a number of optimal abstractions that are mutually incomparable, or because the corresponding approximation orderings have not been defined, or are pre- instead of partial orders. In this paper, we start from a given Galois insertion on the level of states, and induce a notion of abstraction between transition systems, where concrete systems are unordered while the approximation order on abstract systems is a pre-order. Although there may exist different representations of the most precise abstraction of a given concrete system, all these representations share the same L_μ properties. In Section 3, an abstraction function is given that maps any concrete system to one such representative. Hence, by comparing a computed abstraction to this representative, we are able to discuss optimality issues.

A more extensive introduction to Abstract Interpretation and overview of its uses can be found in Cousot and Cousot [1992a] and Cousot and Cousot [1992b].

2.4 The relation transformers $\cdot^{\exists\exists}$ and $\cdot^{\forall\exists}$; (bi)simulations

We use two relation transformers that are used to lift transition relations on states to relations on sets of states.

Definition 2.4.1. Let A and B be sets and $R \subseteq A \times B$. The relations $R^{\exists\exists}, R^{\forall\exists} \subseteq \mathcal{P}(A) \times \mathcal{P}(B)$ are defined as follows.

—$R^{\exists\exists} = \{(X, Y) \mid \exists_{x \in X}\ \exists_{y \in Y}\ R(x, y)\}$
—$R^{\forall\exists} = \{(X, Y) \mid \forall_{x \in X}\ \exists_{y \in Y}\ R(x, y)\}$

So, if R is a transition relation, $R^{\exists\exists}(X, Y)$ iff some state in X can make an R-transition to some state in Y, and $R^{\forall\exists}(X, Y)$ iff every state in X can make an R-transition to some state in Y.

Finally, we recall the definitions of simulation [Milner 1971] and bisimulation [Park 1981].

Definition 2.4.2. Let $\mathcal{T}_1 = (\Sigma_1, I_1, R_1)$ and $\mathcal{T}_2 = (\Sigma_2, I_2, R_2)$ be transition systems. A relation $\sigma \subseteq \Sigma_1 \times \Sigma_2$ is a *simulation (from $\mathcal{T}_1$ to $\mathcal{T}_2$)* iff $\sigma^{-1}R_1 \subseteq R_2\sigma^{-1}$ (juxtaposition denotes composition of relations). In this case we say that R_1 *σ-simulates* R_2. σ is a *bisimulation* if in addition σ^{-1} is a simulation from $\mathcal{T}_2$ to $\mathcal{T}_1$. A simulation σ is *consistent* iff $\sigma(s_1, s_2)$ implies $\forall_{p \in Lit}\ s_2 \models p \ \Rightarrow\ s_1 \models p$.

An equivalent definition of simulation is the following[6] (see, e.g., Milner [1971]). Whenever $\sigma(s_1, s_2)$ and $R_1(s_1, s_1')$, then there exists s_2' such that $R_2(s_2, s_2')$ and $\sigma(s_1', s_2')$.

3. ABSTRACT TRANSITION SYSTEMS

The definition of an abstract system $\mathcal{A}$ starts from a given poset $({}_\alpha\Sigma, \preceq)$ of *abstract states* together with a Galois insertion (α, γ) from $(\mathcal{P}(\Sigma), \subseteq)$ to $({}_\alpha\Sigma, \preceq)$ that determines its relation to the concrete states. We usually write $\alpha(c)$ for $\alpha(\{c\})$. We investigate how to define abstract models in such a way that L_μ is preserved from

[6] The intuition is that R_2 can "mimic" everything that R_1 can do. From this point of view, the terminology "R_1 simulates R_2", which was introduced in Milner [1971], is awkward.

the abstract to the concrete model. Our goal is to define, given the abstraction of states, an abstract transition system $\mathcal{A}$ that satisfies as many L_μ properties as possible. The choices that are made in this section are motivated by this goal. In Section 4, we formally establish the optimality of this canonical abstraction $\mathcal{A}$.

As we start from a notion of abstraction of states, it is natural to require preservation of formulae on the level of individual states ("statewise preservation"):

$$\forall_{\varphi \in L_\mu, a \in {}_\alpha\Sigma}\ (\mathcal{A}, a) \models \varphi \ \Rightarrow\ (\mathcal{C}, \gamma(a)) \models \varphi \tag{1}$$

We take this requirement as the starting point in defining the abstract model $\mathcal{A}$. Besides ${}_\alpha\Sigma$, which is already given, we need three more ingredients for the definition of such a model: a function ${}_\alpha\|\cdot\|_{Lit}$ specifying the interpretation of literals over abstract states, a set ${}_\alpha I$ of abstract initial states, and an abstract transition relation ${}_\alpha R$. These points are considered in the following subsections.

3.1 Valuation of literals

In order to satisfy (1) for the literals in L_μ, we must have $(\mathcal{A}, a) \models p \ \Rightarrow\ (\mathcal{C}, \gamma(a)) \models p$ for every literal p and every abstract state a. On the other hand, as we intend to use the abstract model in order to infer properties of the concrete model, we would like as many literals as possible to hold in each abstract state.

Definition 3.1.1. For $p \in Lit$, define $\quad {}_\alpha\|p\|_{Lit} = \{a \in {}_\alpha\Sigma \mid \gamma(a) \subseteq \|p\|_{Lit}\}$.

This choice determines the valuation of literals in abstract states. Namely, the value of ${}_\alpha\|p\|e$, where ${}_\alpha\|\cdot\|\cdot$ has the functionality $L_\mu \times (Var \to \mathcal{P}({}_\alpha\Sigma)) \to \mathcal{P}({}_\alpha\Sigma)$, is defined as in the first clause of Definition 2.2.1, where $\|\cdot\|_{Lit}$ has to be replaced by ${}_\alpha\|\cdot\|_{Lit}$. By this choice it can now easily be shown that $(\mathcal{A}, a) \models p \ \Leftrightarrow\ (\mathcal{C}, \gamma(a)) \models p$, for every $a \in {}_\alpha\Sigma$ and $p \in Lit$, which implies that as many literals as possible hold in each abstract state. Note that if $a \in {}_\alpha\Sigma$ is such that $\gamma(a)$ contains concrete states in which p holds *and* concrete states in which $\neg p$ holds, then $a \notin {}_\alpha\|p\|_{Lit}$ but also $a \notin {}_\alpha\|\neg p\|_{Lit}$. So, although it is always the case that either $a \models p$ holds, or its negation $a \not\models p$, and similarly for $\neg p$, it may occur that for some a we have neither $a \models p$ nor $a \models \neg p$. In particular, $a \not\models p$ does not necessarily imply that $a \models \neg p$.

Furthermore, less-precise states satisfy fewer literals:

LEMMA 3.1.2. *Let $a, a' \in {}_\alpha\Sigma$. If $a' \succeq a$, then for all $p \in Lit$ $a' \models p \Rightarrow a \models p$.*

3.2 Abstract initial states

The abstract initial states should be chosen in such a way that the requirement (1) of statewise preservation implies preservation on the level of models: $\mathcal{A} \models \varphi$ should imply $\mathcal{C} \models \varphi$, for all $\varphi \in L_\mu$. A sufficient condition for this is $\bigcup\{\gamma(a) \mid a \in {}_\alpha I\} \supseteq I$. On the other hand, the set of abstract initial states has to be as small as possible, so that the condition $\mathcal{A} \models \varphi$ to be verified is as weak as possible. In general, it is not possible to choose ${}_\alpha I$ such that $\bigcup\{\gamma(a) \mid a \in {}_\alpha I\} = I$. However, the following choice for ${}_\alpha I$ yields the smallest set $\bigcup\{\gamma(a) \mid a \in {}_\alpha I\}$ that still includes I.

Definition 3.2.1. ${}_\alpha I = \{\alpha(c) \mid c \in I\}$

One may wonder why we did not take $\alpha(I)$ as the (single) abstract initial state. The reason is that each element of $\{\alpha(c) \mid c \in I\}$ is in general at least as precise as $\alpha(I)$: because α distributes over $\bigcup$ (see, e.g., Cousot and Cousot [1979]), we have

$\alpha(I) = \bigvee\{\alpha(s) \mid s \in I\}$ (where $\bigvee$ denotes the least upper bound in ${}_\alpha\Sigma$), so that for each $s \in I$, $\alpha(s) \preceq \alpha(I)$. Therefore, the set $\bigcup\{\gamma(a) \mid a \in {}_\alpha I\}$ of concrete states to which ${}_\alpha I$ (as defined above) corresponds, is a subset of the concretisation $\gamma(\alpha(I))$.

3.3 Abstract transition relations

We want to abstract a concrete by an abstract transition relation in such a way that both existential and universal properties are preserved. However, such an abstract transition relation ${}_\alpha R$ will only exist if there exists a consistent bisimulation from (Σ, R) to $({}_\alpha\Sigma, {}_\alpha R)$ (see, e.g., Loiseaux et al. [1995]), which is a much too strong condition, as it results in the falsehood of L_μ formulae being preserved as well. Our solution is to define instead *two* transition relations on ${}_\alpha\Sigma$; one preserving universal properties, and the other existential properties.

It is not difficult to see that if properties of the form $\Diamond\varphi$ are to be preserved, then abstract state b may only be a successor of a if $R^{\forall\exists}(\gamma(a), Y)$ for some $Y \subseteq \Sigma$ with $Y \subseteq \gamma(b)$. For reasons of optimality we also would like a to have as many successors as possible[7], and, furthermore, each of them should be a description of Y that is as precise as possible. The first requirement is satisfied by letting b be a successor of a *whenever* $R^{\forall\exists}(\gamma(a), Y)$; the second by choosing Y to be minimal and b to be the most precise description of it, as specified by α. A similar consideration for the preservation of universal properties leads to the requirement that b is a successor of a iff $R^{\exists\exists}(\gamma(a), Y)$, Y is minimal, and $\alpha(Y) = b$:

Definition 3.3.1.

(1) ${}_\alpha R^F(a, b) \;\Leftrightarrow\; b \in \{\alpha(Y) \mid Y \in \min\{Y' \mid R^{\exists\exists}(\gamma(a), Y')\}\}$
(2) ${}_\alpha R^C(a, b) \;\Leftrightarrow\; b \in \{\alpha(Y) \mid Y \in \min\{Y' \mid R^{\forall\exists}(\gamma(a), Y')\}\}$

${}_\alpha R^F$ and ${}_\alpha R^C$ are called the *free* and *constrained* (abstract transition) relations respectively. Note that for any $a \in {}_\alpha\Sigma$, the minimal sets Y' such that $R^{\exists\exists}(\gamma(a), Y')$, are all singletons[8]. Also, note that by the requirement of minimality of Y in the definitions, it is not in general the case that ${}_\alpha R^C \subseteq {}_\alpha R^F$.

In order to accommodate these two different transition relations in a single transition system, we give the following definition.

Definition 3.3.2. A *mixed transition system* is a quadruple $M = (S, I, F, C)$ consisting of a set S of states, a set I of initial states, and two transition relations F and C called the free and constrained (transition) relations respectively. A *free path* is a path with all its transitions in F; a *constrained path* is a path with all its transitions in C. The notion of reachability is taken relative to the union $F \cup C$ of

---

[7]In Definition 3.3.1, we only consider minimal sets of successors in order to keep the relation small. Of course, in the presence of BDD representations this does not necessarily imply that the representation will actually be smaller. In general, any abstract transition relation in which these minimal successors are present will be as good.

[8]With regard to the definition of ${}_\alpha R^C$, we should point out that the set $\{Y' \mid R^{\forall\exists}(\gamma(a), Y')\}$ may have no minimal elements. (As an example, take R to be the ordering $\geq$ on the integers and $\gamma(a)$ the set of all integers.) By dropping the requirement of minimality in this case, every abstract state will have an ${}_\alpha R^C$-successor and as a result more existential formulae will hold in the abstract model. The preservation results to be presented will still hold: observe that their proofs do not depend on the minimality requirements.

both transition relations, unless explicitly specified otherwise. The interpretation of L_μ formulae over a mixed system is defined slightly different from Definition 2.2.1. Besides replacing $\|\cdot\|_{Lit}$ by the valuation function that is associated with M, R has to be replaced by F in the clause for $\|\Box\varphi\|e$ and by C in the clause for $\|\Diamond\varphi\|e$.

An environment that maps propositional variables to sets of abstract states is called an *abstract environment*. In particular, ${}_\alpha\bot_{env}$ maps every $x \in \mathit{Var}$ to $\emptyset$.

These mixed systems form our notion of abstract transition system. We now define the canonical, "best" abstraction $\alpha^M(\mathcal{C})$ of $\mathcal{C}$ as the mixed transition system $\mathcal{A}^M = ({}_\alpha\Sigma, {}_\alpha I, {}_\alpha R^F, {}_\alpha R^C)$. The valuation of literals in states of $\mathcal{A}^M$ is specified by the function ${}_\alpha\|\cdot\|_{Lit}$ defined in Subsection 3.1 above.

We then have:

THEOREM 3.3.3. *For every* $\varphi \in L_\mu$, $\quad \mathcal{A}^M \models \varphi \;\Rightarrow\; \mathcal{C} \models \varphi$.

PROOF. It is easy to see that it suffices to prove statewise preservation for every φ in L_μ: for every state $a \in {}_\alpha\Sigma$, $(\mathcal{A}^M, a) \models \varphi \;\Rightarrow\; (\mathcal{C}, \gamma(a)) \models \varphi$. Let $\breve{\gamma}$ be the extension of γ to sets of abstract states, defined by $\breve{\gamma}(A) = \bigcup\{\gamma(a) \mid a \in A\}$. By definition of $\models$, it suffices to prove the stronger claim that

> $\breve{\gamma}({}_\alpha\|\varphi\|d) \subseteq \|\varphi\|e$ whenever the abstract environment d and the concrete environment e are such that for every $x \in \mathit{Var}$, $\breve{\gamma}(d(x)) \subseteq e(x)$.

The result follows because clearly for every $x \in \mathit{Var}$, $\breve{\gamma}({}_\alpha\bot_{env}(x)) \subseteq \bot_{env}(x)$.

This fact is easily proven for the base cases. If φ is a literal p, then it follows from Definition 3.1.1. If φ is a variable x, then it follows from the assumption on d and e.

Conjunctions and disjunctions are also easily proven. As for the next-state operators, we show the case for $\Diamond\varphi$ — the case $\Box\varphi$ is similar. We have to show that $\breve{\gamma}({}_\alpha\|\Diamond\varphi\|d) \subseteq \|\Diamond\varphi\|e$. Let $c \in \breve{\gamma}({}_\alpha\|\Diamond\varphi\|d)$, which by definition of $\breve{\gamma}$ means that we can choose $a \in {}_\alpha\|\Diamond\varphi\|d$ such that $c \in \gamma(a)$. We have to show that $c \in \|\Diamond\varphi\|e$, i.e., that there exists $d \in \Sigma$ such that $R(c, d)$ and $d \in \|\varphi\|e$. Now, $a \in {}_\alpha\|\Diamond\varphi\|d$ means that we can choose $b \in {}_\alpha\Sigma$ such that (i) ${}_\alpha R^C(a, b)$ and (ii) $b \in {}_\alpha\|\varphi\|d$. From (i), it follows by definition of ${}_\alpha R^C$ that $b = \alpha(Y)$ for some $Y \subseteq \Sigma$ satisfying $R^{\forall\exists}(\gamma(a), Y)$. As $c \in \gamma(a)$, we can choose a $d \in Y$ such that $R(c, d)$. By monotonicity of α we have $\alpha(d) \preceq \alpha(Y) = b$. From the latter, we obtain by monotonicity of γ that $\gamma(\alpha(d)) \subseteq \gamma(b)$. Because $\gamma \circ \alpha(C) \supseteq C$, it now follows that $d \in \gamma(b)$. From (ii), it follows by the induction hypothesis that $\gamma(b) \subseteq \|\varphi\|e$, so we get $d \in \|\varphi\|e$.

From the fixpoint formulae, we show the ν case — μ is similar. Consider $\breve{\gamma}({}_\alpha\|\nu x.\varphi\|d)$. By definition of ${}_\alpha\|\nu x.\varphi\|d$, this is equal to $\breve{\gamma}(\bigcup\{A \mid A \subseteq {}_\alpha\|\varphi\|d[x \mapsto A]\})$. Clearly, $\breve{\gamma}$ distributes over $\bigcup$, so that this equals $\bigcup\{\breve{\gamma}(A) \mid A \subseteq {}_\alpha\|\varphi\|d[x \mapsto A]\}$ (iii). Below, we show that $\{\breve{\gamma}(A) \mid A \subseteq {}_\alpha\|\varphi\|d[x \mapsto A]\} \subseteq \{C \mid C \subseteq \|\varphi\|e[x \mapsto C]\}$. As a result, the expression (iii) is a subset of $\bigcup\{C \mid C \subseteq \|\varphi\|e[x \mapsto C]\}$, which is by definition equal to $\|\nu x.\varphi\|e)$.

Consider the expression $A \subseteq {}_\alpha\|\varphi\|d[x \mapsto A]$. By monotonicity of $\breve{\gamma}$, it implies $\breve{\gamma}(A) \subseteq \breve{\gamma}({}_\alpha\|\varphi\|d[x \mapsto A])$. By the induction hypothesis, the right-hand side of this is a subset of $\|\varphi\|e[x \mapsto \breve{\gamma}(A)]$. Therefore, $\{\breve{\gamma}(A) \mid A \subseteq {}_\alpha\|\varphi\|d[x \mapsto A]\}$ is a subset of $\{\breve{\gamma}(A) \mid \breve{\gamma}(A) \subseteq \|\varphi\|e[x \mapsto \breve{\gamma}(A)]\}$, which is in turn easily seen to be a subset of $\{C \mid C \subseteq \|\varphi\|e[x \mapsto C]\}$. $\Box$

So, mixed abstractions allow verification of full L_μ while the degree of reduction is determined by the choice of the abstract domain and may hence be arbitrarily large. In contrast, reductions with regard to bisimulation equivalence [Bouajjani et al. 1992] only allow a limited reduction. These facts may seem contradictory, but the reader should note that by the definition of satisfaction of L_μ formulae over mixed abstractions, it is possible that neither φ, nor $\neg\varphi$ holds; this is not possible with bisimulation reduction.

4. APPROXIMATION AND OPTIMALITY

We have defined an abstraction function α^M mapping each concrete system $\mathcal{C}$ to the best abstract system $\mathcal{A}^M$. It involves the definitions of the valuation of literals in abstract states (${}_\alpha\|\cdot\|_{Lit}$), of the abstract initial states (${}_\alpha I$), and of two abstract transition relations (${}_\alpha R^F$ and ${}_\alpha R^C$). Each of these definitions, given in the previous section, was motivated by the objective to define the transition system $\alpha^M(\mathcal{C})$ in such a way that it satisfies as many L_μ properties as possible. For example, the free and constrained transition relations in $\mathcal{A}^M$ were defined by always choosing, for the successors of an abstract state, the $\preceq$-least descriptions (as provided by α) of $\subseteq$-minimal sets of concrete states, and furthermore by having a minimal number of free and a maximal number of constrained successors. In this section, we give a formal justification of these choices by defining an approximation ordering $\trianglelefteq$ on abstract systems. Just as α^M may be seen as the lifting of the abstraction function α from individual states to transition systems, $\trianglelefteq$ is the lifting of $\preceq$. We show that, under certain conditions, $\trianglelefteq$ coincides with the "L_μ-property ordering" (i.e. we show that $\mathcal{A}' \trianglelefteq \mathcal{A}''$ iff $\mathcal{A}'$ enjoys at least the same L_μ-properties as $\mathcal{A}''$) and furthermore that $\alpha^M(\mathcal{C})$ is the $\trianglelefteq$-least abstract transition system (over the given set ${}_\alpha\Sigma$ of abstract states[9]) that is "safe" in the sense that L_μ properties are preserved from $\alpha^M(\mathcal{C})$ to $\mathcal{C}$. Thus, this formally establishes the optimality of $\alpha^M(\mathcal{C})$. Another reason for introducing approximations is that they turn out to occur in a natural way when computing abstractions directly from a program, as will be done in the next section.

Definition 4.0.1. Let $\mathcal{A}' = ({}_\alpha\Sigma, I', F', C')$ and $\mathcal{A}'' = ({}_\alpha\Sigma, I'', F'', C'')$ be mixed transition systems. A relation $\sigma \subseteq {}_\alpha\Sigma \times {}_\alpha\Sigma$ is a *mixed simulation* (from $\mathcal{A}'$ to $\mathcal{A}''$) iff (1) σ is consistent[10], (2) F' σ-simulates F'', and (3) C'' σ^{-1}-simulates C'.

$\mathcal{A}' \trianglelefteq \mathcal{A}''$ iff there exists a mixed simulation σ from $\mathcal{A}'$ to $\mathcal{A}''$ such that (4) for every $a' \in I'$ there exists $a'' \in I''$ such that $\sigma(a', a'')$.

The following lemma expresses that this approximation order coincides with the L_μ-property ordering. A similar theorem was proven in Larsen [1989]. That paper considers Hennessy-Milner Logic, which does not feature a fixpoint operator, however, the proof for fixpoint formulae is analogous to the fixpoint case in the proof of Theorem 3.3.3.

[9] One point to mark is that all the mixed transition systems that we are comparing have the same set ${}_\alpha\Sigma$ of abstract states. Obviously, one can always construct better abstractions by refining this abstract domain — up to the point where it contains the same amount of detail as the set Σ of concrete states. The merit of an optimality result is that it identifies the most precise abstraction *given* the loss of information that is inherent to the choice of ${}_\alpha\Sigma$.

[10] σ is consistent iff $\sigma(s_1, s_2)$ implies $\forall_{p \in Lit}\ s_2 \models p \ \Rightarrow\ s_1 \models p$, see Definition 2.4.2.

THEOREM 4.0.2. *Let $\mathcal{A}'$ and $\mathcal{A}''$ be as in Definition 4.0.1. If $\mathcal{A}' \trianglelefteq \mathcal{A}''$, then for every $\varphi \in L_\mu$, $\mathcal{A}'' \models \varphi \;\Rightarrow\; \mathcal{A}' \models \varphi$. Furthermore, if F' and C'' are finitely branching*[11]*, then the reverse holds as well: if for every $\varphi \in L_\mu$, $\mathcal{A}'' \models \varphi \;\Rightarrow\; \mathcal{A}' \models \varphi$, then $\mathcal{A}' \trianglelefteq \mathcal{A}''$.*

Note that the relation $\trianglelefteq$ over abstract systems is a pre-order but not a partial order (i.e. $\mathcal{A}' \trianglelefteq \mathcal{A}''$ and $\mathcal{A}'' \trianglelefteq \mathcal{A}'$ does not imply $\mathcal{A}' = \mathcal{A}''$). We turn it into a partial order by identifying systems $\mathcal{A}'$ and $\mathcal{A}''$ whenever both $\mathcal{A}' \trianglelefteq \mathcal{A}''$ and $\mathcal{A}'' \trianglelefteq \mathcal{A}'$ hold; by Theorem 4.0.2 we are justified to do this. To improve readability, we do not explicitly distinguish between equivalence classes and representants.

As an immediate corollary of this theorem and Theorem 3.3.3, we have:

COROLLARY 4.0.3. *If $\mathcal{A} \trianglerighteq \alpha^M(\mathcal{C})$, then for every $\varphi \in L_\mu$, $\quad \mathcal{A} \models \varphi \;\Rightarrow\; \mathcal{C} \models \varphi$.*

4.1 Optimality

Together, α^M and $\trianglelefteq$ induce the following abstraction (or *description*) relation (cf. page 117):

Definition 4.1.1. $\mathcal{A}$ is an abstraction of $\mathcal{C}$ iff $\mathcal{A} \trianglerighteq \alpha^M(\mathcal{C})$.

Corollary 4.0.3 above states that such an abstraction $\mathcal{A}$ is safe for $\mathcal{C}$. However, we still do not know how good (in the sense of $\trianglelefteq$) the abstraction $\alpha^M(\mathcal{C})$ is with regard to other transition systems over ${}_\alpha\Sigma$ that are safe for $\mathcal{C}$. The following theorem shows that under a few additional conditions, $\alpha^M(\mathcal{C})$ is indeed the best abstraction of $\mathcal{C}$.

THEOREM 4.1.2. *Let $\mathcal{A}' = ({}_\alpha\Sigma, F', C', I')$ be a mixed transition system with an associated valuation function $\|\cdot\|'_{Lit}$ that is such that for every $c \in \Sigma$ and every $a \in {}_\alpha\Sigma$*

$$[\forall_{p \in Lit}\; a \models p \;\Rightarrow\; c \models p] \;\Rightarrow\; c \in \gamma(a). \tag{2}$$

Assume that $\forall_{\varphi \in L_\mu}\; (\mathcal{A}', a') \models \varphi \;\Rightarrow\; (\mathcal{C}, \gamma(a')) \models \varphi$ and furthermore that R (the concrete transition relation) and F' are finitely branching. Then $\preceq$ is a mixed simulation from $\alpha^M(\mathcal{C})$ to $\mathcal{A}'$.

If, in addition to statewise preservation, we require global preservation to hold, i.e. $\forall_{\varphi \in L_\mu}\; \mathcal{A}' \models \varphi \;\Rightarrow\; \mathcal{C} \models \varphi$, and furthermore I' is finite, then $\mathcal{A}' \trianglerighteq \alpha^M(\mathcal{C})$.

Assumption (2) in the theorem intuitively means that γ is maximal in the sense that whenever the truth of all literals is preserved from some a to some c, then a is indeed a description of c. Note that this does not necessarily hold for the canonical valuation function ${}_\alpha\|\cdot\|_{Lit}$ of Section 3.1. It may be the case that two abstract states a and a' satisfy the same literals, but nevertheless describe different sets of concrete states. Assumption (2) forbids this, so that a and a' cannot be interchanged.

PROOF OF THEOREM 4.1.2. Let $\mathcal{A} = \alpha^M(\mathcal{C}) = ({}_\alpha\Sigma, {}_\alpha R^F, {}_\alpha R^C, {}_\alpha I)$. Points (1) through (4) below correspond to the points in Definition 4.0.1. The first three points deal with the first part of the theorem while the second part is proven in point (4).

[11] A relation $R \subseteq A \times B$ is *finitely branching*, or *image-finite*, iff for every $a \in A$, the set $\{b \in B \mid R(a,b)\}$ has finite cardinality.

(1) Consistency of $\preceq$ follows from Lemma 3.1.2.

(2) Let $a \preceq a'$ and ${}_\alpha R^F(a,b)$. By definition of ${}_\alpha R^F$ this means that we can choose $c, d \in \Sigma$ such that $c \in \gamma(a)$, $R(c,d)$ and $b = \alpha(\{d\})$. Because $a \preceq a'$, we also have $c \in \gamma(a')$. By the assumption of statewise preservation, it now follows that there exists b' such that $F'(a', b')$ and $d \in \gamma(b')$. Namely, suppose there is not, i.e. $\forall_{b'}\ F'(a',b') \ \Rightarrow\ d \notin \gamma(b')$. Then by assumption (2) in the theorem, we can choose literals $p_{b'}$ such that $b' \models p_{b'}$ for every such b', but $d \not\models p_{b'}$ for any such b'. Then $a' \models \Box \bigvee_{b':F'(a',b')} p_{b'}$, while $c \not\models \Box \bigvee_{b':F'(a',b')} p_{b'}$. Because F' is finitely branching, $\Box \bigvee_{b':F'(a',b')} p_{b'}$ is a L_μ formula, and hence we have arrived at a contradiction.

Furthermore, from $\{d\} \subseteq \gamma(b')$ it follows that $\alpha(\{d\}) \preceq \alpha \circ \gamma(b')$, i.e. $b \preceq b'$.

(3) Let $a \preceq a'$ and ${}_\alpha R^C(a', b')$. Let $c \in \gamma(a)$. So also $c \in \gamma(a')$. Then we can choose $d \in \gamma(b')$ such that $R(c,d)$. Namely, suppose $\forall_d\ R(c,d) \ \Rightarrow\ d \notin \gamma(b')$. Then by assumption (2) in the theorem, we can choose literals p_d such that $b' \models p_d$ for every such d, but for any such d, $d \not\models p_d$. Then $a' \models \Diamond \bigwedge_{d:R(c,d)} p_d$, while $c \not\models \Diamond \bigwedge_{d:R(c,d)} p_d$. Because R is finitely branching, $\Diamond \bigwedge_{d:R(c,d)} p_d$ is a L_μ formula, and hence we have arrived at a contradiction.

So, $R^{\forall\exists}(\gamma(a), \gamma(b'))$. Then we can choose Y to be a $\subseteq$-minimal Y' for which $R^{\forall\exists}(\gamma(a), Y')$ in such a way that $Y \subseteq \gamma(b')$. By definition of ${}_\alpha R^C$, we have ${}_\alpha R^C(a, \alpha(Y))$. As $Y \subseteq \gamma(b')$, $\alpha(Y) \preceq \alpha(\gamma(b'))$, i.e. $\alpha(Y) \preceq b'$.

(4) Next, we show that the additional assumption of $\forall_{\varphi \in L_\mu}\ \mathcal{C} \models \varphi \ \Leftarrow\ \mathcal{A}' \models \varphi$ implies that $\forall_{a \in {}_\alpha I}\ \exists_{a' \in I'}\ a \preceq a'$.

Suppose this does not hold, i.e. we can choose $a \in {}_\alpha I$ such that $\forall_{a' \in I'}\ a \not\preceq a'$. By definition of $\preceq$ this means $\forall_{a' \in I'}\ \gamma(a) \not\subseteq \gamma(a')$. By definition of ${}_\alpha I$ (namely, $\{\alpha(\{c\}) \mid c \in I\}$), we can choose $c \in I$ such that $a = \alpha(\{c\})$. Below, we will show that $\forall_{a' \in I'}\ c \notin \gamma(a')$. By assumption (2) in the theorem, we can then choose literals $p_{a'}$ such that $a' \models p_{a'}$ for every such a' but $c \not\models p_{a'}$ for any such a'. This implies that $\mathcal{A}' \models \bigvee_{a' \in I'} p_{a'}$, while $\mathcal{C} \not\models \bigvee_{a' \in I'} p_{a'}$. By the assumption that I' is finite, this formula is in L_μ. So we have arrived at a contradiction.

We need to show that $\forall_{a' \in I'}\ c \notin \gamma(a')$. Suppose this does *not* hold, say $c \in \gamma(a')$ for some $a' \in I'$. Then $\{c\} \subseteq \gamma(a')$, hence $\alpha(\{c\}) \preceq a'$, i.e. $a \preceq a'$. Contradiction.

□

Note that we require both global and statewise preservation in Theorem 4.1.2. It turns out that if we weaken the condition to global preservation ($\forall_{\varphi \in L_\mu}\ \mathcal{A}' \models \varphi \ \Rightarrow\ \mathcal{C} \models \varphi$) alone, we cannot prove $\mathcal{A}' \trianglerighteq \alpha^M(\mathcal{C})$ anymore. This means that $\alpha^M(\mathcal{C})$ is not the $\trianglelefteq$-least safe description of $\mathcal{C}$ in that case. However, it can still be shown that then $\mathcal{A}' \ntrianglelefteq \alpha^M(\mathcal{C})$, meaning that $\alpha^M(\mathcal{C})$ is $\trianglelefteq$-*minimal*, although not unique.

This section was motivated by Cleaveland et al. [1995] that also establishes optimality results, be it in a more restricted context (see also Section 8). In that restricted framework, their *canonical abstraction* (similar to our $\alpha^M(\mathcal{C})$) is the unique best description of $\mathcal{C}$.

5. COMPUTING ABSTRACT MODELS BY ABSTRACT INTERPRETATION

After having defined abstract models and proven their preservation properties, we now get to the topic of how to compute such models directly from a program. We will do this through abstract interpretation of the program text. An abstract interpretation may be viewed as a non-standard semantics defined over a domain of data-descriptions, where the functions are given corresponding non-standard interpretations. The abstract states are then valuations of program variables over the domain of data-descriptions, and the abstract transitions are computed by evaluation of the abstract semantic functions over these domains.

In order to further develop the theory, we first need to fix a programming language. We use a language that is based on *action systems* [Back and Kurki-Suonio 1983], which, although being very simple, will help to grasp the idea of how to abstractly interpret operations in "real" programming languages, as it contains rudimentary forms of the common notions of assignment, test and loop. A program is a set of *actions* of the form $c_i(\bar{x}) \rightarrow t_i(\bar{x}, \bar{x}')$, where i ranges over some index set J, $\bar{x}$ represents the vector of program variables, c_i is a condition on their values and t_i specifies a transformation of their values into the new vector $\bar{x}'$. A program is run by repeatedly nondeterministically choosing an action whose condition c_i yields *true* and updating the program variables as specified by the associated transformation[12] t_i. In the following, we let P be the program $\{c_i(\bar{x}) \rightarrow t_i(\bar{x}, \bar{x}') \mid i \in J\}$, Val the set of values that the vector $\bar{x}$ may take, and $IVal \subseteq Val$ the set of values that it may have initially. Thus, each c_i is a predicate over Val and each t_i a relation on $Val \times Val$.

Definition 5.0.1. P's (concrete) interpretation $\mathcal{I}(P)$ is the transition system (Σ, I, R) defined as follows.

—$\Sigma = Val$

—$I = IVal$

—$R = \{(\bar{v}, \bar{v}') \in Val^2 \mid \exists_{i \in J}\ c_i(\bar{v}) \ \wedge\ t_i(\bar{v}, \bar{v}')\}$

Henceforth, we identify $\mathcal{I}(P)$ with the concrete Kripke structure $\mathcal{C}$.

Next, we assume a set ${}_\alpha Val$ of descriptions of sets of values in Val, via a Galois insertion (α, γ), and define two types of non-standard, *abstract* interpretations of the c_i's and t_i's over ${}_\alpha Val$ in such a way that abstractions of the concrete models of programs may be computed by interpreting the operators in the program correspondingly. Note the similarity of the following definitions with Definition 3.3.1 of ${}_\alpha R^F$ and ${}_\alpha R^C$.

Definition 5.0.2. For $i \in J$, let c_i^F, c_i^C be conditions on ${}_\alpha Val$ and t_i^F, t_i^C be transformations on ${}_\alpha Val \times {}_\alpha Val$.

—c_i^F is a *free abstract interpretation of* c_i iff for every $a \in {}_\alpha Val$,
$c_i^F(a) \Leftrightarrow \exists_{\bar{v} \in \gamma(a)}\ c_i(\bar{v})$.

—t_i^F is a *free abstract interpretation of* t_i iff for every $a, b \in {}_\alpha Val$,

[12] As t_i is a relation, there may be several different updated states $\bar{x}'$. In this case, one of these is selected nondeterministically.

$t_i^F(a,b) \;\Leftrightarrow\; b \in \{\alpha(Y) \mid Y \in \min\{Y' \mid t_i{}^{\exists\exists}(\gamma(a), Y')\}\}$.

—c_i^C is a *constrained abstract interpretation of* c_i iff for every $a \in {}_\alpha Val$,
$c_i^C(a) \Leftrightarrow \forall_{\bar{v}\in\gamma(a)}\; c_i(\bar{v})$.

—t_i^C is a *constrained abstract interpretation of* t_i iff for every $a, b \in {}_\alpha Val$,
$t_i^C(a,b) \;\Leftrightarrow\; b \in \{\alpha(Y) \mid Y \in \min\{Y' \mid t_i{}^{\forall\exists}(\gamma(a), Y')\}\}$.

Furthermore, we define the abstract interpretation ${}_\alpha\mathcal{I}(P)$ of P as the system $\widehat{\mathcal{A}^M} = ({}_\alpha\Sigma, {}_\alpha I, \widehat{{}_\alpha R^F}, \widehat{{}_\alpha R^C})$ where:

—${}_\alpha\Sigma = {}_\alpha Val$

—${}_\alpha I = \{\alpha(\bar{v}) \mid \bar{v} \in IVal\}$

—$\widehat{{}_\alpha R^F} = \{(a,b) \in {}_\alpha Val^2 \mid \exists_{i\in J}\; c_i^F(a) \;\wedge\; t_i^F(a,b)\}$

—$\widehat{{}_\alpha R^C} = \{(a,b) \in {}_\alpha Val^2 \mid \exists_{i\in J}\; c_i^C(a) \;\wedge\; t_i^C(a,b)\}$

$\widehat{{}_\alpha R^F}$ and $\widehat{{}_\alpha R^C}$ are called the *computed* free and constrained transition relations respectively.

Of course, the abstract interpretations c_i^F, t_i^F and c_i^C, t_i^C should be effectively computable. The idea of abstract interpretation is that an analysis tool, when provided with the domain of abstract values and corresponding abstractions of the operators, should be able to automatically evaluate the abstract semantics ${}_\alpha\mathcal{I}(P)$ of any program P.

5.1 Abstract interpretation gives approximations

The following theorem expresses that the abstract interpretations given above can be used to compute approximations to $\mathcal{A}^M$ $(= \alpha^M(\mathcal{C}))$.

THEOREM 5.1.1. *$\widehat{\mathcal{A}^M} \trianglerighteq \mathcal{A}^M$ (i.e., ${}_\alpha\mathcal{I}(P) \trianglerighteq \alpha(\mathcal{I}(P))$).*

PROOF. We show that $\preceq$ satisfies points (1) through (4) in Definition 4.0.1. (1) follows from Lemma 3.1.2. As to points (2) and (3), observe that it easily follows from the definitions of $\widehat{{}_\alpha R^F}$ and ${}_\alpha R^C$ that $\widehat{{}_\alpha R^F}$ $\preceq$-simulates $\widehat{{}_\alpha R^F}$ and that ${}_\alpha R^C$ $\succeq$-simulates ${}_\alpha R^C$. Hence, by Lemmata 5.1.2 and 5.1.3 below, ${}_\alpha R^F$ $\preceq$-simulates $\widehat{{}_\alpha R^F}$ and $\widehat{{}_\alpha R^C}$ $\succeq$-simulates ${}_\alpha R^C$. Finally, (4) is immediate as the initial states of $\widehat{\mathcal{A}^M}$ and $\mathcal{A}^M$ are identical. □

LEMMA 5.1.2. *Let R' and R'' be transition relations over ${}_\alpha\Sigma$.*

(1) If for all $a, b \in {}_\alpha\Sigma$, $R'(a,b) \;\Rightarrow\; \exists_{b'\succeq b}\; R''(a,b')$ and R'' $\preceq$-simulates R'', then R' $\preceq$-simulates R''.

(2) If for all $a, b' \in {}_\alpha\Sigma$, $R''(a,b') \;\Rightarrow\; \exists_{b\preceq b'}\; R'(a,b)$ and R' $\succeq$-simulates R', then R'' $\succeq$-simulates R'.

Note that $R'(a,b) \;\Rightarrow\; \exists_{b'\succeq b}\; R''(a,b')$ is satisfied if $R' \subseteq R''$, and that $R''(a,b') \;\Rightarrow\; \exists_{b\preceq b'} R'(a,b)$ holds if $R'' \subseteq R'$.

LEMMA 5.1.3.

(1) ${}_\alpha R^F \subseteq \widehat{{}_\alpha R^F}$.

(2) *For all* $a, b \in {}_{\alpha}\Sigma$, $\widehat{{}_{\alpha}R^{C}}(a,b) \;\Rightarrow\; \exists_{b'' \preceq b}\; {}_{\alpha}R^{C}(a, b'')$.

PROOF.

(1) Let $a, b \in {}_{\alpha}Val$ and suppose $(a,b) \in {}_{\alpha}R^{F}$. By Definition 3.3.1 of ${}_{\alpha}R^{F}$, Definition 2.4.1 of $R^{\exists\exists}$ and Definition 5.0.1 of R, this is equivalent to $b \in \{\alpha(Y) \mid Y \in \min\{Y' \mid \exists_{\bar{v}\in\gamma(a),\bar{w}\in Y'}\; [\exists_{i\in J}\; [c_i(\bar{v}) \wedge t_i(\bar{v},\bar{w})]]\}\}$. Exchanging existential quantifiers yields the equivalent $b \in \{\alpha(Y) \mid Y \in \min\{Y' \mid \exists_{i\in J}\; [\exists_{\bar{v}\in\gamma(a),\bar{w}\in Y'}\; [c_i(\bar{v}) \wedge t_i(\bar{v},\bar{w})]]\}\}$.
Because the elements of the set $\min\{Y' \mid \exists_{i\in J}\; \ldots\}$ are singletons (see the remark below Definition 3.3.1), the subterm $Y \in \min\{Y' \mid \exists_{i\in J}\; \ldots\}$ is easily seen to be equivalent with $\exists_{i\in J}\; Y \in \min\{Y' \mid \ldots\}$. After performing this replacement, we can bring the $\exists_{i\in J}$ outside, resulting in the equivalent formula

$$\exists_{i\in J}\; [b \in \{\alpha(Y) \mid Y \in \min\{Y' \mid \exists_{\bar{v}\in\gamma(a),\bar{w}\in Y'}\; [c_i(\bar{v}) \wedge t_i(\bar{v},\bar{w})]\}\}]. \qquad (3)$$

Now this is *weakened* by distributing the innermost existential quantifier over the $\wedge$:

$$\exists_{i\in J}\; [b \in \{\alpha(Y) \mid Y \in \min\{Y' \mid \exists_{\bar{v}\in\gamma(a)}\; [c_i(\bar{v})] \wedge \exists_{\bar{v}\in\gamma(a),\bar{w}\in Y'}\; [t_i(\bar{v},\bar{w})]\}\}]. \qquad (4)$$

Because both the innermost and outermost sets do not depend on $\exists_{\bar{v}\in\gamma(a)}\; [c_i(\bar{v})]$, this conjunct may be taken out of the set brackets. Using Definition 2.4.1 of $t_i{}^{\exists\exists}$ and Definition 5.0.2 of c_i^F, t_i^F, and $\widehat{{}_{\alpha}R^{F}}$, the resulting equivalent term can then be rewritten to $(a,b) \in \widehat{{}_{\alpha}R^{F}}$.

(2) Let $a, b \in {}_{\alpha}Val$ and suppose $(a,b) \in \widehat{{}_{\alpha}R^{C}}$. By Definition 5.0.2 of $\widehat{{}_{\alpha}R^{C}}$, c_i^C and t_i^C, and Definition 2.4.1 of $t_i{}^{\forall\exists}$, this is equivalent to $\exists_{i\in J}[\forall_{\bar{v}\in\gamma(a)}\; [c_i(\bar{v})] \;\wedge\; b \in \{\alpha(Y) \mid Y \in \min\{Y' \mid \forall_{\bar{v}\in\gamma(a)}\; \exists_{\bar{w}\in Y'}\; [t_i(\bar{v},\bar{w})]\}\}]$. This expression can be rewritten to the equivalent:

$$b \in \{\alpha(Y) \mid \exists_{i\in J}\; [Y \in \min\{Y' \mid \forall_{\bar{v}\in\gamma(a)}\; \exists_{\bar{w}\in Y'}\; [c_i(\bar{v}) \wedge t_i(\bar{v},\bar{w})]\}]\} \qquad (5)$$

(define $\min\emptyset = \emptyset$ in this proof). Now consider the subexpression

$$\exists_{i\in J}\; [Y \in \min\{Y' \mid \forall_{\bar{v}\in\gamma(a)}\; \exists_{\bar{w}\in Y'}\; [c_i(\bar{v}) \wedge t_i(\bar{v},\bar{w})]\}]. \qquad (6)$$

Compare this to the expression that is obtained by pushing the $\exists_{i\in J}$ inside:

$$Y \in \min\{Y' \mid \exists_{i\in J}\; [\forall_{\bar{v}\in\gamma(a)}\; \exists_{\bar{w}\in Y'}\; [c_i(\bar{v}) \wedge t_i(\bar{v},\bar{w})]]\}. \qquad (7)$$

If Y satisfies (6), then there exists an $i \in J$ such that Y is minimal among all "$\forall\exists$-successors" of $\gamma(a)$ that correspond to action i. On the other hand, if Y satisfies (7), then Y is minimal among *all* the $\forall\exists$-successors of $\gamma(a)$, regardless of the specific i. Hence, this latter Y will be a subset of (or possibly equal to) the Y that satisfies (6). So, for each set that satisfies (6), there exists a subset of it that satisfies (7), so that if b satisfies (5), there exists $b' \preceq b$ that satisfies:

$$b' \in \{\alpha(Y) \mid Y \in \min\{Y' \mid \exists_{i\in J}\; [\forall_{\bar{v}\in\gamma(a)}\; \exists_{\bar{w}\in Y'}\; [c_i(\bar{v}) \wedge t_i(\bar{v},\bar{w})]]\}\}. \qquad (8)$$

A similar step can be made again: if b' satisfies (8), then there exists $b'' \preceq b'$ satisfying:

$$b'' \in \{\alpha(Y) \mid Y \in \min\{Y' \mid \forall_{\bar{v}\in\gamma(a)}\; \exists_{\bar{w}\in Y'}\; \exists_{i\in J}\; [c_i(\bar{v}) \wedge t_i(\bar{v},\bar{w})]\}\} \qquad (9)$$

which is, by Definition 5.0.1 of R, Definition 2.4.1 of $R^{\forall\exists}$ and Definition 3.3.1 of ${}_{\alpha}R^{C}$, equivalent to ${}_{\alpha}R^{C}(a, b'')$.

□

5.2 Optimal abstract interpretations

Construction of an abstract model by abstract interpretation of the "elementary" operations (the c_i and t_i) occurring in a program is a natural thing to do — it resembles the way abstractions are computed in traditional applications of Abstract Interpretation. However, the computed abstract models (Definition 5.0.2) are, in general, less precise than the optimal abstractions of Definition 3.3.1. How much precision is lost exactly depends on the program to be analysed and the choice of the abstract domain. In order to get some insight, we discuss two approaches to obtain optimality. Firstly, we derive sufficient conditions on the abstract domain (and program) for the computed abstract models of Definition 5.0.2 to be optimal (i.e. equal to $\alpha^M(\mathcal{C})$). Secondly, we briefly sketch how, alternatively, the abstract interpretation of programs may be adapted in such a way that computed models are optimal.

Conditions on the abstract domain. In order to pinpoint the reasons why the computed abstractions are not optimal, we analyse the proof of Lemma 5.1.3. In part 1, concerning the free abstraction, the only place where the formula being manipulated is (strictly) weakened, is when the term $\exists_{\bar{v}\in\gamma(a),\bar{w}\in Y'}\ [c_i(\bar{v})\wedge t_i(\bar{v},\bar{w})]$ (T1) is replaced by $\exists_{\bar{v}\in\gamma(a)}\ [c_i(\bar{v})] \wedge \exists_{\bar{v}\in\gamma(a),\bar{w}\in Y'}\ [t_i(\bar{v},\bar{w})]$ (T2).The following small example illustrates what happens. Suppose that the concrete state space consists of a single integer variable v, and that the abstract domain contains values e and o, being descriptions of the even and the odd numbers respectively. Assume that P contains as action i: $v = 4 \;\rightarrow\; v := v/4$ (specifying $c_i(v)$ to be $v = 4$ and $t_i(v, w)$ to be $w = v/4$). Then (T1), with e for a and $\gamma(\mathsf{e})$ for Y', does not hold. On the other hand, (T2) *does* hold: there exists an even number that is equal to 4 and there exists a (different) even number that, when divided by 4, yields an even number. In order to enforce equivalence of (T1) and (T2), we can impose a condition on the abstract states. For an abstract state a, this condition intuitively requires that the concrete states in $\gamma(a)$ behave uniformly with respect to every condition c_i.

Lemma 5.2.1. *Let $\widehat{\mathcal{A}^M} = {}_{\alpha}\mathcal{I}(P)$ be the abstract model computed according to Definition 5.0.2, and let $a \in {}_{\alpha}\Sigma$. If for every $i \in J$, we have either $\forall_{\bar{v}\in\gamma(a)}\ c_i(\bar{v})$ or $\forall_{\bar{v}\in\gamma(a)}\ \neg c_i(\bar{v})$, then every outgoing ${}_{\alpha}\widehat{R^F}$-transition of a is in ${}_{\alpha}R^F$.*

Proof. The precondition of the lemma is easily seen to imply equivalence of the terms T1 and T2 and hence of the formulae (3) and (4) in the proof of Lemma 5.1.3. The conclusion then follows directly. □

So, under the given condition, all outgoing free transitions of state a are optimal. Clearly, the condition is very strong for "$\preceq$-large" states. E.g. if $a = \top$, then it is only satisfied if all program conditions c_i are either tautologies or unsatisfiable. However, as far as properties $\varphi \in \Box L_\mu$ are concerned, it is sufficient to require the condition to hold only for those states on which φ depends. In that case, the result of model checking φ over the computed model $\widehat{\mathcal{A}^M}$ will be the same as when checking it over the optimal model $\mathcal{A}^M$. As a consequence, unreachable states may be ignored alltogether. As to the reachable states, observe that only the *atoms* of the abstract domain, i.e. the elements $\{\alpha(\{c\}) \mid c \in \Sigma\}$, can be reachable via a free transition. This follows from the observation, below Definition 3.3.1, that α is only applied to singletons. Hence, we should preferably choose the abstract domain in such a way that these atoms are $\preceq$-small[13]. However, if φ is a subformula of a formula that contains $\Diamond$'s, then also certain states that are reachable via constrained transitions may have to satisfy the condition of Lemma 5.2.1.

Although sufficient, the condition required in Lemma 5.2.1 is not necessary. However, it is a reasonable condition that can be checked rather easily: one has to check that for each atomic abstract state a and each condition c_i of the program, either "$a \Rightarrow c_i$" or "$a \cap c_i = \emptyset$". For instance, in the example above, "being even" neither implies nor excludes "being equal to 4", so the condition is not met. The condition also gives a deeper insight in how to design "good" abstract domains given a program(ming language).

For the constrained relation, we analyse part 2 of the proof of Lemma 5.1.3. The last two steps in this proof introduce the differences between $\widehat{{}_\alpha R^C}$ and ${}_\alpha R^C$. We consider these steps in reverse direction, going from ${}_\alpha R^C$ to $\widehat{{}_\alpha R^C}$. While in formula (9) the "$\forall\exists$-successors" Y' of $\gamma(a)$ are taken relative to transitions via *any action* (i.e. all states in $\gamma(a)$ must be able to make a transition to some state in Y' via no matter which action i), the $\forall\exists$-successors Y' of $\gamma(a)$ in (8) are taken "per action", i.e. for a single action $i \in J$, all states in $\gamma(a)$ must be able to make a transition to some state in Y' *via action* i. This means that in the latter case, certain $\forall\exists$-successors Y' may be "missed" and consequently, $\widehat{{}_\alpha R^C}$ may contain fewer transitions than ${}_\alpha R^C$. However, note that if for such a transition, say from a to b, which is in ${}_\alpha R^C$ but not in $\widehat{{}_\alpha R^C}$, there exists another transition in $\widehat{{}_\alpha R^C}$ from a to a more precise state $b' \preceq b$, this loss does not matter: $\widehat{{}_\alpha R^C}$ will not be less precise (in the sense of Definition 4.0.1) than ${}_\alpha R^C$ because of this. It is this observation on which the condition in Lemma 5.2.2 below is based.

Now consider the step from (8) to (5) — more precisely, the replacement of subformula (7) by (6). In (7), the minimality of Y is determined globally over all actions, while in (6) all Y's that are minimal relative to a single action $i \in J$ are taken. As a result, the set of successors under $\widehat{{}_\alpha R^C}$ of some abstract state a may be a superset of its successors under ${}_\alpha R^C$ (the fact that $\widehat{{}_\alpha R^C}$ is not strictly *more* precise than ${}_\alpha R^C$ is explained by observing that for each such extra successor b' under $\widehat{{}_\alpha R^C}$ there will be a more precise successor $b \preceq b'$ under ${}_\alpha R^C$). Hence, this effect does not negatively affect the precision of $\widehat{{}_\alpha R^C}$ w.r.t. ${}_\alpha R^C$.

LEMMA 5.2.2. *Let $a \in {}_\alpha\Sigma$ and suppose that both of the following conditions hold:*

(1) For every $i \in J$, $\forall_{\bar{v} \in \gamma(a)}\ c_i(\bar{v})$ or $\forall_{\bar{v} \in \gamma(a)}\ \neg c_i(\bar{v})$.

(2) For all $i, j \in J$ with $i \neq j$ and $b_i, b_j \in {}_\alpha\Sigma$ with $c_i^C(a) \wedge t_i^C(a, b_i)$ and $c_j^C(a) \wedge t_j^C(a, b_j)$: for every $b \in {}_\alpha\Sigma$ with $\gamma(b_i) \cap \gamma(b) \neq \emptyset$ and $\gamma(b_j) \cap \gamma(b) \neq \emptyset$, there

[13] See [Cousot and Cousot 1979] for a variety of techniques for the construction of suitable abstract domains.

exist $k \in J$ and $b_k \in {}_\alpha\Sigma$ with $c_k^C(a) \wedge t_k^C(a, b_k)$ such that $\gamma(b_k) \subseteq \gamma(b)$.

Then for every $b' \in {}_\alpha\Sigma$, ${}_\alpha R^C(a, b') \Rightarrow \exists_{b \preceq b'}\ \widehat{{}_\alpha R^C}(a, b)$.

Note that condition (1) in this lemma is similar to the condition in Lemma 5.2.1 above. Condition (2) specifies that two abstract successors b_i and b_j of a corresponding to different actions ($i \neq j$) may only both be (partially or completely) overlapped by a third state b if b completely overlaps some successor b_k (possibly $k = i$ or $k = j$) of a.

Proof of Lemma 5.2.2. It may be helpful to realise that we are, roughly speaking, trying to reverse the direction of the argument in point (2) of the proof of Lemma 5.1.3. Let $b' \in {}_\alpha\Sigma$ and assume ${}_\alpha R^C(a, b')$. By Definition 3.3.1 of ${}_\alpha R^C$, Definition 2.4.1 of $R^{\forall\exists}$ and Definition 5.0.1 of R, this is equivalent to saying that b' is an element of

$$\{\alpha(Y) \mid Y \in \min\{Y' \mid \forall_{\bar{v} \in \gamma(a)}\ \exists_{\bar{w} \in Y'}\ \exists_{i \in J}\ [c_i(\bar{v}) \wedge t_i(\bar{v}, \bar{w})]\}\}. \tag{10}$$

Next, consider the set that is obtained by taking the $\exists_{i \in J}$ outside of the scope of $\forall_{\bar{v} \in \gamma(a)} \exists_{\bar{w} \in Y'}$:

$$\{\alpha(Y) \mid Y \in \min\{Y' \mid \exists_{i \in J}\ [\forall_{\bar{v} \in \gamma(a)}\ \exists_{\bar{w} \in Y'}\ [c_i(\bar{v}) \wedge t_i(\bar{v}, \bar{w})]]\}\}. \tag{11}$$

We consider two cases. If b' is an element of (11), then we proceed as follows. The subexpression

$$Y \in \min\{Y' \mid \exists_{i \in J}\ [\forall_{\bar{v} \in \gamma(a)}\ \exists_{\bar{w} \in Y'}\ [c_i(\bar{v}) \wedge t_i(\bar{v}, \bar{w})]]\} \tag{12}$$

of (11) is *weakened* by bringing the $\exists_{i \in J}$ outside:

$$\exists_{i \in J}\ [Y \in \min\{Y' \mid \forall_{\bar{v} \in \gamma(a)}\ \exists_{\bar{w} \in Y'}\ [c_i(\bar{v}) \wedge t_i(\bar{v}, \bar{w})]\}]. \tag{13}$$

Therefore, b' is also an element of the set obtained by replacing subexpression (12) of (11) by (13), resulting in the set

$$\{\alpha(Y) \mid \exists_{i \in J}\ [Y \in \min\{Y' \mid \forall_{\bar{v} \in \gamma(a)}\ \exists_{\bar{w} \in Y'}\ [c_i(\bar{v}) \wedge t_i(\bar{v}, \bar{w})]\}]\}. \tag{14}$$

Similar to the first two steps in point (2) of the proof of Lemma 5.1.3, but in reverse order, this implies that $(a, b') \in \widehat{{}_\alpha R^C}$.

The other case is that b' is not in (11). Let $Z \subseteq \Sigma$ be such that $b' = \alpha(Z)$. Because b' is in (10) but not in (11), we can choose $i, j \in J$ with $i \neq j$, $\bar{v}_i, \bar{v}_j \in \gamma(a)$ and $\bar{w}_i, \bar{w}_j \in Z$ such that $c_i(\bar{v}_i) \wedge t_i(\bar{v}_i, \bar{w}_i)$ and $c_j(\bar{v}_j) \wedge t_j(\bar{v}_j, \bar{w}_j)$. Because, by condition (1) of the lemma, $\forall_{\bar{v} \in \gamma(a)}\ c_i(\bar{v})$ and $\forall_{\bar{v} \in \gamma(a)}\ c_j(\bar{v})$, we can also choose $Z_i, Z_j \subseteq \Sigma$ such that $\bar{w}_i \in Z_i$ and $\bar{w}_j \in Z_j$. Hence, $\alpha(Z)$ must have a non-empty intersection with both $\alpha(Z_i)$ and with $\alpha(Z_j)$. Condition (2) of the lemma then requires that there exist $k \in J$ and $b_k \in {}_\alpha\Sigma$ with $c_k^C(a) \wedge t_k^C(a, b_k)$ (and therefore $(a, b_k) \in \widehat{{}_\alpha R^C}$) such that $b_k \preceq \alpha(Z)$, i.e. $b_k \preceq b'$. □

Again, for a formula $\varphi \in \Diamond L_\mu$ being checked, it suffices to impose the conditions of this lemma only on those states on which the interpretation of φ depends. If φ contains □-operators as well, then the condition of Lemma 5.2.1 should hold in those states on whose outgoing free transitions φ depends and the conditions of Lemma 5.2.2 should hold in those states on whose outgoing constrained transitions

φ depends. In that case, model checking φ over the computed mixed abstraction $\widehat{\mathcal{A}^M}$ of Definition 5.0.2 gives the same result as checking it over the optimal $\mathcal{A}^M$.

Adapting the abstract interpretation. Instead of imposing conditions guaranteeing optimality of abstract models computed as specified by Definition 5.0.2, we may change the definition of these abstract interpretations themselves in such a way that the "loss" of Lemma 5.1.3 does not occur. For the free abstract interpretation, this means that it may not be distributed over the individual condition and transformation parts of an action. In case of the example given above, this would mean that an abstract interpretation act_i^F has to be provided for the action $act_i(v,w) \Leftrightarrow v = 4 \wedge w = v/4$ as a whole, satisfying $act_i^F(a,b) \Leftrightarrow \exists_{v \in \gamma(a)} [c_i(v) \wedge b \in \{\alpha(Y) \mid Y \in \min\{Y' \mid t_i{}^{\exists\exists}(\{v\}, Y')\}\}]$.

In case of the constrained abstract interpretations, loss of optimality already occurs at the point where they are distributed over the individual actions of a program. Here, the adaptation would require the generalisation of the abstract interpretation of actions by taking into account the effect of executing an arbitrary number of actions "at the same time" by defining $act_{\{i_1,\dots,i_k\}}(a,b) \Leftrightarrow \exists_{a_1,\dots,a_k,b_1,\dots,b_k \in {}_\alpha\Sigma}$ $[a_1 \vee \cdots \vee a_k = a, b_1 \vee \cdots \vee b_k = b, \forall_{j \in \{1,\cdots,k\}}\ c_{i_j}^C(a_j) \ \wedge\ t_{i_j}^C(a_j, b_j)]$ for subsets $\{i_1, \dots, i_k\}$ of J ($\vee$ denotes the least upper bound on ${}_\alpha\Sigma$). This approach corresponds to the *merge over all paths analysis* of Cousot and Cousot [1979].

5.3 Computing approximations

One may choose to compute non-optimal abstractions by specifying approximations to the abstract interpretations of the c_i and t_i. A reason for doing so may be that the computation of optimal abstract interpretations is too complex, when the c_i and t_i involve intricate operations for example. In that case, even if the abstract interpretations are optimal, it may be cumbersome to actually prove so, and one may settle for proving approximation without bothering about optimality.

Definition 5.3.1. The definition of approximation is extended to abstract interpretations of the transformation operators, as follows. For abstract operations[14] $t, \overline{t} \in {}_\alpha Val \times {}_\alpha Val$,

$$\overline{t} \succeq t \quad\Leftrightarrow\quad \forall_{a,b,\overline{b}\ \in\ {}_\alpha Val} \left[t(a,b) \Rightarrow \exists_{\overline{b} \succeq b}\ \overline{t}(a,\overline{b})\right] \quad\wedge\quad \left[\overline{t}(a,\overline{b}) \Rightarrow \exists_{b \preceq \overline{b}}\ t(a,b)\right]$$

Approximations $\overline{t_i^F} \succeq t_i^F$ and $\overline{t_i^C} \succeq t_i^C$ (for every $i \in J$) to the free and constrained interpretations (see Definition 5.0.2) induce the abstract model $\overline{\mathcal{A}^M} = ({}_\alpha\Sigma, {}_\alpha I, \overline{{}_\alpha R^F}, \overline{{}_\alpha R^C})$, where:

—${}_\alpha\Sigma = {}_\alpha Val$

—${}_\alpha I = \{\alpha(\bar{v}) \mid \bar{v} \in IVal\}$

—$\overline{{}_\alpha R^F} = \{(a,b) \in {}_\alpha Val^2 \mid \exists_{i \in I}\ c_i^F(a) \ \wedge\ \overline{t_i^F}(a,b)\}$

—$\overline{{}_\alpha R^C} = \{(a,b) \in {}_\alpha Val^2 \mid \exists_{i \in I}\ c_i^C(a) \ \wedge\ \overline{t_i^C}(a,b)\}$

Lemma 5.3.2. $\overline{\mathcal{A}^M} \trianglerighteq \widehat{\mathcal{A}^M}$.

[14] Recall that such "operations" are binary relations.

Proof. We show that $\preceq$ is a mixed simulation from $\widehat{\mathcal{A}^M}$ to $\overline{\mathcal{A}^M}$. The non-trivial parts are to show that (1) $\widehat{{}_\alpha R^F}$ $\preceq$-simulates $\overline{{}_\alpha R^F}$ and (2) $\overline{{}_\alpha R^C}$ $\succeq$-simulates $\widehat{{}_\alpha R^C}$.

(1) Let $a, a_1, \overline{a} \in {}_\alpha\Sigma$ with $\widehat{{}_\alpha R^F}(a, a_1)$ and $\overline{a} \succeq a$. We show that there exists $\overline{a}_1 \succeq a_1$ such that $\overline{{}_\alpha R^F}(\overline{a}, \overline{a}_1)$. By Definition 5.0.2 of $\widehat{{}_\alpha R^F}$, $\widehat{{}_\alpha R^F}(a, a_1)$ equivales $\exists_{i \in I}\ [c_i^F(a) \wedge t_i^F(a, a_1)]$. Because $\overline{a} \succeq a$, we have $c_i^F(a) \Rightarrow c_i^F(\overline{a})$ and also $t_i^F(a, a_1) \Rightarrow t_i^F(\overline{a}, a_1)$ (see Definition 5.0.2 of c_i^F and t_i^F and Definition 2.4.1 of $\cdot^{\exists\exists}$), so $\exists_{i \in I}\ [c_i^F(\overline{a}) \wedge t_i^F(\overline{a}, a_1)]$. By Definition 5.3.1 of $\overline{t_i^F}$, there exists $\overline{a}_1 \succeq a_1$ such that $\exists_{i \in I}\ [c_i^F(\overline{a}) \wedge \overline{t_i^F}(\overline{a}, \overline{a}_1)]$, i.e. $\overline{{}_\alpha R^F}(\overline{a}, \overline{a}_1)$.

(2) Let $a, a_1, a' \in {}_\alpha\Sigma$ with $\overline{{}_\alpha R^C}(a, a_1)$ and $a' \preceq a$. We show that there exists $a'_1 \preceq a_1$ such that $\widehat{{}_\alpha R^C}(a', a'_1)$. We have $\overline{{}_\alpha R^C}(a, a_1)$. By Definition 5.3.1 of $\overline{{}_\alpha R^C}$ and $\overline{t_i^C}$, there exists $a''_1 \preceq a_1$ such that $\exists_{i \in I}\ [c_i^C(a) \wedge t_i^C(a, a''_1)]$. Because $a' \preceq a$, we have $c_i^C(a) \Rightarrow c_i^C(a')$, and also there must be some $a'_1 \preceq a''_1$ such that $t_i^C(a', a'_1)$ (see Definition 5.0.2 of c_i^C and t_i^C and Definition 2.4.1 of $\cdot^{\forall\exists}$). So $\exists_{i \in I}\ [c_i^C(a') \wedge t_i^C(a', a'_1)]$, i.e., $\widehat{{}_\alpha R^C}(a', a'_1)$, and, by transitivity of $\preceq$, $a'_1 \preceq a_1$.

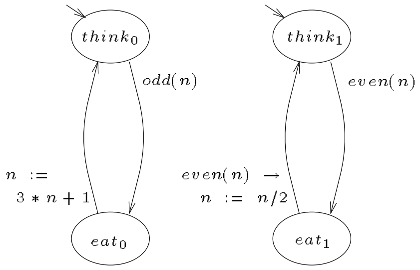
□

5.4 Practical application

The use of abstract interpretation to model check a property φ for a program P is characterised by the following phases. First, an abstract domain ${}_\alpha Val$ has to be chosen and for all operation symbols occurring in P, abstract interpretations have to be provided. Typically, the tests and transformations are defined in terms of more elementary operations, in which case abstract interpretations may be provided for these. Depending on the property φ to be checked, free and/or constrained interpretations should be given; these have to satisfy Definition 5.0.2. Then, the abstract model can be constructed by a symbolic evaluation of the program over the abstract domain, interpreting the operations according to their abstract interpretations. Finally, φ is model checked over the abstract model under the adapted definition of satisfaction (Definition 3.3.2). It is important to notice that only *positive* results of this model checking carry over to the concrete model: because the $\Box$ and $\Diamond$ next-state operators of φ are interpreted along different kinds of transitions of the abstract model, a negative result $\mathcal{A} \not\models \varphi$ does *not* imply that $\mathcal{A} \models \neg\varphi$ and hence does not justify the conclusion that $\mathcal{C} \models \neg\varphi$, in spite of the fact that $\neg\varphi$ is (an abbreviation of) a L_μ formula. However, it may be possible to resolve such a negative answer for φ by checking the negation $\neg\varphi$. If *true* is returned, then we know that $\mathcal{A} \models \neg\varphi$ and hence $\mathcal{C} \models \neg\varphi$, i.e. $\mathcal{C} \not\models \varphi$. As $\neg\varphi$ contains the dual[15] operators of those in φ, its satisfaction by the mixed Kripke structure $\mathcal{A}$ may depend on different paths — in particular, it may not hold either. So, whether this "trick" to resolve negative answers is successful, depends on how the dual abstractions (in the sense of free vs. constrained) are chosen. We do not further investigate this point here; the interested reader is referred to Kelb et al. [1995].

The same idea of constructing an abstract model by abstract interpretation of program operations, although based on a different theoretical framework [Loiseaux

[15] Recall that $\neg\varphi$ is the abbreviation of a L_μ formula in negation-normal form.



(a)

$$\begin{array}{c} \ell_0 = think,\ odd(n) \longrightarrow \ell_0 := eat \\ \ell_0 = eat \longrightarrow \ell_0 := think,\ n := 3 * n + 1 \\[6pt] \ell_1 = think,\ even(n) \longrightarrow \ell_1 := eat \\ \ell_1 = eat,\ even(n) \longrightarrow \ell_1 := think,\ n := n/2 \end{array}$$

(b)

Fig. 1. The dining mathematicians (a); expressed as an action system (b).

et al. 1995; Loiseaux 1994] (see Section 8 for a comparison), is applied to a "real-life" example in Graf [1994]. Graf shows in that paper how a distributed cache memory, which is in principle an infinite state system because request queues are unbounded, can be verified by providing a finite abstract domain and corresponding abstract operations.

Although the model checking procedure itself is an automated process, it is not obvious how the choice of an appropriate abstract domain with corresponding abstract operations, as well as the proofs that these operations satisfy the conditions of Definition 5.0.2, can be performed in an automated fashion. So far we have assumed that the abstract domain is provided by the user of the method; an example of this may be found in Graf [1994]. In Loiseaux [1994], the process is automated, but this is only possible because the concrete domain is required to be finite. The proofs for the abstract operators may form a difficult step in the method. In Kelb et al. [1995], approximations to the transition relation of StateCharts [Harel 1987] are used to verify μ-calculus properties of a production cell [Damm et al. 1995] in a compositional fashion. In Dams et al. [1993] and Dams et al. [1994], a method is developed that aims at full automation of these steps. However, it is potentially more expensive than to evaluate properties over the concrete domain.

6. EXAMPLE

In this section we illustrate the theory on a small example. Consider the system consisting of two concurrent processes depicted in Figure 1(a), which is a parallel variant of the Collatz $3n + 1$ program. We chose this example because it is small but nevertheless displays a non-trivial interplay between data and control. The properties that we will verify concern certain control aspects that depend on the values that the integer variable n takes under the various operations that are performed on it. Because the state space is infinite, data-abstraction will be necessary in order to verify aspects of the control-flow. It serves as an illustration of the fact that abstraction techniques bring into reach the model checking of systems that cannot be verified through the standard approach.

The program may be viewed as a protocol controlling the mutually exclusive access to a common resource of two concurrent processes, modelling the behaviour of two mathematicians, numbered 0 and 1. They both cycle through an infinite sequence of "think" and "eat" states. The right to enjoy a meal in strict solitude is regulated by having them inspect the value of n before eating, letting the one go ahead only if n has an odd value, and the other only if n is even. Upon exit from the dining room, each mathematician has its own procedure for assigning a new value to n. Transitions can only be taken when the enabling conditions are satisfied, e.g., mathematician 1 can only leave the dining room if n is divisible by 2. An execution is a maximal sequence of (arbitrarily) interleaved steps of both processes that starts in a state where both mathematicians are in their thinking state, and n is set to some arbitrary positive integer value. We want to verify mutual exclusion and the absence of individual starvation along every execution. In order to formalise this, we first express the program as an action system[16]: see Figure 1(b). As data and control are treated uniformly in such systems, we introduce variables ℓ_0 and ℓ_1, both ranging over $\{think, eat\}$, to encode the effect of "being in a location" $think_i$ or eat_i. The state space Σ of this program is the set $\{think, eat\}^2 \times \mathbb{N}\backslash\{0\}$ of values that the vector $\langle \ell_0, \ell_1, n\rangle$ of program variables may assume. The initial states are $I = \{\langle think, think, n\rangle \mid n \in \mathbb{N}\backslash\{0\}\}$. The transitions are defined as in Definition 5.0.1, using the standard interpretations of the tests $=$, *even*, *odd* and operations $3*$, $+1$ and $/2$ (the latter three are considered as operations on one argument, i.e. functional binary relations, where $/2$ is assumed to be defined for even numbers only).

The properties to be verified are expressed in L_μ as follows.

$$\nu x.(\neg(\ell_0 = eat \wedge \ell_1 = eat) \wedge \Box x) \tag{15}$$

$$\nu x.(\ell_0 = eat \rightarrow (\mu y.(\ell_1 = eat \vee (\Diamond true \wedge \Box y))) \wedge \Box x) \tag{16}$$

$$\nu x.(\ell_1 = eat \rightarrow (\mu y.(\ell_0 = eat \vee (\Diamond true \wedge \Box y))) \wedge \Box x) \tag{17}$$

Formula (15) says that the property $\neg(\ell_0 = eat \wedge \ell_1 = eat)$ holds in every state along every execution. (16) expresses that whenever a state is reached in which $\ell_0 = eat$ holds, then along every continuation, there must eventually be a state in which $\ell_1 = eat$; (17) says the same with ℓ_0 and ℓ_1 reversed. The first formula is in $\Box L_\mu$, while the other two are in neither fragment of L_μ. However, we will see later in this section that the program is deadlock free, so that by Lemma 2.2.2, the $\Diamond true$ conjuncts may be dropped from the formulae. The resulting formulae are contained in $\Box L_\mu$, so that we can verify them via a free abstraction.

The abstract domain is defined by providing abstractions of the components that comprise the concrete domain. We choose to leave the components $\{think, eat\}$ the same. Formally, this means that we take an abstract domain containing elements think and eat whose concretisations are $\{think\}$ and $\{eat\}$ respectively. To abstract $\mathbb{N}\backslash\{0\}$, we choose an abstract domain in which n may take the values e and o, describing the even and odd positive integers respectively, i.e. $\gamma(\mathsf{e}) = \{2, 4, 6, \ldots\}$ and $\gamma(\mathsf{o}) = \{1, 3, 5\ldots\}$. To both abstract domains, we add a top element $\top$. The set ${}_\alpha\Sigma$ of abstract states is thus defined as follows.

$${}_\alpha\Sigma = \{\mathsf{think}, \mathsf{eat}, \top\}^2 \times \{\mathsf{e}, \mathsf{o}, \top\}$$

[16]We use the more operational notion of assignment, :=, rather than the primed variables of Definition 5.0.1.

FREE:	(e, e)	(e, o)	(e, ⊤)	(o, e)	(o, o)	(o, ⊤)	(⊤, e)	(⊤, o)	(⊤, ⊤)
$3*^F$	*true*	*false*	*false*	*false*	*true*	*false*	*true*	*true*	*false*
$+1^F$	*false*	*true*	*false*	*true*	*false*	*false*	*true*	*true*	*false*
$/2^F$	*true*	*true*	*false*	*false*	*false*	*false*	*true*	*true*	*false*

(a)

FREE:	e	o	⊤
$even^F$	*true*	*false*	*true*
odd^F	*false*	*true*	*true*

(b)

CONSTR.:	(o, 100)	(o, e)	(o, o)	(o, ⊤)
$3*^C$	*false*	*false*	*true*	*false*
$+1^C$	*false*	*true*	*false*	*false*
$/2^C$	*false*	*false*	*false*	*false*

(c)

CONSTR.:	100	e	o	⊤
$even^C$	*true*	*true*	*false*	*false*
odd^C	*false*	*false*	*true*	*false*

(d)

CONSTR.:	think	eat	⊤
$= think^C$	*true*	*false*	*false*
$= eat^C$	*false*	*true*	*false*

(e)

Fig. 2. Free abstract interpretations of operations (a) and some of the tests (b). Constrained abstract interpretations of some operations (c) and tests (d and e).

Its top element is $\langle \top, \top, \top \rangle$, while the approximation relation $\preceq$ is the obvious extension of the orderings on each of the three components. It is easily verified that the concretisation function thus defined determines a Galois insertion from $(\mathcal{P}(\Sigma), \subseteq)$ to $({}_{\alpha}\Sigma, \preceq)$. The valuations of the propositions $\ell_0 = eat$ and $\ell_1 = eat$ over the abstract states, given by Definition 3.1.1, are as expected. For the abstract initial states we have, according to Definition 3.2.1:

$$ {}_{\alpha}I = \{\langle \mathsf{think}, \mathsf{think}, \mathsf{e} \rangle, \langle \mathsf{think}, \mathsf{think}, \mathsf{o} \rangle\} $$

Having chosen an abstract domain, we also have to provide abstract interpretations, over this domain, of the operations that appear in the program, along the lines of Definition 5.0.2. Tables (a) and (b) in Figure 2 give the definitions of the free abstract interpretations of some of the transformations and tests on the abstract domain $\{\mathsf{e}, \mathsf{o}, \top\}$. The operations $3*$, $+1$ and $/2$ are considered single symbols. The tables have to be interpreted as indicated by the following examples. The entry *true* in Table (b), row $even^F$, column e, indicates that $even^F(\mathsf{e})$ holds, i.e. (cf. Definition 5.0.2) $\exists_{n \in \gamma(\mathsf{e})}\, even(n)$. The entry *false* in Table (a), row $+1^F$, column (e, e), means that $+1^F(\mathsf{e}, \mathsf{e})$ is false, i.e. for any minimal Y such that $+1^{\exists\exists}(\gamma(\mathsf{e}), Y)$, we have $\alpha(Y) \neq \mathsf{e}$ (see Definitions 5.0.2 and 2.4.1). From these diagrams we see for example that $/2^F$ is not functional (Table (a), row $/2^F$, first two columns, as well as the columns for $(\top, \mathsf{e})$ and $(\top, \mathsf{o})$), illustrating that a function may become a relation when abstracted. The abstract interpretation of the composed operation $3 * ... + 1$ that occurs as a transformation in the program, is now obtained by composition of the abstract interpretations of the constituents. The tables (c)–(e) are explained below.

Now we can abstractly interpret the program over this abstract domain, using the interpretations given in the tables. Such an abstract execution yields the abstract

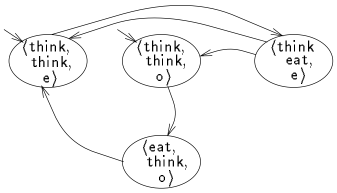


Fig. 3. The free abstract model.

model of Figure 3. In this model, only those states are shown that are reachable along the computed free transition relation. We see that in no reachable state the property $\ell_0 = \mathsf{eat} \wedge \ell_1 = \mathsf{eat}$ holds. Hence we have established property (15). Furthermore, every path from the state where $\ell_0 = \mathsf{eat}$, reaches $\ell_1 = \mathsf{eat}$ within 2 steps, so we have also verified property (16).

However, the abstraction does not allow verification of the other non-starvation property, (17): a counterexample in the abstract model is the path cycling infinitely between $\langle \mathsf{think}, \mathsf{think}, \mathsf{e} \rangle$ and $\langle \mathsf{think}, \mathsf{eat}, \mathsf{e} \rangle$. It turns out that the negation of property (17) can also not be established via the constrained transition relation. So, only refinement of the abstract domain may bring the answer. In this case, the abstract states where $n = \mathsf{e}$ would have to be unraveled into infinitely many states representing the cases where n is divisible by 4, by 8, by 16, ... Hence, with our methodology, it is impossible to verify property (17) through a finite abstraction.

Nevertheless, an interesting question is how the refinement of an abstract model, in order to decide indeterminate results, can be computed. Ongoing work, which concentrates on including fairness constraints in the abstract models, has meanwhile yielded results that enable the verification of property (17) via a finite abstraction.

It is easy to check that for each condition c_i of the action system in Figure 1 (b) and each abstract state a of the system of Figure 3, either c_i evaluates to true in all concrete states in a's concretisation, or it evaluates to false in all those states. This implies by Lemma 5.2.1 that (the reachable part of) the computed free abstraction coincides with (the reachable part of) the optimal free abstraction as defined by Definition 3.3.1.

In order to illustrate the use of the constrained abstraction, we consider a small extension to the program: we add a third concurrent process that can "restart" the system by setting n to value 100. This may only be done when both mathematicians are thinking, otherwise there may be executions possible that violate the mutual exclusion property. To this effect, the following fifth action is added to the program:

$$\ell_0 = think,\ \ell_1 = think \ \longrightarrow\ n := 100$$

We want to check whether along every possible execution path, in every state there is a possible continuation that will eventually reach a "restart" state. Writing $restart$ for $\ell_0 = think \wedge \ell_1 = think \wedge n = 100$, this property is expressed in L_μ by:

$$\nu x.(\mu y.(restart \vee \Diamond y) \wedge \Box x) \tag{18}$$

We extend the abstract domain for n by the value $\mathsf{100}$, where $\gamma(\mathsf{100}) = \{100\}$. Formula (18) not being in $\Box L_\mu$ or $\Diamond L_\mu$, we need a mixed abstraction. The tables (c)–(e) in Figure 2 provide some of the entries that will be needed in an abstract execution of the program. The other tables from Figure 2 have to be extended in

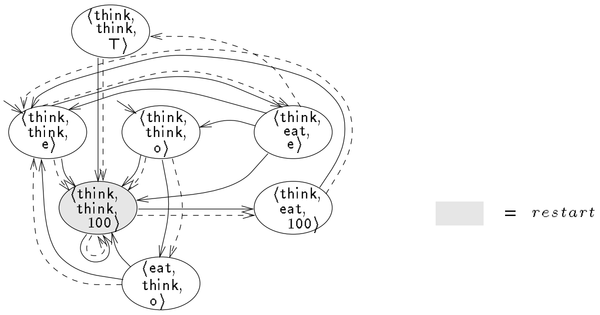


Fig. 4. The mixed abstract model for the modified program.

order to take into account the new abstract value $\mathsf{100}$. Being straightforward, these extensions are left to the reader.

The resulting abstraction is depicted in Figure 4. Solid arrows denote free transitions, dashed arrows represent constrained transitions. Not all reachable states are shown. The complete model would include 10 more states, which are only reachable via free transitions starting from $\langle\mathsf{think},\mathsf{think},\top\rangle$. However, as the formula to be checked does not depend on these, they have been omitted. On the other hand, the presence of the state $\langle\mathsf{think},\mathsf{think},\top\rangle$, though not reachable via free transitions alone, *is* essential in proving the property. Note that it is not in general the case that ${}_{\alpha}R^C \subseteq {}_{\alpha}R^F$, as is illustrated by the arrow from $\langle\mathsf{think},\mathsf{eat},\mathsf{e}\rangle$ to $\langle\mathsf{think},\mathsf{think},\top\rangle$.

Property (18) is verified on this model, interpreting the $\Box$ modalities along the free transitions, and the $\Diamond$s along the constrained transitions. It can easily be seen that (18) holds, hence, we have established its validity in the concrete program. Also, we can see from this abstraction that the program is deadlock free, as it satisfies the formula $\nu x.(\Diamond true \wedge \Box x)$ which expresses that there exists a successor in every reachable state.

In order to check optimality of transitions in the model of Figure 4, we verify the preconditions of Lemmata 5.2.1 and 5.2.2 for its states. Earlier, we checked the precondition of Lemma 5.2.1 for those states that already occur in the free model of Figure 3. The concretisations of the states $\langle\mathsf{think},\mathsf{think},\mathsf{100}\rangle$ and $\langle\mathsf{think},\mathsf{eat},\mathsf{100}\rangle$ are singletons, from which it directly follows that also for these states, the precondition of Lemma 5.2.1 is satisfied, whence their outgoing free transitions are optimal. For $\langle\mathsf{think},\mathsf{think},\top\rangle$, the precondition of Lemma 5.2.1 does not hold. Indeed, the free transitions starting from this state that are not shown in the figure, are not optimal. Those optimal free edges would lead back to $\langle\mathsf{think},\mathsf{eat},\mathsf{e}\rangle$ and to $\langle\mathsf{eat},\mathsf{think},\mathsf{o}\rangle$.

Next, we check the constrained transitions. As condition (1) of Lemma 5.2.2 is the same as in Lemma 5.2.1, it remains to check condition (2). The only states that have two different (constrained) successors (corresponding to the b_i and b_j in Lemma 5.2.2) are $\langle\mathsf{think},\mathsf{think},\mathsf{e}\rangle$, $\langle\mathsf{think},\mathsf{think},\mathsf{o}\rangle$ and $\langle\mathsf{think},\mathsf{think},\mathsf{100}\rangle$. The successors of $\langle\mathsf{think},\mathsf{think},\mathsf{e}\rangle$ are $\langle\mathsf{think},\mathsf{eat},\mathsf{e}\rangle$ and $\langle\mathsf{think},\mathsf{think},\mathsf{100}\rangle$. The states in ${}_{\alpha}\Sigma$ (corresponding to the b in Lemma 5.2.2) that (partly) overlap both of these are $\langle\mathsf{think},\top,\mathsf{100}\rangle$ and all states that are $\preceq$-greater than it. Because $\langle\mathsf{think},\top,\mathsf{100}\rangle$ completely overlaps a successor of $\langle\mathsf{think},\mathsf{think},\mathsf{e}\rangle$, namely $\langle\mathsf{think},\mathsf{think},\mathsf{100}\rangle$ (b_k

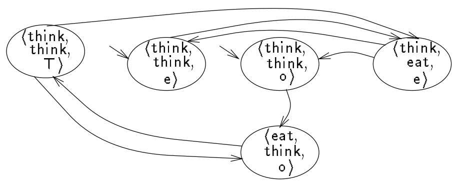


Fig. 5. An approximation to the free abstraction.

in Lemma 5.2.2), all states that are $\preceq$-greater than $\langle \mathsf{think}, \top, 100\rangle$ also overlap this successor. In a similar way, condition (2) of Lemma 5.2.2 can be shown to hold for $\langle \mathsf{think}, \mathsf{think}, \mathsf{o}\rangle$ and for $\langle \mathsf{think}, \mathsf{think}, 100\rangle$ too. The conclusion is that only the constrained transition that starts in the state $\langle \mathsf{think}, \mathsf{think}, \top\rangle$ may be non-optimal[17], as condition (1) did not hold.

As an example of the computation of approximations by choosing non-optimal abstract interpretations $\overline{t_i}$ of operations in the program, consider the dining mathematicians without the "restart" extension. Take the same free abstract interpretations as before for all operations except 3∗, for which we take the following approximation: $\overline{3*^F}(\mathsf{o}, \top) = \mathit{true}$ while $\overline{3*^F}(\mathsf{o}, \mathsf{e}) = \overline{3*^F}(\mathsf{o}, \mathsf{o}) = \mathit{false}$. This is easily checked to satisfy Definition 5.3.1. We get the free abstraction of Figure 5, from which still various properties may be deduced, such as the fact that at least one mathematician will keep engaged in a cycle of thinking and eating.

7. STRONG PRESERVATION

In Section 5.2, we have identified conditions under which computed abstract models are optimal in the standard sense of Abstract Interpretation. This notion of optimality concerns the quality of the abstract interpretation used to compute an abstraction, ${}_{\alpha}\mathcal{I}(P)$, relative to the "ideal" abstraction $\alpha(\mathcal{I}(P))$. In particular, it is optimality *with respect to a given abstract domain* ${}_{\alpha}\Sigma$.

A different notion of quality is the aptness of the abstract domain itself for the set of properties to be checked. In other words, given an abstract domain, how often will we get indeterminate answers as in the case of property (17) in Section 6? In posing this question, we enter in fact the area of specific applications of the framework presented so far: the answer to the question depends very much on the specific set of properties to be checked and on the programming language that is analysed. Nevertheless, we present a few general results that characterise the preservation quality of abstract domains. We say that a set of properties is *strongly preserved* when for every property, it is satisfied in the abstract model if and only if it is satisfied in the concrete model.

According to a well-known result in modal logic [van Benthem et al. 1994], two image-finite transition systems satisfy the same closed L_μ formulae if and only if they are bisimilar. In our case this implies the following.

[17] In fact, it *is* optimal, as is easily seen. This indicates that Lemma 5.2.2 only gives a *sufficient* condition.

LEMMA 7.0.1. *Consider* $\mathcal{C} = (\Sigma, I, R)$ *and* $\mathcal{A}^{\cup} = ({}_{\alpha}\Sigma, {}_{\alpha}I, {}_{\alpha}R^F \cup {}_{\alpha}R^C)$ *and let* Σ' *and* ${}_{\alpha}\Sigma'$ *be the reachable states of* $\mathcal{C}$ *and* $\mathcal{A}^{\cup}$ *resp. If the relation* $\rho \subseteq \Sigma' \times {}_{\alpha}\Sigma'$ *defined by* $\rho(c, a) \Leftrightarrow \alpha(c) = a$ *is a bisimulation from* $\mathcal{C}$ *to* $\mathcal{A}^{\cup}$ *and both* ρ *and* ρ^{-1} *are consistent*[18], *then every closed formula in* L_μ *is strongly preserved.*

PROOF. Using induction on the structure of L_μ formula. Note that ρ is total by totality of α. □

It is not difficult to see that a sufficient condition for this in terms of ${}_{\alpha}\Sigma$ is that the partitioning $\{\gamma(a) \mid a \in atoms({}_{\alpha}\Sigma)\}$ of Σ (where $atoms({}_{\alpha}\Sigma) = \{\alpha(\{c\}) \mid c \in \Sigma\}$) is at least as fine as the partitioning induced by the coarsest bisimulation on Σ.

Reversely, assuming that $\mathcal{C}$ and $\mathcal{A}^{\cup}$ have image-finite transition relations, if every closed formula in L_μ is strongly preserved, then there exists a consistent bisimulation from $\mathcal{C}$ to $\mathcal{A}^{\cup}$, although this need not be the relation that is induced by the Galois insertion.

In search for a strongly preserving abstract domain, the following may also be useful.

LEMMA 7.0.2. *Let* ρ *be as in Lemma 7.0.1.* $\mathcal{A}$ *is an abstraction that is* ρ*-bisimilar to* $\mathcal{C}$ *if and only if* ${}_{\alpha}R^F = {}_{\alpha}R^C$.

The price for strong preservation of full L_μ is that the attainable reduction of the concrete model is bounded by its quotient under bisimulation equivalence. However, it may well be the case that we can identify a subset of L_μ in which all the properties of interest can be expressed. Such a subset induces a coarser equivalence on the concrete states, in general. In Dams et al. [1993] and Dams et al. [1994], we develop algorithms that can be used to reduce the system with respect to the equivalences induced by $\forall$CTL* and by a single $\forall$CTL formula.

8. RELATED WORK

Property-preserving abstractions of reactive systems have been the topic of intensive research lately. Most of these efforts are based on the notion of simulation (see Definition 2.4.2). *Homomorphisms* (see e.g. Ginzburg [1968]), used in automata theory to construct language preserving reductions of automata, can be viewed as a precursor of this. Adapted to our notion of transition system, $h : \Sigma \rightarrow {}_{\alpha}\Sigma$ is a homomorphism iff $c \in I$ implies $h(c) \in {}_{\alpha}I$ and $R(c, d)$ implies ${}_{\alpha}R(h(c), h(d))$, where ${}_{\alpha}I$ is the set of initial abstract states and ${}_{\alpha}R$ the abstract transition relation. In Milner [1971], Milner introduced the term simulation to denote a homomorphism between deterministic systems. Since then, it has been re-adapted to nondeterministic transition systems and has become popular in the areas of program refinement and verification; Sifakis [1982], Sifakis [1983] and Hennessy and Milner [1980] are some early papers on this topic. Dill [1989] and Kurshan [1990] focus on trace (linear time) semantics and universal safety and liveness properties. Some of the first papers that consider the (strong) preservation of full CTL* and L_μ are Clarke et al. [1992] and Bensalem et al. [1992].

Following Kurshan [1990], Clarke et al. [1992] defines the relation between the concrete and abstract model by means of a homomorphism h, which induces an

[18] See Definition 2.4.2.

equivalence relation $\sim$ on the concrete states, defined by $c \sim d \Leftrightarrow h(c) = h(d)$. The abstract states are then representations of the equivalence classes of $\sim$. It is shown that universal properties (expressed in $\forall$CTL*) are preserved from the abstract to the concrete model. Preservation of full CTL* is shown to hold when h is *exact*, which boils down to requiring the concrete and abstract Kripke structures to be bisimilar. Consequently, CTL* is *strongly* preserved in that case, thus only allowing for relatively small reductions in the size of model. A notion of approximation between abstract systems is given based on the subset ordering on transition relations. As a result, an abstraction that is based on an exact h cannot be approximated, except by itself. Our approach to defining approximations in Section 4 is a generalisation of this — see Lemma 5.1.2 and the remark below it. Furthermore, our notion of abstract Kripke structure *does* allow proper approximations in the context of weak preservation of the full logic, in this case L_μ. Clarke et al. [1992] also explains the construction of abstract models and approximations thereof, by abstract interpretation of elementary operations (called *abstract compilation*), and illustrates this with a number of examples. A journal version appeared as Clarke et al. [1994]. Long [1993] also contains these results, presented in a slightly more general setting.

Bensalem et al. [1992] presents similar ideas in a more general setting by considering simulation relations to connect the concrete and abstract transition systems. Preservation of both $\Diamond L_\mu$ and $\Box L_\mu$ is dealt with in the setting of weak preservation. It is shown that if there exists a simulation from $\mathcal{C}$ to $\mathcal{A}$ that is total on Σ, then properties expressed in $\Box L_\mu$ are preserved from $\mathcal{A}$ to $\mathcal{C}$, while properties in $\Diamond L_\mu$ are preserved from $\mathcal{C}$ to $\mathcal{A}$. Again, preservation of the full μ-calculus is only shown for abstractions that are bisimilar to $\mathcal{C}$. The construction of abstract models, which is only briefly touched upon in Bensalem et al. [1992], is worked out further in the journal version, Loiseaux et al. [1995], where it is shown in addition how the abstraction of a concurrent system can be constructed compositionally from the abstractions of the individual components. In Loiseaux [1994], this theory is not only worked out in full detail, but the implementation of a tool based on it is described and analysed too. A closely related paper is Graf and Loiseaux [1993]. The approach is similar to that taken in Section 5, although the results deviate because the underlying frameworks are slightly different.

Loiseaux et al. [1995] also uses Galois connections to relate concrete and abstract states spaces, but in a different way than we do. It is shown that in their case, this is equivalent to using simulation relations. However, being between $\mathcal{P}(\Sigma)$ and $\mathcal{P}({}_\alpha\Sigma)$, these connections do not impose structure on the set ${}_\alpha\Sigma$ of abstract states. In particular, no approximation ordering $\preceq$ to relate the precision of abstract states is defined. As a result, that approach is more general, but fails to capture the notion of optimality, both on the level of states and on the level of complete transition systems. On the other hand, our approach is a proper instance of the simulation-based framework, and does distinguish between optimal abstractions of transition systems (as captured by the abstraction function α^M) and approximations (expressed by the relation $\trianglelefteq$ on abstract transition systems, see Definition 4.0.1). This is further discussed in Section 8.1 below.

Kelb [1995] also discusses the preservation of universal and existential μ-calculus properties within the framework of Abstract Interpretation. As in Bensalem et al.

[1992], the relation between abstract and concrete systems is defined through simulations cast in the form of Galois connections. The interpretation of a μ-calculus formula, which is a set of states, is approximated from below and above. By combining these dual approximate interpretations, using one for the $\Box$-operator and the other for the $\Diamond$, weak preservation of arbitrary μ-calculus formulae is obtained. This technique is similar to the mixed abstractions presented in this paper. A strong point of Kelb's thesis is the integration of these theoretical results with symbolic (BDD-based) representations. Indeed, Kelb [1995] proceeds by decribing practical experiments on the symbolic verification of StateChart programs, including part of the material from Kelb et al. [1995].

Cleaveland et al. [1994] is based on an early version, Dams et al. [1993], of this paper, and independently develops the idea of mixing both free and constrained abstractions in a single abstract system to attain preservation of full CTL*. More recently, Cleaveland et al. [1995] focusses on the issue of optimality. An approximation ordering $\propto_h$ (relative to a homomorphism h) on abstract transition systems is defined and shown to coincide with the CTL*-property ordering, i.e. $\mathcal{T}_1 \propto_h \mathcal{T}_2$ ($\mathcal{T}_1$ is an approximation of $\mathcal{T}_2$) if and only if any CTL*-property satisfied by $\mathcal{T}_1$ is satisfied by $\mathcal{T}_2$ as well. Cleaveland et al. [1995] defines an abstraction function (cf. our α^M) that maps transition systems to "$\propto_h$-optimal"[19] abstractions. Our approach in Section 4 is similar, but it should be noted that the framework of Cleaveland et al. [1995], being based on functional homomorphisms h, is less general than ours.

Cleaveland and Riely [1994] presents a framework for the abstract interpretation of processes that pass values. Application of Abstract Interpretation to verify properties of CCS is described in De Francesco et al. [1995].

While developed independently, and from a different perspective, Abstract Kripke structures bear some resemblance to the *modal transition systems* of Larsen and Thomsen [1988], which also combine two types of transition relations ("may" and "must"-transitions) in one system. Modal transition systems have been developed in the area of specification. Must-transitions specify what is required while may-transitions specify what is admissible. In Larsen and Thomsen [1988], a notion of refinement is defined such that the must-transitions in the specification simulate those in the refined system, while the may-transitions in the refined system simulate those in the specification. This is similar to our definition of approximation between Abstract Kripke structures (Definition 4.0.1). On the other hand, in modal transition systems, the must-relation is required to be a subset of the may-relation. Also, there is no notion of approximation ordering between states.

A recent paper, Kelb et al. [1995], reports on an application of abstract-interpretation techniques to the verification of properties of a production cell.

8.1 Comparing the simulation-based and Galois-insertion approaches

Above, we have discussed several related papers in which the relation between the concrete and the abstract systems is defined in terms of simulation relations, much in the same way as we have defined the approximation relation $\trianglelefteq$ among abstract systems in Section 4. Our definition of abstraction, in terms of Galois insertions, is a special case of this. In this subsection, we compare these alternative approaches

[19] In Cleaveland et al. [1995], a different notation is used for this.

in some more depth.

The conditions under which L_μ is preserved from a mixed transition system $\mathcal{A}' = (\Sigma', I', F', C')$ to the concrete system $\mathcal{C} = (\Sigma, I, R)$ may be formulated entirely in terms of simulations as follows. There should exist a relation $\rho \subseteq \Sigma \times \Sigma'$ such that:

(1) ρ is consistent.

(2) R ρ-simulates F'.

(3) C' ρ^{-1}-simulates R.

(4) For every $c \in I$ there exists $a \in I'$ such that $\rho(c, a)$.

Such a simulation-based approach, as we call it, is a generalisation of our approach using Galois insertions, as expressed by the following lemma.

LEMMA 8.1.1. *Let $\mathcal{A}' = (\Sigma', I', F', C')$ such that $\mathcal{A}' \trianglerighteq \alpha^M(\mathcal{C})$. Then there is a relation $\rho \subseteq \Sigma \times \Sigma'$ such that the conditions (1)–(4) above are satisfied.*

PROOF. By Definition 4.0.1, $\mathcal{A}' \trianglerighteq \alpha^M(\mathcal{C})$ implies that Σ' is equal to the set ${}_\alpha\Sigma$ of abstract states that is connected to the concrete states via (α, γ) and on which the valuation ${}_\alpha\|\cdot\|_{Lit}$ of literals has been defined. Consider the underlying description relation $\rho' \subseteq \Sigma \times \Sigma'$ defined by $\rho'(c, a) \Leftrightarrow c \in \gamma(a)$. Furthermore, let σ be the mixed simulation from $\alpha^M(\mathcal{C})$ to $\mathcal{A}'$ that exists by Definition 4.0.1, satisfying points (1) through (4) from that definition. We show that ρ defined as $\rho'\sigma$ satisfies the conditions (1)–(4) above.

(1) Suppose that $\rho(c, a)$ and $a \models p$. By definition of ρ we can choose $a' \in {}_\alpha\Sigma$ such that $\rho'(c, a')$ and $\sigma(a', a)$. By condition (1) in Definition 4.0.1, $a' \models p$, i.e. $a' \in {}_\alpha\|p\|_{Lit}$. $\rho'(c, a')$ implies that $\alpha(c) \preceq a'$, so, by Lemma 3.1.2, $\alpha(c) \models p$. By Definition 3.1.1 and the fact that $\gamma(\alpha(c)) \supseteq \{c\}$, it follows that $c \in \|p\|_{Lit}$.

(2) & (3) For the optimal abstraction $\alpha^M(\mathcal{C})$, it is easily shown that R ρ'-simulates ${}_\alpha R^F$ and ${}_\alpha R^C$ ρ'^{-1}-simulates R. From the fact that $\mathcal{A}' \trianglerighteq \alpha^M(\mathcal{C})$ it follows by Definition 4.0.1 that ${}_\alpha R^F$ σ-simulates F' and C' σ^{-1}-simulates ${}_\alpha R^C$. By transitivity of simulation it now follows that R ρ-simulates F' and C' ρ^{-1}-simulates R.

(4) Let $c \in I$. Then by Definition 3.2.1 of abstract initial states, $\alpha(c) \in {}_\alpha I$ (the initial states of $\alpha^M(\mathcal{C})$). From point (4) in Definition 4.0.1 of $\mathcal{A}' \trianglerighteq \alpha^M(\mathcal{C})$ it now follows that there exists $a \in I'$ with $\sigma(\alpha(c), a)$. Because we have $\rho'(c, \alpha(c))$, it follows that $\rho(c, a)$.

□

Conversely, if for a mixed system $\mathcal{A}' = (\Sigma', I', F', C')$, there exists a relation $\rho \subseteq \Sigma \times \Sigma'$ such that conditions (1)–(4) above hold, then it need not be the case that $\mathcal{A}' \trianglerighteq \alpha^M(\mathcal{C})$: although every concretisation function γ induces a relation ρ satisfying conditions (1)–(4) (see the proof above), an arbitrary ρ does not necessarily induce the concretisation function of a Galois insertion. So why use Galois insertions when the simulation-based approach is more general? We see two main advantages of our framework.

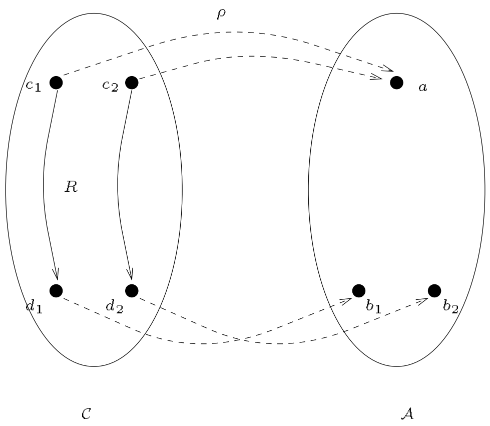


Fig. 6. Finding simulations in both ways.

Useful constrained abstractions. First, in order to define useful abstractions that preserve existential properties, where with useful we mean that a fair amount of existential properties indeed hold in the abstract system, the abstract states need to be partially ordered anyway. We illustrate this by an example. Consider Figure 6. c_1, c_2, d_1 and d_2 are concrete states while a, b_1 and b_2 are abstract. The relation ρ, indicated by dashed arrows, gives the relation between concrete and abstract states, so one could say that c_1 is described by a, or that c_1 is in the concretisation of a (although a Galois insertion does not necessarily exist). In order for *universal* properties to be preserved from a to its concretisation, R has to ρ-simulate the abstract transition relation. This implies that there have to be abstract transitions from a to both b_1 and to b_2. It is not hard to see that as long as ρ is total on the concrete states, it is always possible to find a total abstract transition relation F such that R ρ-simulates F. In order for *existential* properties to be preserved, the abstract transition relation C should ρ^{-1}-simulate R. For the situation of Figure 6, no C-transition from a is possible under this requirement. This shows that an abstract domain that is suitable for defining useful abstractions preserving universal properties is not necessarily also suitable for defining useful abstractions that preserve existential properties. In the case of this example, if we want to also have an outgoing C-transition from a, we need to extend the abstract domain with a state that describes (at least) both d_1 and d_2. In general, in order to define useful abstractions preserving existential properties, the abstract domain should contain states describing subsets of concrete states of various sizes. In particular, if it should always be possible for C to be total, the abstract domain should in particular have a "top" element describing all concrete states. The abstract state $\langle \mathsf{think}, \mathsf{think}, \top \rangle$ in Figure 4 is a good example of this. Without it, property (15) (page 134) could not have been verified.

Optimal abstractions vs. approximations. Second, the mere requirement that a simulation must exist in order for preservation to hold has the drawback that it leaves too much freedom in the choice of "good" abstractions. The Galois-insertion

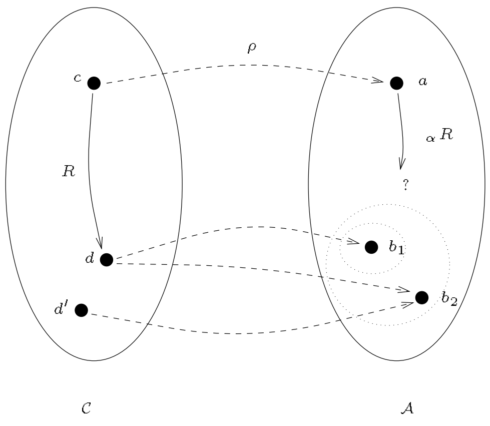


Fig. 7. Abstraction with states of comparable precision.

framework in which we developed our results may be seen as a successful attempt to try and quantify the notion of *precision* of abstractions by distinguishing between the notions of *(optimal) abstraction* and *approximation*. This point is discussed in the following comparison of our work with that of Loiseaux et al. [1995].

We focus on free abstractions. Given the concrete transition system $\mathcal{C} = (\Sigma, I, R)$ and a set Σ' of abstract states that is related to the concrete states by a "description relation" $\rho \subseteq \Sigma \times \Sigma'$ (total on Σ), the requirement $\rho^{-1}R \subseteq {}_{\alpha}R\rho^{-1}$ (R simulates ${}_{\alpha}R$), viewed as an inequality over ${}_{\alpha}R$, has many solutions. From the point of view of property preservation, the $\subseteq$-minimal solutions are interesting. However, even such minimal solutions may have comparable quality, as illustrated by the example in Figure 7, where the problem is to choose an ${}_{\alpha}R$-successor of a such that $\rho^{-1}R \subseteq {}_{\alpha}R\rho^{-1}$ is satisfied. The $\subseteq$-minimal solutions are obtained by taking either b_1 or b_2 as successor (but not both). However, choosing b_1 will generally give better property preservation, as it describes fewer concrete states.

However, instead of exploring this freedom in order to refine their notion of quality of abstract transition relations, Loiseaux et al. propose a condition under which all minimal solutions are bisimilar to each other. This condition is

$$\rho\rho^{-1}\rho = \rho. \tag{19}$$

Expressed in words, it says that if two concrete states share a description (abstract state), then they share all descriptions. For example, in Figure 7 also the states d' and b_1 would have to be related by ρ. It is easy to see that the generality of simulations over Galois insertions, namely the possibility to have several optimal but mutually incomparable abstractions of a set of concrete states, is eliminated by this condition. In fact, requirement (19) implies that it is useless to have a ρ that is not functional. This is expressed in the following lemma (which can be found in Loiseaux et al. [1995]). It implies that whenever $\rho(c, a)$ and $\rho(c, a')$ ($a \neq a'$) for some c — i.e, ρ is not functional — then a and a' are bisimilar. If the goal of abstraction is to produce abstract systems with a minimal number of states[20], then one of a and a' should be removed from $\mathcal{A}$ in this case.

LEMMA 8.1.2. *If ρ is total on Σ, ${}_\alpha R$ is a $\subseteq$-minimal relation such that R ρ-simulates ${}_\alpha R$, and $\rho\rho^{-1}\rho = \rho$, then $\rho\rho^{-1}$ is a bisimulation on $\mathcal{A}$.*

PROOF. We have to show that $\rho\rho^{-1}$ and $(\rho\rho^{-1})^{-1}$ are simulations on $\mathcal{A}$. Because $(\rho\rho^{-1})^{-1} = \rho\rho^{-1}$, it suffices to show that $\rho\rho^{-1}$ is a simulation, i.e. (by Definition 2.4.2), that $(\rho\rho^{-1})^{-1}{}_\alpha R \subseteq {}_\alpha R(\rho\rho^{-1})^{-1}$, i.e., $\rho^{-1}\rho\ {}_\alpha R \subseteq {}_\alpha R\ \rho^{-1}\rho$ (*). Because any minimal solution ${}_\alpha R$ satisfies ${}_\alpha R = \rho^{-1}R\rho$ (see Loiseaux et al. [1995]), (*) is equivalent to $\rho^{-1}\rho\rho^{-1}R\rho \subseteq \rho^{-1}R\rho\rho^{-1}\rho$. Because $\rho\rho^{-1}\rho = \rho$ and therefore also $\rho^{-1}\rho\rho^{-1} = \rho^{-1}$, this is equivalent to $\rho^{-1}R\rho \subseteq \rho^{-1}R\rho$, which is true. $\square$

So, in order to be able to distinguish optimal abstractions from approximations, Loiseaux et al. [1995] makes assumption (19), which renders their framework less general than the Galois-insertion approach, because, under the reasonable assumption that the abstract system does not contain bisimilar states, it forces ρ to be functional.

Consider Figure 7 again. In our framework, the simulation relation ρ induces the following Galois insertion on sets of states: for any $a \in \Sigma'$, $\gamma(a) = \{c \mid \rho(c,a)\}$ and for any $C \subseteq \Sigma$, $\alpha(C) = \bigwedge\{a \mid \gamma(a) \supseteq C\}$, where $\bigwedge$ denotes the meet operation corresponding to the ordering $\preceq$ defined by $a \preceq a' \Leftrightarrow \gamma(a) \subseteq \gamma(a')$. Taking ${}_\alpha R$ to be ${}_\alpha R^F$ as specified by Definition 3.3.1, point (1), yields b_1 as the only successor of a, as desired.

9. CONCLUSIONS

The results of this paper may be seen from two points of view. From the position of Abstract Interpretation, we have presented a generalisation of the framework, extending it to the analysis of reactive properties. This generalisation consists in allowing the next-state relation of a non-deterministic transition system to be abstracted to a relation, and not to a function as is common practice. This allows the analysis, via the abstraction, of not only universal properties — expressing that something holds along all possible executions —, but also existential properties — expressing the *existence* of paths satisfying some property. Furthermore, both safety as well as liveness properties are preserved. We have proven that the truth of every property expressible in L_μ is preserved from abstract to concrete model. As is common in Abstract Interpretation, the attained reduction depends solely on the choice of the abstraction function, thus allowing better reductions than is the case with minimisation based on bisimulation. This was possible by considering abstract transition systems having two different transition relations, each preserving a separate fragment of L_μ. The use of a Galois insertion to relate concrete and abstract states allowed the definition of both types of transitions over the same set of abstract states, resulting in the preservation of full L_μ. The price to be paid is that there exist formulae — and increasingly many when the abstraction becomes

[20] In general, the goal is to have minimal *representations* of the system. When states are not represented explicitly, but by BDDs for example, the size of the representation may actually shrink as the number of states grows. In such cases, it may indeed be useful to have some "redundant" states around.

coarser — that do not hold in the abstraction, and neither do their negations. In case of persistence on *strong* preservation (i.e. preservation of both truth and falsehood of formulae), which renders the abstract model bisimilar to the concrete model, we have shown the implications for the form that the abstract domain takes.

From the viewpoint of property-preserving characteristics of simulation relations, we have managed to define a notion of precision that allows us to "separate the wheat from the chaff". An abstraction function α^M specifies the optimal abstract model for a given concrete system, while an approximation order $\trianglelefteq$ distinguishes the relative precision between abstract models. The embedding of the property-preservation results for simulation in the framework of Abstract Interpretation opens up the possibility of constructing abstract models directly from the text of a program, thereby avoiding the intermediate construction of the full concrete model. This construction is possible by associating non-standard, *abstract* interpretations with the operators in a programming language, which allows their evaluation over *descriptions* of data. To this purpose, we chose a simple programming language and defined abstract interpretations of its tests and operations. Conditions were given under which the free and constrained abstract transition relations thus computed coincide with the optimal relations as specified by α. Furthermore, a notion of approximation on the level of operations was given by which the user may simplify the task without loosing the preservation results. Such approximations can accelerate the computation of abstract models, be it at the risk of obtaining a model that does not contain enough information in order to verify the property. It was illustrated by an example that these techniques can be applied to verify properties of systems with an infinite state space.

Further work. As pointed out in Section 6, the construction of abstract models that strongly preserve a given property of interest may require refinement of the abstract domain. The framework of Abstract Interpretation, being based on a given abstract domain, does not offer a methodological approach to such refinement. A trial-and-error approach would benefit much from the development of heuristics that are specific to the domain of application, while also a set of powerful diagnostic tools in addition to the model checker are invaluable in that case.

In the light of the quest for fully automated verification methods, we are currently investigating the use of *partition refinement algorithms* for the construction of strongly preserving models; see Dams [1996]. Other, rather preliminary ideas point in the direction of using theorem provers and algebraic manipulation tools. Although the problem is undecidable in general, there may well be interesting subclasses that can be decided efficiently.

In a recent paper, Kelb et al. [1995], we apply the ideas developed in this paper and in Kelb [1994] to verify μ-calculus properties of a production cell [Damm et al. 1995] in a compositional fashion.

Interesting extensions of the framework that we plan to investigate are fairness and action labelling.

ACKNOWLEDGMENTS

We thank Susanne Graf for many interesting and stimulating discussions, and Nissim Francez for his helpful comments. Anthony McIsaac pointed out some mistakes.

We are much indebted to the anonymous referees, who contributed many substantial improvements.

Orna Grumberg has been partially supported by the E. and J. Bishop research fund, and Rob Gerth has been partially supported by ESPRIT project P6021: "Building Correct Reactive Systems (REACT)". We are particularly grateful to the Netherlands Organisation for International Co-operation in Higher Education (Nuffic) for funding a working visit to the Technion in Haifa.

REFERENCES

BACK, R. J. R. AND KURKI-SUONIO, R. 1983. Decentralization of process nets with centralized control. In *2nd ACM SIGACT–SIGOPS Symp. on Principles of Distributed Computing*. ACM, New York, 131–142.

BENSALEM, S., BOUAJJANI, A., LOISEAUX, C., AND SIFAKIS, J. 1992. Property preserving simulations. In *Computer-Aided Verification*, G. von Bochmann and D. Probst, Eds. Number 663 in LNCS. Springer-Verlag, New York, 251–263.

BOUAJJANI, A., FERNANDEZ, J.-C., HALBWACHS, N., RAYMOND, P., AND RATEL, C. 1992. Minimal state graph generation. *Sci. Comput. Program. 18*, 247–269.

CLARKE, E., EMERSON, E., AND SISTLA, A. 1986. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst. 8,* 2 (Apr.), 244–263.

CLARKE, E., GRUMBERG, O., AND LONG, D. 1992. Model checking and abstraction. In *19th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. ACM SIGACT/SIGPLAN.

CLARKE, E., GRUMBERG, O., AND LONG, D. 1994. Model checking and abstraction. *ACM Trans. Program. Lang. Syst. 16,* 5 (Sept.), 1512–1542.

CLEAVELAND, R., IYER, P., AND YANKELEVICH, D. 1995. Optimality in abstractions of model checking. In *Static Analysis*, A. Mycroft, Ed. Number 983 in LNCS. Springer-Verlag, New York, 51–63.

CLEAVELAND, R., IYER, S. P., AND YANKELEVICH, D. 1994. Abstractions for preserving all CTL* formulae. Tech. Rep. 94-03, Dept. of Comp. Sc., North Carolina State University, Raleigh, NC 27695. Apr.

CLEAVELAND, R. AND RIELY, J. 1994. Testing-based abstractions for value-passing systems. In *CONCUR '94: Concurrency Theory*, B. Jonsson and J. Parrow, Eds. Number 836 in LNCS. Springer-Verlag, Berlin, 417–432.

CODISH, M., FALASCHI, M., AND MARRIOTT, K. 1994. Suspension analysis for concurrent logic programs. *ACM Trans. Program. Lang. Syst. 16,* 3 (May), 649–686.

COURCOUBETIS, C., Ed. 1993. *Computer Aided Verification*. Number 697 in LNCS. Springer-Verlag, Berlin.

COUSOT, P. AND COUSOT, R. 1977. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. 4th ACM Symp. on Principles of Programming Languages*. Los Angeles, California, 238–252.

COUSOT, P. AND COUSOT, R. 1979. Systematic design of program analysis frameworks. In *Proc. 6th ACM Symp. on Principles of Programming Languages*. San Antonio, Texas, 269–282.

COUSOT, P. AND COUSOT, R. 1992a. Abstract interpretation and application to logic programs. *Journal of Logic Programming 13*, 103–179.

COUSOT, P. AND COUSOT, R. 1992b. Abstract interpretation frameworks. *J. Logic and Comput. 2,* 4, 511–547.

DAM, M. 1994. CTL* and ECTL* as fragments of the modal μ-calculus. *Theor. Comput. Sci. 126*, 77–96.

DAMM, W., HUNGAR, H., KELB, P., AND SCHLÖR, R. 1995. Statecharts: Using graphical specification languages and symbolic model checking in the verification of a production cell. In *Formal Development of Reactive Systems: Case Study Production Cell*, C. Lewerenz and T. Lindner, Eds. Number 891 in LNCS. Springer-Verlag, Berlin, 131–149.

DAMS, D., GERTH, R., DÖHMEN, G., HERRMANN, R., KELB, P., AND PARGMANN, H. 1994. Model checking using adaptive state and data abstraction. See Dill [1994], 455–467.

DAMS, D., GERTH, R., AND GRUMBERG, O. 1993. Generation of reduced models for checking fragments of CTL. See Courcoubetis [1993], 479–490.

DAMS, D., GRUMBERG, O., AND GERTH, R. 1993. Abstract interpretation of reactive systems: Abstractions preserving ACTL*, ECTL* and CTL*. Draft.

DAMS, D., GRUMBERG, O., AND GERTH, R. 1995. Abstract interpretation of reactive systems: Preservation of CTL*. Logic Group Preprint Series 132, Utrecht University, Dept. of Philosophy, Heidelberglaan 8, 3584 CS Utrecht, The Netherlands. May. Also appears as Computing Science Note 95/16, Eindhoven University of Technology, Dept. of Math. and Comp. Sc.

DAMS, D. R. 1996. Abstract interpretation and partition refinement for model checking. Ph.D. thesis, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands.

DE FRANCESCO, N., FANTECHI, A., GNESI, S., AND INVERARDI, P. 1995. Model checking of non-finite state processes by finite approximations. In *Tools and Algorithms for the Construction and Analysis of Systems*, E. Brinksma, W. R. Cleaveland, K. G. Larsen, T. Margaria, and B. Steffen, Eds. Number 1019 in LNCS. Springer, Berlin, 195–215.

DILL, D. 1989. *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits.* ACM Distinguished Dissertations. The MIT Press, London.

DILL, D. L., Ed. 1994. *Computer Aided Verification.* Number 818 in LNCS. Springer-Verlag, Berlin.

EMERSON, E. AND HALPERN, J. 1986. Sometimes and not never revisited: On branching versus linear time. *Journal of the Association for Computing Machinery 33,* 1, 151–178.

GINZBURG, A. 1968. *Algebraic Theory of Automata.* ACM Monograph Series. Academic Press, New York/London.

GRAF, S. 1994. Verification of a distributed cache memory by using abstractions. See Dill [1994], 207–219. To appear in *Distributed Computing.*

GRAF, S. AND LOISEAUX, C. 1993. A tool for symbolic program verification and abstraction. See Courcoubetis [1993], 71–84.

HAREL, D. 1987. Statecharts: A visual formalism for complex systems. *Sci. Comput. Program. 8*, 231–274.

HENNESSY, M. AND MILNER, R. 1980. On observing nondeterminism and concurrency. In *Proc. of the 7th International Colloquium on Automata Languages and Programming (ICALP)*, J. de Bakker and J. van Leeuwen, Eds. Number 85 in LNCS. Springer-Verlag, Berlin, 299–309.

KELB, P. 1994. Model checking and abstraction: A framework preserving both truth and failure information. University of Oldenburg, Germany. Unpublished note.

KELB, P. 1995. Abstraktionstechniken für automatische verifikationsmethoden. Ph.D. thesis, Carl von Ossietzky University of Oldenburg, Germany.

KELB, P., DAMS, D., AND GERTH, R. 1995. Efficient symbolic model checking of the full μ-calculus using compositional abstractions. Computing Science Reports 95/31, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands. Oct.

KOZEN, D. 1983. Results on the propositional μ-calculus. *Theor. Comput. Sci. 27*, 333–354.

KRIPKE, S. 1963. A semantical analysis of modal logic I: normal modal propositional calculi. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik 9*, 67–96.

KURSHAN, R. P. 1990. Analysis of discrete event coordination. In *Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness*, J. W. de Bakker, W.-P. de Roever, and G. Rozenberg, Eds. Number 430 in LNCS. Springer-Verlag, Berlin, 414–453.

KURSHAN, R. P. 1994. *Computer-Aided Verification of Coordinating Processes: The Automata-Theoretic Approach.* Princeton Series in Computer Science. Princeton University Press, Princeton, NJ.

LARSEN, K. G. 1989. Modal specifications. In *Automatic Verification Methods for Finite State Systems (CAV89)*, J. Sifakis, Ed. Number 407 in LNCS. Springer-Verlag, Berlin, 232–246.

Larsen, K. G. and Thomsen, B. 1988. A modal process logic. In *1988 IEEE Symposium on Logic in Computer Science*. TC-MFC, Computer Society Press, Washington, 203–210.

Lichtenstein, O. and Pnueli, A. 1985. Checking that finite state concurrent programs satisfy their linear specification. In *12th Annual ACM Symposium on Principles of Programming Languages*. ACM SIGACT/SIGPLAN, 97–107.

Loiseaux, C. 1994. Vérification symbolique de systèmes réactifs à l'aide d'abstractions. Ph.D. thesis, Université Joseph Fourier – Grenoble I, Grenoble, France.

Loiseaux, C., Graf, S., Sifakis, J., Bouajjani, A., and Bensalem, S. 1995. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design 6*, 11–44.

Long, D. E. 1993. Model checking, abstraction, and compositional verification. Ph.D. thesis, School of Comp. Sc., Carnegie Mellon University, Pittsburgh, PA 15213.

Marriott, K. 1993. Frameworks for abstract interpretation. *Acta Inf. 30,* 2, 103–129.

Milner, R. 1971. An algebraic definition of simulation between programs. In *2nd International Joint Conference on Artificial Intelligence*. British Computer Society, London, 481–489.

Park, D. 1981. Concurrency and automata on infinite sequences. In *Theoretical Computer Science*, P. Deussen, Ed. Number 104 in LNCS. Springer-Verlag, Berlin, 167–183.

Queille, J. P. and Sifakis, J. 1982. Specification and verification of concurrent systems in CESAR. In *International Symposium on Programming*, M. Dezani-Ciancaglini and U. Montanari, Eds. Number 137 in LNCS. Springer-Verlag, Berlin, 337–351.

Sifakis, J. 1982. Property preserving homomorphisms and a notion of simulation for transition systems. Rapport de Recherche 332, IMAG, Grenoble, France. Nov.

Sifakis, J. 1983. Property preserving homomorphisms of transition systems. In *4thW orkshop on Logics of Programs*, E. Clarke and D. Kozen, Eds. Number 164 in LNCS. Springer-Verlag, Berlin, 458–473.

van Benthem, J., van Eijck, J., and Stebletsova, V. 1994. Modal logic, transition systems and processes. *J. of Logic and Comput. 4,* 5, 811–855.

Vardi, M. Y. and Wolper, P. 1986. An automata-theoretic approach to automatic program verification (preliminary report). In *Logic in Computer Science*. IEEE TC-MFC, IEEE Computer Society Press, 332–344.

Received May 1994; revised May 1995; November 1995; September 1996; accepted October 1996