Abstract Interpretation of Reactive Systems: Preservation of CTL^*

Dennis Dams[‡] Orna Grumberg[§] Rob Gerth[¶]

Abstract

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 the stumbling block in many situations. 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 on the property-preserving characteristics of *simulation relations*. A major drawback of all these results is that they do not offer a satisfactory formalization of the notions of *precision* and *optimality* of abstractions. Furthermore, the use of simulation relations poses difficulties when formalizing the preservation of both existential and universal properties over the same abstract domain.

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 formalizes the notion of optimality, while allowing to trade precision for speed by computing sub-optimal *approximations*. However, Abstract Interpretation has traditionally been focussed on the analysis of deterministic programs, in particular through abstractions that preserve invariance properties, i.e., properties that hold in all states along every possible execution path.

In this paper, we extend Abstract Interpretation to non-deterministic systems. This results in a uniform basis for the analysis of both existential and universal reactive properties, as expressible in the branching time logic CTL^*. It is shown how optimal abstract models may be constructed by symbolic execution of programs and that approximation may be used to find an optimum between precision and speed. Examples are given to illustrate this. We indicate conditions under which also the falsity of formulae is preserved. Finally, we compare our approach to those based on simulation relations.

Submitted for publication.

This report is a revision of [DGG94] and also appears as Computing Science Note 95/16 of the Department of Computing Science, Eindhoven University of Technology.

[‡]Dept. of Philosophy, Utrecht University, Heidelberglaan 8, 3584 CS Utrecht, The Netherlands. Dennis.Dams@phil.ruu.nl.

[§]Computer Science Dept., Technion, Haifa 32000, Israel. orna@cs.technion.ac.il. Partially supported by the U.S.-Israeli Binational Science Foundation.

[¶]Dept. of Math. and Computing Science, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands. robg@win.tue.nl. Currently working in ESPRIT project P6021: “Building Correct Reactive Systems (REACT)”.

1 Introduction

In the *model-checking* approach [CES86, LP85, QS81] 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, the formulae express certain required properties of such behaviours. Obviously, the size of the model is the 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 formalized in the framework of *Abstract Interpretation* [CC77], which was originally conceived as a unifying theory of compile-time (data-flow) analyses. Applications of Abstract Interpretation have traditionally 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 which 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 which 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 which, in addition to information about single states, also provides 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 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 formalized by expressing them in a formal logic whose formulae can be interpreted over transition systems. One commonly used logic is CTL^* (*computation tree logic*, see [EH86]). It contains universal and existential quantification over execution paths, as well as temporal operators that express that, along a path, some property will hold (a) in the next state, (b) in every state (safety), or (c) in some state (liveness).

The structure of this paper is as follows. The next section introduces the formal machinery to be used. In Sect. 3, a notion of abstract transition system is developed which preserves properties from CTL^*. Such an abstract system is an optimal (in a sense that is made precise in Sect. 2) description

[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 elsewhere often just termed "safety" or "invariance".

of a (concrete) transition system. In computing such abstract models, one may decide to sacrifice optimality in exchange for a gain in efficiency. The *approximation* ordering between abstract systems, defined in Sect. 4, formalises this notion of sub-optimality. Sect. 5 then shows how abstractions 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, and an alternative construction method is indicated which yields optimal models even in absence of those conditions. Furthermore, it is shown that sub-optimal models are constructed when computing approximations to the lifted operations. An elaborate example is presented in Sect. 6. Sect. 7 briefly indicates the consequences of insisting on *strong preservation*, meaning that not only truth, but also falsehood of formulae is preserved. Sect. 8 compares ours to related work, and Sect. 9 concludes.

2 Preliminaries

Temporal logic Given is a set *Prop* of *propositions*. We choose to define CTL^* in its negation normal form, i.e., negations only appear in front of propositions. This facilitates the definition of universal and existential CTL^*. The set of *literals* is defined by $Lit = \mathit{Prop} \cup \{\neg p \mid p \in \mathit{Prop}\}$.

2.0.1 DEFINITION. *The logic* CTL^* *is the set of state formulae which is inductively defined by the following clauses.*

1. *If* $p \in Lit$*, then* p *is a state formula.*
2. *If* φ *and* ψ *are state formulae, then so are* $\varphi \wedge \psi$ *and* $\varphi \vee \psi$*.*
3. *Any state formula is also a path formula.*
4. *If* φ *and* ψ *are path formulae, then so are* $\varphi \wedge \psi$*,* $\varphi \vee \psi$*,* $X\varphi$*,* $\varphi\ U\ \psi$ *and* $\varphi\ V\ \psi$*.*
5. *If* φ *is a path formula, then* $\forall\varphi$ *and* $\exists\varphi$ *are state formulae.*

For $\varphi \in \mathrm{CTL}^*$*, the formula* $\neg\varphi$ *is considered to be an abbreviation of the equivalent* CTL^* *formula in negation normal form (obtained in the usual way). The abbreviations true, false and* $\rightarrow$ *can then be defined as usual. For a path formula* φ*,* $F\varphi$ *and* $G\varphi$ *abbreviate true* $U\ \varphi$ *and false* $V\ \varphi$ *respectively.*

$\forall\mathrm{CTL}^*$ *and* $\exists\mathrm{CTL}^*$ *(universal and existential* CTL^**) are subsets of* CTL^* *in which the only allowed path quantifiers are* $\forall$ *and* $\exists$ *respectively.*

Transition systems CTL^* 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 is a *transition relation* over Σ which is total and image-finite, so that for every $s \in \Sigma$ there exists a positive, finite number of $t \in \Sigma$ for which $R(s,t)$ holds. A *path* in $\mathcal{T}$ is an infinite sequence $\pi = s_0 s_1 \cdots$ of states such that for every $i \in I\!N$, $R(s_i, s_{i+1})$ holds. The notation π^n denotes the suffix of π which begins at s_n. For $s \in \Sigma$, a $(\mathcal{T}, s)$-*path* (or s-*path* when $\mathcal{T}$ is clear from the context) is a path in $\mathcal{T}$ that starts in s.

We assume a function $||\cdot|| : Lit \rightarrow \mathcal{P}(\Sigma)$, satisfying $||p|| \cap ||\neg p|| = \emptyset$ for every proposition $p \in$ *Prop*, which specifies the interpretation of literals over states. Intuitively, $||p||$ is the set of states where p holds. Transition systems thus defined are essentially the same as Kripke structures ([Kri63]). The only difference is that we have the function $||\cdot||$ instead of a labelling function from Σ to sets of literals.

2.0.2 DEFINITION. *Satisfaction of formulae is defined inductively as follows. Let $p \in Lit$, $s \in \Sigma$ and π a path in $\mathcal{T}$.*

1. $s \models p$ *iff* $s \in \|p\|$.

2. $s \models \varphi \wedge \psi$ *iff* $s \models \varphi$ *and* $s \models \psi$, $s \models \varphi \vee \psi$ *iff* $s \models \varphi$ *or* $s \models \psi$.

3. $\pi \models \varphi$, *where* $\pi = s_0 s_1 \cdots$ *and* φ *is a state formula, iff* $s_0 \models \varphi$.

4. $\pi \models \varphi \wedge \psi$ *iff* $\pi \models \varphi$ *and* $\pi \models \psi$, $\pi \models \varphi \vee \psi$ *iff* $\pi \models \varphi$ *or* $\pi \models \psi$.
   $\pi \models X\varphi$ *iff* $\pi^1 \models \varphi$.
   $\pi \models \varphi\ U\ \psi$ *iff there exists* $n \in I\!N$ *such that* $\pi^n \models \psi$ *and for all* $i < n$, $\pi^i \models \varphi$.
   $\pi \models \varphi\ V\ \psi$ *iff for all* $n \in I\!N$, *if* $\pi^i \not\models \varphi$ *for all* $i < n$, *then* $\pi^n \models \psi$.

5. $s \models \forall\varphi$ *iff for every* s*-path* π, $\pi \models \varphi$, $s \models \exists\varphi$ *iff there exists an* s*-path* π *such that* $\pi \models \varphi$.

For a set S of states or paths, 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 $(\mathcal{T}, I) \models \varphi$.

These formulae can be used to express a variety of properties of transition systems. For example, $s \models \forall Gp$ expresses that p is true in all states which are reachable from s. If proposition r characterizes reset states, then $s \models \forall G\exists Fr$ means that along every possible execution path from s, in every state there is a possible continuation which will eventually reach a reset state. V is the dual of U ($\varphi\ V\ \psi$ is semantically equivalent to $\neg(\neg\varphi\ U\ \neg\psi)$) and has the intuitive meaning of "release": ψ must be true as long as φ is false, and only if φ becomes true, ψ may become false afterwards. It has been added as the dual of U to compensate for the fact that formulae like $\neg(\varphi\ U\ \psi)$ are not well-formed. For the same reason both $\wedge$ and $\vee$ are primitive in the logic.

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

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[2] 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 ("state explosion"), we look for abstractions of it which 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 (CTL^*) property that holds for the abstract model, should hold for the original concrete model as well. Sect. 3 focuses 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 ${}_\alpha\mathcal{I}$ which maps each program P to an abstraction of $\mathcal{I}(P)$. Sect. 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 formalize 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

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

formally captured by introducing a set of *abstract states* and a concretisation function γ mapping each abstract state to a set of concrete states that have become indistinguishable. For example, we may wish to abstract from everything but the fact whether variable x is greater than 5, in which case we introduce two abstract states named, e.g., **grt_5** and **leq_5**, acting as descriptions of concrete states as specified by the function γ with $\gamma(\textbf{grt\_5}) = \{s \in \Sigma \mid s(x) > 5\}$ and $\gamma(\textbf{leq\_5}) = \{s \in \Sigma \mid s(x) \leq 5\}$ (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 **grt_5** and **leq_5**. For example, the execution of the assignment $x := x + 1$ in the abstract state **grt_5** results in the state **grt_5** again. On the other hand, executing it in the state **leq_5** may yield either **grt_5** or **leq_5**. This can be modelled by introducing more abstract states in such a way that for every subset C of Σ, there exists an abstract state a with $\gamma(a) \supseteq C$. An *abstraction function* α may then be introduced which maps each C to the "smallest" description (see below). Correctness ("safety") of the abstract interpretation of the assignment $x := x + 1$ can then be stated by requiring that for each C, $\gamma(\alpha(C)\text{“}+1\text{”}) \supseteq C + 1$ (where on the left-hand-side of the $\supseteq$, the quotes indicates that the function $+1$ has to be interpreted abstractly, over the domain of abstract values, and on the right-hand-side, $+1$ is pointwise extended to sets).

Formally, Abstract Interpretation presupposes the existence of partially ordered sets (often complete lattices) $(C, \sqsubseteq)$ and $(A, \preceq)$, the *concrete* and *abstract domains* respectively. The elements of these domains denote (concrete resp. abstract) interpretations of programs or data values (in the example above, the concrete domain is $(\mathcal{P}(\Sigma), \subseteq)$). The *approximation orderings* $\sqsubseteq$ and $\preceq$ relate the information content, or *precision*, of elements. If $a \preceq a'$ we say that a is more precise than a', or that a' approximates a; similarly for $\sqsubseteq$. Optimal means maximally precise in this sense. The abstraction relation between elements from these domains is specified by an abstraction function $\alpha : C \to A$ which maps every concrete element to its most precise, that is $\preceq$-least, abstraction; so α by definition yields an (indeed: *the*) optimal abstraction. Alternatively, it can be specified through a concretization function $\gamma : A \to C$ which yields for every $a \in A$ the most general, that is $\sqsubseteq$-greatest, element that is abstracted by a. Under these assumptions, the pair (α, γ) forms a Galois connection from $(A, \preceq)$ to $(C, \sqsubseteq)$:

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

For an index set $J \subseteq I\!N$, if $\{c_j \in C \mid j \in J\}$ has a least upper bound in C (denoted by $\bigsqcup$), then α distributes over it: $\alpha(\bigsqcup_{j \in J} c_j) = \bigvee_{j \in J} \alpha(c_j)$ (where $\bigvee$ denotes the least upper bound in A). See, e.g., [CC79].

Often, the abstract approximation order is defined in terms of the concrete by $a \preceq a' \;\Leftrightarrow\; \gamma(a) \sqsubseteq \gamma(a')$. In that case, 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, a concrete interpretation function is often defined to be the least fixpoint *lfp* of some function $f : C \to C$ (where f depends on P in some way). Given an abstract domain A and the connection (α, γ), one looks for a function ${}_{\alpha}f : A \to A$ such that $\alpha(\mathit{lfp}(f)) \preceq \mathit{lfp}({}_{\alpha}f)$. This is satisfied if for all $c \in C$, $\alpha \circ f(c) \preceq {}_{\alpha}f \circ \alpha(c)$ ([CC79]). In this case, ${}_{\alpha}f$ is said to be *safe* w.r.t. f.

The requirement $\alpha(\mathcal{I}(P)) \preceq {}_\alpha\mathcal{I}(P)$ can easily be satisfied by defining ${}_\alpha\mathcal{I}$ to be the constant function mapping everything to an element $\top \in A$ which is less precise than all other elements and hence gives no information at all. In order to avoid such solutions and reason about the quality of abstractions, the notion of optimality is lifted to functions. An abstract interpretation function ${}_\alpha\mathcal{I}$ is said to be *optimal* w.r.t. $\mathcal{I}$ iff $\alpha(\mathcal{I}(P)) = {}_\alpha\mathcal{I}(P)$. Likewise, in case of the above example, ${}_\alpha f$ is optimal w.r.t. f iff $\alpha \circ f(c) = {}_\alpha f \circ \alpha(c)$; this is easily seen to imply that *lfp*$({}_\alpha f)$ is optimal w.r.t. *lfp*(f). In the Galois *insertion* framework, if such an optimal function ${}_\alpha f$ exists, it is defined by ${}_\alpha f = \alpha \circ f \circ \gamma$.

A number of weaker frameworks than the Galois connection framework have been proposed in which the abstraction and/or concretization functions are replaced by relations (see [Mar93] 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 which 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 CTL^* properties. An abstraction function is given which maps any concrete system to one such representative. Hence, by comparing a computed abstraction to this representative, we are still able to discuss optimality issues.

A more extensive introduction to Abstract Interpretation and overview of its uses can be found in [CC92a] and [CC92b].

The relation transformers $\cdot^{\exists\exists}$ and $\cdot^{\forall\exists}$; (bi)simulations We use two relation transformers which lift transition relations on states to relations on sets of states.

2.0.4 DEFINITION. *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.*

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

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 ([Mil71]) and bisimulation ([Par81]).

2.0.5 DEFINITION. *Let $R_1 \subseteq \Sigma_1 \times \Sigma_1$ and $R_2 \subseteq \Sigma_2 \times \Sigma_2$ be transition relations. A relation $\sigma \subseteq \Sigma_1 \times \Sigma_2$ is a simulation (from Σ_1 to Σ_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 Σ_2 to Σ_1. A (bi)simulation σ is consistent iff $\sigma(s_1, s_2)$ implies $\{p \in Lit \mid \|p\| \ni s_1\} = \{p \in Lit \mid \|p\| \ni s_2\}$.*

An equivalent definition of simulation is the following (see, e.g., [Mil71]). 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].

[3] The intuition is that R_2 can "mimic" everything that R_1 can do. From this point of view, the by now standard terminology "R_1 simulates R_2", which was introduced in [Mil71], is awkward.

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 $({}_{\alpha}\Sigma, \preceq)$ which determines its relation to the concrete states. We will usually write $\alpha(c)$ for $\alpha(\{c\})$. For a path $\rho = a_0 a_1 \cdots$ in $\mathcal{A}$, $\gamma(\rho) = \{c_0 c_1 \cdots \mid \forall_i\ R(c_i, c_{i+1}) \wedge\ c_i \in \gamma(a_i)\}$. We investigate how to define abstract models in such a way that CTL^* is preserved from abstract to concrete model. The intention is that a concrete state c is described by any abstraction $a \succeq \alpha(c)$ of it. From this point, it is natural to require preservation of formulae on the level of individual states:

$$\forall_{\varphi \in \mathrm{CTL}^*, a \in {}_{\alpha}\Sigma}\ (\mathcal{A}, a) \models \varphi\ \Rightarrow\ (\mathcal{C}, \gamma(a)) \models \varphi \tag{1}$$

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 ${}_{\alpha}||\cdot||$ 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 CTL^*, we must have $(\mathcal{A}, a) \models p\ \Rightarrow\ (\mathcal{C}, \gamma(a)) \models p$ for every literal p. 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 as possible literals to hold in each abstract state.

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

This choice determines the valuation of literals in abstract states. Namely, the relation $\models\ \subseteq {}_{\alpha}\Sigma \times Lit$ is defined as in clause 1 of Def. 2.0.2, where s now denotes an abstract state and $||\cdot||$ has to be replaced by ${}_{\alpha}||\cdot||$. 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 as possible literals 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 $a \notin {}_{\alpha}||p||$ but also $a \notin {}_{\alpha}||\neg p||$. So, although the rule of the excluded third from classical logic holds on the level of interpretation of literals over abstract states (we have either $a \models p$ or $a \not\models p$, and similarly for $\neg p$), this rule does *not* hold on the level of the logic itself: $a \not\models p$ does not necessarily imply that $a \models \neg p$.

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 $\varphi \in \mathrm{CTL}^*$. A sufficient condition for this is $\cup\{\gamma(a) \mid a \in {}_{\alpha}I\} \supseteq I$. 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 $\cup\{\gamma(a) \mid a \in {}_{\alpha}I\} = I$. However, the following choice for ${}_{\alpha}I$ yields the smallest set $\cup\{\gamma(a) \mid a \in {}_{\alpha}I\}$ which still includes I.

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

Note that the singleton $\alpha(I)$ is in general less precise than this: because α distributes over $\cup$ (see the remark after Def. 2.0.3), we have $\alpha(I) = \bigvee\{\alpha(s) \mid s \in I\}$, so that for each $s \in I$, $\alpha(s) \preceq \alpha(I)$.

3.3 Abstract transition relations

Most traditional applications of the framework of Abstract Interpretation are developed in a context where both the concrete and the abstract transition relations are deterministic, i.e., they are transition functions f and ${}_{\alpha}f$. In such cases, the (unique) abstract successor ${}_{\alpha}f(a)$ of a would be the best description of the successors of the states in $\gamma(a)$: ${}_{\alpha}f(a) = \alpha(f(\gamma(a)))$[4]. On the other hand, 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., [BCG88]), which is a much too strong condition, as it results in the falsity of CTL^* 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 "there exists a successor state satisfying property φ" are to be preserved, then abstract state b may only be a successor of a if $R^{\forall\exists}(\gamma(a), Y)$ for some $Y \subseteq \Sigma$ and $Y \subseteq \gamma(b)$. For reasons of optimality, we also would like a to have as many successors as possible, and furthermore each of them should be as precise a description of Y 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 best 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 $\alpha(Y) = b$:

3.3.1 DEFINITION.

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. A few remarks are in place. Note that for any $a \in {}_{\alpha}\Sigma$, the minimal sets Y' such that $R^{\exists\exists}(\gamma(a), Y')$, are all singletons. By the totality of R it follows that ${}_{\alpha}R^F$ is total[5]. By image-finiteness, the set $\{Y' \mid R^{\forall\exists}(\gamma(a), Y')\}$ always has minimal elements so that also ${}_{\alpha}R^C$ is total; this follows from the following lemma.

3.3.2 LEMMA. *If $R \subseteq A \times B$ is image-finite, then for every $X \subseteq A$ there exists a minimal $Y \subseteq B$ such that $R^{\forall\exists}(X, Y)$.*

PROOF. Let $X \subseteq A$. Take C to be the collection of sets Z such that, for every $x \in X$, $R(x) = \{y \mid R(x, y)\}$ is not contained in Z. For a chain $Z_0 \subseteq Z_1 \subseteq Z_2 \ldots$ of sets in C, let Z^* be the union of the Z_i. Z^* is an upper bound of the chain of sets; we will prove that Z^* is in C. If Z^* is not in C, there is some $x \in X$ such that $R(x)$ is contained in Z^*. Since $R(x)$ is finite, $R(x)$ must be contained in some Z_i. Contradiction.

We can now apply Zorn's Lemma[6] to conclude that there is a maximal element of C. Its complement is a minimal set Y as required. □

Finally, note that by the requirement of minimality of Y in the definitions, it is not in general the case that ${}_{\alpha}R^C \subseteq {}_{\alpha}R^F$.

We have the following preservation properties for paths.

[4] Functions are pointwise extended to sets.

[5] We make one exception, namely, the abstract state $\perp \in {}_{\alpha}\Sigma$ with $\gamma(\perp) = \emptyset$ (if present) has no successors. However, note that $\perp$ itself is not reachable anyway.

[6] Zorn's Lemma states that if in a non-empty collection C of sets, every $\subseteq$-increasing chain of sets in C has an upper bound in C, then C has a maximal element.

3.3.3 LEMMA.

1. *Let* $a \in {}_{\alpha}\Sigma$*,* $c \in \gamma(a)$ *and* π *be a* $((\Sigma, I, R), c)$*-path. Then there exists an* $(({}_{\alpha}\Sigma, {}_{\alpha}I, {}_{\alpha}R^F), a)$*-path* ρ *such that* $\pi \in \gamma(\rho)$*.*
2. *Let* $a \in {}_{\alpha}\Sigma$*,* $c \in \gamma(a)$ *and* ρ *be an* $(({}_{\alpha}\Sigma, {}_{\alpha}I, {}_{\alpha}R^C), a)$*-path. Then there exists a* $((\Sigma, I, R), c)$*-path* π *in* $\gamma(\rho)$*.*

PROOF.

1. Assume $\pi = c_0 c_1 \cdots$ with $c_0 = c$. Define $\rho = a_0 a_1 \cdots$ with $a_0 = a$ and $a_i = \alpha(c_i)$ for $i \geq 1$. Because (α, γ) is a Galois Insertion, we have $\gamma(\alpha(c_i)) \supseteq \{c_i\}$, so $c_i \in \gamma(a_i)$ for $i \geq 1$. Also, $c_0 \in \gamma(a_0)$. So $\pi \in \gamma(\rho)$.

   Because for all $i \geq 0$, $R(c_i, c_{i+1})$, $c_i \in \gamma(a_i)$, and $a_i = \alpha(c_i)$, we have by Def. 3.3.1 of ${}_{\alpha}R^F$: ${}_{\alpha}R^F(a_i, a_{i+1})$.
2. Assume $\rho = a_0 a_1 \cdots$ with $a_0 = a$. We show that there exists an infinite sequence $\pi = c_0 c_1 \cdots$ of states in Σ such that for all $i \geq 0$, $c_i \in \gamma(a_i)$ and $R(c_i, c_{i+1})$. π is constructed inductively, as follows. Let $c_0 = c$. Then by definition, $c_0 \in \gamma(a_0)$. Now suppose that for some $n \geq 0$, c_n is given such that $c_n \in \gamma(a_n)$. Because ${}_{\alpha}R^C(a_n, a_{n+1})$, there must be (by Def. 3.3.1 of ${}_{\alpha}R^C$) Y such that $R^{\forall\exists}(\gamma(a_n), Y)$ and $\alpha(Y) = a_{n+1}$. By definition of $R^{\forall\exists}$, there exists $c_{n+1} \in Y$ such that $R(c_n, c_{n+1})$. Because (α, γ) is a Galois Insertion, we have $\gamma(\alpha(Y)) \supseteq Y$, so $c_{n+1} \in \gamma(a_{n+1})$.

   Thus, π is a $((\Sigma, I, R), c)$-path and $\pi \in \gamma(\rho)$. □

We now define the abstraction of $\mathcal{C}$ as the four-tuple $\mathcal{A}^M = ({}_{\alpha}\Sigma, {}_{\alpha}I, {}_{\alpha}R^F, {}_{\alpha}R^C)$, representing a transition system with two transition relations. This is called the *mixed* abstraction. A *free path* is a path with all its transitions in ${}_{\alpha}R^F$; a *constrained path* is a path with all its transitions in ${}_{\alpha}R^C$. The interpretation of CTL^* formulae over a mixed abstraction is defined slightly different from Def. 2.0.2. $\|\cdot\|$ has to be replaced by ${}_{\alpha}\|\cdot\|$, and furthermore clause 5 is replaced by

5′. $s \models \forall\varphi$ *iff for every free* s*-path* π*,* $\pi \models \varphi$*;* $s \models \exists\varphi$ *iff there exists a constrained* s*-path* π *such that* $\pi \models \varphi$*.*

In the rest of this paper, we implicitly assume this adapted definition when interpreting formulae over mixed abstractions or approximations thereof (see Sect. 4). We then have:

3.3.4 THEOREM. *For every* $\varphi \in \mathrm{CTL}^*$*,* $\quad \mathcal{A}^M \models \varphi \;\Rightarrow\; \mathcal{C} \models \varphi$*.*

PROOF. It is easy to see that it suffices to prove statewise preservation for every φ in CTL^*. This is done by induction on the structure of φ, proving for state formulae that for every state $a \in {}_{\alpha}\Sigma$, $(\mathcal{A}^M, a) \models \varphi \;\Rightarrow\; (\mathcal{C}, \gamma(a)) \models \varphi$ and for path formulae that for every path ρ in $\mathcal{A}^M$, $(\mathcal{A}^M, \rho) \models \varphi \;\Rightarrow\; (\mathcal{C}, \gamma(\rho)) \models \varphi$. The base case, $\varphi \in \mathit{Lit}$, follows directly from Def. 3.1.1. The cases that φ is a conjunction or disjunction of state or path formulae, a state formula interpreted over a path, or a path formula with principal operator X, U or V, are straightforward. For φ of the form $\forall\psi$, let a be a state such that $(\mathcal{A}^M, a) \models \forall\psi$, let $c \in \gamma(a)$, and consider a $(\mathcal{C}, c)$-path π. By Lemma 3.3.3, we know that there exists an $(({}_{\alpha}\Sigma, {}_{\alpha}I, {}_{\alpha}R^F), a)$-path ρ such that $\pi \in \gamma(\rho)$, so, because $(\mathcal{A}^M, a) \models \forall\psi$, $(\mathcal{A}^M, \rho) \models \psi$; and by the ind. hyp. this gives $(\mathcal{C}, \pi) \models \psi$. So $(\mathcal{C}, \gamma(a)) \models \forall\psi$. The argument for φ of the form $\exists\psi$ is similar. □

So, mixed abstractions allow verification of full CTL^* while the degree of reduction is determined by the choice of the abstract domain and may hence be arbitrarily large. In contrast, reductions w.r.t. bisimulation equivalence [BFH$^+$92] only allow a limited reduction. These facts may seem contradicting, but the reader should note that by the definition of satisfaction of CTL^* formulae over mixed abstractions, it is possible that neither φ, nor $\neg\varphi$ holds; this is not possible with bisimulation reduction.

4 Approximations

We have (implicitly) specified an abstraction function α^M mapping each concrete system $\mathcal{C}$ to an abstract system $\mathcal{A}^M$. As explained in Subsection 3.3, the free and constrained transition relations in $\mathcal{A}^M$ were chosen in an optimal way by always choosing, for the successors of an abstract state, the $\preceq$-least descriptions of certain sets of concrete states, and furthermore by having a minimal number of free and a maximal number of constrained successors. In this section, we formally define an approximation ordering on abstract systems which allows non-optimal successors to be chosen, resulting in less precise abstract models, satisfying fewer CTL^* properties. Such approximations turn out to occur in a natural way when computing abstractions directly from a program P, as will be done in the next section. Lemma 4.0.3 shows that this notion of approximation can be seen as the "lifting" of the approximation order $\preceq$ on ${}_\alpha\Sigma$ to the level of abstract transition systems.

4.0.1 DEFINITION. *Let $\mathcal{A} = ({}_\alpha\Sigma, I, F, C)$ and $\mathcal{A}' = ({}_\alpha\Sigma, I', F', C')$ be abstract transition systems. $\mathcal{A} \preceq \mathcal{A}'$ iff (1) for every $a \in I$ there exists $a' \in I'$ such that $a \preceq a'$, (2) F $\preceq$-simulates F', and (3) C' $\succeq$-simulates C. For paths $\rho = a_0a_1\cdots$ and $\overline{\rho} = \overline{a}_0\overline{a}_1\cdots$, $\rho \preceq \overline{\rho}$ is defined by $\forall_{i\geq 0}\ a_i \preceq \overline{a}_i$.*

Note that $\preceq$ over abstract systems is a pre-order but no partial order (i.e., $\mathcal{A} \preceq \mathcal{A}'$ and $\mathcal{A}' \preceq \mathcal{A}$ does not imply $\mathcal{A} = \mathcal{A}'$).

4.0.2 THEOREM. *If $\mathcal{A} \preceq \mathcal{A}'$, then for every $\varphi \in \mathrm{CTL}^*$, $\quad \mathcal{A}' \models \varphi \Rightarrow \mathcal{A} \models \varphi$.*

PROOF. It is easily seen that if $\mathcal{A} \preceq \mathcal{A}'$, then for every free $(\mathcal{A}, a)$-path ρ there is a free $(\mathcal{A}', a)$-path $\rho' \succeq \rho$, and for every constrained $(\mathcal{A}', a)$-path ρ' there is a constrained $(\mathcal{A}, a)$-path $\rho \preceq \rho'$. Using these observations instead of Lemma 3.3.3, the rest of the proof is similar to that of Thm. 3.3.4. □

As an immediate corollary, we have that if $\mathcal{A}' \succeq \alpha^M(\mathcal{C})$, then CTL^* is preserved from $\mathcal{A}'$ to $\mathcal{C}$. For $\mathcal{A}' = ({}_\alpha\Sigma, I', F', C')$, it can indeed easily be shown that the condition $\mathcal{A}' \succeq \alpha^M(\mathcal{C})$ is equivalent to the requirement that R ρ-simulates F' and C' ρ^{-1}-simulates R (recall that R is the concrete transition relation), where $\rho(c, a) \Leftrightarrow \alpha(c) \preceq a$. The advantage of our framework over the use of simulations is that the distinction between the notions of *abstraction* (α^M) and *approximation* ($\preceq$) in our approach allows to distinguish optimal abstractions from other descriptions. In the following section, we explore this distinction in order to define optimal abstract interpretations which compute the optimal abstract systems directly from the text of a program.

The following lemma, which is easily proven, gives sufficient conditions for $\preceq$- and $\succeq$-simulation. It will be used in the following sections.

4.0.3 LEMMA. *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 $R'(a, b') \Rightarrow \exists_{b\preceq b'}\ R(a, b)$ if $R' \subseteq R$.

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 which is based on *action systems* [BKS83], 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 $\bar{x}$ represents the vector of program variables, c_i is a condition on their values and t_i specifies a transformation[7] of their values into the new vector $\bar{x}'$ (i ranges over some index set J). Executing an action means evaluating its condition c_i and, if and only if this yields *true*, updating the program variables as specified by the associated transformation t_i. A program is run by repeatedly nondeterministically choosing an action and executing it. We let Val denote 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^2.

5.0.1 DEFINITION. *Let P be the program $\{c_i(\bar{x}) \rightarrow t_i(\bar{x}, \bar{x}') \mid i \in J\}$. Its concrete model $\mathcal{C}$ is defined as follows:*

- $\Sigma = Val$,
- $I = IVal$,
- $R = \{(\bar{v}, \bar{v}') \in Val^2 \mid \exists_{i \in J}\, c_i(\bar{v}) \wedge t_i(\bar{v}, \bar{v}')\}$.

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 approximations of the abstract models of a program may be computed by interpreting the operators in the program correspondingly.

5.0.2 DEFINITION. *For $a, b \in {}_{\alpha}Val$,*

- $c_i^F(a) \Leftrightarrow \exists_{\bar{v} \in \gamma(a)}\, c_i(\bar{v})$;
- $t_i^F(a, b) \Leftrightarrow b \in \{\alpha(Y) \mid Y \in \min\{Y' \mid t_i^{\exists\exists}(\gamma(a), Y')\}\}$;
- $c_i^C(a) \Leftrightarrow \forall_{\bar{v} \in \gamma(a)}\, c_i(\bar{v})$;
- $t_i^C(a, b) \Leftrightarrow b \in \{\alpha(Y) \mid Y \in \min\{Y' \mid t_i^{\forall\exists}(\gamma(a), Y')\}\}$.

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

[7] We could have represented this transformation as the simultaneous assignment $\bar{x}' := t_i(\bar{x})$. However, by abstracting the function t_i, it may become a relation, and we prefer to denote both the concrete and the abstract transformations in the same way.

- ${}_{\alpha}\Sigma = {}_{\alpha}Val$;
- ${}_{\alpha}I = \{\alpha(\bar{v}) \mid \bar{v} \in IVal\}$;
- ${}_{\alpha}\widehat{R^F} = \{(a,b) \in {}_{\alpha}Val^2 \mid \exists_{i\in J}\; c_i^F(a) \;\wedge\; t_i^F(a,b)\}$;
- ${}_{\alpha}\widehat{R^C} = \{(a,b) \in {}_{\alpha}Val^2 \mid \exists_{i\in J}\; c_i^C(a) \;\wedge\; t_i^C(a,b)\}$.

Totality of ${}_{\alpha}\widehat{R^F}$ follows from the totality of R. In order for ${}_{\alpha}\widehat{R^C}$ to be total, we need to require that for every abstract state $a \in {}_{\alpha}\Sigma$, there exists $i \in J$ such that $\forall_{\bar{v}\in\gamma(a)}\, c_i(\bar{v}) =$ *true*. Note that in particular, this should hold for the abstract state $\top = \alpha(\Sigma)$ (although it need not be reachable). We satisfy this requirement by assuming that there is always an action of the form *true* $\rightarrow skip$ in P; an alternative solution is discussed in Sect. 5.1.2.

The following lemma and theorem express that the abstract interpretations given above can be used to compute approximations to $\mathcal{A}^M$.

5.0.3 LEMMA.

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

PROOF.

1. Let $a, b \in {}_{\alpha}Val$ and suppose $(a,b) \in {}_{\alpha}R^F$. By Def. 3.3.1 of ${}_{\alpha}R^F$, Def. 2.0.4 of $R^{\exists\exists}$ and Def. 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 $\min\{Y' \mid \ldots\}$ are singletons, 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 $\exists_{i\in J}\, [b \in \{\alpha(Y) \mid Y \in \min\{Y' \mid \exists_{\bar{v}\in\gamma(a),\bar{w}\in Y'}\, [c_i(\bar{v}) \wedge t_i(\bar{v},\bar{w})]\}\}]$. Now this is weakened by distributing the innermost existential quantifier over the $\wedge$: $\exists_{i\in J}\, [b \in \{\alpha(Y) \mid Y \in \min\{Y' \mid \exists_{\bar{v}\in\gamma(a)}\, [c_i(\bar{v})] \wedge \exists_{\bar{v}\in\gamma(a),\bar{w}\in Y'}\, [t_i(\bar{v},\bar{w})]\}\}]$. Because the set does not depend on $\exists_{\bar{v}\in\gamma(a)}\, [c_i(\bar{v})]$, this conjunct may be taken out of the (inner) set brackets. Using Def. 2.0.4 of $t_i(\bar{v},\bar{w})^{\exists\exists}$ and Def. 5.0.2 of $c_i^F(a)$, $t_i^F(a,b)$, and ${}_{\alpha}\widehat{R^F}$, the resulting equivalent term can then be rewritten to $(a,b) \in {}_{\alpha}\widehat{R^F}$.

2. Let $a, b \in {}_{\alpha}Val$ and suppose $(a,b) \in {}_{\alpha}\widehat{R^C}$. By Def. 5.0.2 of ${}_{\alpha}\widehat{R^C}$, $c_i^C(a)$ and $t_i^C(a,b)$, and Def. 2.0.4 of $t_i(\gamma(a),Y)^{\forall\exists}$, 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:

$$b \in \{\alpha(Y) \mid \exists_{i\in J}\, [Y \in \min\{Y' \mid \forall_{\bar{v}\in\gamma(a)}\, \exists_{\bar{w}\in Y'}\, [c_i(\bar{v}) \wedge t_i(\bar{v},\bar{w})]\}]\} \tag{2}$$

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

$$\exists_{i\in J}\, [Y \in \min\{Y' \mid \forall_{\bar{v}\in\gamma(a)}\, \exists_{\bar{w}\in Y'}\, [c_i(\bar{v}) \wedge t_i(\bar{v},\bar{w})]\}] \tag{3}$$

By pushing the $\exists_{i\in J}$ inside, this subexpression is weakened:

$$Y \in \min\{Y' \mid \exists_{i\in J}\, [\forall_{\bar{v}\in\gamma(a)}\, \exists_{\bar{w}\in Y'}\, [c_i(\bar{v}) \wedge t_i(\bar{v},\bar{w})]]\} \tag{4}$$

Clearly, for each Y that satisfies (3), there exists a subset of it that satisfies (4), so that if b satisfies (2), there exists $b' \preceq b$ which satisfies:

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

A similar step can be made again: if b' satisfies (5), then there exists $b'' \preceq b'$ satisfying:

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

which is, by Def. 5.0.1 of R, Def. 2.0.4 of $R^{\forall\exists}$ and Def. 3.3.1 of ${}_\alpha R^C$, equivalent to ${}_\alpha R^C(a, b'')$. □

5.0.4 THEOREM. $\widehat{\mathcal{A}^M} \succeq \mathcal{A}^M$.

PROOF. The proof is based on Lemmata 5.0.3 and 4.0.3. From the definitions of $\widehat{{}_\alpha R^F}$ and ${}_\alpha R^C$ it easily follows that ${}_\alpha R^F$ $\preceq$-simulates $\widehat{{}_\alpha R^F}$ and that ${}_\alpha R^C$ $\succeq$-simulates ${}_\alpha R^C$. □

5.1 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 (Def. 5.0.2) are, in general, strictly less precise than the optimal abstractions of Def. 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 Def. 5.0.2 to be optimal. Secondly, we indicate how, alternatively, the abstract interpretation of programs may be adapted in such a way that computed models are optimal.

5.1.1 Conditions on the abstract domain

In order to pinpoint the reasons why the computed abstractions are not optimal, we analyse the proof of Lemma 5.0.3. 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 $\boldsymbol{e}$ and $\boldsymbol{o}$, 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 $\boldsymbol{e}$ for a and $\gamma(\boldsymbol{e})$ for Y', does not hold. On the other hand, (T2) *does* hold: there exists an even number which is equal to 4 and there exists a (different) even number which, when divided by 4, yields an even number. In order to avoid this situation and enforce equivalence of (T1) and (T2), we can impose a condition on the abstract states.

5.1.1 LEMMA. *Let $\widehat{\mathcal{A}^M}$ be the abstract model computed according to Def. 5.0.2, and ${}_\alpha\Sigma' \subseteq {}_\alpha\Sigma$ and $\widehat{{}_\alpha R^F}'$ $\subseteq \widehat{{}_\alpha R^F}$ be the sets of states and transitions resp. which are reachable along transitions from $\widehat{{}_\alpha R^F}$. Let ${}_\alpha R^{F'}$ be the set of transitions of the optimal abstraction $\mathcal{A}^M$ which are reachable along transitions from ${}_\alpha R^F$. If for all $a \in {}_\alpha\Sigma'$ and $i \in J$, $\forall_{\bar{v}\in\gamma(a)}\ c_i(\bar{v})$ or $\forall_{\bar{v}\in\gamma(a)}\ \neg c_i(\bar{v})$, then $\widehat{{}_\alpha R^F}' = {}_\alpha R^{F'}$.*

PROOF. The same condition for *all* states, i.e., for all $a \in {}_\alpha\Sigma$ and $i \in J$, $\forall_{\bar{v}\in\gamma(a)}\ c_i(\bar{v})$ or $\forall_{\bar{v}\in\gamma(a)}\ \neg c_i(\bar{v})$, is easily seen to imply equivalence of the terms (T1) and (T2). Therefore, $\widehat{{}_\alpha R^F} = {}_\alpha R^F$ in that case. It easily follows that the restriction of the condition to reachable states implies a similar result for the reachable transitions. □

So, under the given condition, the reachable part of the computed abstract model — which is the only part that matters for evaluation of CTL^* formulae — coincides with the reachable part of the optimal model. The given condition is quite strong if the set of reachable states contains "large" states like $\top$. However, as can be seen from Def. 3.3.1 and the remark, following it, about the sets Y' being singletons, only the *atoms* of the abstract domain can be reachable, i.e., the elements $\{\alpha(\{c\}) \mid c \in \Sigma\}$. Hence, we should preferably choose the abstract domain in such a way that these atoms are "small". See [CC79] for a variety of techniques for the construction of suitable abstract domains.

Though being sufficient, the condition required in Lemma 5.1.1 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 5.0.3. The last two steps in this proof introduce the differences between ${}_\alpha R^C$ and $\widehat{{}_\alpha R^C}$. By replacing subformula (3) by (4), the minimality of the "$\forall\exists$-successors" Y of $\gamma(a)$ over the different actions is "divided out" and made global to all actions. As a result, each of the $\forall\exists$-successors under $\widehat{{}_\alpha R^C}$ of $\gamma(a)$ is a superset of an $\forall\exists$-successor under ${}_\alpha R^C$ of $\gamma(a)$. In the step from formula (5) to (6), the "action-wise" computation of $\forall\exists$-successors (for a single action $i \in J$, Y must be an $\forall\exists$-successor of $\gamma(a)$, i.e., all states in $\gamma(a)$ must be able to make a transition to some state in Y via action i) is replaced by an "action-wide" computation (all states in $\gamma(a)$ must be able to make a transition to some state in Y via *any* action). This difference *does* entail a true loss of information.

Both effects are eliminated by imposing a condition on the "overlapping" of abstract successor states in different directions (i.e., taking different actions), as follows.

5.1.2 LEMMA. *If for all $i, j \in J$ with $i \neq j$ and $a, b_i, b_j \in {}_\alpha\Sigma$ such that $c_i^C(a) \wedge t_i^C(a, b_i)$ and $c_j^C(a) \wedge t_j^C(a, b_j)$, we have $\forall_{b \in {}_\alpha\Sigma}\ [\gamma(b_i) \cap \gamma(b) = \emptyset \ \vee\ \gamma(b_j) \cap \gamma(b) = \emptyset \ \vee\ (\gamma(b_i) \subseteq \gamma(b) \wedge \gamma(b_j) \subseteq \gamma(b))]$, then $\widehat{{}_\alpha R^C} = {}_\alpha R^C$.*

Intuitively, the condition in this lemma specifies that two abstract successors corresponding to different actions may only both be overlapped by a third state if they are completely subsumed by this third state.

PROOF. It is easy to check that the condition implies that for $i, j \in J$ with $i \neq j$ and $a \in {}_\alpha\Sigma$ both of the following hold:

1. there cannot exist $Y_i \in \min\{Y' \mid \forall_{\bar{v} \in \gamma(a)}\ \exists_{\bar{w} \in Y'}\ c_i(\bar{v}) \wedge t_i(\bar{v}, \bar{w})\}$ and $Y_j \in \min\{Y' \mid \forall_{\bar{v} \in \gamma(a)}\ \exists_{\bar{w} \in Y'}\ c_j(\bar{v}) \wedge t_j(\bar{v}, \bar{w})\}$ such that $Y_i \subset Y_j$.
2. $\min\{Y' \mid \forall_{\bar{v} \in \gamma(a)}\ \exists_{\bar{w} \in Y'}\ [c_i(\bar{v}) \wedge t_i(\bar{v}, \bar{w})] \vee \forall_{\bar{v} \in \gamma(a)}\ \exists_{\bar{w} \in Y'}\ [c_j(\bar{v}) \wedge t_j(\bar{v}, \bar{w})]\} =$
$\min\{Y' \mid \forall_{\bar{v} \in \gamma(a)}\ \exists_{\bar{w} \in Y'}\ [(c_i(\bar{v}) \wedge t_i(\bar{v}, \bar{w})) \vee (c_j(\bar{v}) \wedge t_j(\bar{v}, \bar{w}))]\}$

Point 1 now implies that in the step from formula (2) to (5) in part 2 of the proof of Lemma 5.0.3, computing the minimal Y globally to all actions does not affect the computed constrained transition relation. Point 2 ensures that also the step from (5) to (6) in the proof yields an equivalent formula. □

5.1.2 Adapting the abstract interpretation

Instead of imposing conditions guaranteeing optimality of abstract models computed as specified by Def. 5.0.2, we may change the definition of these abstract interpretations themselves in such a way that the "loss" of Lemma 5.0.3 does not occur. For the free abstract interpretation, this means that it cannot 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 generalization 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 $act_{\{i_1,\ldots,i_k\}}(a,b) \Leftrightarrow \exists_{a_1,\ldots,a_k,b_1,\ldots,b_k \in {}_\alpha\Sigma}$ $[a_1 \vee \cdots \vee a_k = a, b_1 \vee \cdots \vee b_k = b, \forall_{j\in\{1,\cdots,k\}}\, c_{i_j}^C(a_j) \wedge t_{i_j}^C(a_j,b_j)]$ for subsets $\{i_1,\ldots,i_k\}$ of J (recall that $\bigvee$ and $\vee$ denote least upper bounds on ${}_\alpha\Sigma$). This approach corresponds to the *merge over all paths analysis* of [CC79].

5.2 Computing approximations

One may deliberately 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, e.g., the c_i and t_i involve intricate operations. 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.

5.2.1 DEFINITION. *The definition of approximation is extended to abstract interpretations of the transformation operators, as follows. For abstract operations*[8] $t, \overline{t} \in {}_\alpha Val \times {}_\alpha Val$,

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

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

- ${}_\alpha\Sigma = {}_\alpha Val$;
- ${}_\alpha I = \{\alpha(\bar{v}) \mid \bar{v} \in IVal\}$;
- $\overline{{}_\alpha R^F} = \{(a,b) \in {}_\alpha Val^2 \mid \exists_{i\in I}\, c_i^F(a) \wedge \overline{t_i^F}(a,b)\}$
- $\overline{{}_\alpha R^C} = \{(a,b) \in {}_\alpha Val^2 \mid \exists_{i\in I}\, c_i^C(a) \wedge \overline{t_i^C}(a,b)\}$

5.2.2 LEMMA. $\overline{\mathcal{A}^M} \succeq \widehat{\mathcal{A}^M}$.

PROOF. We 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 Def. 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 Def. 5.0.2 of c_i^F and t_i^F and Def. 2.0.4 of $\cdot^{\exists\exists}$), so $\exists_{i\in I}\, [c_i^F(\overline{a}) \wedge t_i^F(\overline{a},a_1)]$. By Def. 5.2.1 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., $\overline{{}_\alpha R^F}(\overline{a},\overline{a}_1)$.

[8] Remember that such "operations" are binary relations.

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 Def. 5.2.1 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 Def. 5.0.2 of c_i^C and t_i^C and Def. 2.0.4 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 $\preceq$, $a'_1 \preceq a_1$. □

5.3 Practical application

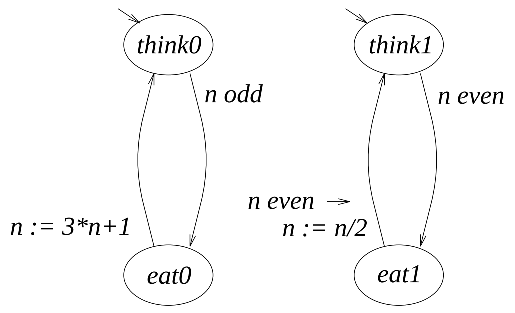
The use of abstract interpretation to model check a property φ for a program P is characterized by the following phases. First, an abstract domain ${}_\alpha Val$ has to be chosen and for all operation symbols occurring in P, abstract interpretations have to be provided. Typically, the tests and transformations 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 Def. 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. It is important to notice that only *positive* results of this model checking carry over to the concrete 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 CTL^* 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 $\mathcal{C} \models \neg\varphi$, i.e., $\mathcal{C} \not\models \varphi$. Of course, the dual abstraction should be well-chosen for this. We do not further investigate this point; solutions can be found in [DGD⁺94].

The same idea of constructing an abstract model by abstract interpretation of program operations, although based on a different theoretical framework ([LGS⁺93], see Sect. 8 for a comparison), is applied to a “real-life” example in [Gra94]. 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 Def. 5.0.2, can be performed in an automated fashion. In [Gra94], the abstract domain has to be provided by the user of the method, and the proofs for the abstract operators form a difficult step in the method. In [KDG95], approximations to the transition relation of StateCharts ([Har87]) are used to verify μ-calculus properties of a production cell ([DHKS95]) in a compositional fashion. In [DGG93a] and [DGD⁺94], a method is developed which aims at full automation of these steps.

6 Example

In this section we illustrate the theory on a small example. Consider the system consisting of two concurrent processes depicted in Fig. 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.



(a)

$$\ell_0 = think,\ odd(n) \longrightarrow \ell_0 := eat$$
$$\ell_0 = eat \longrightarrow \ell_0 := think,\ n := 3 * n + 1$$
$$\ell_1 = think,\ even(n) \longrightarrow \ell_1 := eat$$
$$\ell_1 = eat,\ even(n) \longrightarrow \ell_1 := think,\ n := n/2$$

(b)

Figure 1: The dining mathematicians (a). Expressed as an action system (b).

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 any infinite sequence of (arbitrarily) interleaved steps of both processes which 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 formalize this, we first express the program as an action system. 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. See Fig. 1(b). The state space Σ of this program is the set $\{think, eat\}^2 \times I\!N \setminus \{0\}$ of values that the vector $\langle \ell_0, \ell_1, n\rangle$ of program variables may assume. The initial states are $I = \{\langle think, think, n\rangle \mid n \in I\!N \setminus \{0\}\}$. Its transitions are defined as in Def. 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).

The properties to be verified are expressed in CTL^* as follows.

$$\forall G \neg(\ell_0 = eat \wedge \ell_1 = eat) \tag{7}$$

$$\forall G(\ell_0 = eat \rightarrow \forall F \ell_1 = eat) \tag{8}$$

$$\forall G(\ell_1 = eat \rightarrow \forall F \ell_0 = eat) \tag{9}$$

As these formulae are in $\forall\mathrm{CTL}^*$, we can verify them via a free abstraction.

The abstract domain is defined by providing abstractions of the components which comprise the concrete domain. We choose to leave the component $\{think, eat\}^2$ the same. Formally, this means that we take an abstract domain with two elements whose concretizations are $\{think\}$ and $\{eat\}$, however, for readability we just denote these elements by $think$ and eat respectively. To abstract $I\!N \setminus \{0\}$, we choose an abstract domain in which n may take the values $\boldsymbol{e}$ and $\boldsymbol{o}$, describing the even and odd positive integers respectively, i.e, $\gamma(\boldsymbol{e}) = \{2, 4, 6, \ldots\}$ and $\gamma(\boldsymbol{o}) = \{1, 3, 5 \ldots\}$. To both abstract domains, we add a top element $\top$. The set ${}_{\alpha}\Sigma$ of abstract states is now defined as follows.

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

It is easily verified that the concretization function thus defined determines a Galois insertion (α, γ) from $\mathcal{P}(\Sigma)$ to ${}_{\alpha}\Sigma$. For the abstract initial states we have:

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

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 Def. 5.0.2. The tables (a) and (b) in Fig. 2 give the definitions of the free abstract interpretations of the transformations and tests on the abstract domain $\{\boldsymbol{e}, \boldsymbol{o}, \top\}$. 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 $\boldsymbol{e}$, indicates that $even^F(\boldsymbol{e})$ holds, i.e. (cf. Def. 5.0.2), $\exists_{n \in \gamma(\boldsymbol{e})}\ even(n)$. The entry *false* in table (a), row $+1^F$, column $(\boldsymbol{e}, \boldsymbol{e})$, means that $+1^F(\boldsymbol{e}, \boldsymbol{e})$ is false, i.e., for any minimal Y such that $+1^{\exists\exists}(\gamma(\boldsymbol{e}), Y)$, we have $\alpha(Y) \neq \boldsymbol{e}$ (see Defs. 5.0.2 and 2.0.4). From these diagrams we see for ex-

FREE:	$(\boldsymbol{e},\boldsymbol{e})$	$(\boldsymbol{e},\boldsymbol{o})$	$(\boldsymbol{e},\top)$	$(\boldsymbol{o},\boldsymbol{e})$	$(\boldsymbol{o},\boldsymbol{o})$	$(\boldsymbol{o},\top)$	$(\top,\boldsymbol{e})$	$(\top,\boldsymbol{o})$	$(\top,\top)$
$3*^F$	*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:	$\boldsymbol{e}$	$\boldsymbol{o}$	$\top$
$even^F$	*true*	*false*	*true*
odd^F	*false*	*true*	*true*

(b)

CONSTR.:	$(\boldsymbol{o},\mathbf{100})$	$(\boldsymbol{o},\boldsymbol{e})$	$(\boldsymbol{o},\boldsymbol{o})$	$(\boldsymbol{o},\top)$
$3*^F$	*false*	*false*	*true*	*false*
$+1^F$	*false*	*true*	*false*	*false*

(c)

CONSTR.:	$\mathbf{100}$	$\boldsymbol{e}$	$\boldsymbol{o}$	$\top$
$even^C$	*true*	*true*	*false*	*false*
odd^C	*false*	*false*	*true*	*false*

(d)

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

(e)

Figure 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).

ample that $/2^F$ is not functional (table (a), row $/2^F$, first two columns), illustrating that a function may become a relation when abstracted. 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 model of Fig. 3. In this model, only

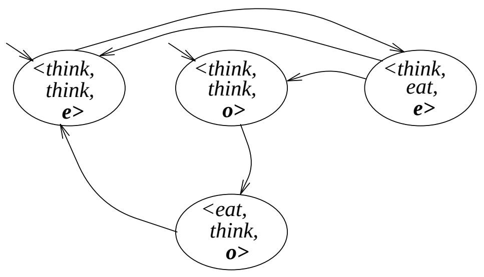


Figure 3: The free abstract model.

the reachable abstract states are shown. We see that in no reachable state the property $\ell_0 = eat \wedge \ell_1 = eat$ holds. Hence we have established property (7). Furthermore, the only path from the state where $\ell_0 = eat$ reaches $\ell_1 = eat$ within 2 steps, so we have also verified property (8).

However, the abstraction does not allow verification of the other non-starvation property, (9): a counterexample in the abstract models is the path cycling infinitely between $\langle think, think, \boldsymbol{e}\rangle$ and $\langle think, eat, \boldsymbol{e}\rangle$. It turns out that the negation of property (9) 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 $n = \boldsymbol{e}$ would have to be unraveled into infinitely many states representing the cases where n is divisible by 4, by 8, by $16, \ldots$. Hence, with our methodology, it is impossible to verify property (9) 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. In [DGG93a] we identify conditions under which a strongly preserving abstraction may be computed by a successive refinement which is guided by the form of the formula to be checked.

In order to illustrate the use of the constrained abstraction, we consider a small extension to the program: we add a third concurrent process which 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 which 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 it is always possible to reach a "restart" state. Writing $restart$ for $think_0 \wedge think_1 \wedge n = 100$, this property is expressed in CTL^* by:

$$\forall G \exists F restart \tag{10}$$

We extend the abstract domain for n by the value $\mathbf{100}$, where $\gamma(\mathbf{100}) = \{100\}$. Formula (10) being in full CTL^*, we need a mixed abstraction. The tables (c)–(e) in Fig. 2 provide some of the entries which will be needed in an abstract execution of the program. The tables from Fig. 2 have to be extended in order to take into account the new abstract value $\mathbf{100}$. Being straightforward, these extensions are left to the reader.

The resulting abstraction is depicted in Fig. 4. Solid arrows denote free transitions, dashed arrows represent constrained transitions. Note that it is not in general the case that ${}_{\alpha}R^C \subseteq {}_{\alpha}R^F$, as is illustrated by the arrow from $\langle think, eat, \boldsymbol{e}\rangle$ to $\langle think, think, \top\rangle$. Property (10) is verified on this model, interpreting the universal quantification along the free paths, and the existential quantification along

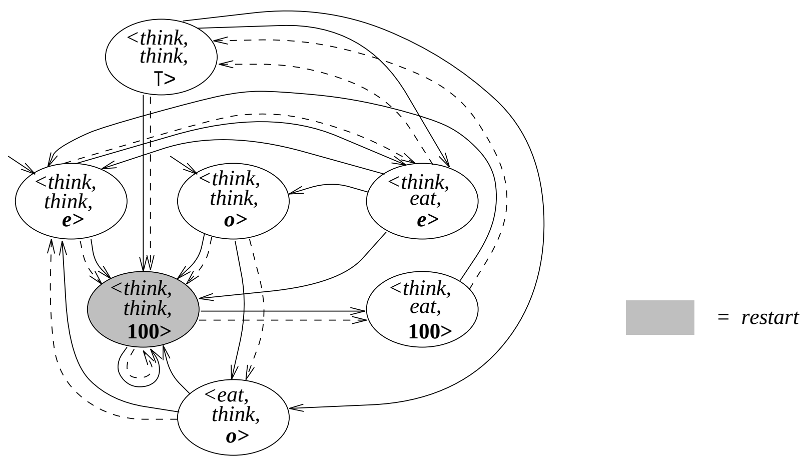


Figure 4: The mixed abstract model for the modified program.

the constrained paths. It can easily be seen that (10) holds, hence, we have established its validity in the concrete program.

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 again (without the "restart" extension). Take optimal free abstract interpretations of all operations but $3*$, for which we take the following approximation: $\overline{3*^F}(\boldsymbol{o}, \top) =$ *true* and $\overline{3*^F}(\boldsymbol{o}, \boldsymbol{e}) = \overline{3*^F}(\boldsymbol{o}, \boldsymbol{o}) =$ *false*. Furthermore, take $\langle think, think, \top\rangle$ as the abstract initial state. This gives the free abstraction of Fig. 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.

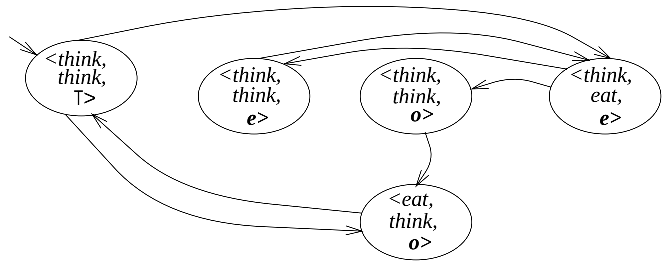


Figure 5: An approximation to the free abstraction.

7 Strong Preservation

In Sect. 5.1, we have identified conditions under which computed abstract models are optimal in the standard sense of Abstract Interpretation (see Sect. 2). 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*.

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 (9) in Sect. 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 characterize 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 ([BCG88]) in process equivalences, two (image-finite) transition systems satisfy the same CTL^* formulae if and only if they are bisimilar. In our case this implies that

7.0.1 LEMMA. CTL^* *is strongly preserved iff the relation* $\rho \subseteq \Sigma \times {}_\alpha\Sigma$ *defined by* $\rho(c,a) \Leftrightarrow \alpha(c) = a$ *is a consistent bisimulation from* (Σ, I, R) *to* $({}_\alpha\Sigma, {}_\alpha I, {}_\alpha R^F \cup {}_\alpha R^C)$.

PROOF. See [BCG88]. 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 finer than the partitioning induced by the coarsest bisimulation on Σ.

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

7.0.2 LEMMA. $\mathcal{A}$ *is an abstraction which is* ρ*-bisimilar to* $\mathcal{C}$ *if and only if* ${}_\alpha R^F = {}_\alpha R^C$.

These results show that the price for strong preservation of full CTL^* 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 CTL^* in which all the properties of interest can be expressed. Such a subset induces a coarser equivalence on the concrete states, in general. In [DGG93a] and [DGD$^+$94], we develop algorithms which can be used to reduce the system with respect to the equivalences induced by $\forall\mathrm{CTL}^*$ and by a single ACTL 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. *Homomophisms* (see, e.g., [Gin68]), used in automata theory to construct language preserving reductions of automata, can be seen as a precursor of this. Adapted to our notion of transition system, $h : \Sigma \to {}_\alpha\Sigma$ 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 [Mil71], 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, see, e.g., [Sif82, Sif83]. [Dil89, Kur89] focus on trace (linear time) semantics and universal safety and liveness properties. More recently, [CGL94, GL93] consider branching time semantics and preservation of both $\forall\mathrm{CTL}^*$ as well as CTL^*. In these papers, as well as in [Kur89], the relation between concrete and abstract model is defined by means of a homomorphism h, which induces an 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 are preserved from the abstract to the concrete model.

[CGL94] also indicates how abstract models may be computed by abstract interpretations of the operations in a program. However, their notion of approximation is based on the subset ordering on abstract transition relations; they do not have the approximation relation $\preceq$ which allows non-optimal abstract interpretations of individual operations. Also, preservation of existential properties is only possible via abstractions which are bisimilar to the concrete model, thus only allowing for relatively small reductions in the size of models.

In [LGS$^+$93], the preservation results for simulations are generalized for the case of the μ-calculus. Loiseaux *et al.* show in that paper that if there exists a consistent simulation from $\mathcal{C}$ to $\mathcal{A}$ which is total on Σ, then properties expressed in the universal μ-calculus ($\Box L_\mu$) are preserved from $\mathcal{A}$ to $\mathcal{C}$, and existential properties ($\Diamond L_\mu$) are preserved from $\mathcal{C}$ to $\mathcal{A}$. Again, preservation of the full μ-calculus is only shown for bisimilar abstractions.

A recent paper by Kelb, [Kel94], also discusses the preservation of universal and existential μ-calcu- lus properties within the framework of Abstract Interpretation. As in [LGS$^+$93], the relation between abstract and concrete systems is defined through simulations. In addition, Kelb shows how formulae from the full μ-calculus may be verified by combining two types of abstractions through a so-called *truth-failure-connection*.

[CIY94] is based on an early version ([DGG93b]) of this paper and independently develops the idea of mixing both free and constrained abstractions in a single abstract system in order to attain preservation of full CTL^*, and presents an optimality result about this.

[CR94] presents a framework for the abstract interpretation of processes that pass values.

Simulations to specify the relation between concrete and abstract systems are a generalization of our approach using Galois connections. It can easily be shown that R ρ-simulates ${}_\alpha R^F$ while ${}_\alpha R^C$ ρ-simulates R, where $\rho(c, a) \quad \Leftrightarrow \quad \alpha(c) \preceq a$. Nevertheless, the use of Galois connections has two important advantages. Firstly, it allows to derive in a uniform fashion results on the preservation of both universal and existential properties, whereas general simulation relations do not guarantee the existence of abstract models in this case. Secondly, the mere requirement that a simulation must exist in order for preservation to hold has the drawback that it leaves far too much freedom in the choice of "good" abstractions. The Galois connection 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 [LGS$^+$93].

8.1 Comparing the simulation and Galois connection approaches

We focus on free abstractions. Given a concrete transition system and a set of abstract states which is related to the concrete states by a "description relation" $\rho \subseteq \Sigma \times {}_\alpha\Sigma$ (total on Σ), the requirement $\rho^{-1}R \subseteq {}_\alpha R\rho^{-1}$ (R simulates ${}_\alpha R$) has many solutions in ${}_\alpha R$. 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 Fig. 6, where the problem is to choose an optimal ${}_\alpha R$-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

$$\rho\rho^{-1}\rho = \rho \tag{11}$$

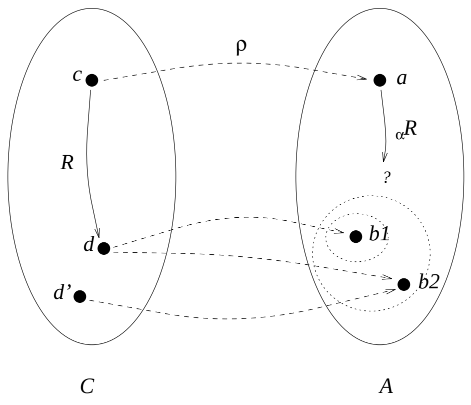


Figure 6: Abstraction with states of comparable precision.

Expressed in words, it says that if two concrete states share a description (abstract state), then they share all descriptions. 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 (11) implies that it is useless to have a ρ which is not functional. This is expressed in the following lemma (which can be found in [LGS$^+$93]). 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. Consequently, one of a and a' should be removed from $\mathcal{A}$, as, after all, the goal of abstraction is to produce minimal abstract systems.

8.1.1 LEMMA. *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 Def. 2.0.5), 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 [LGS$^+$93]), (*) 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, [LGS$^+$93] makes assumption (11), which renders their framework less general than the Galois connection approach, because, under the reasonable assumption that the abstract system does not contain bisimilar states, it forces ρ to be functional.

Consider Fig. 6 again. In our framework, the simulation relation ρ induces the following Galois insertion on sets of states: for any $a \in {}_{\alpha}\Sigma$, $\gamma(a) = \{c \mid \rho(c, a)\}$ 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 Def. 3.3.1 (item 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 generalization of the framework extending it to the analysis of reactive properties. This generalization consists in allowing the next-state relation of a non-deterministic transition system to be abstracted to a relation, and not 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 CTL^* 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 minimization based on bisimulation. This was possible by considering abstract transition systems having two different transition relations, each preserving a seperate fragment of CTL^*. 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 CTL^*. The price to be paid is that there exist formulae — and increasingly many when the abstraction becomes coarser — which 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 falsity 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 which "allowes to separate the wheat from the chaff". An abstraction function α specifies the optimal abstract model for a given concrete system, while an approximation order $\preceq$ 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 nonstandard, *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. Furthermore, 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 Sect. 6, the construction of abstract models that strongly preserve a given property of interest requires 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 which 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 the papers [DGG93a] and [DGD$^+$94]. 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, [KDG95], we apply the ideas developed in this paper and in [Kel94] to verify μ-calculus properties of a production cell ([DHKS95]) in a compositional fashion.

Acknowledgements We thank Susanne Graf for many interesting and stimulating discussions, and Nissim Francez for his helpful comments. Anthony McIsaac pointed out some mistakes. The anonymous referees are acknowledged for their valuable suggestions.

References

[BCG88] M. C. Brown, E. M. Clarke, and O. Grumberg. Characterizing finite Kripke structures in propositional temporal logic. *Theoretical Computer Science*, 59:115–131, 1988.

[BFH$^+$92] A. Bouajjani, J.-C. Fernandez, N. Halbwachs, P. Raymond, and C.Ratel. Minimal state graph generation. *Science of Computer Programming*, 18(3):247–271, June 1992.

[BKS83] R.J.R. Back and R. Kurki-Suonio. Decentralization of process nets with centralized control. In *2nd ACM SIGACT–SIGOPS Symp. on PoDC*, pages 131–142. ACM, 1983.

[CC77] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings 4th ACM Symp. Principles Prog. Lang.*, pages 238–252, Los Angeles, California, 1977.

[CC79] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Proceedings 6th ACM Symp. Principles Prog. Lang.*, pages 269–282, San Antonio, Texas, 1979.

[CC92a] P. Cousot and R. Cousot. Abstract interpretation and application to logic programs. *Journal of Logic Programming*, 13:103–179, 1992.

[CC92b] P. Cousot and R. Cousot. Abstract interpretation frameworks. *Journal of Logic and Computation*, 2(4):511–547, 1992.

[CES86] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, January 1986.

[CFM94] M. Codish, M. Falaschi, and K. Marriott. Suspension analysis for concurrent logic programs. *ACM Transactions on Programming Languages and Systems*, 16(3):649–686, May 1994.

[CGL94] E.M. Clarke, O. Grumberg, and D.E. Long. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems*, 16(5), September 1994.

[CIY94] R. Cleaveland, S.P. Iyer, and D. Yankelevich. Abstractions for preserving all CTL* formulae. Technical Report 94-03, N.C. State University, April 1994.

[CR94] R. Cleaveland and J. Riely. Testing-based abstractions for value-passing systems. In B. Jonsson and J. Parrow, editors, *CONCUR '94: Concurrency Theory*, Lecture Notes in Computer Science 836, pages 417–432. Springer-Verlag, August 1994.

[DGD$^+$94] D. Dams, R. Gerth, G. Döhmen, R. Herrmann, P. Kelb, and H. Pargmann. Model checking using adaptive state and data abstraction. In Dill [Dil94].

[DGG93a] D. Dams, R. Gerth, and O. Grumberg. Generation of reduced models for checking fragments of CTL. In C. Courcoubetis, editor, *Proc. Fifth Conf. on Computer-Aided Verification (CAV)*, number 697 in Lecture Notes in Computer Science, pages 479–490. Springer-Verlag, July 1993.

[DGG93b] D. Dams, O. Grumberg, and R. Gerth. Abstract interpretation of reactive systems: Abstractions preserving ACTL*, ECTL* and CTL*. Draft, July 1993.

[DGG94] D. Dams, O. Grumberg, and R. Gerth. Abstract interpretation of reactive systems: Abstractions preserving $\forall$CTL*, $\exists$CTL* and CTL*. In E.-R. Olderog, editor, *Proceedings of the IFIP WG2.1/WG2.2/WG2.3 Working Conference on Programming Concepts, Methods and Calculi (PROCOMET)*, IFIP Transactions, Amsterdam, June 1994. North-Holland/Elsevier.

[DHKS95] W. Damm, H. Hungar, P. Kelb, and R. Schlör. Using graphical specification languages and symbolic model checking in the verification of a production cell. In C. Lewerenz and T. Lindner, editors, *Formal Development of Reactive Systems: Case Study "Production Cell"*, number 891 in Lecture Notes in Computer Science. Springer, 1995.

[Dil89] D.L. Dill. *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits*. ACM Distinguished Dissertations. The MIT Press, 1989.

[Dil94] D. Dill, editor. *Proc. Sixth Conf. on Computer-Aided Verification (CAV)*, number 818 in Lecture Notes in Computer Science, 1994.

[EH86] E.A. Emerson and J.Y. Halpern. Sometimes and not never revisited: On branching versus linear time. *Journal of the ACM*, 33(1):151–178, 1986.

[Gin68] A. Ginzburg. *Algebraic Theory of Automata*. ACM Monograph Series. Academic Press, New York/London, 1968.

[GL93] S. Graf and C. Loiseaux. A tool for symbolic program verification and abstraction. In C. Courcoubetis, editor, *Proceedings of the Fifth Conference on Comput.-Aided Verification*, LNCS 697. Springer-Verlag, July 1993.

[Gra94] S. Graf. Verification of a distributed cache memory by using abstractions. In Dill [Dil94]. To appear in *Distributed Computing*.

[Har87] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, 1987.

[KDG95] P. Kelb, D. Dams, and R. Gerth. Efficient symbolic model checking of the full μ-calculus using compositional abstractions. To appear, March 1995.

[Kel94] P. Kelb. Model checking and abstraction: A framework preserving both truth and failure information, 1994. OFFIS, Oldenburg, Germany.

[Kri63] S. Kripke. A semantical analysis of modal logic I: normal modal propositional calculi. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 9:67–96, 1963. Announced in *Journal of Symbolic Logic*, **24**, 1959, p. 323.

[Kur89] R. P. Kurshan. Analysis of discrete event coordination. In J. W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Proceedings of the Workshop on Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness*, volume 430 of *Lecture Notes in Computer Science*, pages 414–454. Springer, 1989.

[LGS$^+$93] C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, and S. Bensalem. Property preserving abstractions for the verification of concurrent systems. Spectre technical report RTC40, LGI/IMAG, Grenoble, France, 1993. To appear in *Formal Methods in System Design*.

[LP85] O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Proceedings of the Twelfth ACM Symposium on the Principles of Programming Languages (POPL)*, pages 97–107, New Orleans, Louisiana, January 1985. ACM Press.

[Mar93] K. Marriott. Frameworks for abstract interpretation. *Acta Informatica*, 30(2):103–129, 1993.

[Mil71] R. Milner. An algebraic definition of simulation between programs. In *Proceedings of the Second International Joint Conference on Artificial Intelligence*, pages 481–489. BCS, 1971.

[Par81] D. Park. Concurrency and automata on infinite sequences. In *5th GI-Conference on Theoretical Computer Science*, number 104 in Lecture Notes in Computer Science. Springer-Verlag, 1981.

[QS81] J. P. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *Proceedings of the 5th International Symposium on Programming*, volume 137 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 1981.

[Sif82] J. Sifakis. Property preserving homomorphisms and a notion of simulation for transition systems. Rapport de Recherche 332, IMAG, Grenoble, France, November 1982.

[Sif83] J. Sifakis. Property preserving homomorphisms of transition systems. In E. Clarke and D. Kozen, editors, *4th Workshop on Logics of Programs*, number 164 in Lecture Notes in Computer Science, pages 458–473, Pittsburgh, June 1983. Springer Verlag.