Abstract interpretation of reactive systems : abstractions preserving $\forall$CTL*, $\exists$CTL* and CTL*

Eindhoven University of Technology

Department of Mathematics and Computing Science

Abstract Interpretation of Reactive Systems: Abstractions Preserving ∀CTL*, ∃CTL* and CTL*

by

D. Dams, O. Grumberg and R. Gerth

94/24

Computing Science Note 94/24
Eindhoven, May 1994

COMPUTING SCIENCE NOTES

This is a series of notes of the Computing Science Section of the Department of Mathematics and Computing Science Eindhoven University of Technology.
Since many of these notes are preliminary versions or may be published elsewhere, they have a limited distribution only and are not for review.
Copies of these notes are available from the author.

Copies can be ordered from:
Mrs. M. Philips
Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O. Box 513
5600 MB EINDHOVEN
The Netherlands
ISSN 0926-4515

editors: prof.dr.M.Rem
prof.dr.K.M.van Hee.

Abstract Interpretation of Reactive Systems: Abstractions Preserving $\forall$CTL*, $\exists$CTL* and CTL*

Dennis Dams* Orna Grumberg† Rob Gerth*§

Abstract

The need for formal verification of systems on one hand, and the advent of complex reactive systems on the other, call 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. However, a coherent basis for these notions is still lacking.

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 such abstract models directly from the text of a program, thereby avoiding the need for intermediate storage of a full-blown model. Finally, it allows trading precision for speed by computing sub-optimal abstractions; this is formalized in the notion of approximation.

However, Abstract Interpretation has traditionally been focussed on abstractions that preserve safety properties of programs, properties that hold in all states along every possible execution path. In this paper, we show how it can be extended to the analysis of different kinds of reactive properties. To this purpose, we introduce two notions of abstraction of non-deterministic systems. One preserves $\forall$CTL*, the fragment of the branching time logic CTL* which allows only universal path quantification. Another preserves $\exists$CTL*, the fragment only allowing existential path quantification. Furthermore we show that a combination of both notions preserves full CTL*.

This brings several powerful techniques from Abstract Interpretation within the reach of model checking, including the construction of abstract models by symbolic execution of programs and the use of approximations to find an optimum between precision and speed. Examples are given to illustrate these.

Keywords: verification, model checking, approximation, simulation

*Dept. of Math. and Computing Science, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands. {wsindd,robg}@win.tue.nl.

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

§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 (see [AH87] and [CC92] for an overview and bibliography). A universal property is one that holds along all possible executions of the program. The notion of safety (or: *invariance*) is used relative to an execution and means that a property holds in all states of that execution[1]. An example of a universal safety property is "whenever program P reaches location ℓ, the value of x will be positive", which may alternatively be seen as expressing invariance of the predicate "P is not in ℓ or x is positive". Such information may for example be used to perform dead-code detection; to suppress certain run-time operations like type-checks, occur-checks (in logic programs), or garbage collection; to transform programs (partial evaluation, parallelization); and to verify and debug programs.

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*. An example of an existential safety property is "there is a run of P which keeps cycling in location ℓ", indicating for example that there is a danger of livelock. 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 they 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

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

holds in the original system, it suffices to know that it holds in the abstracted system.

The different classes of properties may be formalized by expressing them in a formal logic whose formulae can be interpreted over transition systems. One such logic, which is commonly used, 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).

In this paper we investigate different ways to abstract a nondeterministic transition system. We present (in Sect. 3) notions of abstraction which preserve universal and existential properties respectively, and show how these abstractions may be combined in order to get preservation of full CTL*. Then, in Sect. 4, it is shown how these abstractions may be computed directly from a program text by "lifting" the operations of a programming language to a domain of data *descriptions*. This is illustrated by an example in Sect. 5. Sect. 6 defines a notion of *approximation* which is an ordering on abstract systems. It allows the formalization of the idea of approximate computation, which speeds up the computation of abstract models at the cost of a less precise result. The coarser the approximations made, the fewer will be the number of properties that hold in the abstract model, and hence the fewer the number of properties that may be concluded to hold in the original system (note that if a property does *not* hold on the abstracted model, this does not give any clue as to whether it holds on the original model or not). Sect. 7 compares ours to related work, and Sect. 8 concludes.

These results are not only valuable to the practice of verification, where they may lead to an advance of the limits of feasibility of the model checking approach, but also they provide an interesting generalization of Abstract Interpretation in the analysis of a more general class of properties than is usually considered (namely universal safety properties).

2 Preliminaries

2.1 Temporal logic

We assume given a set *Prop* of *propositions*. We choose to define CTL* in its positive normal form, i.e., negations only appear in front of propositions. This facilitates the definition of universal and existential CTL*. The set of *atoms* is defined by $Atom = Prop \cup \{\neg p \mid p \in Prop\}$.

2.1.1 DEFINITION. *The logic CTL* is the set of state formulae given by the following inductive definition.*

1. *If $p \in Atom$, 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 $\varphi \wedge \psi$ and $\varphi \vee \psi$.*
5. *If φ and ψ are path formulae, then so are $X\varphi$, $\varphi\ U\ \psi$ and $\varphi\ V\ \psi$.*
6. *If φ is a path formula, then $\forall\varphi$ and $\exists\varphi$ are state formulae.*

The abbreviations true, false and $\rightarrow$ are defined as usual. For a path formula φ, $F\varphi$ and $G\varphi$ abbreviate $(true\ U\ \varphi)$ and $(false\ V\ \varphi)$ respectively.

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

2.2 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 total *transition relation* over Σ (R is total means $\forall_{x\in\Sigma}\, \exists_{y\in\Sigma}\, R(x,y)$). A *path* in $\mathcal{T}$ is an infinite sequence $\pi = s_0 s_1 \cdots$ of states such that for every $i \in \mathbb{N}$, $R(s_i, s_{i+1})$. 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 $\|\cdot\| : Atom \to \mathcal{P}(\Sigma)$ specifying the interpretation of atoms over states. Intuitively, $\|p\|$ is the set of states where p holds. Transition systems thus defined are essentially the same as Kripke structures. The only difference is that we have the function $\|\cdot\|$ instead of a labelling function from Σ to sets of atoms. We require that for every proposition $p \in Prop$, $\|p\| \cap \|\neg p\| = \emptyset$.

2.2.1 DEFINITION. *Satisfaction of formulae is defined inductively as follows. Let $p \in Atom$, $s \in \Sigma$ 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* $\pi \models \psi$, $\pi \models \varphi \vee \psi$ *if* $\pi \models \varphi$ *or* $\pi \models \psi$.
5. (a) $\pi \models X\varphi$ *iff* $\pi^1 \models \varphi$.
   (b) $\pi \models \varphi\ U\ \psi$ *iff there exists* $n \in \mathbb{N}$ *such that* $\pi^n \models \psi$ *and for all* $i < n$, $\pi^i \models \varphi$.
   (c) $\pi \models \varphi\ V\ \psi$ *iff for all* $n \in \mathbb{N}$, *if* $\pi^i \not\models \varphi$ *for all* $i < n$, *then* $\pi^n \models \psi$.
6. $s \models \forall\varphi$ *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 of states or paths S, 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. Apart from state based properties expressed by formulae built from atoms and boolean connectors, properties of paths may be expressed through the *next-state* operator X, the *until* operator U and its dual V (*release*, see below). For example, $\pi \models X(p\ U\ q)$ expresses that along path π, from the next state on, p will hold in all states until we will eventually get to a state where q holds. $\pi \models XXXp$ says that p holds in the third state of π. $\pi \models Fp$ and $\pi \models Gp$ state that p will hold eventually resp. always along π. Note that, strictly speaking, path formulae are not in CTL*: they have to be preceded by $\forall$ or $\exists$, resulting in state formulae.

$s \models \forall Gp$ expresses that p is true in all states which are reachable from s. If atom 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 ($\varphi\ V\ \psi = \neg(\neg\varphi\ U\ \neg\psi)$) 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.

2.3 Abstract Interpretation

A transition sytem represents the possible behaviours of a program. For the moment, it is irrelevant what a program is. The only thing that matters right now is that there is a function, called *interpretation*, which maps every program to the system that represents its possible behaviours. In other words, the interpretation maps programs to their models. Properties of a program may be analysed by studying its model. However, the state explosion problem forces us to consider abstractions of this model. It is important to realize that such abstractions preferably are to be constructed directly from the program, and not by first building a full model and then abstracting it. The way in which such an abstract model is constructed from a program is called *abstract interpretation*. So, an abstract interpretation maps programs to abstract models.

In the rest of this section and in the following section we concentrate on the definition of abstract models and their preservation properties. Then, in Sect. 4, we will touch upon the computation of abstract models by abstract interpretation.

Abstract states The definition of an abstract system $\mathcal{A}$ starts by choosing a set ${}_\alpha\Sigma$ of *abstract states*. Intuitively, each abstract state is a description of a number of concrete states. This is formalized by a *concretization function* $\gamma : {}_\alpha\Sigma \to \mathcal{P}(\Sigma)$. For each $a \in {}_\alpha\Sigma$, $\gamma(a)$ is the set of all concrete states described by a. Conversely, every set C of concrete states has a "best" description $\alpha(C)$; α is called the *abstraction function*. "Best" means that $\alpha(C)$ is the least description of C w.r.t. the *approximation* (or *precision*) ordering $\preceq$ on ${}_\alpha\Sigma$ which is defined by

$$a \preceq a' \quad \Leftrightarrow \quad \gamma(a) \subseteq \gamma(a')$$

If $a \preceq a'$ we say that a' approximates a, or that a is more precise than a'. Usually, these requirements are captured by the condition that the pair (α, γ) forms a *Galois Insertion* from $(\mathcal{P}(\Sigma), \subseteq)$ to $({}_\alpha\Sigma, \preceq)$.

2.3.1 DEFINITION. *(α, γ) is a* Galois Insertion *from $(\mathcal{P}(\Sigma), \subseteq)$ to $({}_\alpha\Sigma, \preceq)$ iff*

1. *α and γ are total and monotonic,*
2. *for all $C \in \mathcal{P}(\Sigma)$, $\gamma \circ \alpha(C) \supseteq C$, and*
3. *for all $a \in {}_\alpha\Sigma$, $\alpha \circ \gamma(a) = a$.*

Note that these requirements imply that α is surjective, γ is injective and that there is a *top* element $\top$ in ${}_\alpha\Sigma$ for which $\alpha(\Sigma) = \top$ and $\gamma(\top) = \Sigma$. We will usually write $\alpha(c)$ for $\alpha(\{c\})$. For a path $\rho = a_0 a_1 \cdots$ in $\mathcal{A}$, $\gamma(\rho) = \{c_0 c_1 \cdots \mid c_0 c_1 \cdots$ is a path in C and $\forall_i\, c_i \in \gamma(a_i)\}$.

Different approaches to specifying the relation between abstract and concrete states may be taken. One possibility is to define a surjection $h : \Sigma \to {}_\alpha\Sigma$ which maps every concrete state to its abstraction. This is the apprach taken in, among others, [CGL92, Kur89]. Such a function h 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 $\sim$. If h is a homomorphism, then universal properties are preserved from the abstract to the concrete model. See [CGL92] for details. The restriction to functional homomorphisms is not necessary, as is demonstrated in [Sif82, Sif83]. There, it is shown that a relation $\rho \subseteq \Sigma \times {}_\alpha\Sigma$, if it is a *weak homomorphism* [Gin68] between Σ and ${}_\alpha\Sigma$, guarantees preservation of certain properties from abstract to concrete, and of others from concrete to abstract

models. Such relational homomorphisms are also known as *simulations*, a term introduced by Milner, originally in the context of deterministic programs [Mil71].

Galois Insertions (α, γ) to specify the relation between abstract and concrete states are more general than functions and less general than relations. The reasons for choosing a slightly less general formalism than relations are that (1) in the resulting framework we are able to derive in a uniform fashion results on the preservation of both universal and existential properties, whereas general relations do not always guarantee the existence of abstract models in this case; and (2) restrictions have to be imposed on relations anyway when formalizing the notion of *optimality* of abstract models. For a further discussion on this see Sect. 7.

2.4 Relation transformers

In the next section, we define transition systems over abstract states. The definition of an abstract transition relation boils down to lifting the transition relation to sets of states. We use two such liftings.

2.4.1 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.*

$$R^{\exists\exists} = \{(X, Y) \mid Y \text{ is a minimal (w.r.t. } \subseteq\text{) set such that } \exists_{x \in X}\ \exists_{y \in Y}\ R(x, y)\}$$

$$R^{\forall\exists} = \{(X, Y) \mid Y \text{ is a minimal (w.r.t. } \subseteq\text{) set such that } \forall_{x \in X}\ \exists_{y \in Y}\ R(x, y)\}$$

Note that if $R^{\exists\exists}(X, Y)$, then $|Y| = 1$. An example is given in Fig. 1. Let $R = \{(v, w), (x, y), (x, z)\}$ be the relation represented by the thin arrows. Then $(\{v, x\}, \{w\})$, $(\{v, x\}, \{y\})$ and $(\{v, x\}, \{z\})$ are in $R^{\exists\exists}$ (fat solid arrows), but, e.g., not $(\{v, x\}, \{w, y\})$. Furthermore, we have $(\{v, x\}, \{w, y\})$ and $(\{v, x\}, \{w, z\})$ in $R^{\forall\exists}$ (fat dashed arrows), but not $(\{v, x\}, \{w, y, z\})$.

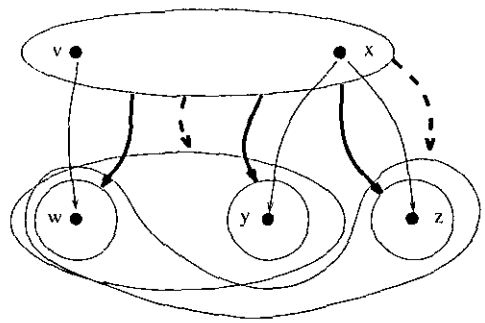

Figure 1: Relation transformers.

3 Abstract Transition Systems

Assume given a set ${}_{\alpha}\Sigma$ of abstract states and a Galois Insertion (α, γ) which determines its relation to the concrete states. We investigate how to define abstract models in such a way that certain classes of formulae are preserved from abstract to concrete model. This means, letting $\mathcal{A}$ denote the abstract model, that we want the following to hold:

$$\forall_{\varphi \in \Phi}\ \mathcal{A} \models \varphi \Rightarrow C \models \varphi \qquad \text{"preservation"} \qquad (1)$$

where Φ is some subset of CTL*.

The intention of abstracting concrete states is to *describe* them by abstract states. I.e., if $\alpha(c) = a$, then a should "behave like" c. From this point, it is natural to require preservation of formulae on the level of individual states:

$$\forall_{\varphi\in\Phi, a\in{}_\alpha\Sigma}\ (\mathcal{A}, a) \models \varphi \ \Rightarrow\ (\mathcal{C}, \gamma(a)) \models \varphi \qquad \text{"statewise preservation"} \qquad (2)$$

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:

1. a function ${}_\alpha\|\cdot\|$ specifying the interpretation of atoms over abstract states,
2. a set ${}_\alpha I$ of abstract initial states, and
3. an abstract transition relation ${}_\alpha R$.

These points are considered in the following subsections.

3.1 Valuation of atoms

We start with considering $\Phi = Atom$ in requirement (2): we want the truth of atoms to be preserved. As atoms are asserting something about individual states and not about computations, the requirement thus obtained does not concern the definition of the abstract transition relation. In order to satisfy (2), we must have for every atom p: $(\mathcal{A}, a) \models p \ \Rightarrow\ (\mathcal{C}, \gamma(a)) \models p$. 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 formulae to hold in each abstract state.

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

This choice determines the valuation of atoms in abstract states. Namely, the relation $\models \subseteq {}_\alpha\Sigma \times Atom$ is defined as in clause 1 of Def. 2.2.1, where s now denotes an abstract state and $\|\cdot\|$ has to be replaced by ${}_\alpha\|\cdot\|$. That this choice is "optimal", in the sense that as many as possible atoms hold in each abstract state, is implied by the following lemma.

3.1.2 LEMMA. *For every $a \in {}_\alpha\Sigma$ and $p \in Atom$, $(\mathcal{A}, a) \models p \ \Leftrightarrow\ (\mathcal{C}, \gamma(a)) \models p$.*

PROOF. Direct from the definitions of $\models$, $\|\cdot\|$ and ${}_\alpha\|\cdot\|$. □

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 atomic predicates 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

Requirement (2) does not imply (1). The left-hand side in (1), $\mathcal{A} \models \varphi$, is an abbreviation for $\forall_{a \in {}_\alpha I}\ (\mathcal{A}, a) \models \varphi$. By (2), this implies $\forall_{c \in \cup\{\gamma(a) \mid a \in {}_\alpha I\}}\ (\mathcal{C}, c) \models \varphi$. A sufficient condition for this to imply the right-hand side of (1), $\mathcal{C} \models \varphi$, which is an abbreviation for $\forall_{c \in I}\ (\mathcal{C}, c) \models \varphi$, is that $\cup\{\gamma(a) \mid a \in {}_\alpha I\} \supseteq I$. When we have preservation, (1), a property φ of the concrete model can be verified by checking it on the abstract model, i.e., by verifying that $\mathcal{A} \models \varphi$. Preferably, this condition is as weak as possible. That means that the set of abstract initial states has to be as small as possible. In general, it is not possible to choose ${}_\alpha I$ such that $\cup\{\gamma(a) \mid a \in {}_\alpha I\} = I$. However, it can easily be seen that 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. ${}_\alpha I = \{\alpha(c) \mid c \in I\}$

As argued above, statewise preservation, (2), now implies preservation, (1). The following property states something slightly more general, namely that if statewise preservation holds, and furthermore some property holds in a superset of the abstract initial states, then it holds in (the initial states of) the concrete system. Its proof is straightforward.

3.2.2 PROPERTY. *If* $\forall_{\varphi \in \Phi, a \in {}_\alpha\Sigma}\ (\mathcal{A}, a) \models \varphi \Rightarrow (\mathcal{C}, \gamma(a)) \models \varphi$ *and* ${}_\alpha I' \supseteq {}_\alpha I$, *then* $\forall_{\varphi \in \Phi}\ (\mathcal{A}, {}_\alpha I') \models \varphi \Rightarrow \mathcal{C} \models \varphi$.

3.3 Abstract transition relations

We now investigate the definition of the abstract transition relation: when is there a transition from abstract state a to b? As we said earlier, the abstract state a is intended to be a description of the concrete states in $\gamma(a)$. Hence, the possible transitions from a should depend on those possible from each of the states in $\gamma(a)$. Many 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., a transition function 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 $\gamma(a)$: ${}_\alpha f(a) = \alpha(f(\gamma(a)))$[2].

In the current case, we have a non-deterministic concrete transition relation, and the question is how this non-determinism is going to be reflected on the abstract side. We consider two possible points of view. One is to allow in a all nondeterminism that is present in *any* state from $\gamma(a)$: if a state c in $\gamma(a)$ can make a transition to d, then a can make a transition to the description $\alpha(d)$ of d. We call this the *free* abstraction. The other possibility is to only allow in a the nondeterminism that is present in *all* states of $\gamma(a)$: a can only make transitions to those states which are the description of sets of states which are successor to every c in $\gamma(a)$. This we call the *constrained* abstraction. Formally:

3.3.1 DEFINITION.

1. ${}_\alpha R^F(a,b) \Leftrightarrow \exists_Y\ R^{\exists\exists}(\gamma(a), Y) \wedge \alpha(Y) = b$
2. ${}_\alpha R^C(a,b) \Leftrightarrow \exists_Y\ R^{\forall\exists}(\gamma(a), Y) \wedge \alpha(Y) = b$

Note that by the requirement of minimality of Y (in Def. 2.4.1 of $R^{\exists\exists}$ and $R^{\forall\exists}$), it is not in general the case that ${}_\alpha R^F \subseteq {}_\alpha R^C$. We denote $\mathcal{A}^F = ({}_\alpha\Sigma, {}_\alpha I, {}_\alpha R^F)$ and $\mathcal{A}^C = ({}_\alpha\Sigma, {}_\alpha I, {}_\alpha R^C)$. As a result of these choices, we have the following preservation properties for paths.

[2] Functions are extended to sets pointwisely.

3.3.2 LEMMA.

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

2. *Let $a \in {}_\alpha\Sigma$, $c \in \gamma(a)$ and ρ be an $(\mathcal{A}^C, a)$-path. Then there exists a $(\mathcal{C}, c)$-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 $(\mathcal{C}, c)$-path and $\pi \in \gamma(\rho)$.

□

3.4 Preservation of $\forall$CTL* and $\exists$CTL*

Satisfaction over abstract models Having defined abstract models, we can check formulae over them. The satisfaction relation $\models \subseteq {}_\alpha\Sigma \times \text{CTL}^*$ is defined by Def. 2.2.1, where $\|\cdot\|$ has to be replaced by ${}_\alpha\|\cdot\|$. We then have the following result.

3.4.1 THEOREM.

1. *For every $\varphi \in \forall CTL^*$, $\quad \mathcal{A}^F \models \varphi \;\Rightarrow\; \mathcal{C} \models \varphi$.*

2. *For every $\varphi \in \exists CTL^*$, $\quad \mathcal{A}^C \models \varphi \;\Rightarrow\; \mathcal{C} \models \varphi$.*

PROOF. By Property 3.2.2, it suffices to prove statewise preservation in both cases.

1. Let $\mathcal{A}$ denote $\mathcal{A}^F$. We prove the first item by induction on the structure of φ. So, for state formulae we prove that for every state $a \in {}_\alpha\Sigma$

   - $(\mathcal{A}, a) \models \varphi \;\Rightarrow\; (\mathcal{C}, \gamma(a)) \models \varphi$

   and for path formulae we prove that for every path ρ in $\mathcal{A}$

   - $(\mathcal{A}, \rho) \models \varphi \;\Rightarrow\; (\mathcal{C}, \gamma(\rho)) \models \varphi$.

   There are 6 cases, corresponding to those in Def. 2.1.1.

   ("base" case:)

1. TO PROVE: For every atom $p \in Atom$ and every state $a \in {}_a\Sigma$
$$(\mathcal{A}, a) \models p \;\Rightarrow\; (\mathcal{C}, \gamma(a)) \models p$$
PROOF: Follows directly from Def. 3.1.1.

("step" cases:)

2–5. The cases that φ is a conjunction or disjunction of state or path formulae (cases 2 and 5), a state formula interpreted over a path (case 3), or a path formula with principal operator X, U or V (case 4), are straightforward.

6. TO PROVE: If φ is a path formula and
(ih) for every path ρ in $\mathcal{A}$, $(\mathcal{A}, \rho) \models \varphi \;\Rightarrow\; (\mathcal{C}, \gamma(\rho)) \models \varphi$,
then
for every state $a \in {}_a\Sigma$, $(\mathcal{A}, a) \models \forall\varphi \;\Rightarrow\; (\mathcal{C}, \gamma(a)) \models \forall\varphi$.
PROOF: Let a be a state such that $(\mathcal{A}, a) \models \forall\varphi$, let $c \in \gamma(a)$, and consider a $(\mathcal{C}, c)$-path π. By Lemma 3.3.2, we know that there exists an $(\mathcal{A}, a)$-path ρ such that $\pi \in \gamma(\rho)$, so, because $(\mathcal{A}, a) \models \forall\varphi$, $(\mathcal{A}, \rho) \models \varphi$. By (ih) we have $(\mathcal{C}, \pi) \models \varphi$. So $(\mathcal{C}, \gamma(a)) \models \forall\varphi$.

2. Cases 1–5 are the same as in item 1, provided that $\mathcal{A}$ now denotes $\mathcal{A}^C$. Case 6 is a straightforward variation of case 6 under item 1; it uses Lemma 3.3.2.

□

In Property 3.2.2 we saw that the set of abstract initial states may be enlarged without violating preservation. Similarly, the following theorem states that it is "safe" to approximate the abstract transition relations by taking a superset of the transition relation ${}_aR^F$ or a subset[3] of ${}_aR^C$. This result will be used in the following section. In order to distinguish these notions of approximation from the approximation ordering $\preceq$ (which will be extended to relations in Sect. 6), we will refer to them as $\supseteq$-approximation and $\subseteq$-approximation.

3.4.2 THEOREM. *Let* $R_1 \supseteq {}_aR^F$, $\mathcal{A}_1 = ({}_a\Sigma, {}_aI, R_1)$, $R_2 \subseteq {}_aR^C$ *and* $\mathcal{A}_2 = ({}_a\Sigma, {}_aI, R_2)$.

1. *For every* $\varphi \in \forall CTL^*$, $\quad \mathcal{A}_1 \models \varphi \;\Rightarrow\; \mathcal{C} \models \varphi$.
2. *For every* $\varphi \in \exists CTL^*$, $\quad \mathcal{A}_2 \models \varphi \;\Rightarrow\; \mathcal{C} \models \varphi$.

PROOF. By Property 3.2.2, it is sufficient to prove that

1. for every $\varphi \in \forall\text{CTL}^*$ and every $a \in {}_a\Sigma$, $\quad (\mathcal{A}_1, a) \models \varphi \;\Rightarrow\; (\mathcal{A}^F, a) \models \varphi$, and
2. for every $\varphi \in \exists\text{CTL}^*$ and every $a \in {}_a\Sigma$, $\quad (\mathcal{A}_2, a) \models \varphi \;\Rightarrow\; (\mathcal{A}^C, a) \models \varphi$.

Together with Thm. 3.4.1 these imply Thm. 3.4.2. Both items are easily proven by induction on the structure of φ. When φ has $\forall$ or $\exists$ as principal operator, the argument is as follows.

1. Suppose $\varphi = \forall\varphi'$. Because $R_1 \supseteq {}_aR^F$, every path in $\mathcal{A}^F$ is also a path in $\mathcal{A}_1$. Therefore, if φ' holds for all $(\mathcal{A}_1, a)$-paths, it also holds for all $(\mathcal{A}^F, a)$-paths.
2. Suppose $\varphi = \exists\varphi'$. Because $R_2 \subseteq {}_aR^C$, every path in $\mathcal{A}_2$ is also a path in $\mathcal{A}^C$. Therefore, if φ' holds for some $(\mathcal{A}_2, a)$-path, it also holds for some $(\mathcal{A}^C, a)$-path.

□

[3] Remember however that the transition relation has to remain total.

3.5 Mixed Abstraction: preservation of CTL*

When we have a free as well as a constrained abstraction of a system, formulae from both ∀CTL* and ∃CTL* can be verified. However, the union of ∀CTL* and ∃CTL* is not CTL*. In order to have an abstraction which preserves all CTL* formulae, we can combine the free and constrained transition relations in one model, which we call *mixed abstraction*. A similar idea is presented in [Kel94]. There, CTL* formulae are interpreted over a *pair* of abstract models.

3.5.1 DEFINITION. *The* mixed abstraction *is the system* $\mathcal{A}^M = ({}_\alpha\Sigma, {}_\alpha I, {}_\alpha R^M)$ *where* ${}_\alpha R^M = {}_\alpha R^F \cup {}_\alpha R^C$. *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. 2.2.1: clause 6 is replaced by

6′. $s \models \forall\varphi$ *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 (both the $(\supseteq, \subseteq)$-approximations to be introduced below as well as the $\preceq$-approximation in Sect. 6). We then have:

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

PROOF. A combination of the proofs of 1 and 2 in Thm. 3.4.1. □

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 [Mil71] 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.

The result of Thm. 3.4.2 is also adapted for mixed abstractions in a straightforward way. Because superset- and subset-approximations are combined, we speak of $(\supseteq, \subseteq)$-*approximations* of mixed abstractions.

3.5.3 THEOREM. *Let* $R_1 \supseteq {}_\alpha R^F$, $R_2 \subseteq {}_\alpha R^C$ *and* $\mathcal{A} = ({}_\alpha\Sigma, {}_\alpha I, R_1 \cup R_2)$. *Then for every* $\varphi \in CTL^*$, $\mathcal{A} \models \varphi \Rightarrow C \models \varphi$.

PROOF. Similar to the proof of Thm. 3.4.2. □

4 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. From [CDY94] we extract the following informal definition:

An abstract interpretation is 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 simple language which is based on *action systems* [BKS83]. 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[4] of their values into the new vector $\bar{x}'$ (i ranges over some index set I). Executing an action means evaluating its condition c_i and, 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. The conditions and transformations will typically be specified in terms of more elementary operations over (components of) the values in Val.

We chose this language because (a) by its simplicity, notably the uniform treatment of data and control, it allows for a comprehensive presentation of the following results on abstract interpretation—choosing a more concrete language would needlessly complicate matters, yet, (b) it contains rudimentary forms of the common notions of assignment, test and loop, which will help to grasp the idea of how to abstractly interpret operations in "real" programming languages.

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

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

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 $\supseteq$- and $\subseteq$-approximations (see Thm. 3.4.2) for the free and constrained models respectively of a program may be computed by interpreting the operators in the program correspondingly.

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

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

[4] 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. Hence we denote both the concrete and the abstract transformations in the same way.

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

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

The following lemma expresses that the abstract interpretations given above can be used to compute $\subseteq$- and $\supseteq$-approximations.

4.0.3 LEMMA. $\quad {}_\alpha\widehat{R^F} \supseteq {}_\alpha R^F \quad$ *and* $\quad {}_\alpha\widehat{R^C} \subseteq {}_\alpha R^C$

PROOF. Let $a, b \in {}_\alpha Val$.

1. $(a,b) \in {}_\alpha R^F$

   $\Leftrightarrow$ { apply Def. 3.3.1 of ${}_\alpha R^F$, Def. 2.4.1 of $R^{\exists\exists}$ and Def. 4.0.1 of R }

   $\exists_Y$ [Y is a minimal set such that $\exists_{\bar v\in\gamma(a),\bar w\in Y}$ $[\exists_{i\in I}\ [c_i(\bar v) \wedge t_i(\bar v,\bar w)]] \wedge \alpha(Y) = b$]

   $\Leftrightarrow$ { $\alpha(Y) = b$ does not depend on i, so we may move the $\exists_{i\in I}$ outside }

   $\exists_{i\in I}\ \exists_Y$ [Y is a minimal set such that $\exists_{\bar v\in\gamma(a),\bar w\in Y}$ $[c_i(\bar v) \wedge t_i(\bar v,\bar w)] \wedge \alpha(Y) = b$]

   $\Rightarrow$ { weaken by distributing the $\exists_{\bar v\in\gamma(a),\bar w\in Y}$ over the first $\wedge$ }

   $\exists_{i\in I}\ \exists_Y$ [Y is a minimal set such that $\exists_{\bar v\in\gamma(a)}$ $[c_i(\bar v)] \wedge \exists_{\bar v\in\gamma(a),\bar w\in Y}$ $[t_i(\bar v,\bar w)] \wedge \alpha(Y) = b$]

   $\Leftrightarrow$ { move the $\exists_Y$ inside }

   $\exists_{i\in I}\ [\exists_{\bar v\in\gamma(a)}\ [c_i(\bar v)] \wedge \exists_Y$ [Y is a minimal set such that $\exists_{\bar v\in\gamma(a),\bar w\in Y}$ $[t_i(\bar v,\bar w)] \wedge \alpha(Y) = b]]$

   $\Leftrightarrow$ { apply Def. 2.4.1 of $t_i(\bar v,\bar w)^{\exists\exists}$ and Def. 4.0.2 of $c_i^F(a)$, $t_i^F(a,b)$, and ${}_\alpha\widehat{R^F}$ }

   $(a,b) \in {}_\alpha\widehat{R^F}$

2. $(a,b) \in {}_\alpha\widehat{R^C}$

   $\Leftrightarrow$ { apply Def. 4.0.2 of ${}_\alpha\widehat{R^C}$, $c_i^C(a)$ and $t_i^C(a,b)$, and Def. 2.4.1 of $t_i(\gamma(a),Y)^{\forall\exists}$ }

   $\exists_{i\in I}\ [\forall_{\bar v\in\gamma(a)}\ [c_i(\bar v)] \wedge \exists_Y$ [Y is a minimal set such that $\forall_{\bar v\in\gamma(a)}\ \exists_{\bar w\in Y}$ $[t_i(\bar v,\bar w)] \wedge \alpha(Y) = b]]$

   $\Leftrightarrow$ { $\forall_{\bar v\in\gamma(a)}$ $[c_i(\bar v)]$ does not depend on Y, so we may move the $\exists_Y$ outside }

   $\exists_{i\in I}\ \exists_Y$ [Y is a minimal set such that $\forall_{\bar v\in\gamma(a)}$ $[c_i(\bar v)] \wedge \forall_{\bar v\in\gamma(a)}\ \exists_{\bar w\in Y}$ $[t_i(\bar v,\bar w)] \wedge \alpha(Y) = b$]

   $\Leftrightarrow$ { $c_i(\bar v)$ does not depend on Y, so we may bring the $\exists_{\bar w\in Y}$ outside the first $\wedge$ }

   $\exists_{i\in I}\ \exists_Y$ [Y is a minimal set such that $\forall_{\bar v\in\gamma(a)}\ \exists_{\bar w\in Y}$ $[c_i(\bar v) \wedge t_i(\bar v,\bar w)] \wedge \alpha(Y) = b$]

   $\Rightarrow$ { weaken by moving the $\exists_{i\in I}$ inside, over the other quantifiers }

   $\exists_Y$ [Y is a minimal set such that $\forall_{\bar v\in\gamma(a)}\ \exists_{\bar w\in Y}$ $[\exists_{i\in I}\ [c_i(\bar v) \wedge t_i(\bar v,\bar w)]] \wedge \alpha(Y) = b$]

   $\Leftrightarrow$ { apply Def. 4.0.1 of R, Def. 2.4.1 of $R^{\forall\exists}$ and Def. 3.3.1 of ${}_\alpha R^C$ }

   $(a,b) \in {}_\alpha R^C$

□

The (approximations to) free, constrained and mixed abstractions thus computed preserve the formulae of $\forall$CTL*, $\exists$CTL* and CTL* respectively.

4.0.4 COROLLARY.

1. *For every* $\varphi \in \forall CTL^*$, $\widehat{\mathcal{A}^F} \models \varphi \Rightarrow \mathcal{C} \models \varphi$.
2. *For every* $\varphi \in \exists CTL^*$, $\widehat{\mathcal{A}^C} \models \varphi \Rightarrow \mathcal{C} \models \varphi$.
3. *For every* $\varphi \in CTL^*$, $\widehat{\mathcal{A}^M} \models \varphi \Rightarrow \mathcal{C} \models \varphi$.

PROOF. From Thms. 3.4.2, 3.5.3 and Lemma 4.0.3. □

If the c_i and t_i are built up from more elementary operations, the abstract interpretations of the c_i and t_i can be defined in terms of those, however, it must be seen to that the result of Lemma 4.0.3 is maintained.

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. Depending on the property φ to be checked, free and/or constrained interpretations should be given; these have to satisfy Def. 4.0.2. Then, the (free, constrained or mixed) 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 equivalent to a CTL* formula.

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. 7.1 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. 4.0.2, 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 [DGG93] and [DGD+94], a method is developed which aims at full automation of these steps.

5 Example

In this section we illustrate the theory on a small example. Consider the system consisting of two concurrent processes depicted in Fig. 2, which is a parallel variant of the famous $3n + 1$ 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.

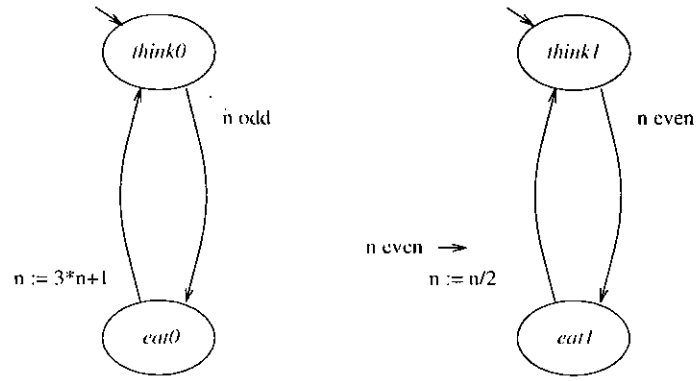

Figure 2: The dining mathematicians.

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 that along every execution

- the mathematicians have mutually exclusive access to the dining room, and
- mathematician 1 will not starve, i.e., when mathematician 0 is eating, then, eventually, mathematician 1 will get access to the dining room.

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.

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

$$\ell_1 = think,\ even(n) \longrightarrow \ell_1 := eat$$
$$\ell_1 = eat,\ even(n) \longrightarrow \ell_1 := think,\ n := n/2$$

Note that although we translate a concurrent into a sequential system, we do not have to "unfold" the inherent non-determinism: the two processes which describe the mathematicians can be recognized in the first two lines and last two lines of this program. 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. 4.0.1, 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.

$$\forall G \neg(\ell_0 = eat \wedge \ell_1 = eat) \tag{3}$$

$$\forall G(\ell_0 = eat \rightarrow \forall F \ell_1 = eat) \tag{4}$$

As both formulae are in ∀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. 4.0.2. The tables (a) and (b) in Fig. 3 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. For completeness, Fig. 3 (c) gives the table with free abstract interpretations of the tests $= think$ and $= eat$ (to be considered single symbols) on the domain $\{think, eat, \top\}$. The tables have to be interpreted as indicated by the following examples. The entry *true* in row $even^F$, column $\boldsymbol{e}$ of table (b) indicates that $even^F(\boldsymbol{e})$ holds, i.e. (cf. Def. 4.0.2), $\exists_{n \in \gamma(\boldsymbol{e})}\ even(n)$. The entry ***false*** in row $+1^F$, column $(\boldsymbol{e}, \boldsymbol{e})$ of table (a) means that $+1^F(\boldsymbol{e}, \boldsymbol{e})$ is false, i.e., for any Y such that $+1^{\exists\exists}(\gamma(\boldsymbol{e}), Y)$, we have $\alpha(Y) \neq \boldsymbol{e}$ (see Defs. 4.0.2 and 2.4.1). From these diagrams we see for example that $/2^F$ is not functional, illustrating that a function may become a relation when abstracted.

Now we can abstractly interpret the program over this abstract domain, using the interpretations given in the tables. We start in the two initial states $\langle think, think, \boldsymbol{e}\rangle$ and $\langle think, think, \boldsymbol{o}\rangle$. Consider for example $\langle think, think, \boldsymbol{e}\rangle$. According to the tables (b) and (c), the only action from the program whose condition c_i evaluates to *true* is the 3rd line. As a result of the corresponding transformation $(\ell_1 := eat)$, the (only) successor of $\langle think, think, \boldsymbol{e}\rangle$ is $\langle think, eat, \boldsymbol{e}\rangle$. Continuing from this state, the only action that applies is the 4th line of the program. From the entries for the operation $/2^F$ on the value $\boldsymbol{e}$, we see that the results can be both $\boldsymbol{e}$ and $\boldsymbol{o}$. Hence, we get free abstract transitions from $\langle think, eat, \boldsymbol{e}\rangle$ back to $\langle think, think, \boldsymbol{e}\rangle$, and also to $\langle think, think, \boldsymbol{o}\rangle$. Such an abstract execution yields the abstract model of Fig. 4. We see that in no state the property $\ell_0 = eat \wedge \ell_1 = eat$ holds. Hence we have established property (3). Furthermore, the only path from the state where $\ell_0 = eat$ reaches $\ell_1 = eat$ within 2 steps, so we have also verified property (4).

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:

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

(b)

FREE:	*think*	*eat*	$\top$
$(= think)^F$	*true*	*false*	*true*
$(= eat)^F$	*false*	*true*	*true*

(c)

Figure 3: Free abstract interpretations of operations (a) and tests (b and c).

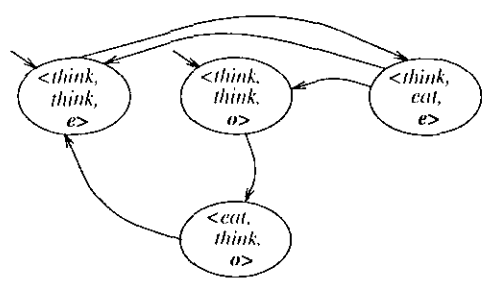

Figure 4: The free abstract model.

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

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:

$$\forall G \exists F restart \tag{5}$$

We extend the abstract domain for n by the value $\mathbf{100}$, where $\gamma(\mathbf{100}) = \{100\}$. Formula (5) being in full CTL*, we need a mixed abstraction. Instead of providing the constrained abstract interpretations of all tests and operations over all abstract values, Figure 5 only provides those entries which will be needed in an abstract execution of the program. Also the tables from Fig. 3 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.

CONSTR.:	$(e, \mathbf{100})$	(e, e)	(e, o)	$(e, \top)$
$/2^F$	*false*	*false*	*false*	*true*

(a)

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

(b)

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

(c)

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

(d)

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

(e)

Figure 5: Constrained abstract interpretations of operations (a, b, c) and tests (d and e).

The resulting abstraction is depicted in Fig. 6. Solid arrows denote free transitions, dashed arrows represent constrained transitions. Note that it is not in general the case that ${}_a R^C \not\subseteq {}_a R^F$, as is illustrated by the arrow from $\langle think, eat, e \rangle$ to $\langle think, think, \top \rangle$. Property (5) is verified on this model by interpreting the universal quantification along the free paths, and the existential quantification along the constrained paths. It can easily be seen that (5) holds, hence, we have established its validity in the concrete program.

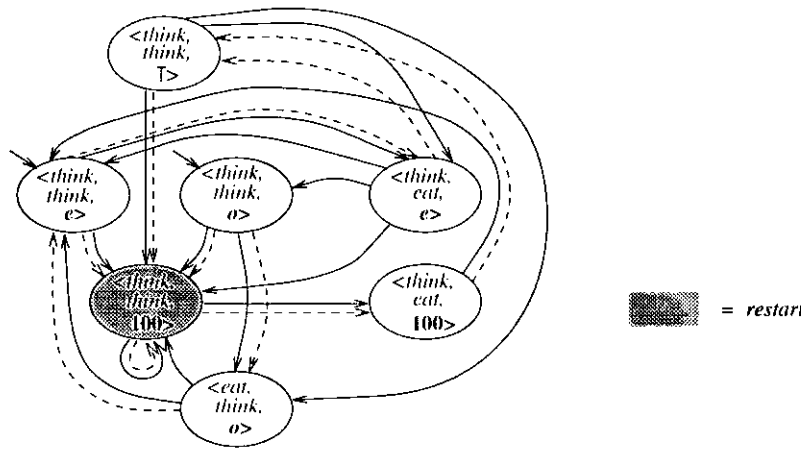

Figure 6: The mixed abstract model for the modified program.

6 Approximations

So far we have considered abstractions which were *optimal* in the sense that in the definition of the free and the constrained transition relations, the successor of an abstract state is the *best* description of certain sets of concrete states. This appears from the fact that these descriptions are obtained by the application of α in Def. 3.3.1. In practical applications of Abstract Interpretation, *approximations*, w.r.t. $\preceq$, of such best descriptions are often taken instead. One reason is that the computation of optimal abstract interpretations may be too complex. Another reason is that even if the abstract interpretation is optimal, it can be cumbersome to actually prove this. The following definition extends the notion of approximation ($\succeq$, see Sect. 2.3) to paths and to models[5], where also the initial states may be approximated. It applies to free, constrained as well as mixed abstractions. In the rest of this paper, the term "approximation", unless explicitly stated otherwise, refers to the notion defined here and should not be confused with $\supseteq$- and $\subseteq$-approximation.

6.0.1 DEFINITION. *Let* $\overline{\mathcal{A}} = ({}_\alpha\Sigma, {}_\alpha\overline{I}, \overline{{}_\alpha R^F \cup {}_\alpha R^C})$ *be a transition system where transitions in* $\overline{{}_\alpha R^F}$ *are called free and in* $\overline{{}_\alpha R^C}$ *are called constrained.*

1. *For paths* $\rho = a_0 a_1 \cdots$ *and* $\overline{\rho} = \overline{a}_0 \overline{a}_1 \cdots$, $\rho \preceq \overline{\rho}$ *iff* $a_0 \preceq \overline{a}_0$, $a_1 \preceq \overline{a}_1, \ldots$.
2. $\overline{\mathcal{A}} \succeq^F \mathcal{A}$ *iff (a) and (b)i,* $\overline{\mathcal{A}} \succeq^C \mathcal{A}$ *iff (a) and (b)ii, and* $\overline{\mathcal{A}} \succeq \mathcal{A}$ *iff* $\overline{\mathcal{A}} \succeq^F \mathcal{A}$ *and* $\overline{\mathcal{A}} \succeq^C \mathcal{A}$*:*
   (a) *for every* $a \in {}_\alpha I$ *there is an* $\overline{a} \in {}_\alpha\overline{I}$ *such that* $a \preceq \overline{a}$
   (b) *for every* $a \in {}_\alpha\Sigma$
      i. *for every free* $(\mathcal{A}, a)$*-path* ρ *there is a free* $(\overline{\mathcal{A}}, a)$*-path* $\overline{\rho} \succeq \rho$
      ii. *for every constrained* $(\overline{\mathcal{A}}, a)$*-path* $\overline{\rho}$ *there is a constrained* $(\mathcal{A}^C, a)$*-path* $\rho \preceq \overline{\rho}$.

We have the following preservation results for approximations.

[5]The extension of $\preceq$ to models is not a partial order anymore, but a pre-order.

6.0.2 THEOREM.

1. *If* $\overline{\mathcal{A}} \succeq^F \mathcal{A}$, *then for every* $\varphi \in \forall CTL^*$, $\quad \overline{\mathcal{A}} \models \varphi \;\Rightarrow\; \mathcal{A} \models \varphi$.
2. *If* $\overline{\mathcal{A}} \succeq^C \mathcal{A}$, *then for every* $\varphi \in \exists CTL^*$, $\quad \overline{\mathcal{A}} \models \varphi \;\Rightarrow\; \mathcal{A} \models \varphi$.
3. *If* $\overline{\mathcal{A}} \succeq \mathcal{A}$, *then for every* $\varphi \in CTL^*$, $\quad \overline{\mathcal{A}} \models \varphi \;\Rightarrow\; \mathcal{A} \models \varphi$.

PROOF. We prove 3; the other proofs are simplifications of this. By condition (a) in Def. 6.0.1, it suffices to prove statewise preservation. We prove this by induction on the structure of φ. The 6 cases correspond to those in Def. 2.1.1 of CTL*. For *state* formulae $\varphi \in$ CTL* we prove that

for every $a \in {}_a\Sigma$, $(\overline{\mathcal{A}}, a) \models \varphi \;\Rightarrow\; (\mathcal{A}, a) \models \varphi$.

For *path* formulae $\varphi \in$ CTL*, we prove that

for every $a \in {}_a\Sigma$ and every $(\mathcal{A}, a)$-path ρ and $(\overline{\mathcal{A}}, a)$-path $\overline{\rho}$ such that $\rho \preceq \overline{\rho}$,
$(\overline{\mathcal{A}}, \overline{\rho}) \models \varphi \;\Rightarrow\; (\mathcal{A}, \rho) \models \varphi$.

1–5. The cases that φ is an atom (case 1), a conjunction or disjunction of state or path formulae (cases 2 and 5), a state formula interpreted over a path (case 3), or a path formula with principal operator X, U or V (case 4), are straightforward.

6. TO PROVE: If $\varphi \in$ CTL* is a path formula such that

   (ih) for every $a \in {}_a\Sigma$ and every $(\mathcal{A}, a)$-path ρ and $(\overline{\mathcal{A}}, a)$-path $\overline{\rho}$ such that $\rho \preceq \overline{\rho}$,
   $(\overline{\mathcal{A}}, \overline{\rho}) \models \varphi \;\Rightarrow\; (\mathcal{A}, \rho) \models \varphi$

   then

   (a) for every $a \in {}_a\Sigma$, $\quad (\overline{\mathcal{A}}, a) \models \forall\varphi \;\Rightarrow\; (\mathcal{A}, a) \models \forall\varphi$, and
   (b) for every $a \in {}_a\Sigma$, $\quad (\overline{\mathcal{A}}, a) \models \exists\varphi \;\Rightarrow\; (\mathcal{A}, a) \models \exists\varphi$.

   PROOF:

   (a) Let $a \in {}_a\Sigma$ be a state such that $(\overline{\mathcal{A}}, a) \models \forall\varphi$. Let ρ be a free $(\mathcal{A}, a)$-path. By Def. 6.0.1 of approximation, there exists a free $(\overline{\mathcal{A}}, a)$-path $\overline{\rho}$ such that $\rho \preceq \overline{\rho}$. So, because $(\overline{\mathcal{A}}, a) \models \forall\varphi$, $(\overline{\mathcal{A}}, \overline{\rho}) \models \varphi$. By (ih), this implies $(\mathcal{A}, \rho) \models \varphi$. So, $(\mathcal{A}, a) \models \forall\varphi$.
   (b) Let $a \in {}_a\Sigma$ be a state such that $(\overline{\mathcal{A}}, a) \models \exists\varphi$. So there exists a constrained $(\overline{\mathcal{A}}, a)$-path $\overline{\rho}$ such that $(\overline{\mathcal{A}}, \overline{\rho}) \models \varphi$. By Def. 6.0.1 of approximation, this implies that there exists a constrained $(\mathcal{A}, a)$-path ρ such that $\rho \preceq \overline{\rho}$. By (ih), this implies $(\mathcal{A}, \rho) \models \varphi$. So, $(\mathcal{A}, a) \models \exists\varphi$.

□

6.0.3 COROLLARY.

1. *If* $\overline{\mathcal{A}} \succeq^F \mathcal{A}^F$, *then for every* $\varphi \in \forall CTL^*$, $\quad \overline{\mathcal{A}} \models \varphi \;\Rightarrow\; C \models \varphi$.
2. *If* $\overline{\mathcal{A}} \succeq^C \mathcal{A}^C$, *then for every* $\varphi \in \exists CTL^*$, $\quad \overline{\mathcal{A}} \models \varphi \;\Rightarrow\; C \models \varphi$.
3. *If* $\overline{\mathcal{A}} \succeq \mathcal{A}^M$, *then for every* $\varphi \in CTL^*$, $\quad \overline{\mathcal{A}} \models \varphi \;\Rightarrow\; C \models \varphi$.

PROOF. From Thm. 6.0.2 and Thms. 3.4.1 and 3.5.2. □

6.1 Computing approximations

Approximations to models may also be computed by abstract interpretation of a program. In this case, the abstract interpretations of conditions and transformations do not have to be optimal. Here, we will illustrate such approximation for the transformation operators (t_i). It can easily be extended for the conditions (c_i) as well.

6.1.1 DEFINITION. *The definition of approximation is extended to abstract interpretations of the transformation operators, as follows. For abstract operations*[6] $t, \overline{t} \in {}_{\alpha}Val \times {}_{\alpha}Val$,

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

Such approximate free and constrained interpretations $\overline{t_i^F} \succeq t_i^F$ *and* $\overline{t_i^C} \succeq t_i^C$ *(for all* $i \in I$*) induce the abstract models* $\overline{\mathcal{A}^F} = ({}_{\alpha}\Sigma, {}_{\alpha}I, \overline{{}_{\alpha}R^F})$, $\overline{\mathcal{A}^C} = ({}_{\alpha}\Sigma, {}_{\alpha}I, \overline{{}_{\alpha}R^C})$, *and* $\overline{\mathcal{A}^M} = ({}_{\alpha}\Sigma, {}_{\alpha}I, \overline{{}_{\alpha}R^F} \cup \overline{{}_{\alpha}R^C})$, *where:*

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

6.1.2 LEMMA. *1.* $\overline{\mathcal{A}^F} \succeq \widehat{\mathcal{A}^F}$, *2.* $\overline{\mathcal{A}^C} \succeq \widehat{\mathcal{A}^C}$, *and 3.* $\overline{\mathcal{A}^M} \succeq \widehat{\mathcal{A}^M}$.

PROOF.

1. Let $a \in {}_{\alpha}\Sigma$. We have to prove that for every $(\widehat{\mathcal{A}^F}, a)$-path ρ there exists an $(\overline{\mathcal{A}^F}, a)$-path $\overline{\rho}$ such that $\rho \preceq \overline{\rho}$.

   Let $\rho = aa_1a_2\cdots$ be an $(\widehat{\mathcal{A}^F}, a)$-path. We show that for any $\overline{a} \succeq a$, there is an $(\overline{\mathcal{A}^F}, \overline{a})$-path $\overline{\rho}$ such that $\rho \preceq \overline{\rho}$. (In particular, this implies that there is an $(\overline{\mathcal{A}^F}, a)$-path $\overline{\rho}$ such that $\rho \preceq \overline{\rho}$.) Let $\overline{a} \succeq a$. We show that there exists $\overline{a}_1 \succeq a_1$ such that $\overline{{}_{\alpha}R^F}(\overline{a}, \overline{a}_1)$. We have $\widehat{{}_{\alpha}R^F}(a, a_1)$. By Def. 4.0.2 of $\widehat{{}_{\alpha}R^F}$, this 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. 4.0.2 of c_i^F and t_i^F and Def. 2.4.1 of $\exists\exists$), so $\exists_{i \in I}\, [c_i^F(\overline{a}) \wedge t_i^F(\overline{a}, a_1)]$. By Def. 6.1.1 of $\overline{t_i^F}$, there exists $\overline{a}_1 \succeq a_1$ such that $\exists_{i \in I}\, [c_i^F(\overline{a}) \wedge t_i^F(\overline{a}, \overline{a}_1)]$, i.e., $\overline{{}_{\alpha}R^F}(\overline{a}, \overline{a}_1)$. This argument may be applied inductively (cf. the proof of Lemma 3.3.2) to construct the $(\overline{\mathcal{A}^F}, \overline{a})$-path $\overline{\rho} = \overline{a}\,\overline{a}_1\overline{a}_2\cdots$.

2. Let $a \in {}_{\alpha}\Sigma$. We have to prove that for every $(\overline{\mathcal{A}^C}, a)$-path $\overline{\rho}$ there exists an $(\widehat{\mathcal{A}^C}, a)$-path ρ such that $\rho \preceq \overline{\rho}$.

   Let $\overline{\rho} = aa_1a_2\cdots$ be an $(\overline{\mathcal{A}^C}, a)$-path. We show that for any $a' \preceq a$, there is an $(\widehat{\mathcal{A}^C}, a')$-path ρ such that $\rho \preceq \overline{\rho}$. Let $a' \preceq a$. 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. 6.1.1 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 $a' \preceq a$,

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

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. 4.0.2 of c_i^C and t_i^C and Def. 2.4.1 of $\cdot^{\forall\exists}$). So $\exists_{i\in I}\,[c_i^C(a') \wedge t_i^C(a', a_1')]$, i.e., $\circ\widehat{R^C}(a', a_1')$, and, by transitivity of $\preceq$, $a_1' \preceq a_1$. This argument may be applied inductively (cf. the proof of Lemma 3.3.2) to construct the $(\widehat{\mathcal{A}^C}, a')$-path $\rho = a'a_1'a_2' \cdots$.

3. From 1 and 2.

□

So, we can compute approximations to the abstract models by choosing non-optimal abstract interpretations $\overline{t_i}$ of operations in the program. As an example, 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: $\overline{3*^F}(\boldsymbol{o}, \top) = \mathit{true}$ and $\overline{3*^F}(\boldsymbol{o}, \boldsymbol{e}) = \overline{3*^F}(\boldsymbol{o}, \boldsymbol{o}) = \mathit{false}$. Furthermore, take $\langle \mathit{think}, \mathit{think}, \top \rangle$ as the abstract initial state. This gives the free abstraction of Fig. 7, 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.

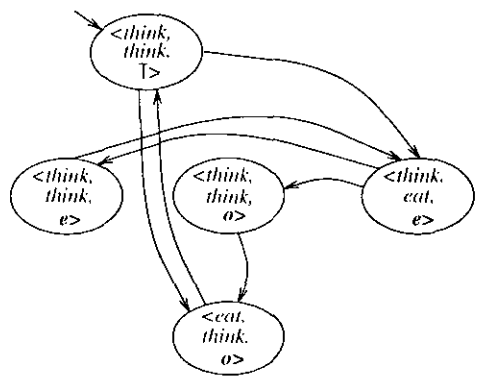

Figure 7: An approximation to the free abstraction.

7 Related Work

Property preserving abstractions of reactive systems have been the topic of intensive research lately. The results can be classified according to the type of semantics that is considered and the class of properties to be preserved. For example, [Dil89, Kur89] focus on trace (linear time) semantics and universal safety and liveness properties. More recently, [CGL92, GL93] consider branching time semantics and preservation of both $\forall$CTL* as well as CTL*. In those two papers, the relation between concrete and abstract model is defined by means of a *homomorphism*. [CGL92] 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 subsetordering on abstract transition relations (cf. our Thm. 3.4.2, item 1); 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 automata theory, property preserving homomorphisms provide a classical method to construct language preserving reductions of automata. 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. We recall the definition of simulation. Juxtaposition of relations denotes composition as usual.

7.0.1 DEFINITION. *A relation* $\rho \subseteq \Sigma \times {}_{\alpha}\Sigma$ *is a* simulation (from $\mathcal{C}$ to $\mathcal{A}$) *iff* $\rho^{-1}R \subseteq {}_{\alpha}R\rho^{-1}$. $\mathcal{C}$ simulates $\mathcal{A}$ *iff there exists a simulation from* $\mathcal{C}$ *to* $\mathcal{A}$.

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 $\mathcal{C}$ simulates $\mathcal{A}$, 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.

The framework of Galois Insertions, which forms a special case of these simulations, has the advantage that the distinction between the notions of *abstraction* and *approximation* renders the approach closer to the original approach to Abstract Interpretation [CC77, CC79] and allows a better control of the precision of abstractions. This point is further discussed below.

A recent paper by Kelb, [Kel94], also discusses the preservation of universal and existential μ-calculus 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.*

7.1 Precision

It can be shown that $\mathcal{C}$ simulates $\mathcal{A}^F$ and that $\mathcal{A}^C$ simulates $\mathcal{C}$. So, the notion of simulation is a generalization of that of a Galois Insertion (conversely, not every simulation determines a Galois Insertion). However, we will show that once *optimality* is taken into account, the situation changes and the Galois Insertion framework is the more general. We focus on the free abstraction.

In the sequel we need the following definition.

7.1.1 DEFINITION. *Let* A *and* B *be sets and* $R \subseteq A \times B$. *The functions* $post_R : \mathcal{P}(A) \to \mathcal{P}(B)$ *and* $pre_R : \mathcal{P}(B) \to \mathcal{P}(A)$ *are defined as follows.*

$post_R(X) = \{b \in B \mid \exists_{a \in X}\ R(a,b)\}$.

$pre_R(Y) = \{a \in A \mid \exists_{b \in Y}\ R(a,b)\}$.

Let $\widetilde{pre}_R$ *denote the dual of* pre_R, *i.e.,* $\widetilde{pre}_R(Y) = \overline{pre_R(\overline{Y})}$ *(overlining denotes complement here).*

In [LGS+93], it is shown that the existence of a simulation ρ from $\mathcal{C}$ to $\mathcal{A}$ is equivalent to the existence of a Galois *Connection*[7] (φ, ψ) from $(\mathcal{P}(\Sigma), \subseteq)$ to $(\mathcal{P}({}_{\alpha}\Sigma), \subseteq)$. More precisely, they show the following.

- If ρ is a simulation, then $(post_\rho, \widetilde{pre}_\rho)$ is a Galois Connection from $(\mathcal{P}(\Sigma), \subseteq)$ to $(\mathcal{P}({}_{\alpha}\Sigma), \subseteq)$ and

$$post_\rho \circ pre_R \circ \widetilde{pre}_\rho \subseteq pre_{{}_{\alpha}R} \tag{6}$$

  ($\circ$ denotes function composition).

[7] A Galois Connection is a generalization of a Galois Insertion where $\varphi \circ \psi$ is required to be $\subseteq id$ rather than $= id$.

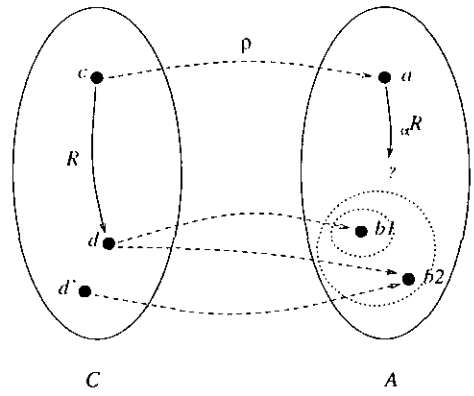

Figure 8: Abstraction with states of comparable precision.

- Conversely, if (φ, ψ) is a Galois Connection from $(\mathcal{P}(\Sigma), \subseteq)$ to $(\mathcal{P}({}_{\alpha}\Sigma), \subseteq)$ for which $\varphi \circ pre_R \circ \psi \subseteq pre_{{}_{\alpha}R}$, then there exists a simulation $\rho \subseteq \Sigma \times {}_{\alpha}\Sigma$ such that $\varphi = post_{\rho}$ and $\psi = \widetilde{pre}_{\rho}$.

This alternative characterization, (6), of simulation is claimed to be useful to compute the abstract relation ${}_{\alpha}R$ when $\mathcal{C}$, ${}_{\alpha}\Sigma$ and ρ are given. Requirement (6) has many solutions ${}_{\alpha}R$, of which the minimal ones are interesting from the point of view of property preservation. Namely, the smaller is ${}_{\alpha}R$, the greater the number of properties that hold in $\mathcal{A}$. When ρ is total, then $\widetilde{pre}_{\rho} \subseteq pre_{\rho}$, and taking ${}_{\alpha}R$ such that

$$post_{\rho} \circ pre_R \circ pre_{\rho} = pre_{{}_{\alpha}R} \tag{7}$$

is claimed to define an "interesting" abstraction. However, this is not always an optimal choice, as illustrated by the following example.

7.1.2 EXAMPLE. *Consider the systems $\mathcal{C}$ and $\mathcal{A}$ depicted in Fig. 8. The problem is to choose an optimal ${}_{\alpha}R$-successor of a such that $\rho^{-1}R \subseteq {}_{\alpha}R\rho^{-1}$, or, equivalently, (6) above is satisfied. If we take $pre_{{}_{\alpha}R}$ such that (7) holds, then both b_1 and b_2 become successors of a. This is not an optimal choice, since taking only b_1 would suffice in order to satisfy (6).*

In order to avoid such "bad" abstractions, Loiseaux *et al.* propose a condition under which (7) yields an optimal abstract relation. Optimality of ${}_{\alpha}R$ in this case means that the resulting abstract system is bisimilar to any other abstract system $\mathcal{A}' = ({}_{\alpha}\Sigma, {}_{\alpha}R')$ with ${}_{\alpha}R'$ a minimal solution of (6). This condition is

$$\rho\rho^{-1}\rho = \rho \tag{8}$$

Expressed in words, it says that if two concrete states share a description, 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 (8) 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 $c\rho a$ and $c\rho a'$ $(a \neq a')$ 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 the goal of abstraction is to produce minimal abstract systems after all.

7.1.3 LEMMA. *If ρ is total, ${}_\alpha R$ defined by (7), 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. 7.0.1), 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 Def. (7) is equivalent to ${}_\alpha R = \rho^{-1}R\rho$ (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. □

So, if one wants to be able to distinguish optimal abstractions from approximations, then assumption (8) has to be made, in which case the framework of [LGS+93] becomes less general because, under the reasonable assumption that the abstract system does not contain bisimilar states, it forces ρ to be functional.

Consider Fig. 8 again. In our framework, the simulation relation ρ induces the following Galois Insertion on sets of states: for any $a \in {}_\alpha\Sigma$, $\gamma(a) = pre_\rho(\{a\})$ and for any $C \subseteq \Sigma$, $\alpha(C) = \bigwedge\{a \mid \gamma(a) = C\}$, where $\bigwedge$ denotes the meet operation corresponding to the ordering $\preceq$ (the existence of a Galois Insertion guarantees that this meet exists). 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.

8 Conclusions

We have presented a generalization of the framework of Abstract Interpretation 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 us to verify, via the abstraction, 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. Two possible ways to abstract a transition relation were presented, differing in the way they abstract from the choice points occurring in a non-deterministic system. The *free* abstract relation ${}_\alpha R^F$ yields an abstract model which may be used to verify properties in ∀CTL* (universal CTL* properties). The *constrained* abstraction ${}_\alpha R^C$ results in a model for which existential properties are preserved: ∃CTL*. By combining the two abstract transition relations within one model, which we called the *mixed abstraction*, it is furthermore possible to verify full CTL* while obtaining better reductions than is the case with minimization based on bisimulation. The price that has to be payed is that there will be formulae which do not hold in the abstraction, and neither do their negations. We have chosen CTL* instead of the more expressive μ-calculus because of its better readability. However, the presented results generalize to the μ-calculus as well.

We showed that, after fixing the abstract domain (the set of abstract states), both the free and the constrained abstract models can be constructed directly from the text of a program, thereby avoiding the intermediate construction of the full concrete model. This construction is possible by associating non-standard, *abstract* interpretations with the operators in a programming language which allows their evaluation over *descriptions* of data. To this purpose, we chose a simple programming language and defined free and constrained abstract interpretations of its tests and operations. The abstract transition relations thus computed, ${}_\alpha\widehat{R^F}$ and ${}_\alpha\widehat{R^C}$, were shown to be $\supseteq$- and $\subseteq$-approximations respectively to the free and constrained relations ${}_\alpha R^F$ and ${}_\alpha R^C$ (Lemma 4.0.3), keeping up the preservation results. It was illustrated by an example that this technique can be applied to verify properties of systems with an infinite state space.

The human interaction required in this approach consists in providing abstract interpretations of elementary operations over data descriptions. The notion of *approximation*, formalized as an ordering $\preceq$ on the abstract domain, and not to be confused with the $\supseteq$- and $\subseteq$-approximations mentioned above, offers a degree of freedom herein. By providing approximations to the abstract interpretations, 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.

Finally, we compared our approach to related work on property preserving abstractions. The main conclusion is that, once the notion of precision is taken into account, Galois Insertions are a more general means to relate concrete to abstract states than homomorphisms or simulation relations.

Further work In the light of the quest for fully automated verification methods, two points remain open problems: the choice of an abstract domain that is appropriate to allow verification of the properties of interest, and the computation of abstract interpretations of operations over such a domain. We are currently investigating these problems; see the papers [DGG93] 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.

Acknowledgements We thank Susanne Graf for many interesting and stimulating discussions, and Nissim Francez for his helpful comments.

References

[AH87] S. Abramsky and C. Hankin, editors. *Abstract Interpretation of Declarative Languages.* Ellis Horwood, 1987.

[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.

[CC92] P. Cousot and R. Cousot. Comparing the Galois connection and widening/narrowing approaches to abstract interpretation. In *Proceedings of the Conference on Programming Language Implementation and Logic Programming (PLILP'92)*, pages 269–295. Springer-Verlag, August 1992. Lecture Notes in Computer Science 631.

[CDY94] M. Codish, D. Dams, and E. Yardeni. Bottom-up abstract interpretation of logic programs. *Journal of Theoretical Computer Science*, 1(124):93–125, February 1994.

[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.

[CFM] M. Codish, M. Falaschi, and K. Marriott. Suspension analysis for concurrent logic programs. Submitted and revised for *ACM Transactions on Programming Languages and Systems*, (In Press); Also: University of Pisa Technical Report TR-12/92; December 1991.

[CGL92] E. M. Clarke, O. Grumberg, and D. E. Long. Model checking and abstraction. In *Proc. 19th Ann. ACM Symp. on Principles of Prog. Lang.*, January 1992.

[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].

[DGG93] 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)*, Lecture Notes in Computer Science 697, pages 479–490. Springer Verlag, July 1993.

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

[Dil94] D. Dill, editor. *Proc. Sixth Conference on Comput.-Aided Verification*, 1994.

[EH86] E.A. Emerson and J.Y. Halpern. 'Sometimes' and 'Not Never' revisited: on branching time versus linear time temporal logic. *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, *Proc. 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*.

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

[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-Verlag, 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.

[Mil71] R. Milner. An algebraic definition of simulation between programs. In *Proc. 2nd Int. Joint Conf. on Artificial Intelligence*, pages 481–489. BCS, 1971.

[QS81] J.P. Quielle 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-Verlag, 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.

Computing Science Notes

Department of Mathematics and Computing Science
Eindhoven University of Technology

In this series appeared:

91/01	D. Alstein	Dynamic Reconfiguration in Distributed Hard Real-Time Systems, p. 14.
91/02	R.P. Nederpelt H.C.M. de Swart	Implication. A survey of the different logical analyses "if...,then...", p. 26.
91/03	J.P. Katoen L.A.M. Schoenmakers	Parallel Programs for the Recognition of *P*-invariant Segments, p. 16.
91/04	E. v.d. Sluis A.F. v.d. Stappen	Performance Analysis of VLSI Programs, p. 31.
91/05	D. de Reus	An Implementation Model for GOOD, p. 18.
91/06	K.M. van Hee	SPECIFICATIEMETHODEN, een overzicht, p. 20.
91/07	E.Poll	CPO-models for second order lambda calculus with recursive types and subtyping, p. 49.
91/08	H. Schepers	Terminology and Paradigms for Fault Tolerance, p. 25.
91/09	W.M.P.v.d.Aalst	Interval Timed Petri Nets and their analysis, p.53.
91/10	R.C.Backhouse P.J. de Bruin P. Hoogendijk G. Malcolm E. Voermans J. v.d. Woude	POLYNOMIAL RELATORS, p. 52.
91/11	R.C. Backhouse P.J. de Bruin G.Malcolm E.Voermans J. van der Woude	Relational Catamorphism, p. 31.
91/12	E. van der Sluis	A parallel local search algorithm for the travelling salesman problem, p. 12.
91/13	F. Rietman	A note on Extensionality, p. 21.
91/14	P. Lemmens	The PDB Hypermedia Package. Why and how it was built, p. 63.
91/15	A.T.M. Aerts K.M. van Hee	Eldorado: Architecture of a Functional Database Management System, p. 19.
91/16	A.J.J.M. Marcelis	An example of proving attribute grammars correct: the representation of arithmetical expressions by DAGs, p. 25.

91/17	A.T.M. Aerts P.M.E. de Bra K.M. van Hee	Transforming Functional Database Schemes to Relational Representations, p. 21.
91/18	Rik van Geldrop	Transformational Query Solving, p. 35.
91/19	Erik Poll	Some categorical properties for a model for second order lambda calculus with subtyping, p. 21.
91/20	A.E. Eiben R.V. Schuwer	Knowledge Base Systems, a Formal Model, p. 21.
91/21	J. Coenen W.-P. de Roever J.Zwiers	Assertional Data Reification Proofs: Survey and Perspective, p. 18.
91/22	G. Wolf	Schedule Management: an Object Oriented Approach, p. 26.
91/23	K.M. van Hee L.J. Somers M. Voorhoeve	Z and high level Petri nets, p. 16.
91/24	A.T.M. Aerts D. de Reus	Formal semantics for BRM with examples, p. 25.
91/25	P. Zhou J. Hooman R. Kuiper	A compositional proof system for real-time systems based on explicit clock temporal logic: soundness and complete ness, p. 52.
91/26	P. de Bra G.J. Houben J. Paredaens	The GOOD based hypertext reference model, p. 12.
91/27	F. de Boer C. Palamidessi	Embedding as a tool for language comparison: On the CSP hierarchy, p. 17.
91/28	F. de Boer	A compositional proof system for dynamic proces creation, p. 24.
91/29	H. Ten Eikelder R. van Geldrop	Correctness of Acceptor Schemes for Regular Languages, p. 31.
91/30	J.C.M. Baeten F.W. Vaandrager	An Algebra for Process Creation, p. 29.
91/31	H. ten Eikelder	Some algorithms to decide the equivalence of recursive types, p. 26.
91/32	P. Struik	Techniques for designing efficient parallel programs, p. 14.
91/33	W. v.d. Aalst	The modelling and analysis of queueing systems with QNM-ExSpect, p. 23.
91/34	J. Coenen	Specifying fault tolerant programs in deontic logic, p. 15.

91/35	F.S. de Boer J.W. Klop C. Palamidessi	Asynchronous communication in process algebra, p. 20.
92/01	J. Coenen J. Zwiers W.-P. de Roever	A note on compositional refinement, p. 27.
92/02	J. Coenen J. Hooman	A compositional semantics for fault tolerant real-time systems, p. 18.
92/03	J.C.M. Baeten J.A. Bergstra	Real space process algebra, p. 42.
92/04	J.P.H.W.v.d.Eijnde	Program derivation in acyclic graphs and related problems, p. 90.
92/05	J.P.H.W.v.d.Eijnde	Conservative fixpoint functions on a graph, p. 25.
92/06	J.C.M. Baeten J.A. Bergstra	Discrete time process algebra, p.45.
92/07	R.P. Nederpelt	The fine-structure of lambda calculus, p. 110.
92/08	R.P. Nederpelt F. Kamareddine	On stepwise explicit substitution, p. 30.
92/09	R.C. Backhouse	Calculating the Warshall/Floyd path algorithm, p. 14.
92/10	P.M.P. Rambags	Composition and decomposition in a CPN model, p. 55.
92/11	R.C. Backhouse J.S.C.P.v.d.Woude	Demonic operators and monotype factors, p. 29.
92/12	F. Kamareddine	Set theory and nominalisation, Part I, p.26.
92/13	F. Kamareddine	Set theory and nominalisation, Part II, p.22.
92/14	J.C.M. Baeten	The total order assumption, p. 10.
92/15	F. Kamareddine	A system at the cross-roads of functional and logic programming, p.36.
92/16	R.R. Seljée	Integrity checking in deductive databases; an exposition, p.32.
92/17	W.M.P. van der Aalst	Interval timed coloured Petri nets and their analysis, p. 20.
92/18	R.Nederpelt F. Kamareddine	A unified approach to Type Theory through a refined lambda-calculus, p. 30.
92/19	J.C.M.Baeten J.A.Bergstra S.A.Smolka	Axiomatizing Probabilistic Processes: ACP with Generative Probabilities, p. 36.
92/20	F.Kamareddine	Are Types for Natural Language? P. 32.

92/21	F.Kamareddine	Non well-foundedness and type freeness can unify the interpretation of functional application, p. 16.
92/22	R. Nederpelt F.Kamareddine	A useful lambda notation, p. 17.
92/23	F.Kamareddine E.Klein	Nominalization, Predication and Type Containment, p. 40.
92/24	M.Codish D.Dams Eyal Yardeni	Bottum-up Abstract Interpretation of Logic Programs, p. 33.
92/25	E.Poll	A Programming Logic for Fω, p. 15.
92/26	T.H.W.Beelen W.J.J.Stut P.A.C.Verkoulen	A modelling method using MOVIE and SimCon/ExSpect, p. 15.
92/27	B. Watson G. Zwaan	A taxonomy of keyword pattern matching algorithms, p. 50.
93/01	R. van Geldrop	Deriving the Aho-Corasick algorithms: a case study into the synergy of programming methods, p. 36.
93/02	T. Verhoeff	A continuous version of the Prisoner's Dilemma, p. 17
93/03	T. Verhoeff	Quicksort for linked lists, p. 8.
93/04	E.H.L. Aarts J.H.M. Korst P.J. Zwietering	Deterministic and randomized local search, p. 78.
93/05	J.C.M. Baeten C. Verhoef	A congruence theorem for structured operational semantics with predicates, p. 18.
93/06	J.P. Veltkamp	On the unavoidability of metastable behaviour, p. 29
93/07	P.D. Moerland	Exercises in Multiprogramming, p. 97
93/08	J. Verhoosel	A Formal Deterministic Scheduling Model for Hard Real-Time Executions in DEDOS, p. 32.
93/09	K.M. van Hee	Systems Engineering: a Formal Approach Part I: System Concepts, p. 72.
93/10	K.M. van Hee	Systems Engineering: a Formal Approach Part II: Frameworks, p. 44.
93/11	K.M. van Hee	Systems Engineering: a Formal Approach Part III: Modeling Methods, p. 101.
93/12	K.M. van Hee	Systems Engineering: a Formal Approach Part IV: Analysis Methods, p. 63.
93/13	K.M. van Hee	Systems Engineering: a Formal Approach

		Part V: Specification Language, p. 89.
93/14	J.C.M. Baeten J.A. Bergstra	On Sequential Composition, Action Prefixes and Process Prefix, p. 21.
93/15	J.C.M. Baeten J.A. Bergstra R.N. Bol	A Real-Time Process Logic, p. 31.
93/16	H. Schepers J. Hooman	A Trace-Based Compositional Proof Theory for Fault Tolerant Distributed Systems, p. 27
93/17	D. Alstein P. van der Stok	Hard Real-Time Reliable Multicast in the DEDOS system, p. 19.
93/18	C. Verhoef	A congruence theorem for structured operational semantics with predicates and negative premises, p. 22.
93/19	G-J. Houben	The Design of an Online Help Facility for ExSpect, p.21.
93/20	F.S. de Boer	A Process Algebra of Concurrent Constraint Programming, p. 15.
93/21	M. Codish D. Dams G. Filé M. Bruynooghe	Freeness Analysis for Logic Programs - And Correctness?, p. 24.
93/22	E. Poll	A Typechecker for Bijective Pure Type Systems, p. 28.
93/23	E. de Kogel	Relational Algebra and Equational Proofs, p. 23.
93/24	E. Poll and Paula Severi	Pure Type Systems with Definitions, p. 38.
93/25	H. Schepers and R. Gerth	A Compositional Proof Theory for Fault Tolerant Real-Time Distributed Systems, p. 31.
93/26	W.M.P. van der Aalst	Multi-dimensional Petri nets, p. 25.
93/27	T. Kloks and D. Kratsch	Finding all minimal separators of a graph, p. 11.
93/28	F. Kamareddine and R. Nederpelt	A Semantics for a fine λ-calculus with de Bruijn indices, p. 49.
93/29	R. Post and P. De Bra	GOLD, a Graph Oriented Language for Databases, p. 42.
93/30	J. Deogun T. Kloks D. Kratsch H. Müller	On Vertex Ranking for Permutation and Other Graphs, p. 11.
93/31	W. Körver	Derivation of delay insensitive and speed independent CMOS circuits, using directed commands and production rule sets, p. 40.
93/32	H. ten Eikelder and H. van Geldrop	On the Correctness of some Algorithms to generate Finite Automata for Regular Expressions, p. 17.

93/33	L. Loyens and J. Moonen	ILIAS, a sequential language for parallel matrix computations, p. 20.
93/34	J.C.M. Baeten and J.A. Bergstra	Real Time Process Algebra with Infinitesimals, p.39.
93/35	W. Ferrer and P. Severi	Abstract Reduction and Topology, p. 28.
93/36	J.C.M. Baeten and J.A. Bergstra	Non Interleaving Process Algebra, p. 17.
93/37	J. Brunekreef J-P. Katoen R. Koymans S. Mauw	Design and Analysis of Dynamic Leader Election Protocols in Broadcast Networks, p. 73.
93/38	C. Verhoef	A general conservative extension theorem in process algebra, p. 17.
93/39	W.P.M. Nuijten E.H.L. Aarts D.A.A. van Erp Taalman Kip K.M. van Hee	Job Shop Scheduling by Constraint Satisfaction, p. 22.
93/40	P.D.V. van der Stok M.M.M.P.J. Claessen D. Alstein	A Hierarchical Membership Protocol for Synchronous Distributed Systems, p. 43.
93/41	A. Bijlsma	Temporal operators viewed as predicate transformers, p. 11.
93/42	P.M.P. Rambags	Automatic Verification of Regular Protocols in P/T Nets, p. 23.
93/43	B.W. Watson	A taxomomy of finite automata construction algorithms, p. 87.
93/44	B.W. Watson	A taxonomy of finite automata minimization algorithms, p. 23.
93/45	E.J. Luit J.M.M. Martin	A precise clock synchronization protocol,p.
93/46	T. Kloks D. Kratsch J. Spinrad	Treewidth and Patwidth of Cocomparability graphs of Bounded Dimension, p. 14.
93/47	W. v.d. Aalst P. De Bra G.J. Houben Y. Kornatzky	Browsing Semantics in the "Tower" Model, p. 19.
93/48	R. Gerth	Verifying Sequentially Consistent Memory using Interface Refinement, p. 20.

94/01	P. America M. van der Kammen R.P. Nederpelt O.S. van Roosmalen H.C.M. de Swart	The object-oriented paradigm, p. 28.
94/02	F. Kamareddine R.P. Nederpelt	Canonical typing and Π-conversion, p. 51.
94/03	L.B. Hartman K.M. van Hee	Application of Marcov Decision Processe to Search Problems, p. 21.
94/04	J.C.M. Baeten J.A. Bergstra	Graph Isomorphism Models for Non Interleaving Process Algebra, p. 18.
94/05	P. Zhou J. Hooman	Formal Specification and Compositional Verification of an Atomic Broadcast Protocol, p. 22.
94/06	T. Basten T. Kunz J. Black M. Coffin D. Taylor	Time and the Order of Abstract Events in Distributed Computations, p. 29.
94/07	K.R. Apt R. Bol	Logic Programming and Negation: A Survey, p. 62.
94/08	O.S. van Roosmalen	A Hierarchical Diagrammatic Representation of Class Structure, p. 22.
94/09	J.C.M. Baeten J.A. Bergstra	Process Algebra with Partial Choice, p. 16.
94/10	T. verhoeff	The testing Paradigm Applied to Network Structure. p. 31.
94/11	J. Peleska C. Huizing C. Petersohn	A Comparison of Ward & Mellor's Transformation Schema with State- & Activitycharts, p. 30.
94/12	T. Kloks D. Kratsch H. Müller	Dominoes, p. 14.
94/13	R. Seljée	A New Method for Integrity Constraint checking in Deductive Databases, p. 34.
94/14	W. Peremans	Ups and Downs of Type Theory, p. 9.
94/15	R.J.M. Vaessens E.H.L. Aarts J.K. Lenstra	Job Shop Scheduling by Local Search, p. 21.
94/16	R.C. Backhouse H. Doornbos	Mathematical Induction Made Calculational, p. 36.
94/17	S. Mauw M.A. Reniers	An Algebraic Semantics of Basic Message Sequence Charts, p. 9.

94/18	F. Kamareddine R. Nederpelt	Refining Reduction in the Lambda Calculus, p. 15.
94/19	B.W. Watson	The performance of single-keyword and multiple-keyword pattern matching algorithms, p. 46.
94/20	R. Bloo F. Kamareddine R. Nederpelt	Beyond β-Reduction in Church's λ→, p. 22.
94/21	B.W. Watson	An introduction to the Fire engine: A C++ toolkit for Finite automata and Regular Expressions.
94/22	B.W. Watson	The design and implementation of the FIRE engine: A C++ toolkit for Finite automata and regular Expressions.
94/23	S. Mauw and M.A. Reniers	An algebraic semantics of Message Sequence Charts, p. 43.