Abstract interpretation of reactive systems: preservation of CTL*

Citation for published version (APA):

Document status and date:
Published: 01/01/1995

Document Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher’s website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.

Download date: 11. Nov. 2019
Abstract Interpretation of Reactive Systems: Preservation of CTL*

by

Dennis Dams, Oma Grumberg and Rob Gerth

Computing Science Report 95/16
Eindhoven, May 1995
Abstract Interpretation of Reactive Systems: 
Preservation of CTL*

Dennis Dams†  Orna Grumberg§  Rob Gerth¶

Abstract

The advent of ever more complex reactive systems in increasingly critical areas calls for the development of automated verification techniques. Model checking is one such technique, which has proven quite successful. However, the state explosion problem remains the stumbling block in many situations. Recent experience indicates that solutions are to be found in the application of techniques for property preserving abstraction and successive approximation of models. Most such applications have so far been based on the property-preserving characteristics of simulation relations. A major drawback of all these results is that they do not offer a satisfactory formalization of the notions of precision and optimality of abstractions. Furthermore, the use of simulation relations poses difficulties when formalizing the preservation of both existential and universal properties over the same abstract domain.

The theory of Abstract Interpretation offers a framework for the definition and justification of property preserving abstractions. Furthermore, it provides a method for the effective computation of abstract models directly from the text of a program, thereby avoiding the need for intermediate storage of a full-blown model. Finally, it formalizes the notion of optimality, while allowing to trade precision for speed by computing sub-optimal approximations. However, Abstract Interpretation has traditionally been focussed on the analysis of deterministic programs, in particular through abstractions that preserve invariance properties, i.e., properties that hold in all states along every possible execution path.

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

Submitted for publication.

This report is a revision of Computing Science Note 94/24 (also see [DGG94]) and also appears as report No. 132 in the Logic Group Preprint Series of the Department of Philosophy, Utrecht University.

†Dept. of Philosophy, Utrecht University, Heidelberglaan 8, 3584 CS Utrecht, The Netherlands. Dennis.Dams@phil.ruu.nl.
§Computer Science Dept., Technion, Haifa 32000, Israel. orna@cs.technion.ac.il. Partially supported by the U.S.-Israeli Binational Science Foundation.
1 Introduction

In the model-checking approach [CES86, LP85, QS81] to program verification, a model of the program is constructed over which formulae are checked for satisfaction. The model reflects the possible behaviours of the program, the formulae express certain required properties of such behaviours. Obviously, the size of the model is the limiting factor to the feasibility of the model-checking approach. In the worst case, it doubles with every extra bit of memory that the program may access. This problem is referred to as the state explosion problem. One solution to it is the application of abstraction techniques, which aim to abstract the model to a smaller one, in such a way that if some property holds for the abstracted model, it also holds for the original model.

Such abstraction techniques are formalized in the framework of Abstract Interpretation [CC77], which was originally conceived as a unifying theory of compile-time (data-flow) analyses. Applications of Abstract Interpretation have traditionally been focussed on the analysis of universal safety properties, that hold in all states (safety) along all possible executions (universality) of the program.\(^1\)

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

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

The properties may be formalized by expressing them in a formal logic whose formulae can be interpreted over transition systems. One commonly used logic is CTL* (computation tree logic, see [EH86]). It contains universal and existential quantification over execution paths, as well as temporal operators that express that, along a path, some property will hold (a) in the next state, (b) in every state (safety), or (c) in some state (liveness).

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

\(^1\)The notions of universality and safety of a property are not always distinguished as explicitly as we do in this paper. What we call "universal safety" is elsewhere often just termed "safety" or "invariance".
of a (concrete) transition system. In computing such abstract models, one may decide to sacrifice optimality in exchange for a gain in efficiency. The approximation ordering between abstract systems, defined in Sect. 4, formalises this notion of sub-optimality. Sect. 5 then shows how abstractions may be computed directly from a program text by "lifting" the operations of a programming language to a domain of data descriptions. Conditions are given under which the constructed models are optimal, and an alternative construction method is indicated which yields optimal models even in absence of those conditions. Furthermore, it is shown that sub-optimal models are constructed when computing approximations to the lifted operations. An elaborate example is presented in Sect. 6. Sect. 7 briefly indicates the consequences of insisting on strong preservation, meaning that not only truth, but also falsehood of formulae is preserved. Sect. 8 compares ours to related work, and Sect. 9 concludes.

2 Preliminaries

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

2.0.1 Definition. The logic $\text{CTL}^*$ is the set of state formulae which is inductively defined by the following clauses.

1. If $p \in \text{Lit}$, then $p$ is a state formula.
2. If $\varphi$ and $\psi$ are state formulae, then so are $\varphi \land \psi$ and $\varphi \lor \psi$.
3. Any state formula is also a path formula.
4. If $\varphi$ and $\psi$ are path formulae, then so are $\varphi \land \psi$, $\varphi \lor \psi$, $\exists \varphi$, $\forall \varphi$, $\varphi \land \psi$, and $\varphi \lor \psi$.
5. If $\varphi$ is a path formula, then $\forall \varphi$ and $\exists \varphi$ are state formulae.

For $\varphi \in \text{CTL}^*$, the formula $\neg \varphi$ is considered to be an abbreviation of the equivalent $\text{CTL}^*$ formula in negation normal form (obtained in the usual way). The abbreviations $\text{true}$, $\text{false}$ and $\rightarrow$ can then be defined as usual. For a path formula $\varphi$, $F\varphi$ and $G\varphi$ abbreviate $\text{true} \land \varphi$ and $\text{false} \lor \varphi$ respectively. $\forall \text{CTL}^*$ and $\exists \text{CTL}^*$ (universal and existential $\text{CTL}^*$) are subsets of $\text{CTL}^*$ in which the only allowed path quantifiers are $\forall$ and $\exists$ respectively.

Transition systems $\text{CTL}^*$ formulae are interpreted over transition systems $T = (\Sigma, I, R)$ where $\Sigma$ is a set of states, $I \subseteq \Sigma$ is a set of initial states, and $R$ is a transition relation over $\Sigma$ which is total and image-finite, so that for every $s \in \Sigma$ there exists a positive, finite number of $t \in \Sigma$ for which $R(s, t)$ holds. A path in $T$ is an infinite sequence $\pi = s_0s_1 \cdots$ of states such that for every $i \in \mathbb{N}$, $R(s_i, s_{i+1})$ holds. The notation $\pi^n$ denotes the suffix of $\pi$ which begins at $s_n$. For $s \in \Sigma$, a $(T, s)$-path (or s-path when $T$ is clear from the context) is a path in $T$ that starts in $s$.

We assume a function $\llbracket \cdot \rrbracket : \text{Lit} \rightarrow \mathcal{P}(\Sigma)$, satisfying $\llbracket p \rrbracket \cap \llbracket \neg p \rrbracket = \emptyset$ for every proposition $p \in \text{Prop}$, which specifies the interpretation of literals over states. Intuitively, $\llbracket p \rrbracket$ is the set of states where $p$ holds. Transition systems thus defined are essentially the same as Kripke structures ([Kri63]). The only difference is that we have the function $\llbracket \cdot \rrbracket$ instead of a labelling function from $\Sigma$ to sets of literals.
2.0.2 Definition. Satisfaction of formulae is defined inductively as follows. Let \( p \in \text{Lit}, s \in \Sigma \) and \( \pi \) a path in \( T \).

1. \( s \models p \) iff \( s \in \| p \| \).
2. \( s \models \varphi \land \psi \) iff \( s \models \varphi \) and \( s \models \psi \), \( s \models \varphi \lor \psi \) iff \( s \models \varphi \) or \( s \models \psi \).
3. \( \pi \models \varphi \), where \( \pi = s_0s_1 \ldots \) and \( \varphi \) is a state formula, iff \( s_0 \models \varphi \).
4. \( \pi \models \varphi \land \psi \) iff \( \pi \models \varphi \) and \( \pi \models \psi \), \( \pi \models \varphi \lor \psi \) iff \( \pi \models \varphi \) or \( \pi \models \psi \).
5. \( s \models \forall \varphi \) iff for every \( s \)-path \( \pi \), \( \pi \models \varphi \), \( s \models \exists \varphi \) iff there exists an \( s \)-path \( \pi \) such that \( \pi \models \varphi \).

For a set \( S \) of states or paths, the notation \( S \models \varphi \) abbreviates \( \forall s \in S \) \( s \models \varphi \). When there may be confusion between different systems, we write \( (T, s) \models \varphi \) to denote that \( s \models \varphi \) in \( T \), and similar for \( (T, S) \models \varphi \).

These formulae can be used to express a variety of properties of transition systems. For example, \( s \models \forall G p \) expresses that \( p \) is true in all states which are reachable from \( s \). If proposition \( r \) characterizes reset states, then \( s \models \forall G \exists F r \) 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 \( U \) (\( \varphi \lor \psi \) is semantically equivalent to \( \neg(\neg \varphi \lor \neg \psi) \) and has the intuitive meaning of “release”: \( \psi \) must be true as long as \( \varphi \) is false, and only if \( \varphi \) becomes true, \( \psi \) may become false afterwards. It has been added as the dual of \( U \) to compensate for the fact that formulae like \( \neg(\varphi \lor \psi) \) are not well-formed. For the same reason both \( \land \) and \( \lor \) are primitive in the logic.

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

**Abstract Interpretation** A transition system forms the interpretation of a program: it models the possible behaviours. Formally, this is captured by an interpretation function \( \mathcal{I} \) from programs\(^2\) to transition systems. Properties of a program \( P \)'s behaviour may be analysed by studying \( \mathcal{I}(P) \). As this model may be too complex to handle ("state explosion"), we look for abstractions of it which can provide partial information. Two points are of quintessential importance in the definition of such abstracted transition systems. Firstly, the abstraction should preserve the information that we are interested in: any (CTL*) property that holds for the abstract model, should hold for the original concrete model as well. Sect. 3 focuses on this aspect. Secondly, such abstractions are to be constructed directly from the program, and not by first building a full model and then abstracting it. That is, we are looking for an abstract interpretation function, \( \mathcal{I} \) which maps each program \( P \) to an abstraction of \( \mathcal{I}(P) \). Sect. 5 defines such a function, fixing a simple notion of program for this purpose.

In the rest of this section we review the framework of Abstract Interpretation which offers ways to formalize the notion of abstraction and provides means to design "good" abstract interpretation functions through the notions of optimality and approximation. Briefly, the idea is as follows. As the concrete object that we are interested in, in this case the concrete model, is too large to handle, we abstract from certain aspects of the states. Thereby, states that were different now become identified. This is

---

\(^{2}\)In Sect. 5 we will become more specific about the syntax of programs.
formally captured by introducing a set of abstract states and a concretisation function $\gamma$ mapping each abstract state to a set of concrete states that have become indistinguishable. For example, we may wish to abstract from everything but the fact whether variable $x$ is greater than 5, in which case we introduce two abstract states named, e.g., $gr_{\leq 5}$ and $leq_{\leq 5}$, acting as descriptions of concrete states as specified by the function $\gamma$ with $\gamma(gr_{\leq 5}) = \{ s \in \Sigma \mid s(x) > 5 \}$ and $\gamma(leq_{\leq 5}) = \{ s \in \Sigma \mid s(x) \leq 5 \}$ (we view a state as a valuation function on variables here). The goal is then to construct transition systems over abstract states by interpreting the operations occurring in the program over the data descriptions $gr_{\leq 5}$ and $leq_{\leq 5}$. For example, the execution of the assignment $x := x + 1$ in the abstract state $gr_{\leq 5}$ results in the state $gr_{\leq 5}$ again. On the other hand, executing it in the state $leq_{\leq 5}$ may yield either $gr_{\leq 5}$ or $leq_{\leq 5}$. This can be modelled by introducing more abstract states in such a way that for every subset $C$ of $\Sigma$, there exists an abstract state $a$ with $\gamma(a) \supseteq C$. An abstraction function $\alpha$ may then be introduced which maps each $C$ to the “smallest” description (see below). Correctness (“safety”) of the abstract interpretation of the assignment $x := x + 1$ can then be stated by requiring that for each $C$, $\gamma(\alpha(C)^{\prime} + 1) \supseteq C + 1$ (where on the left-hand-side of the $\supseteq$ the quotes indicates that the function $+1$ has to be interpreted abstractly, over the domain of abstract values, and on the right-hand-side, $+1$ is pointwise extended to sets).

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

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

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

Often, the abstract approximation order is defined in terms of the concrete by $a \preceq a' \iff \gamma(a) \subseteq \gamma(a')$. In that case, we have $\alpha \circ \gamma(a) = a$ for all $a \in A$; $(\alpha, \gamma)$ is then called a Galois insertion from $(C, \sqsubseteq)$ to $(A, \preceq)$.

Under the assumptions of the Galois connection framework, $a \in A$ is an abstraction of $c \in C$ iff $\alpha(c) \preceq a$ (or, equivalently, $c \sqsubseteq \gamma(a)$). Given a program $P$, the goal is to compute, in an efficient manner an abstraction $\alpha I(P)$ of $I(P)$. Usually, $\alpha I$ is constructed by providing an abstract “counterpart” for each operation used in the definition of $I$. For example, a concrete interpretation function is often defined to be the least fixpoint $I_P$ of some function $I : C \rightarrow C$ (where $I$ depends on $P$ in some way). Given an abstract domain $A$ and the connection $(\alpha, \gamma)$, one looks for a function $\alpha I : A \rightarrow A$ such that $\alpha(I_P(\alpha I(a))) \preceq I_P(\alpha I(a))$. This is satisfied if for all $c \in C$, $\alpha I(a) \preceq_I \alpha(\alpha I(a))$ ([CC79]). In this case, $\alpha I$ is said to be safe w.r.t. $I$. 

5
The requirement \( \alpha(I(P)) \preceq \imath(I(P)) \) can easily be satisfied by defining \( \imath \) to be the constant function mapping everything to an element \( T \in A \) which is less precise than all other elements and hence gives no information at all. In order to avoid such solutions and reason about the quality of abstractions, the notion of optimality is lifted to functions. An abstract interpretation function \( \imath \) is said to be \textit{optimal} w.r.t. \( \imath \) iff \( \alpha(I(P)) = \imath(I(P)) \). Likewise, in case of the above example, \( \imath \circ f \) is optimal w.r.t. \( \imath \circ f \circ \gamma \); this is easily seen to imply that \( I\gamma(I\gamma(f)) \) is optimal w.r.t. \( I\gamma(f) \). In the Galois insertion framework, if such an optimal function \( \imath(f) \) exists, it is defined by \( \imath(f) = \alpha(f) \).

A number of weaker frameworks than the Galois connection framework have been proposed in which the abstraction and/or concretization functions are replaced by relations (see [Mar93] for an overview). These cater for situations where most precise abstractions and/or most general concretisation do not exist, for example because concrete elements have a number of optimal abstractions which are mutually incomparable, or because the corresponding approximation orderings have not been defined, or are pre- instead of partial orders. In this paper, we start from a given Galois insertion on the level of states, and induce a notion of abstraction between transition systems, where concrete systems are unordered while the approximation order on abstract systems is a pre-order. Although there may exist different representations of the most precise abstraction of a given concrete system, all these representations share the same \( \text{CTL}^* \) properties. An abstraction function is given which maps any concrete system to one such representative. Hence, by comparing a computed abstraction to this representative, we are still able to discuss optimality issues.

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

**The relation transformers \( \sim_3 \) and \( \forall_3 \); (bi)simulations** We use two relation transformers which lift transition relations on states to relations on sets of states.

2.0.4 DEFINITION. Let \( A \) and \( B \) be sets and \( R \subseteq A \times B \). The relations \( \sim_3, \forall_3 \subseteq \mathcal{P}(A) \times \mathcal{P}(B) \) are defined as follows.

\[
R \sim_3 = \{(X,Y) \mid \exists x \in X \exists y \in Y \ R(x,y)\} \\
R \forall_3 = \{(X,Y) \mid \forall x \in X \exists y \in Y \ R(x,y)\}
\]

So, if \( R \) is a transition relation, \( R \sim_3 (X,Y) \) iff some state in \( X \) can make an \( R \)-transition to some state in \( Y \), and \( R \forall_3 (X,Y) \) iff every state in \( X \) can make an \( R \)-transition to some state in \( Y \).

Finally, we recall the definitions of simulation ([Mil71]) and bisimulation ([Par81]).

2.0.5 DEFINITION. Let \( R_1 \subseteq \Sigma_1 \times \Sigma_1 \) and \( R_2 \subseteq \Sigma_2 \times \Sigma_2 \) be transition relations. A relation \( \sigma \subseteq \Sigma_1 \times \Sigma_2 \) is a simulation (from \( \Sigma_1 \) to \( \Sigma_2 \)) iff \( \sigma^{-1} R_1 \subseteq R_2 \sigma^{-1} \) (juxtaposition denotes composition of relations). In this case we say that \( R_1 \sigma \)-simulates \( R_2 \). \( \sigma \) is a bisimulation if in addition \( \sigma^{-1} \) is a simulation from \( \Sigma_2 \) to \( \Sigma_1 \). A (bi)simulation \( \sigma \) is consistent iff \( \sigma(s_1, s_2) \) implies \( \{p \in \text{Lit} \mid \|p\| \geq s_1\} = \{p \in \text{Lit} \mid \|p\| \geq s_2\} \).

An equivalent definition of simulation is the following (see, e.g., [Mil71]). Whenever \( \sigma(s_1, s_2) \) and \( R_1(s_1, s_1') \), then there exists \( s_2' \) such that \( R_2(s_2, s_2') \) and \( \sigma(s_1', s_2') \).

\footnote{The intuition is that \( R_2 \) can "mimic" everything that \( R_1 \) can do. From this point of view, the by now standard terminology "\( R_1 \) simulates \( R_2 \)" which was introduced in [Mil71], is awkward.}
Abstract Transition Systems

The definition of an abstract system $A$ starts from a given poset $(\Sigma, \leq)$ of abstract states together with a Galois insertion $(\alpha, \gamma)$ from $(P(\Sigma), \subseteq)$ to $(\Sigma, \leq)$ which determines its relation to the concrete states. We will usually write $\alpha(c)$ for $\alpha(\{c\})$. For a path $\rho = a_0a_1 \cdots$ in $A$, $\gamma(\rho) = \{c_0c_1 \cdots | \forall i, R(c_i, c_{i+1}) \wedge c_i \in \gamma(a_i)\}$. We investigate how to define abstract models in such a way that $\text{CTL}^*$ is preserved from abstract to concrete model. The intention is that a concrete state $C$ is described by any abstraction $a \geq \alpha(c)$ of it. From this point, it is natural to require preservation of formulae on the level of individual states:

$$\forall \varphi \in \text{CTL}^*, a \in \alpha \Sigma (A, a) \models \varphi \Rightarrow (C, \gamma(a)) \models \varphi$$

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

3.1 Valuation of literals

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

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

This choice determines the valuation of literals in abstract states. Namely, the relation $\models \subseteq \alpha \Sigma \times \text{Lit}$ is defined as in clause 1 of Def. 2.0.2, where $s$ now denotes an abstract state and $||\cdot||$ has to be replaced by $\alpha||\cdot||$. By this choice it can now easily be shown that $(A, a) \models \phi \Leftrightarrow (C, \gamma(a)) \models \phi$, for every $a \in \alpha \Sigma$ and $p \in \text{Lit}$, which implies that as many as possible literals hold in each abstract state. Note that if $a \in \alpha \Sigma$ is such that $\gamma(a)$ contains concrete states in which $p$ holds and concrete states in which $\neg p$ holds, then $a \not\in \alpha||p||$ but also $a \not\in \alpha||\neg p||$. So, although the rule of the excluded third from classical logic holds on the level of interpretation of literals over abstract states (we have either $a \models p$ or $a \not\models p$, and similarly for $\neg p$), this rule does not hold on the level of the logic itself: $a \not\models p$ does not necessarily imply that $a \models \neg p$.

3.2 Abstract initial states

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

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

Note that the singleton $\alpha(I)$ is in general less precise than this: because $\alpha$ distributes over $\cup$ (see the remark after Def. 2.0.3), we have $\alpha(I) = \bigvee \{\alpha(s) | s \in I\}$, so that for each $s \in I$, $\alpha(s) \leq \alpha(I)$. 7
3.3 Abstract transition relations

Most traditional applications of the framework of Abstract Interpretation are developed in a context where both the concrete and the abstract transition relations are deterministic, i.e., they are transition functions $f$ and $\alpha f$. In such cases, the (unique) abstract successor $\alpha f(a)$ of $a$ would be the best description of the successors of the states in $\gamma(a)$: $\alpha f(a) = \alpha(f(\gamma(a)))$. On the other hand, we want to abstract a concrete by an abstract transition relation in such a way that both existential and universal properties are preserved. However, such an abstract transition relation $\alpha R$ will only exist if there exists a consistent bisimulation from $(\Sigma, R)$ to $(\alpha \Sigma, \alpha R)$ (see, e.g., [BCG88]), which is a much too strong condition, as it results in the falsity of $\text{CTL}^*$ formulae being preserved as well. Our solution is to define instead two transition relations on $\alpha \Sigma$; one preserving universal properties, and the other existential properties.

It is not difficult to see that if properties of the form "there exists a successor state satisfying property $\varphi$" are to be preserved, then abstract state $b$ may only be a successor of $a$ if $R^3(\gamma(a), Y)$ for some $Y \subseteq \Sigma$ and $Y \subseteq \gamma(b)$. For reasons of optimality, we also would like $a$ to have as many successors as possible, and furthermore each of them should be as precise a description of $Y$ as possible. The first requirement is satisfied by letting $b$ be a successor of $a$ whenever $R^3(\gamma(a), Y)$; the second by choosing $Y$ to be minimal and $b$ to be the best description of it, as specified by $\alpha$. A similar consideration for the preservation of universal properties leads to the requirement that $b$ is a successor of $a$ iff $R^3(\gamma(a), Y)$, $Y$ is minimal, and $\alpha(Y) = b$.

3.3.1 Definition.

1. $\alpha R^F(a, b) \iff b \in \{\alpha(Y) \mid Y \in \min\{Y' \mid R^3(\gamma(a), Y')\}\}$
2. $\alpha R^C(a, b) \iff b \in \{\alpha(Y) \mid Y \in \min\{Y' \mid R^3(\gamma(a), Y')\}\}$

$\alpha R^F$ and $\alpha R^C$ are called the free and constrained (abstract transition) relations respectively. A few remarks are in place. Note that for any $a \in \alpha \Sigma$, the minimal sets $Y'$ such that $R^3(\gamma(a), Y')$, are all singletons. By the totality of $R$ it follows that $\alpha R^F$ is total. By image-finiteness, the set $\{Y' \mid R^3(\gamma(a), Y')\}$ always has minimal elements so that also $\alpha R^C$ is total; this follows from the following lemma.

3.3.2 Lemma. If $R \subseteq A \times B$ is image-finite, then for every $X \subseteq A$ there exists a minimal $Y \subseteq B$ such that $R^3(X, Y)$.

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

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

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

We have the following preservation properties for paths.

Footnotes:
4 Functions are pointwise extended to sets.
5 We make one exception, namely, the abstract state $\bot \in \alpha \Sigma$ with $\gamma(\bot) = \emptyset$ (if present) has no successors. However, note that $\bot$ itself is not reachable anyway.
6 Zorn's Lemma states that if in a non-empty collection $C$ of sets, every $\subseteq$-increasing chain of sets in $C$ has an upper bound in $C$, then $C$ has a maximal element.
3.3.3 Lemma.

1. Let \( a \in \Sigma \), \( c \in \gamma(a) \) and \( \pi \) be a \((\Sigma, I, R, c)\)-path. Then there exists an \((\Sigma, I, R^F, c)\)-path \( \rho \) such that \( \pi \in \gamma(\rho) \).

2. Let \( a \in \Sigma \), \( c \in \gamma(a) \) and \( \rho \) be an \((\Sigma, I, R^C, c)\)-path. Then there exists a \((\Sigma, I, R, c)\)-path \( \pi \) in \( \gamma(\rho) \).

Proof.

1. Assume \( \pi = c_0 c_1 \ldots \) with \( c_0 = c \). Define \( \rho = a_0 a_1 \ldots \) with \( a_0 = a \) and \( a_i = \alpha(c_i) \) for \( i \geq 1 \). Because \( (\alpha, \gamma) \) 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 \( \gamma R^F : \gamma R^F(a_i, a_{i+1}) \).

2. Assume \( \rho = a_0 a_1 \ldots \cdot a_0 = a \). We show that there exists an infinite sequence \( \pi = c_0 c_1 \ldots \cdot \) of states in \( \Sigma \) such that for all \( i \geq 0 \), \( c_i \in \gamma(a_i) \) and \( R(c_i, c_{i+1}) \). \( \pi \) 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 \( \gamma R^C(a_n, a_{n+1}) \), there must be (by Def. 3.3.1 of \( \gamma R^C \)) \( Y \) such that \( \gamma R^C(Y(a_n), Y) \) and \( \alpha(Y) = a_{n+1} \). By definition of \( \gamma R^C \), there exists \( c_{n+1} \in Y \) such that \( R(c_n, c_{n+1}) \). Because \( (\alpha, \gamma) \) is a Galois Insertion, we have \( \gamma(\alpha(Y)) \supseteq Y \), so \( c_{n+1} \in \gamma(a_{n+1}) \).

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

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

\[ S'. \ s \models \forall \varphi \text{ iff for every free s-path } \pi, \pi \models \varphi; \ s = \exists \varphi \text{ iff there exists a constrained s-path } \pi \text{ such that } \pi \models \varphi. \]

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

3.3.4 Theorem. For every \( \varphi \in \text{CTL}* \), \( A^M \models \varphi \Rightarrow C \models \varphi \).

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

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

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

4.0.1 DEFINITION. Let $A = (\alpha^M, I, F, C)$ and $A' = (\alpha'^M, I', F', C')$ be abstract transition systems. $A \preceq A'$ iff (1) for every $a \in I$ there exists $a' \in I'$ such that $a \preceq a'$, (2) $F \preceq$simulates $F'$, and (3) $C'$ $\succeq$simulates $C$. For paths $\rho = a_0a_1 \ldots$ and $\overline{\rho} = \overline{a_0}\overline{a_1} \ldots$, $\rho \preceq \overline{\rho}$ is defined by $\forall i \geq 0 a_i \preceq \overline{a_i}$.

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

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

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

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

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

4.0.3 LEMMA. Let $R$ and $R'$ be transition relations over $\alpha^M$.

1. If for all $a, b \in \alpha^M$, $R(a, b) \Rightarrow \exists b' \in \alpha^M \ R'(a', b')$ and $R' \preceq$simulates $R'$, then $R \preceq$simulates $R'$.

2. If for all $a, b' \in \alpha^M$, $R'(a, b') \Rightarrow \exists a' \in \alpha^M \ R(a, b)$ and $R' \succeq$simulates $R$, then $R' \succeq$simulates $R$.

Note that $\bar{R}(a, b) \Rightarrow \exists b' \in \alpha^M \ R'(a, b')$ is satisfied if $R \subseteq R'$ and $R'(a, b') \Rightarrow \exists a' \in \alpha^M \ R(a, b)$ if $R' \subseteq R$. 10
computing abstract models by abstract interpretation

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

In order to further develop the theory, we first need to fix a programming language. We use a language which is based on action systems [BKS83], which, although being very simple, will help to grasp the idea of how to abstractly interpret operations in "real" programming languages, as it contains rudimentary forms of the common notions of assignment, test and loop. A program is a set of actions of the form \(c_i(x) \rightarrow t_i(x, x')\), where \(x\) represents the vector of program variables, \(c_i\) is a condition on their values and \(t_i\) specifies a transformation\(^7\) of their values into the new vector \(x'\) (\(i\) ranges over some index set \(J\)). Executing an action means evaluating its condition \(c_i\) and, if and only if this yields true, updating the program variables as specified by the associated transformation \(t_i\). A program is run by repeatedly nondeterministically choosing an action and executing it. We let \(\text{Val}\) denote the set of values that the vector \(x\) may take, and \(\text{IVal} \subseteq \text{Val}\) the set of values that it may have initially. Thus, each \(c_i\) is a predicate over \(\text{Val}\) and each \(t_i\) a relation on \(\text{Val}\).

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

- \(\Sigma = \text{Val}\),
- \(I = \text{IVal}\),
- \(R = \{(v, v') \in \text{Val}^2 \mid \exists i \in J.\ c_i(v) \land t_i(v, v')\}\).

Next, we assume a set \(\text{Val}\) of descriptions of sets of values in \(\text{Val}\), via a Galois insertion \((\alpha, \gamma)\), and define two types of non-standard, abstract interpretations of the \(c_i\)'s and \(t_i\)'s over \(\text{Val}\) in such a way that approximations of the abstract models of a program may be computed by interpreting the operators in the program correspondingly.

5.0.2 DEFINITION. For \(a, b \in \text{Val}\),

- \(c_i^F(a) \leftrightarrow \exists \gamma(\alpha) c_i(\bar{v})\);
- \(t_i^F(a, b) \leftrightarrow b \in \{\alpha(Y) \mid Y \in \min\{Y' \mid t_i^{33}(\gamma(a), Y')\}\};
- \(c_i^G(a) \leftrightarrow \forall \gamma(\alpha) c_i(\bar{v})\);
- \(t_i^G(a, b) \leftrightarrow b \in \{\alpha(Y) \mid Y \in \min\{Y' \mid t_i^{33}(\gamma(a), Y')\}\}.

Furthermore, we define the abstract model \(\mathcal{M} = (\alpha, \Sigma, \alpha I, \alpha R^F, \alpha R^G)\) where:

\(^7\)We could have represented this transformation as the simultaneous assignment \(x' := t_i(x)\). However, by abstracting the function \(t_i\), it may become a relation, and we prefer to denote both the concrete and the abstract transformations in the same way.
• $\alpha \Sigma = \alpha \text{Val}$;
• $\alpha I = \{\alpha(\bar{v}) \mid \bar{v} \in I \text{Val}\}$;
• $\alpha \overrightarrow{RF} = \{(a, b) \in \alpha \text{Val}^2 \mid \exists i \in J \ c_i^F(a) \land t_i^F(a, b)\}$;
• $\alpha \overrightarrow{RC} = \{(a, b) \in \alpha \text{Val}^2 \mid \exists i \in J \ c_i^C(a) \land t_i^C(a, b)\}$.

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

The following lemma and theorem express that the abstract interpretations given above can be used to compute approximations to $A^P$.

5.0.3 LEMMA.
1. $\overrightarrow{RF} \supseteq \overrightarrow{RC}$
2. For all $a, b \in \alpha \Sigma$, $\overrightarrow{RC}(a, b) \Rightarrow \exists b' \preceq b \overrightarrow{RC}(a, b')$.

PROOF.
1. Let $a, b \in \alpha \text{Val}$ and suppose $(a, b) \in \overrightarrow{RF}$. By Def. 3.3.1 of $\overrightarrow{RF}$, Def. 2.0.4 of $R^F$ and Def. 5.0.1 of $R$, this is equivalent to $b \in \{\alpha(\bar{v}) \mid \bar{v} \in \text{min}\{Y' \mid \exists i \in J \ [c_i(\bar{v}) \land t_i(\bar{v}, \bar{w})]\}\}$. Exchanging existential quantifiers yields the equivalent $b \in \{\alpha(\bar{v}) \mid \bar{v} \in \text{min}\{Y' \mid \exists i \in J \ [c_i(\bar{v}) \land t_i(\bar{v}, \bar{w})]\}\}$. Because the elements of the set $\text{min}\{Y' \mid \ldots \}$ are singletons, the term $\exists i \in J \ [b \in \{\alpha(\bar{v}) \mid \bar{v} \in \text{min}\{Y' \mid \exists i \in J \ [c_i(\bar{v}) \land t_i(\bar{v}, \bar{w})]\}\}]$ can be rewritten as $\exists i \in J \ [b \in \{\alpha(\bar{v}) \mid \bar{v} \in \text{min}\{Y' \mid \exists i \in J \ [c_i(\bar{v}) \land t_i(\bar{v}, \bar{w})]\}\}]$.

2. Let $a, b \in \alpha \text{Val}$ and suppose $(a, b) \in \overrightarrow{RC}$. By Def. 5.0.2 of $\overrightarrow{RC}$, $c_i^C(a)$, $t_i^C(a, b)$, and Def. 2.0.4 of $t_i(\gamma(a), Y)^F$, this is equivalent to $Y \in \text{min}\{Y' \mid \exists i \in J \ [c_i(\bar{v}) \land t_i(\bar{v}, \bar{w})]\}\}$. This expression can be rewritten to the equivalent:

$$b \in \{\alpha(\bar{v}) \mid \exists i \in J \ [Y \in \text{min}\{Y' \mid \exists i \in J \ [c_i(\bar{v}) \land t_i(\bar{v}, \bar{w})]\}\}] \quad (2)$$

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

$$\exists i \in J \ [Y \in \text{min}\{Y' \mid \exists i \in J \ [c_i(\bar{v}) \land t_i(\bar{v}, \bar{w})]\}\}] \quad (3)$$

By pushing the $\exists i \in J$ inside, this subexpression is weakened:

$$Y \in \text{min}\{Y' \mid \exists i \in J \ [c_i(\bar{v}) \land t_i(\bar{v}, \bar{w})]\}\}] \quad (4)$$

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

$$b' \in \{\alpha(\bar{v}) \mid \bar{v} \in \text{min}\{Y' \mid \exists i \in J \ [c_i(\bar{v}) \land t_i(\bar{v}, \bar{w})]\}\}] \quad (5)$$
A similar step can be made again: if $b'$ satisfies (5), then there exists $b'' \preceq b'$ satisfying:

$$b'' \in \{ \alpha(Y) \mid Y \in \min \{ Y' \mid \forall \theta \in \gamma(a) \exists \phi \in \gamma, \exists I \in J \cdot [c_i(\bar{v}) \land t_i(\bar{v}, \bar{w})] \} \}$$

(6)

which is, by Def. 5.0.1 of $R$, Def. 2.0.4 of $R^f$ and Def. 3.3.1 of $R^c$, equivalent to $R^c(a, b'')$. □

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

PROOF. The proof is based on Lemmata 5.0.3 and 4.0.3. From the definitions of $\overline{R}^F$ and $R^c$, it easily follows that $\overline{R}^F \preceq$-simulates $R^f$ and that $R^c \preceq$-simulates $R^c$. □

5.1 Optimal abstract interpretations

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

5.1.1 Conditions on the abstract domain

In order to pinpoint the reasons why the computed abstractions are not optimal, we analyse the proof of Lemma 5.0.3. In part 1, concerning the free abstraction, the only place where the formula being manipulated is (strictly) weakened, is when the term $\exists \bar{v} \in \gamma(a), \exists \bar{w} \in \gamma, [c_i(\bar{v}) \land t_i(\bar{v}, \bar{w})]$ (T1) is replaced by $\exists \bar{v} \in \gamma(a), c_i(\bar{v})$ or $\exists \bar{v} \in \gamma(a), c_i(\bar{v})$ (T2). The following small example illustrates what happens.

Suppose that the concrete state space consists of a single integer variable $v$, and that the abstract domain contains values $e$ and $o$, being descriptions of the even and the odd numbers respectively. Assume that $P$ contains as action $i: v = 4 \rightarrow v := v/4$ (specifying $c_i(v)$ to be $v = 4$ and $t_i(v, w)$ to be $w = v/4$). Then (T1), with $e$ for $a$ and $\gamma(e)$ for $Y'$, does not hold. On the other hand, (T2) does hold: there exists an even number which is equal to 4 and there exists a (different) even number which, when divided by 4, yields an even number. In order to avoid this situation and enforce equivalence of (T1) and (T2), we can impose a condition on the abstract states.

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

PROOF. The same condition for all states, i.e., for all $a \in \alpha \Sigma$ and $i \in J, \forall \bar{v} \in \gamma(a) c_i(\bar{v})$ or $\forall \bar{v} \in \gamma(a) \neg c_i(\bar{v})$, is easily seen to imply equivalence of the terms (T1) and (T2). Therefore, $\alpha R^f' = \alpha R^f$ in that case. It easily follows that the restriction of the condition to reachable states implies a similar result for the reachable transitions. □
So, under the given condition, the reachable part of the computed abstract model — which is the only part that matters for evaluation of CTL* formulae — coincides with the reachable part of the optimal model. The given condition is quite strong if the set of reachable states contains "large" states like $T$. However, as can be seen from Def. 3.3.1 and the remark, following it, about the sets $Y'$ being singletons, only the atoms of the abstract domain can be reachable, i.e., the elements $\{a(\{c\}) \mid c \in \Sigma\}$. Hence, we should preferably choose the abstract domain in such a way that these atoms are "small". See [CC79] for a variety of techniques for the construction of suitable abstract domains.

Though being sufficient, the condition required in Lemma 5.1.1 is not necessary. However, it is a reasonable condition that can be checked rather easily: one has to check that for each atomic abstract state $a$ and each condition $C_i$ of the program, either "$a \Rightarrow c_i$" or "$a \cap c_i = \emptyset$". For instance, in the example above, "being even" neither implies nor excludes "being equal to 4", so the condition is not met. The condition also gives a deeper insight in how to design "good" abstract domains given a programming language.

For the constrained relation, we analyse part 2 of the proof of Lemma 5.0.3. The last two steps in this proof introduce the differences between $\sigma^C$ and $\sigma^\text{RC}$. By replacing subformula (3) by (4), the minimality of the "$\forall$-successors" $Y$ of $\gamma(a)$ over the different actions is "divided out" and made global to all actions. As a result, each of the $\forall$-successors under $\sigma^C$ of $\gamma(a)$ is a superset of an $\forall$-successor under $\sigma^\text{RC}$ of $\gamma(a)$. In the step from formula (5) to (6), the "action-wise" computation of $\forall$-successors (for a single action $i \in J$, $Y$ must be an $\forall$-successor of $\gamma(a)$, i.e., all states in $\gamma(a)$ must be able to make a transition to some state in $Y$ via action $i$) is replaced by an "action-wide" computation (all states in $\gamma(a)$ must be able to make a transition to some state in $Y$ via any action). This difference does entail a true loss of information.

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

5.1.2 LEMMA. If for all $i, j \in J$ with $i \neq j$ and $a, b_i, b_j \in \Sigma$ such that $c^C_i(a) \land t^C_i(a, b_i)$ and $c^C_j(a) \land t^C_j(a, b_j)$, we have $\forall \delta \in \Sigma \{ \gamma(b_i) \cap \gamma(b) = \emptyset \lor \gamma(b_j) \cap \gamma(b) = \emptyset \lor (\gamma(b_i) \subseteq \gamma(b) \land \gamma(b_j) \subseteq \gamma(b)) \}$, then $\sigma^\text{RC} = \sigma^C$.

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

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

1. there cannot exist $Y_i \in \min \{Y' \mid \forall \delta \in \gamma(a) \exists \omega \in \gamma, c_i(\delta) \land t_i(\delta, \omega)\}$ and $Y_j \in \min \{Y' \mid \forall \delta \in \gamma(a) \exists \omega \in \gamma, c_j(\delta) \land t_j(\delta, \omega)\}$ such that $Y_i \subseteq Y_j$.

2. $\min \{Y' \mid \forall \delta \in \gamma(a) \exists \omega \in \gamma, [c_i(\delta) \land t_i(\delta, \omega)] \lor \forall \delta \in \gamma(a) \exists \omega \in \gamma, [c_j(\delta) \land t_j(\delta, \omega)]\} = \min \{Y' \mid \forall \delta \in \gamma(a) \exists \omega \in \gamma, [c_i(\delta) \land t_i(\delta, \omega)] \lor [c_j(\delta) \land t_j(\delta, \omega)]\}$

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

5.1.2 Adapting the abstract interpretation

Instead of imposing conditions guaranteeing optimality of abstract models computed as specified by Def. 5.0.2, we may change the definition of these abstract interpretations themselves in such a way
that the “loss” of Lemma 5.0.3 does not occur. For the free abstract interpretation, this means that it cannot be distributed over the individual condition and transformation parts of an action. In case of the example given above, this would mean that an abstract interpretation act\_c has to be provided for the action act\_i(v, w) \iff v = 4 \land w = v/4 as a whole, satisfying act\_c(a, b) \iff \exists v \in \gamma(a) \{ c_i(v) \land b \in \{ a(Y) \mid Y \in \text{min}\{ Y' \mid t \in \exists \{ v \mid (Y') \}}\}.

In case of the constrained abstract interpretations, loss of optimality already occurs at the point where they are distributed over the individual actions of a program. Here, the adaptation would require the generalization of the abstract interpretation of actions by taking into account the effect of executing an arbitrary number of actions “at the same time” by defining act\_{i_1, ..., i_k}(a, b) \iff \exists a_1, ..., a_k, b_1, ..., b_k \in \Sigma [a_1 \lor \cdots \lor a_k = a, b_1 \lor \cdots \lor b_k = b, \forall j \in \{i_1, ..., i_k\} c_j^c(a_j) \land t_j^c(a_j, b_j)] for subsets \{i_1, ..., i_k\} of \Sigma (recall that \lor and \lor denote least upper bounds on \Sigma). This approach corresponds to the merge over all paths analysis of [CC79].

5.2 Computing approximations

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

5.2.1 DEFINITION. The definition of approximation is extended to abstract interpretations of the transformation operators, as follows. For abstract operations\(^8\) \(t, \bar{t} \in \Sigma \text{Val} \times \Sigma \text{Val},\)

\[
\bar{t} \succ t \iff \forall a, b, \bar{a}, \bar{b} \in \Sigma \text{Val} \left[ t(a, b) \Rightarrow \exists \bar{a} \leq_\Sigma \bar{b} \bar{t}(\bar{a}, \bar{b}) \right] \land \left[ \bar{t}(a, b) \Rightarrow \exists a \leq_\Sigma b t(a, b) \right]
\]

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

- \(\Sigma = \text{Val};\)
- \(I = \{ a(\bar{v}) \mid \bar{v} \in \text{IVal} \};\)
- \(\bar{R}^F = \{ (a, b) \in \Sigma \text{Val}^2 \mid \exists i \in I c_i^F(a) \land t_i^F(a, b) \}\)
- \(\bar{R}^C = \{ (a, b) \in \Sigma \text{Val}^2 \mid \exists i \in I c_i^C(a) \land t_i^C(a, b) \}\)

5.2.2 LEMMA. \(\bar{A}^M \succeq \bar{A}^M.\)

PROOF. We show that (1) \(\bar{R}^F \succeq \text{simulates} \bar{R}^F\) and (2) \(\bar{R}^C \succeq \text{simulates} \bar{R}^C.\)

1. Let \(a, a_1, \bar{a} \in \Sigma\) with \(\bar{a} \succeq a, a_1 \preceq_a a\). We show that there exists \(\bar{a}_1 \preceq a_1\) such that \(\bar{a} \succeq a, a_1 \preceq (a, a_1).\)

By Def. 5.0.2 of \(\bar{R}^F\), \(\bar{R}^C\) (\(a, a_1\)) equivals \(\exists i \in I [c_i^F(a) \land t_i^F(a, a_1)]\). Because \(\bar{a} \preceq a, a_1 \preceq a\), we have \(c_i^F(a) \Rightarrow c_i^F(\bar{a})\) and also \(t_i^F(a, a_1) \Rightarrow t_i^F(\bar{a}, a_1)\) (see Def. 5.0.2 of \(c_i^F\) and \(t_i^F\) and Def. 2.0.4 of [CC79]), so \(\exists i \in I [c_i^F(\bar{a}) \land t_i^F(\bar{a}, a_1)]\). By Def. 5.2.1 of \(t_i^F\), there exists \(\bar{a}_1 \preceq a_1\) such that \(\exists i \in I [c_i^C(\bar{a}) \land t_i^C(\bar{a}, a_1)]\), i.e., \(\bar{R}^F(\bar{a}, a_1)\).

\(\text{Remember that such “operations” are binary relations.}\)
2. Let \(a, a_1, a' \in \Sigma\) with \(\overline{R}^C(a, a_1)\) and \(a' \preceq a\). We show that there exists \(a'_1 \preceq a_1\) such that \(\overline{R}^C(a', a'_1)\). We have \(\overline{R}^C(a, a_1)\). By Def. 5.2.1 of \(\overline{R}^C\) and \(t_i^C\), there exists \(a''_1 \preceq a_1\) such that \(\exists i \in I [c_i^C(a) \land t_i^C(a, a''_1)]\). Because \(a' \preceq a\), we have \(c_i^C(a) \Rightarrow c_i^C(a')\), and also there must be some \(a'_1 \preceq a'_1\) such that \(t_i^C(a', a'_1)\) (see Def. 5.0.2 of \(c_i^C\) and \(t_i^C\) and Def. 2.0.4 of \(R^2\)). So \(\exists i \in I [c_i^C(a') \land t_i^C(a', a'_1)]\), i.e., \(\overline{R}^C(a', a'_1)\), and, by transitivity of \(\preceq\), \(a'_1 \preceq a_1\). □

5.3 Practical application

The use of abstract interpretation to model check a property \(\varphi\) for a program \(P\) is characterized by the following phases. First, an abstract domain \(\text{Val}\) has to be chosen and for all operation symbols occurring in \(P\), abstract interpretations have to be provided. Typically, the tests and transformations are defined in terms of more elementary operations, in which case abstract interpretations may be provided for these. Depending on the property \(\varphi\) to be checked, free and/or constrained interpretations should be given; these have to satisfy Def. 5.0.2. Then, the abstract model can be constructed by a symbolic evaluation of the program over the abstract domain, interpreting the operations according to their abstract interpretations. Finally, \(\varphi\) 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 \(A \not\models \varphi\) does not imply that \(A \models \neg\varphi\) and hence does not justify the conclusion that \(C \models \neg\varphi\), in spite of the fact that \(\neg\varphi\) is an abbreviation of a CTL* formula. However, it may be possible to resolve such a negative answer for \(\varphi\) by checking the negation \(\neg\varphi\). If \(\text{true}\) is returned, then we know that \(C \models \neg\varphi\), i.e., \(C \not\models \varphi\). Of course, the dual abstraction should be well-chosen for this. We do not further investigate this point; solutions can be found in [DGD+94].

The same idea of constructing an abstract model by abstract interpretation of program operations, although based on a different theoretical framework ([LGS+93], see Sect. 8 for a comparison), is applied to a "real-life" example in [Gra94]. Graf shows in that paper how a distributed cache memory, which is in principle an infinite state system because request queues are unbounded, can be verified by providing a finite abstract domain and corresponding abstract operations.

Although the model checking procedure itself is an automated process, it is not obvious how the choice of an appropriate abstract domain with corresponding abstract operations, as well as the proofs that these operations satisfy the conditions of Def. 5.0.2, can be performed in an automated fashion. In [Gra94], the abstract domain has to be provided by the user of the method, and the proofs for the abstract operators form a difficult step in the method. In [KDG95], approximations to the transition relation of StateCharts ([Har87]) are used to verify \(\mu\)-calculus properties of a production cell ([DHKS95]) in a compositional fashion. In [DGG93a] and [DGD+94], a method is developed which aims at full automation of these steps.

6 Example

In this section we illustrate the theory on a small example. Consider the system consisting of two concurrent processes depicted in Fig. 1(a), which is a parallel variant of the Collatz \(3n + 1\) program. We chose this example because it is small but nevertheless displays a non-trivial interplay between data and control. The properties that we will verify concern certain control aspects that depend on the values that the integer variable \(n\) takes under the various operations that are performed on it. Because the state space is infinite, data-abstraction will be necessary in order to verify aspects of the control-flow. It serves as an illustration of the fact that abstraction techniques bring into reach the model checking of systems that cannot be verified through the standard approach.
The program may be viewed as a protocol controlling the mutually exclusive access to a common resource of two concurrent processes, modelling the behaviour of two mathematicians, numbered 0 and 1. They both cycle through an infinite sequence of “think” and “eat” states. The right to enjoy a meal in strict solitude is regulated by having them inspect the value of \( n \) before eating, letting the one go ahead only if \( n \) has an odd value, and the other only if \( n \) is even. Upon exit from the dining room, each mathematician has its own procedure for assigning a new value to \( n \). Transitions can only be taken when the enabling conditions are satisfied, e.g., mathematician 1 can only leave the dining room if \( n \) is divisible by 2. An execution is any infinite sequence of (arbitrarily) interleaved steps of both processes which starts in a state where both mathematicians are in their thinking state, and \( n \) is set to some arbitrary positive integer value. We want to verify mutual exclusion and the absence of individual starvation along every execution. In order to formalize this, we first express the program as an action system. As data and control are treated uniformly in such systems, we introduce variables \( \ell_0 \) and \( \ell_1 \), both ranging over \( \{ \text{think}, \text{eat} \} \), to encode the effect of “being in a location” think or eat. See Fig. 1(b). The state space \( \Sigma \) of this program is the set \( \{ \text{think}, \text{eat} \}^2 \times \mathbb{N} \setminus \{0\} \) of values that the vector \( (\ell_0, \ell_1, n) \) of program variables may assume. The initial states are \( I = \{ (\text{think, think, } n) \mid n \in \mathbb{N} \setminus \{0\} \} \). Its transitions are defined as in Def. 5.0.1, using the standard interpretations of the tests \( =, \text{even, odd} \) 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.

\[
\forall G \neg (\ell_0 = \text{eat} \land \ell_1 = \text{eat})
\]

(7)

\[
\forall G (\ell_0 = \text{eat} \rightarrow \forall F \ell_1 = \text{eat})
\]

(8)

\[
\forall G (\ell_1 = \text{eat} \rightarrow \forall F \ell_0 = \text{eat})
\]

(9)

As these formulae are in \( \forall \text{CTL}^* \), 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 \( \{ \text{think, eat} \}^2 \) the same. Formally, this means that we take an abstract domain with two elements whose concretizations are \( \{ \text{think} \} \) and \( \{ \text{eat} \} \), however, for readability we just denote these elements by \( \text{think} \) and \( \text{eat} \) respectively. To abstract \( \mathbb{N} \setminus \{0\} \), we choose an abstract domain in which \( n \) may take the values \( e \) and \( o \), describing the even and odd positive
integers respectively, i.e., \( \gamma(e) = \{2, 4, 6, \ldots \} \) and \( \gamma(o) = \{1, 3, 5, \ldots \} \). To both abstract domains, we add a top element \( T \). The set \( \alpha \Sigma \) of abstract states is now defined as follows.

\[
\alpha \Sigma = \{ \text{think, eat, } T \}^2 \times \{ e, o, T \}
\]

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

\[
\alpha I = \{ (\text{think}, \text{think}, e), (\text{think}, \text{think}, o) \}
\]

Having chosen an abstract domain, we also have to provide abstract interpretations, over this domain, of the operations that appear in the program, along the lines of Def. 5.0.2. The tables (a) and (b) in Fig. 2 give the definitions of the free abstract interpretations of the transformations and tests on the abstract domain \( \{ e, o, T \} \). The operations \( 3*, +1 \) and \( /2 \) are considered single symbols. The tables have to be interpreted as indicated by the following examples. The entry \text{true} in table (b), row \( \text{even}^F \), column \( e \), indicates that \( \text{even}^F(e) \) holds, i.e., (cf. Def. 5.0.2), \( \exists n \in \gamma(e) \) \( \text{even}(n) \). The entry \text{false} in table (a), row \( +1^F \), column \( (e, e) \), means that \( +1^F(e, e) \) is false, i.e., for any minimal \( Y \) such that \( +1^F(\gamma(e), Y) \), we have \( \alpha(Y) \neq e \) (see Defs. 5.0.2 and 2.0.4). From these diagrams we see for ex-

\[
\begin{array}{cccccccc}
3* & (e, e) & (e, o) & (e, T) & (o, e) & (o, o) & (o, T) & (T, e) & (T, o) & (T, T) \\
\hline
\text{true} & \text{false} & \text{false} & \text{false} & \text{true} & \text{false} & \text{false} & \text{true} & \text{true} & \text{true} \\
\hline
\end{array}
\]

(a)

\[
\begin{array}{ccc}
even^F & e & o & T \\
\hline
\text{true} & \text{false} & \text{true} \\
\hline
dodd^F & \text{false} & \text{true} & \text{true} \\
\hline
\end{array}
\]

(b)

\[
\begin{array}{cccc}
\text{CONSTR.:} & (o, 100) & (o, e) & (o, o) & (o, T) \\
\hline
3* & \text{false} & \text{false} & \text{true} & \text{false} \\
\hline
+1 & \text{false} & \text{true} & \text{false} & \text{false} \\
\hline
\end{array}
\]

(c)

\[
\begin{array}{cccc}
\text{CONSTR.:} & \text{think} & \text{eat} & T \\
\hline
= \text{think} & \text{true} & \text{false} & \text{false} \\
\hline
= \text{eat} & \text{true} & \text{false} & \text{false} \\
\hline
\end{array}
\]

(d)

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

ample that \( /2^F \) is not functional (table (a), row \( /2^F \), first two columns), illustrating that a function may become a relation when abstracted. The tables (c)–(e) are explained below.

Now we can abstractly interpret the program over this abstract domain, using the interpretations given in the tables. Such an abstract execution yields the abstract model of Fig. 3. In this model, only
the reachable abstract states are shown. We see that in no reachable state the property $\ell_0 = \text{eat} \land \ell_1 = \text{eat}$ holds. Hence we have established property (7). Furthermore, the only path from the state where $\ell_0 = \text{eat}$ reaches $\ell_1 = \text{eat}$ within 2 steps, so we have also verified property (8).

However, the abstraction does not allow verification of the other non-starvation property, (9): a counterexample in the abstract models is the path cycling infinitely between $\langle \text{think}, \text{think}, \text{e} \rangle$ and $\langle \text{think}, \text{eat}, \text{e} \rangle$. It turns out that the negation of property (9) can also not be established via the constrained transition relation. So, only refinement of the abstract domain may bring the answer. In this case, the abstract states where $n = e$ would have to be unraveled into infinitely many states representing the cases where $n$ is divisible by 4, by 8, by 16, .... Hence, with our methodology, it is impossible to verify property (9) through a finite abstraction.

Nevertheless, an interesting question is how the refinement of an abstract model, in order to decide indeterminate results, can be computed. In [DGG93a] we identify conditions under which a strongly preserving abstraction may be computed by a successive refinement which is guided by the form of the formula to be checked.

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

$$\ell_0 = \text{think}, \ell_1 = \text{think} \rightarrow n := 100$$

We want to check whether it is always possible to reach a “restart” state. Writing restart for $\text{think}_0 \land \text{think}_1 \land n = 100$, this property is expressed in CTL* by:

$$\forall G F \text{restart}$$

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

The resulting abstraction is depicted in Fig. 4. Solid arrows denote free transitions, dashed arrows represent constrained transitions. Note that it is not in general the case that $\sigma R^C \subseteq \sigma R^F$, as is illustrated by the arrow from $\langle \text{think}, \text{eat}, \text{e} \rangle$ to $\langle \text{think}, \text{think}, \text{T} \rangle$. Property (10) is verified on this model, interpreting the universal quantification along the free paths, and the existential quantification along
the constrained paths. It can easily be seen that (10) holds, hence, we have established its validity in the concrete program.

As an example of the computation of approximations by choosing non-optimal abstract interpretations $\bar{I}_i$ of operations in the program, consider the dining mathematicians again (without the “restart” extension). Take optimal free abstract interpretations of all operations but $3*$, for which we take the following approximation: $3^F(\epsilon, \top) = \text{true}$ and $3^F(\epsilon, e) = 3^F(\epsilon, o) = \text{false}$. Furthermore, take $(\text{think}, \text{think}, \top)$ as the abstract initial state. This gives the free abstraction of Fig. 5, from which still various properties may be deduced, such as the fact that at least one mathematician will keep engaged in a cycle of thinking and eating.

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

Figure 5: An approximation to the free abstraction.

7 Strong Preservation

In Sect. 5.1, we have identified conditions under which computed abstract models are optimal in the standard sense of Abstract Interpretation (see Sect. 2). This notion of optimality concerns the quality of the abstract interpretation used to compute an abstraction $\alpha(I(P))$, relative to the “ideal” abstraction $\alpha(I(P))$. In particular, it is optimality with respect to a given abstract domain.
A different notion of quality is the aptness of the abstract domain itself for the set of properties to be checked. In other words, given an abstract domain, how often will we get indeterminate answers as in the case of property (9) in Sect. 6? In posing this question, we enter in fact the area of specific applications of the framework presented so far: the answer to the question depends very much on the specific set of properties to be checked and on the programming language that is analysed. Nevertheless, we present a few general results that characterize the preservation quality of abstract domains. We say that a set of properties is strongly preserved when for every property, it is satisfied in the abstract model if and only if it is satisfied in the concrete model.

According to a well-known result ([BCG88]) in process equivalences, two (image-finite) transition systems satisfy the same CTL* formulae if and only if they are bisimilar. In our case this implies that

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

Proof. See [BCG88]. Note that $\rho$ is total by totality of $\alpha$. □

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

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

7.0.2 Lemma. $A$ is an abstraction which is $\rho$-bisimilar to $C$ if and only if $\alpha R^F = \alpha R^C$.

These results show that the price for strong preservation of full $\text{CTL}^*$ is that the attainable reduction of the concrete model is bounded by its quotient under bisimulation equivalence. However, it may well be the case that we can identify a subset of $\text{CTL}^*$ in which all the properties of interest can be expressed. Such a subset induces a coarser equivalence on the concrete states, in general. In [DGG93a] and [DGD+94], we develop algorithms which can be used to reduce the system with respect to the equivalences induced by $\text{VCTL}^*$ and by a single $\text{ACTL}$ formula.

8 Related Work

Property-preserving abstractions of reactive systems have been the topic of intensive research lately. Most of these efforts are based on the notion of simulation. Homomorphisms (see, e.g., [Gin68]), used in automata theory to construct language preserving reductions of automata, can be seen as a precursor of this. Adapted to our notion of transition system, $h : \Sigma \rightarrow \alpha \Sigma$ is a homomorphism iff $c \in I$ implies $h(c) \in \alpha I$ and $R(c, d)$ implies $\alpha R(h(c), h(d))$, where $\alpha I$ is the set of initial abstract states and $\alpha R$ the abstract transition relation. In [Mil71], Milner introduced the term simulation to denote a homomorphism between deterministic systems. Since then, it has been re-adapted to nondeterministic transition systems and has become popular in the areas of program refinement and verification, see, e.g., [Sif82, Sif83]. [Dil89, Kur89] focus on trace (linear time) semantics and universal safety and liveness properties. More recently, [CGL94, GL93] consider branching time semantics and preservation of both $\text{VCTL}^*$ as well as $\text{CTL}^*$. In these papers, as well as in [Kur89], the relation between concrete and abstract model is defined by means of a homomorphism $h$, which induces an equivalence relation $\sim$ on the concrete states, defined by $c \sim d \Leftrightarrow h(c) = h(d)$. The abstract states are then representations of the equivalence classes of $\sim$. It is shown that universal properties are preserved from the abstract to the concrete model.
[CGL94] also indicates how abstract models may be computed by abstract interpretations of the operations in a program. However, their notion of approximation is based on the subset ordering on abstract transition relations; they do not have the approximation relation \( \preceq \) which allows non-optimal abstract interpretations of individual operations. Also, preservation of existential properties is only possible via abstractions which are bisimilar to the concrete model, thus only allowing for relatively small reductions in the size of models.

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

A recent paper by Kelb, [KeI94], also discusses the preservation of universal and existential \( \mu \)-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 \( \mu \)-calculus may be verified by combining two types of abstractions through a so-called truth-failure-connection.

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

[CR94] presents a framework for the abstract interpretation of processes that pass values. Simulations to specify the relation between concrete and abstract systems are a generalization of our approach using Galois connections. It can easily be shown that \( R \rho \)-simulates \( \alpha R^\mathbb{F} \) while \( \alpha R^\mathbb{C} \rho \)-simulates \( R \), where \( \rho(c,a) \iff \alpha(c) \leq a \). Nevertheless, the use of Galois connections has two important advantages. Firstly, it allows to derive in a uniform fashion results on the preservation of both universal and existential properties, whereas general simulation relations do not guarantee the existence of abstract models in this case. Secondly, the mere requirement that a simulation must exist in order for preservation to hold has the drawback that it leaves far too much freedom in the choice of "good" abstractions. The Galois connection framework in which we developed our results may be seen as a successful attempt to try and quantify the notion of precision of abstractions by distinguishing between the notions of (optimal) abstraction and approximation. This point is discussed in the following comparison of our work with that of [LGS+93].

### 8.1 Comparing the simulation and Galois connection approaches

We focus on free abstractions. Given a concrete transition system and a set of abstract states which is related to the concrete states by a "description relation" \( \rho \subseteq \Sigma \times \sigma \Sigma \) (total on \( \Sigma \)), the requirement \( \rho^{-1} R \subseteq \sigma R \rho^{-1} \) (\( R \) simulates \( \sigma R \)) has many solutions in \( \sigma R \). From the point of view of property preservation, the \( \subseteq \)-minimal solutions are interesting. However, even such minimal solutions may have comparable quality, as illustrated by the example in Fig. 6, where the problem is to choose an optimal \( \sigma R \)-successor of \( a \) such that \( \rho^{-1} R \subseteq \sigma R \rho^{-1} \) is satisfied. The \( \subseteq \)-minimal solutions are obtained by taking either \( b_1 \) or \( b_2 \) as successor (but not both). However, choosing \( b_1 \) will generally give better property preservation, as it describes fewer concrete states.

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

\[
\rho \rho^{-1} \rho = \rho
\]
Expressed in words, it says that if two concrete states share a description (abstract state), then they share all descriptions. It is easy to see that the generality of simulations over Galois insertions, namely the possibility to have several optimal but mutually incomparable abstractions of a set of concrete states, is eliminated by this condition. In fact, requirement (11) implies that it is useless to have a ρ which is not functional. This is expressed in the following lemma (which can be found in [LGS+93]). It implies that whenever ρ(c, a) and ρ(c, a') (a ≠ 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 A, as, after all, the goal of abstraction is to produce minimal abstract systems.

8.1.1 LEMMA. If ρ is total on Σ, ρR is a ⊆-minimal relation such that R ρ-simulates ρR, and ρR−1ρ = ρ, then ρR−1 is a bisimulation on A.

PROOF. We have to show that ρR−1 and (ρR−1)−1 are simulations on A. Because (ρR−1)−1 = ρR−1, it suffices to show that ρR−1 is a simulation, i.e. (by Def. 2.0.5), that (ρR−1)−1 ⊆ ρR(ρR−1)−1, i.e., ρR−1ρRρR−1 ⊆ ρR−1ρRρR−1R. Because any minimal solution ρR satisfies ρR = ρ−1RρR (see [LGS+93]), (1) is equivalent to ρR−1RρR−1 ⊆ ρ−1RρR−1R. Because ρR−1RρR−1ρ = ρ and therefore also ρ−1RρR−1 = ρ−1, this is equivalent to ρ−1RρR−1ρR ⊆ ρ−1RρR−1ρR, which is true. □

So, in order to be able to distinguish optimal abstractions from approximations, [LGS+93] makes assumption (11), which renders their framework less general than the Galois connection approach, because, under the reasonable assumption that the abstract system does not contain bisimilar states, it forces ρ to be functional.

Consider Fig. 6 again. In our framework, the simulation relation ρ induces the following Galois insertion on sets of states: for any a ∈ A, γ(a) = {c | ρ(c, a)} and for any C ⊆ A, α(C) = \{a | γ(a) ≥ C\}, where \ denotes the meet operation corresponding to the ordering ≤ defined by a ≤ a' ⇔ γ(a) ⊆ γ(a'). Taking aR to be aR as specified by Def. 3.3.1 (item 1) yields b1 as the only successor of a, as desired.

9 Conclusions

The results of this paper may be seen from two points of view. From the position of Abstract Interpretation, we have presented a generalization of the framework extending it to the analysis of reactive
properties. This generalization consists in allowing the next-state relation of a non-deterministic transition system to be abstracted to a relation, and not a function as is common practice. This allows the analysis, via the abstraction, of not only universal properties — expressing that something holds along all possible executions —, but also existential properties — expressing the existence of paths satisfying some property. Furthermore, both safety as well as liveness properties are preserved. We have proven that the truth of every property expressible in $\text{CTL}^*$ is preserved from abstract to concrete model. As is common in Abstract Interpretation, the attained reduction depends solely on the choice of the abstraction function, thus allowing better reductions than is the case with minimization based on bisimulation. This was possible by considering abstract transition systems having two different transition relations, each preserving a separate fragment of $\text{CTL}^*$. The use of a Galois insertion to relate concrete and abstract states allowed the definition of both types of transitions over the same set of abstract states, resulting in the preservation of full $\text{CTL}^*$. The price to be paid is that there exist formulae — and increasingly many when the abstraction becomes coarser — which do not hold in the abstraction, and neither do their negations. In case of persistence on strong preservation (i.e., preservation of both truth and falsity of formulae), which renders the abstract model bisimilar to the concrete model, we have shown the implications for the form that the abstract domain takes.

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

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

In the light of the quest for fully automated verification methods, we are currently investigating the use of partition refinement algorithms for the construction of strongly preserving models; see the papers [DGG93a] and [DGD94]. Other, rather preliminary ideas point in the direction of using theorem provers and algebraic manipulation tools. Although the problem is undecidable in general, there may well be interesting subclasses that can be decided efficiently.

In a recent paper, [KDG95], we apply the ideas developed in this paper and in [Kel94] to verify $\mu$-calculus properties of a production cell ([DHKS95]) in a compositional fashion.
Acknowledgements  We thank Susanne Graf for many interesting and stimulating discussions, and Nissim Francez for his helpful comments. Anthony McIsaac pointed out some mistakes. The anonymous referees are acknowledged for their valuable suggestions.

References


Computing Science Reports

In this series appeared:

93/01 R. van Geldrop
93/02 T. Verhoef
93/03 T. Verhoef
93/04 E.H.L. Aarts J.H.M. Korst P.J. Zwijnenburg
93/05 J.C.M. Baeten C. Verhoef
93/06 J.P. Velikamp
93/07 P.D. Moerland
93/08 J. Verhoosel
93/09 K.M. van Hee
93/10 K.M. van Hee
93/11 K.M. van Hee
93/12 K.M. van Hee
93/13 K.M. van Hee
93/14 J.C.M. Baeten J.A. Bergstra
93/15 J.C.M. Baeten J.A. Bergstra R.N. Bol
93/16 H. Schepers J. Hoornan
93/17 D. Alsteijn P. van der Stok
93/18 C. Verhoef
93/19 G.J. Hoorenb
93/20 F.S. de Boer
93/21 M. Godish D. Dams G. Filé M. Bruynooghe
93/22 E. Poll
93/23 E. de Kogel
93/24 E. Poll and Paula Severi
93/25 H. Schepers and R. Genth
93/26 W.M.P. van der Aalst
93/27 T. Kloks and D. Kratsch
93/28 F. Kamareddine and R. Nederpelt
93/29 R. Post and P. De Bra
93/30 J. Deogun T. Kloks D. Kratsch

Deriving the Aho-Corasick algorithms: a case study into the synergy of programming methods, p. 36.

A continuous version of the Prisoner’s Dilemma, p. 17.

Quicksort for linked lists, p. 8.

Deterministic and randomized local search, p. 78.

A congruence theorem for structured operational semantics with predicates, p. 18.

On the unavoidability of metastable behaviour, p. 29.

A Formal Deterministic Scheduling Model for Hard Real-Time Executions in DEDOS, p. 32.


A Trace-Based Compositional Proof Theory for Fault Tolerant Distributed Systems, p. 27.

Hard Real-Time Reliable Multicast in the DEDOS system, p. 19.

A congruence theorem for structured operational semantics with predicates and negative premises, p. 22.

The Design of an Online Help Facility for ExSpect, p. 21.


Freeness Analysis for Logic Programs - And Correctness, p. 24.

A Typechecker for Bijective Pure Type Systems, p. 28.

Relational Algebra and Equational Proofs, p. 23.

Pure Type Systems with Definitions, p. 38.


Multi-dimensional Petri nets, p. 25.

Finding all minimal separators of a graph, p. 11.

A Semantics for a fine λ-calculus with de Bruijn indices, p. 49.

GOLD, a Graph Oriented Language for Databases, p. 42.

On Vertex Ranking for Permutation and Other Graphs, p. 11.
Derivation of delay insensitive and speed independent CMOS circuits, using directed commands and production rule sets, p. 40.


ILIAS, a sequential language for parallel matrix computations, p. 20.

Real Time Process Algebra with Infinitesimals, p.39.

Abstract Reduction and Topology, p. 28.

Non Interleaving Process Algebra, p. 17.

Design and Analysis of Dynamic Leader Election Protocols in Broadcast Networks, p. 73.

A general conservative extension theorem in process algebra, p. 17.

Job Shop Scheduling by Constraint Satisfaction, p. 22.


Temporal operators viewed as predicate transformers, p. 11.

Automatic Verification of Regular Protocols in P/T Nets, p. 23.

A taxonomy of finite automata construction algorithms, p. 87.

A taxonomy of finite automata minimization algorithms, p. 23.

A precise clock synchronization protocol, p.


Verifying Sequentially Consistent Memory using Interface Refinement, p. 20.

The object-oriented paradigm, p. 28.

Canonical typing and Π-conversion, p. 51.


Graph Isomorphism Models for Non Interleaving Process Algebra, p. 18.


Time and the Order of Abstract Events in Distributed Computations, p. 29.


A Hierarchical Diagrammatic Representation of Class Structure, p. 22.
<table>
<thead>
<tr>
<th>Page</th>
<th>Authors</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>94/09</td>
<td>J.C.M. Baeten, J.A. Bergstra</td>
<td>Process Algebra with Partial Choice, p. 16.</td>
</tr>
<tr>
<td>94/10</td>
<td>T. Verhoeff</td>
<td>The testing Paradigm Applied to Network Structure, p. 31.</td>
</tr>
<tr>
<td>94/13</td>
<td>R. Seljé</td>
<td>A New Method for Integrity Constraint checking in Deductive Databases, p. 34.</td>
</tr>
<tr>
<td>94/14</td>
<td>W. Peremans</td>
<td>Ups and Downs of Type Theory, p. 9.</td>
</tr>
<tr>
<td>94/16</td>
<td>R.C. Backhouse, H. Doornbos</td>
<td>Mathematical Induction Made Calculational, p. 36.</td>
</tr>
<tr>
<td>94/18</td>
<td>F. Kamareddine, R. Nederpelt</td>
<td>Refining Reduction in the Lambda Calculus, p. 15.</td>
</tr>
<tr>
<td>94/19</td>
<td>B.W. Watson</td>
<td>The performance of single-keyword and multiple-keyword pattern matching algorithms, p. 46.</td>
</tr>
<tr>
<td>94/20</td>
<td>R. Bloo, F. Kamareddine, R. Nederpelt</td>
<td>Beyond β-Reduction in Church's λ →, p. 22.</td>
</tr>
<tr>
<td>94/22</td>
<td>B.W. Watson</td>
<td>The design and implementation of the FIRE engine: A C++ toolkit for Finite automata and Regular Expressions.</td>
</tr>
<tr>
<td>94/23</td>
<td>S. Mauw and M.A. Reniers</td>
<td>An algebraic semantics of Message Sequence Charts, p. 43.</td>
</tr>
<tr>
<td>94/24</td>
<td>D. Dams, O. Grumberg, R. Gerth</td>
<td>Abstract Interpretation of Reactive Systems: Abstractions Preserving VCTL*, CTL*, and CTL*.</td>
</tr>
<tr>
<td>94/25</td>
<td>T. Kloks</td>
<td>K₁₀-free and W₆-free graphs, p. 10.</td>
</tr>
<tr>
<td>94/29</td>
<td>J. Hoornan</td>
<td>Correctness of Real Time Systems by Construction, p. 22.</td>
</tr>
<tr>
<td>94/31</td>
<td>B.W. Watson, R.E. Watson</td>
<td>A Boyer-Moore type algorithm for regular expression pattern matching, p. 22.</td>
</tr>
<tr>
<td>94/33</td>
<td>T. Laan</td>
<td>A formalization of the Ramified Type Theory, p.40.</td>
</tr>
<tr>
<td>94/34</td>
<td>R. Bloo, F. Kamareddine, R. Nederpelt</td>
<td>The Barendregt Cube with Definitions and Generalised Reduction, p. 37.</td>
</tr>
<tr>
<td>94/35</td>
<td>J.C.M. Baeten, S. Mauw</td>
<td>Delayed choice: an operator for joining Message Sequence Charts, p. 15.</td>
</tr>
<tr>
<td>94/36</td>
<td>F. Kamareddine, R. Nederpelt</td>
<td>Canonical typing and Π₁-conversion in the Barendregt Cube, p. 19.</td>
</tr>
<tr>
<td>94/38</td>
<td>A. Bijlsma</td>
<td>Point-free substitution, p. 10.</td>
</tr>
<tr>
<td>Page</td>
<td>Authors</td>
<td>Title</td>
</tr>
<tr>
<td>------</td>
<td>---------</td>
<td>-------</td>
</tr>
<tr>
<td>94/39</td>
<td>A. Blokhuis, T. Klop</td>
<td>On the equivalence covering number of splitgraphs, p. 4.</td>
</tr>
<tr>
<td>94/40</td>
<td>D. Altschin</td>
<td>Distributed Consensus and Hard Real-Time Systems, p. 34.</td>
</tr>
<tr>
<td>94/41</td>
<td>T. Klop, D. Kratsch</td>
<td>Computing a perfect edge without vertex elimination ordering of a chordal bipartite graph, p. 6.</td>
</tr>
<tr>
<td>94/43</td>
<td>R.C. Backhouse, M. Blijsterveld</td>
<td>Category Theory as Coherently Constructive Lattice Theory: An Illustration, p. 35.</td>
</tr>
<tr>
<td>94/46</td>
<td>R. Bloo, F. Kamareddine, R. Nederpelt</td>
<td>Mathematics of Program Construction Group</td>
</tr>
<tr>
<td>94/47</td>
<td>Mathematics of Program Construction Group</td>
<td>Fixed-Point Calculus, p. 11.</td>
</tr>
<tr>
<td>94/49</td>
<td>H. Geuvers</td>
<td>A short and flexible proof of Strong Normalization for the Calculus of Constructions, p. 27.</td>
</tr>
<tr>
<td>94/51</td>
<td>W. Penczek, R. Kuiper</td>
<td>A Partial Order Approach to Branching Time Logic Model Checking, p. 20.</td>
</tr>
<tr>
<td>95/01</td>
<td>J.J. Aarts</td>
<td>Concrete process algebra, p. 134.</td>
</tr>
<tr>
<td>95/02</td>
<td>J.C.M. Baeten, C. Verhoef</td>
<td>Formalizing Process Algebraic Verifications in the Calculus of Constructions, p. 49.</td>
</tr>
<tr>
<td>95/03</td>
<td>J.J. Aarts</td>
<td>An Isotopic Invariant for Planar Drawings of Connected Planar Graphs, p. 9.</td>
</tr>
<tr>
<td>95/04</td>
<td>P. Severi</td>
<td>A Type Inference Algorithm for Pure Type Systems, p. 20.</td>
</tr>
<tr>
<td>95/06</td>
<td>G.A.M. de Bruyn, O.S. van Roosmalen</td>
<td>Drawing Execution Graphs by Parsing, p. 10.</td>
</tr>
<tr>
<td>95/07</td>
<td>R. Bloo</td>
<td>Preservation of Strong Normalisation for Explicit Substitution, p. 12.</td>
</tr>
<tr>
<td>95/08</td>
<td>J.C.M. Baeten, J.A. Bergstra</td>
<td>Discrete Time Process Algebra, p. 20.</td>
</tr>
<tr>
<td>Year</td>
<td>Authors</td>
<td>Title</td>
</tr>
<tr>
<td>------</td>
<td>-------------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>95/10</td>
<td>R.C. Backhouse, R. Verhoeven, O. Weber</td>
<td>MathPad: A System for On-Line Preparation of Mathematical Documents, p. 15</td>
</tr>
<tr>
<td>95/11</td>
<td>R. Selje</td>
<td>Deductive Database Systems and integrity constraint checking, p. 36.</td>
</tr>
<tr>
<td>95/12</td>
<td>S. Mauw and M. Reniers</td>
<td>Empty Interworkings and Refinement</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Semantics of Interworkings Revised, p. 19.</td>
</tr>
<tr>
<td>95/14</td>
<td>A. Ponce, C. Verhoef,</td>
<td>De proceedings: ACP'95, p.</td>
</tr>
<tr>
<td></td>
<td>S.F.M. Vlijmen (eds.)</td>
<td></td>
</tr>
</tbody>
</table>