Model Checking Using Adaptive State and Data Abstraction

(extended abstract)

Dennis Dams[1], Rob Gerth[1†],
Gert Döhmen[2‡], Ronald Herrmann[2], Peter Kelb[2],
Hergen Pargmann[3§]

[1] Eindhoven University of Technology, Dept. of Math. and Computing Science, P.O. Box 513, 5600 MB Eindhoven, The Netherlands. Email: {wsindd,robg}@win.tue.nl
[2] OFFIS, Westerstraße 10–12, 26111 Oldenburg, Germany. Email: Peter.Kelb@arbi.informatik.uni-oldenburg.de
[3] University of Oldenburg, 26121 Oldenburg, Germany

Abstract. We present a partitioning algorithm for checking ACTL specifications that distinguishes between states only if this is necessary to ascertain the specification. This algorithm is then generalized to also abstract from the variable values in the states. Here, too, the values between which the algorithm distinguishes are determined by what is needed to decide whether or not the specification holds. The resulting algorithm is being implemented in an ROBDD based model checker for VHDL/S.

Keywords: model checking, ACTL, abstract interpretation, state partitioning, binary decision diagrams (BDDs)

1 Introduction

The major stumbling block for successful application of model checking to complex systems is the size of the state graphs of these systems. This is the so-called *state explosion* problem. Although an impressive step forward has been made by the introduction of ROBDD based techniques [BCM+90], this step has not moved us beyond the block; rather, it has pushed it further ahead. In order to deal with the state explosion problem in a fundamental way, we need versatile abstraction methods, that allow the abstraction from any details that are not relevant to the property being checked.

Examples of such methods are the reduction of models by collapsing states which are bisimulation equivalent [BFH+92] and the partial order approaches that allow parts of the state graph caused by different interleavings of independent, parallel actions to be ignored [Val91, GW91, Pel93]. These methods are safe for large classes of specifications. The first one applies to any property that cannot distinguish between bisimilar states — which is in fact the case for most common specification languages. The latter methods basically apply to linear temporal logic specifications; but also see [GKPP94].

However, if the specific set of properties to be verfied is known beforehand, many more details will become irrelevant and much better reductions can *in principle* be

† Currently working in ESPRIT project P6021 "Building Correct Reactive Systems (REACT)".
‡ Currently working in ESPRIT project P6128 "Formal Methods in Hardware Design (FORMAT)".
§ Currently working in Projekt "Informationssysteme".

effected. The problem then is to find out which are the relevant details needed to check some given set of properties. In the light of the quest for automated methods, we need efficient algorithms to perform this task.

In [DGG93], these problems are dealt with by an approach which is based on iterative refinement of the model under construction by constructing ever finer partitions of the concrete state space; as such, it generalizes the state partitioning method of [BFH+92]. Starting with an ACTL specification[4] φ to be verified, the full model $\mathcal{C}$ and an abstract model that contains no details, the model is successively refined until it contains enough information to either prove or disprove the formula. Each of the models that is generated in such a sequence of iterative refinements has the property that it *preserves* φ; i.e., whenever φ holds in the abstract model $\mathcal{A}$, it also holds in the full model $\mathcal{C}$, called the *concrete model* henceforth. When φ does *not* hold however, there are two possible reasons. One is that the abstract model does not yet contain enough detail, although φ does hold in the concrete; the other is that φ does not hold in the concrete model. Further refinement of the model will then bear out which one of these cases is true: either there will be a point in the refinement process where φ becomes true in the abstract model, or at some point no further refinement is possible—the model has become *stable*—while φ is still false. It is shown that in the latter case, φ is false in the concrete model as well; i.e., stable models *strongly preserve* satisfaction of specifications.

An obvious factor determining the success of an abstraction methodology is the possibility it provides to construct abstractions in a *direct* fashion, i.e., without intermediate construction of the complete detailed model. Otherwise, the target of avoiding the state explosion would clearly be missed. Yet, all the above mentioned abstraction methods require access to the concrete transition relation, which still may be prohibitively expensive for the state partitioning based methods.

This paper starts by formulating a state partitioning algorithm for ACTL within an abstract interpretation framework. The basic algorithm is then generalized so that the concrete transition relation can be abstracted as well; but in such a way that the algorithm will *automatically* adapt the abstraction until it can decide truth or falsehood of the specification. Both the state splittings as well as the abstractions of the transition relation are governed by what is needed to establish the validity of the specification.

We obtain a two level approach. On the first level, an abstraction R_d of the concrete transition relation R_c is choosen, yielding abstract transition systems $\mathcal{D}$, in such a way that satisfaction of the specification φ is preserved. Then, on the second level, the partitioning algorithm constructs models $\mathcal{A}$ that, in their turn, are abstractions of $\mathcal{D}$. This algorithm terminates either if the specification φ is satisfied in the model $\mathcal{A}$ just constructed, or if $\mathcal{A}$ becomes stable. In the latter case φ may still not hold, from which we conclude that φ is invalid in $\mathcal{D}$ as well. However, $\mathcal{D}$ itself is an abstraction of the concrete model $\mathcal{C}$ and we cannot immediately conclude that φ is false in $\mathcal{C}$. Thus we face the problem of separating 'true' counter examples to the satisfaction of φ from artifacts caused by the first level abstraction. In the paper we show that it is possible to determine whether a counter example is genuine or not. Assuming we do not have a true counter example, we must change the first-level abstraction $\mathcal{D}$ so as to include more detail; i.e., the choosen abstraction $\mathcal{D}$ must be refined. Then the second-level partitioning algorithm

[4] ACTL is the universal fragment of CTL.

again will iteratively construct models that now are abstractions of the new $\mathcal{D}$. As we obviously want to retain what we have computed—$\mathcal{D}$ and the stable $\mathcal{A}$—two other problems are raised: how can the previously stable model $\mathcal{A}$ be used as the starting point for the partitioning algorithm and how can a first-level abstraction $\mathcal{D}$ be adapted rather than recomputed. We provide answers by defining a family of abstractions—depth-k abstractions—and by concretely showing how models can be adapted and re-used.

The algorithm is being implemented in a tool for the verification of VHDL/S code. VHDL/S [HSD+93] is a language developed in the ESPRIT project 6128 FORMAT. The goal of FORMAT is to provide an environment for the efficient development of correct VHDL designs, where correctness is pursued along two different lines: by synthesis and verification. VHDL/S integrates four different and self-contained linguistic paradigms: VHDL, state based specifications [Har87], symbolic timing diagrams [SD93], and temporal logic. The former two are operational, the latter two are declarative in nature. State based specifications are translated into ROBDDs [HK94], while specifications written in VHDL are first translated into Petri Nets, which have actions (e.g., assignments) associated with their transitions. In a second step, these nets are translated into ROBDDs so that symbolic model checking can be applied. The iterative refinement algorithm is fit into this second translation phase, as this is the point at which the state explosion occurs.

The implementation of the state partitioning algorithm is also ROBDD-based, with the obvious advantage that many previously implemented modules of the system can be reused. A key point here is that the first-level abstraction allows us to approximate the concrete model. We exploit this by limiting the size of ROBDDs, so that concrete states are approximated by sets of states. Furthermore, by interpreting the actions that are associated with transitions in nets over such abstract states, an approximation of the transition relation is obtained.

The next section gives some background material. In Sect. 3 we develop the ACTL partitioning algorithm and extend it to use data abstraction in Sect. 4. A sketch of an ROBDD implementation embedded in a VHDL/S model checker would have occupied the penultimate section, were it not for the page limitation imposed by Springer. Finally, in Sect. 5 we draw some conclusions and point to future work.

2 Preliminaries

ACTL and ECTL We assume some countable set of local propositional symbols $\text{Prop} = \{p, q, \ldots\}$. We define ACTL (universal Computation Tree Logic) in its positive normal form in which negations only apply to propositions. The set of *well-formed formulae* (written wff) is defined as follows

— for $p \in \text{Prop}$, p and $\neg p$ are wff,
— if φ and ψ are wff, then so are $\varphi \vee \psi$ and $\varphi \wedge \psi$,
— if φ and ψ are wff, then so are $\mathsf{AX}\varphi$, $\mathsf{AU}(\varphi, \psi)$ and $\mathsf{AV}(\varphi, \psi)$.

ECTL (existential Computation Tree Logic) is defined as $\{\neg\varphi \mid \varphi \in ACTL\}$.

The AV-modality is needed because the use of negation is constrained. Otherwise, we would have had $\mathsf{AV}(\varphi, \psi) \equiv \mathsf{A}\neg\mathsf{U}(\neg\varphi, \neg\psi)$ as can be gleaned from the satisfaction definition below. Write AW to denote either AU or AV. Let $\text{Atoms}(\varphi)$ be the set of subformulae of φ that are either propositions or of the form $\mathsf{AX}\psi$ or $\mathsf{AW}(\psi, \psi')$.

Transition systems and satisfaction ACTL is intended to express properties about computations which are generated by transition systems $T = (V, I, R)$ where V is some set of states, $I \subseteq V$ is the set of initial states and R is the transition relation which is always assumed to be *total* to circumvent some technicalities. A *path* σ in T is an infinite sequence $\sigma = s_0 s_1 \cdots$ of states such that $s_i \; R \; s_{i+1}$ for every i. Write σ_n for state s_n in σ. An *s-path* is a path that starts at state s. Define the *precondition function*, pre_R, associated with R by $pre_R(D) = \{c \mid \exists d \in D \;\; c \; R \; d\}$. Write *pre* if the transition relation is clear.

As usual, we need a *valuation function* $\mathbf{V}: V \to 2^{\text{Prop}}$ to define which propositions are true in which states. A (Kripke) model M is a pair $(T, \mathbf{V})$ of a transition system and a valuation function. $M, s \models \varphi$ denotes that the formula φ is *true* at the state s in the transition system T with valuation $\mathbf{V}$. Its inductive definition follows:

- $M, s \models p$ iff $p \in \mathbf{V}(s)$, for $p \in \text{Prop}$,
- $M, s \models \neg p$ iff $p \notin \mathbf{V}(s)$, for $p \in \text{Prop}$,
- $M, s \models \varphi \vee \psi$ iff $M, s \models \varphi$ or $M, s \models \psi$,
- $M, s \models \varphi \wedge \psi$ iff $M, s \models \varphi$ and $M, s \models \psi$,
- $M, s \models \mathsf{AX}\varphi$ iff $M, \sigma_1 \models \varphi$ for every s-path σ,
- $M, s \models \mathsf{AU}(\varphi, \psi)$ iff for every s-path σ there is a $k \geq 0$ such that $M, \sigma_k \models \psi$ and $M, \sigma_i \models \varphi$ for every $i < k$,
- $M, s \models \mathsf{AV}(\varphi, \psi)$ iff there is no s-path σ such that for some $k \geq 0$ $M, \sigma_k \models \neg\psi$ and $M, \sigma_i \models \neg\varphi$ for every $i < k$.

For $S \subseteq V$, define $M, S \models \varphi$ by $\forall s \in S \;\; M, s \models \varphi$. $M \models \varphi$ denotes $M, I \models \varphi$. When clear from the context, we omit M.

For $\mathsf{AU}(\varphi, \psi)$ and $\mathsf{AV}(\varphi, \psi)$-formulae, we define *approximants* as follows:

$$\begin{array}{ll} \mathsf{AU}_0(\varphi, \psi) = \text{false} & \mathsf{AU}_{i+1}(\varphi, \psi) = \psi \vee (\varphi \wedge \mathsf{AXAU}_i(\varphi, \psi)) \\ \mathsf{AV}_0(\varphi, \psi) = \text{true} & \mathsf{AV}_{i+1}(\varphi, \psi) = \psi \wedge (\varphi \vee \mathsf{AXAV}_i(\varphi, \psi)) \end{array}$$

If the transition system T has N states then for every state s

$$M, s \models \left(\mathsf{AU}(\varphi, \psi) \equiv \bigvee_{i<N} \mathsf{AU}_i(\varphi, \psi)\right) \wedge \left(\mathsf{AV}(\varphi, \psi) \equiv \bigwedge_{i<N} \mathsf{AV}_i(\varphi, \psi)\right). \quad (1)$$

So, we also have $M, s \models \mathsf{AW}_N(\varphi, \psi) \equiv \mathsf{AW}_{N+i}(\varphi, \psi)$ for any $i > 0$. In other words, on finite transition systems the truth of AW-formulae is determined by a finite set of approximants.

Abstract Interpretation Many of the results and constructions below are most easily expressed using the language of *Abstract Interpretation* [CC77]; a general framework to define static analyses of programs. The basic tenet is that the operations of a programming language which operate on concrete values can be mimicked by corresponding abstract operations defined over abstract values that describe sets of concrete values.

The starting point is choosing a set of *abstract states*, V_a. Each abstract state $a \in V_a$ describes a set of concrete states. Conversely, every set $C \subseteq V_c$ of concrete states has a 'best', or most precise description. This is formalized via a *concretization function* $\gamma: V_a \to 2^{V_c}$ and an *abstraction function* $\alpha: 2^{V_c} \to V_a$. For each a, $\gamma(a)$ is the set of all concrete states described by a; for each $C \subseteq V_c$, $\alpha(C)$ is the most precise description

in the sense that $C \subseteq \gamma(\alpha(C))$ and $C \subseteq \gamma(a)$ implies $\gamma(\alpha(C)) \subseteq \gamma(a)$ for any $a \in V_a$. Thus, $\alpha(C)$ is the least description of C w.r.t. the *approximation ordering* $\preceq$ on V_c defined by $a \preceq b$ iff $\gamma(a) \subseteq \gamma(b)$. A given γ uniquely determines an appropriate α (if it exists) by setting $\alpha(C)$ to be the least a such that $\gamma(a) \supseteq C$. The α thus defined is written $\gamma^\flat$. We mention that, similarly, α determines a unique appropriate γ as well.

These requirements are often captured by saying that (α, γ) is a *Galois insertion* from $(2^{V_c}, \subseteq)$ to $(V_a, \preceq)$: (i) α and γ are total and monotonic, (ii) for every $C \in 2^{V_c}$ we have $(\gamma \circ \alpha)(C) \supseteq C$, and (iii) for every $a \in V_a$ we have $(\alpha \circ \gamma)(a) = a$.

Given such an abstract interpretation of the data, functions $f: V_c \to V_c$ can be described by *safe* abstract interpretations $f_a: V_a \to V_a$ that satisfy $f_a(a) \succeq \alpha(f(\gamma(a)))$.[5] In particular, there is a *precise* abstract interpretation of f defined by $\bar{f}_a = \alpha \circ f \circ \gamma$ and f_a is safe just in case $f_a \succeq \bar{f}_a$ (pointwise). Safeness means that given a description of the parameter, f_a yields a description of the result value.

A static analysis can then be viewed as an abstract execution of the program in which data and operations are abstractly interpreted, yielding a description of any concrete execution.

Binary decision diagrams Reduced Ordered BBDs [Bry86, Bry92] are a way to economically represent boolean functions in a canonical way. Although for most boolean functions the size of their ROBDD representation is exponentially large, in most practical cases the ROBDDs are sufficiently small. This, together with the fact that boolean operations, equivalence and tautology checking can be done very efficiently on ROBDDs, is the reason why ROBDDs are so popular. ROBDDs only supply a canonical representation relative to an arbitrary but fixed ordering on the boolean (input) variables and this ordering greatly influences the size of the ROBDDs.

The use of ROBDDs in model checking is based on coding transition relations as boolean functions. Given a transition system (V, I, R), take vectors $\mathbf{x}$, $\mathbf{x}'$ of boolean variables long enough to code for all states in V (e.g., take $|\mathbf{x}|, |\mathbf{x}'| \geq {}^2\log(|V|)$). Then, define a boolean function $\ulcorner R \urcorner(\mathbf{x}, \mathbf{x}')$[6] by

$$\ulcorner R \urcorner(\mathbf{x}, \mathbf{x}') = 1 \iff \exists x, x' \in V \;\; x \; R \; x' \;\&\; \beta(x) = \mathbf{x} \;\&\; \beta(x') = \mathbf{x}' \;,$$

where β is the coding function that maps states to bit strings.

Symbolic model checking is based on such ROBDD representations [BCM+90]. The basic operation that needs to be done is computing preconditions, $pre(C)$, which translates into computing *relational products* $\exists \mathbf{x}' \big(\ulcorner R \urcorner(\mathbf{x}, \mathbf{x}') \wedge \ulcorner C \urcorner(\mathbf{x}') \big)$. Obviously, to represent a set as an ROBDD we use its characteristic predicate.

3 ACTL Partitioning

The aim is to develop an algorithm that allows verifying an ACTL-specification without generating the complete state-graph of the system to be verified. Specifically, we want to verify the specification using an *abstraction* of the state-graph. The type of abstraction

[5] $f(C) = \{f(c) \mid c \in C\}$.

[6] We usually do not make a distinction between the boolean function $\ulcorner R \urcorner$ and its ROBDD representation.

that we have in mind is characterized by the following statement based on results from [DGG94].

An *abstraction* of $\mathcal{C} = (V_c, I_c, R_c)$ is a transition system $\mathcal{A} = (V_a, I_a, R_a)$ for which there is a *concretization function* γ: $V_a \rightarrow 2^{V_c}$ such that R_a satisfies

$$\forall a, b \in V_a \; (\; \exists c \in \gamma(a) \; \exists d \in \gamma(b) \;\; c \; R_c \; d \;\; \Rightarrow \;\; a \; R_a \; b \;)$$

and I_a satisfies $\forall c \in I_c \exists a \in I_a \; c \in \gamma(a)$. Satisfaction of a proposition p in an abstract state a is defined by

$$a \models p \;\; \text{iff} \;\; \gamma(a) \models p \; . \tag{2}$$

Satisfaction of other formulae is then defined as usual.

For such abstract systems, the logic ACTL is *weakly preserved*:

$$\forall \varphi \in ACTL, \; a \in V_a \;\; \mathcal{A}, a \models \varphi \;\Rightarrow\; \mathcal{C}, \gamma(a) \models \varphi \tag{3}$$

Write $\gamma_{\mathcal{C}}$ to explicitly indicate that the transition system $\mathcal{C}$ is being abstracted.

If γ is part of a Galois Insertion (α, γ) (as will always be the case in this paper), then we may rephrase the above statement by saying that in our context the proper notion of precise abstraction of a (transition) relation R_c w.r.t. (α, γ) is defined by

$$R_a = \{(a, b) \mid \exists c, d \in V_c \;\; c \in \gamma(a) \;\&\; c \; R_c \; d \;\&\; \alpha(d) = b\} \; .$$

Again, see [DGG94] for more details.

As every transition system trivially is an abstraction of itself (up to the difference between states c and singletons $\{c\}$, which is irrelevant in this context), there is no formal distinction between 'concrete' and 'abstract' transition systems. The existence of the concretization function implies that we can view the states in an abstraction as predicates over the concrete states; which we shall often do (and, hence, take $V_a = 2^{V_c}$). By this interpretation, the concretization function is fixed as the standard interpretation of predicates over V_c.

3.1 The Basic Partitioning Algorithm

Since abstractions only weakly preserve ACTL (i.e., in (3) the implication in the other direction does not hold in general), there is a potential problem if a specification φ happens not to be satisfied in the concrete system, because in general we cannot draw that conclusion given some abstraction. The reason can be gleaned from (2)[7]: we may well have that $\mathcal{A}, a \not\models \varphi$ while there is a concrete state $c \in \gamma(a)$ for which $\mathcal{C}, c \models \varphi$. Such a formula is said to be *not determined* in a.

Is it possible to decide whether $\mathcal{C} \not\models \varphi$ by analysing an abstraction $\mathcal{A}$? The inductive nature of the satisfaction definition suggests that this should be possible in case all subformulae of φ are determined in (all states in) $\mathcal{A}$. A closer look reveals the following [DGG93]:

[7] We stress that this definition of satisfaction is forced by the aim to have weak preservation.

Define the *companion* of φ, $\mathrm{Comp}(\varphi)$ as

$$\begin{aligned}&\mathrm{Atoms}(\varphi) \setminus \{\mathsf{AW}(\psi,\psi') \mid \psi,\psi' \in \mathrm{ACTL}\}\\ &\quad \cup \{\mathsf{AW}_i(\psi,\psi') \mid i>0,\ \mathsf{AW}(\psi,\psi') \in \mathrm{Atoms}(\varphi)\}\ .\end{aligned}$$

If every formula in $\mathrm{Comp}(\varphi)$ is determined in $\mathcal{A}$, then φ is strongly preserved in $\mathcal{A}$:

$$\mathcal{A}, a \models \varphi \iff \mathcal{C}, \gamma(\alpha) \models \varphi\ . \tag{4}$$

Such companion sets can be stratified: on the lowest level are the propositions; on each higher level one finds formulae of the form $\mathsf{AX}\psi$ where ψ is some boolean combination of formulae of lower levels.

This suggests a strategy to construct an abstract model for some given φ in which φ is strongly preserved: start with some abstraction and 'add' more detail by partitioning, or splitting, states in which some formula is not determined, into parts in which it is; one part in which the formula holds and the other part in which it does not. The structure of $\mathrm{Comp}(\varphi)$ ensures that one can always partition states relative to formulae of the form $\mathsf{AX}\psi$ with ψ already determined (except for the first step during which the splitting is relative to propositions). In [DGG93] we developed algorithms along this line for constructing minimal abstractions in which every $\varphi \in \mathrm{ACTL}$ is determined and also for single formulae. Figure 1 gives a generalized version of the single-formulae *partitioning* algorithm.

The pseudo code uses a number of primitive operations, that must satisfy certain requirements.

$\mathcal{A}$:= "Initial abstraction"
for $p \in \mathrm{Atoms}(\varphi) \cap \mathrm{Prop}$ **do** $\mathcal{A} := \mathrm{split}(\mathcal{A}, p, \mathsf{pre})$ **od**
$F := \mathrm{Comp}(\varphi) \setminus \mathrm{Prop}$
repeat
  pick $\psi \in \min_{\mathcal{A}}(F)$; $F := F \setminus \{\psi\}$
  $s := \mathrm{splitter}(\mathsf{pre}, \psi)$; $\mathcal{A} := \mathrm{split}(\mathcal{A}, s, \mathsf{pre})$
until $\mathrm{stable}_{\varphi}(\mathcal{A})$

Fig. 1. Partitioning algorithm for $\varphi \in \mathrm{ACTL}$

Ordering on F The algorithm splits states with respect to the minimal elements of F. So, the requirement on the ordering is that if $\mathsf{AX}\psi \in \min_{\mathcal{A}}(F)$ then ψ should be determined in $\mathcal{A}$.

An abstraction is partitioned w.r.t. a formula ψ in two steps.

Splitter This function determines the states in which ψ holds. Consequently, it satisfies $a = \mathrm{splitter}(\mathsf{pre}, \psi)$ iff $\gamma(a) = \{c \mid \mathcal{C}, c \models \psi\}$. As $\psi = \mathsf{AX}\psi'$, we can compute $\mathrm{splitter}(\mathsf{pre}, \psi)$ as the characteristic predicate of $\{c \mid \forall d\ c\ R_c\ d \rightarrow d \in \gamma(||\psi'||)\}$.

Here, we have confused abstract states and predicates; furthermore, $||\psi'||$ denotes the characteristic predicate of ψ', which has been computed in a previous iteration. Define the *precondition function* pre by $\mathsf{pre}(b) = a$ iff $\gamma(a) = \{c \mid \exists d \;\; c \; R_c \; d \wedge d \in \gamma(b)\}$, or, more abstractly, as $\gamma^\flat \circ pre_c \circ \gamma$.[8] Thus, there is essentially no distinction between pre and pre_c. Later on this will change. Now we can define splitter(pre, ψ) = $\neg\mathsf{pre}(\neg\psi')$.

Splitting Next, we split all abstract states and compute the abstract transition relation. We have $\mathrm{split}(\mathcal{A}, s, \mathsf{pre}) = (V_a', I_a', R_a')$ where $V_a' = \{a \wedge s, \; a \wedge \neg s \mid a \in V_a\}$, $I_a' = \{a' \mid \gamma(a') \cap I \neq \emptyset, \; a' \in V_a'\}$ and $R_a' = \{(a', b') \mid a' \wedge \mathsf{pre}(b') \neq \text{false}, \; a', b' \in V_a'\}$. If we expand definitions, we find that $a' \wedge \mathsf{pre}(b') \neq$ false rewrites to $\exists c \in \gamma(a') \; \exists d \in \gamma(b') \;\; c \; R_c \; d$ as should be the case. Here, too, we have confused abstract states and predicates.

Termination We may stop partitioning states either when φ becomes valid in some abstraction or when $\mathrm{Comp}(\varphi)$ becomes determined. Hence, we may take

$$\mathrm{stable}_\varphi(\mathcal{A}) = (\mathcal{A} \models \varphi) \vee (\mathrm{Comp}(\varphi) \text{ is determined in } \mathcal{A}) \; .$$

Even if $\mathcal{C} \not\models \varphi$, the partitioning algorithm will terminate for finite-state systems: although it is true that any atom $\mathsf{AW}(\psi, \psi')$ will contribute an infinite set of approximants to the companion, by (1) there will only be finitely many among them which will cause states to split; i.e., at most the first N aproximants, where N is the number of concrete states.

$\mathrm{Comp}(\varphi)$ is determined if no companion formula causes an additional split. Satisfaction of φ can be checked with any ACTL (or CTL) model checker.

Optimizations Even on this abstract level there are some optimizations to the basic algorithm possible. We briefly mention some. It is possible to *update* the abstract transition relation after a splitting instead of recomputing it and if the transition relation is deterministic then the computation of splitters simplifies.

More importantly, as there is a notion of initial state in which specifications should hold, it pays off to do a simultaneous reachability analysis while splitting. The reason is that abstractions preserve non-reachability; i.e., if some abstract state a becomes unreachable in an abstraction then every concrete state in $\gamma(a)$ will be unreachable in the concrete model. This is expected to greatly reduce the size of the models. See [DGG93] for details.

4 Data Abstraction

The above method abstracts from the concrete states which induces an abstraction of the transition relation. However, to compute pre, (parts of) the concrete transition relation is needed and this can be quite expensive in terms of both space and time. On the other hand, we may perform the partitioning algorithm w.r.t. an underlying transition that is already an abstraction of $\mathcal{C}$, thus making computing pre easier. Transitivity of abstractions guarantees weak preservation. We obtain two levels of abstraction: the concrete transition system $\mathcal{C}$ is first 'data abstracted' into $\mathcal{D} = (V_d, R_d)$; the partitioning

[8] Formally speaking, we have thus defined pre to be the precise abstract interpretation of pre_c (i.e., of pre_{R_c}) w.r.t the Galois insertion $(\gamma^\flat, \gamma)$.

algorithm then computes an abstraction of $\mathcal{D}$ (w.r.t. a pre that is determined by R_d). An abstract transition between two abstract states A and A' is illustrated in the picture on this page.

The stable abstraction of $\mathcal{D}$ computed by the partitioning algorithm for some specification φ is strongly preserving w.r.t. $\mathcal{D}$ (i.e., (4) holds when $\mathcal{D}$ is substituted for $\mathcal{C}$). Unfortunately, this does not imply that there is strong preservation w.r.t. $\mathcal{C}$. More precisely, we have strong preservation w.r.t. a larger transition relation on the concrete states:

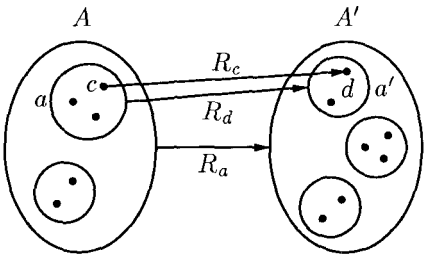
> Let $\mathcal{A}$ be a stable abstraction of $\mathcal{D}$ as computed by the algorithm and let $\mathcal{D}$ be an abstraction of $\mathcal{C}$ with concretization function γ^d. Define $R_{\mathsf{pre}} = \{(c,d) \mid c, d \in V_c,\ c \in \gamma^d(\mathsf{pre}((\gamma^d)^\flat(d))\}$. Note that $R_c \subseteq R_{\mathsf{pre}}$. We have
> (i) (V_c, R_{pre}) is an abstraction of $\mathcal{C}$, and
> (ii) $\mathcal{A} \models \varphi$ iff $(V_c, R_{\mathsf{pre}}) \models \varphi$.

An iterative process is suggested: choose some initial data abstraction $\mathcal{D}$ and compute a stable abstraction $\mathcal{A}$. If it does not satisfy the specification, choose a new data abstraction $\mathcal{D}'$ that is more detailed in the sense that $R_{\mathsf{pre}'} \subseteq R_{\mathsf{pre}}$ (obviously, we still should have safeness: $R_{\mathsf{pre}'} \supseteq R_c$) and start the algorithm again, but now with the (previously) stable model $\mathcal{A}$ as initial abstraction.

If $\mathcal{C} \not\models \varphi$ then it seems that in the end we still need to choose pre to be precise in order to draw that conclusion. However, it turns out that stable abstractions satisfy one more property.

> Let $\mathcal{D}$ be a data abstraction of $\mathcal{C}$ and let $\mathcal{A}$ be a stable abstraction of $\mathcal{D}$ as obtained by partitioning. An edge $(A, A') \in R_a$ in $\mathcal{A}$ is called *exact* if it satisfies
>
> $$\forall a \in \gamma_{\mathcal{D}}(A)\ \ \forall c \in \gamma^d(a)\ \ \exists a' \in \gamma_{\mathcal{D}}(A')\ \exists d \in \gamma^d(a')\ \ c\ R_c\ d\ .$$
>
> A path $\mathbf{A} = A_0 A_1 \cdots$ is *exact* if every edge (A_i, A_{i+1}) on $\mathbf{A}$ is. Then $C \not\models \varphi$ provided there is an exact path $\mathbf{A}$ in $\mathcal{A}$ which is a counterexample for φ and such that the propositions appearing in φ are determined in the states on $\mathbf{A}$.

The exactness condition is illustrated in the picture to the right. Note that for each a and c a corresponding a' and d must be found.



As the algorithm performs its first model check only after the abstract model has been split w.r.t. the propositions, the determinacy constraint is automatically satisfied. Hence, if we can detect whether or not a transition is exact when computing R'_a during a split$(\mathcal{A}, s, \mathsf{pre})$ operation, then we may still be able to conclude that $\mathcal{C} \not\models \varphi$ without splitting w.r.t. the precise pre.

This leads to the algorithm in Fig. 2. In the exit condition we have written $\mathcal{A}^-$ for the abstraction in which only the marked (exact) edges have been retained. As $\neg\varphi$ is an ECTL-formula, $\mathcal{A}^- \not\models \varphi$ can only hold if there is a path $\mathbf{A}$ in $\mathcal{A}^-$ that witnesses $\neg\varphi$ and, hence, is a counterexample to $\mathcal{A} \models \varphi$. By construction of $\mathcal{A}^-$, $\mathbf{A}$ only contains marked edges and hence is exact.

Choose initial data abstraction and a corresponding safe pre
do
    Execute the partitioning algorithm (while marking exact edges)
exit if $(\mathcal{A} \models \varphi) \vee (\mathcal{A}^- \not\models \varphi)$
    Choose a new data abstraction and corresponding safe pre
    Unmark the edges of $\mathcal{A}$
od

Fig. 2. Partitioning algorithm for $\varphi \in$ ACTL with data abstraction

Remaining questions are how to mark edges and how to choose pre's. Especially to the latter question there is obviously no univocal answer. We discuss one possibility.

4.1 Depth-k Abstraction

The precondition function pre_c of $\mathcal{C}$ is determined by its transition relation: $pre_c(D) = \{c \mid \exists d \in D \;\; c \; R_c \; d\}$. So, the choice of abstraction is determined by the aim to lessen the complexity of calculating R_c, and the requirements that it supports the detection of exact edges and that it should be possible to change pre by 'adapting' the current one rather than by recomputing it afresh.

Now, (concrete) transition systems arise as the interpretation of programs. Programs can be abstractly viewed as defining *state-transformers* $t(\mathbf{x}, \mathbf{x}')$ where $\mathbf{x}$ is a vector of program variables and t specifies the relation between the new values $\mathbf{x}'$ of these variables and the old ones that a program step enforces. A program's state is thus a tuple of values in some domain Val (an n-tuple if $\mathbf{x}$ contains n variables). The transition relation associated with it is then $R_t = \{(\mathbf{v}, \mathbf{v}') \mid \mathbf{v}, \mathbf{v}' \in Val^n, \; t(\mathbf{v}, \mathbf{v}')\}$. For ease of exposition we take $Val \subseteq \mathbf{N}$. Obviously, if the program is finite state, i.e., if R_t is finite, then Val is included in some initial segment of $\mathbf{N}$.

Abstraction The abstractions that we propose to use are based on restricting the precision with which variable values are recorded. The idea behind this is that the ROBDDs used to represent sets of states will have a limited depth. To support edge marking we need to know whether a variable's abstract value is precise or not. This leads to abstract value domains $Val_a^k = \{r, (r, \mathsf{O}) \mid r < 2^k, r \in Val\} \cup \{\mathsf{T}\}$ for $k > 0$ where O indicates an 'overflow' in the sense that the value is too large to be represented with k bits; $Val_a^0 = \{\mathsf{T}\}$. Abstract values of the form r represent concrete values smaller than 2^k precisely; a value (r, O) means that there is overflow but that the least k bits are correct and have value r; and T indicates absence of any knowledge. The concretization and abstraction function (i.e., the γ^d and $(\gamma^d)^\flat$ from before) for $k > 0$ are

$$\gamma_C^k(a) = \begin{cases} \{a\}, & \text{if } a < 2^k \\ \left\{ n \;\middle|\; \begin{array}{l} n \geq 2^k, \\ n \bmod 2^k = r \end{array} \right\}, & \text{if } a = (r, \mathsf{O}) \\ \mathbf{N}, & \text{if } a = \mathsf{T} \end{cases}, \quad \alpha_C^k(N) = \begin{cases} r, & \text{if } N = \{r\} \;\&\; r < 2^k \\ (r, \mathsf{O}), & \text{if } N \bmod 2^k = \{r\} \\ & \quad \& \neg(N < 2^k) \\ \mathsf{T}, & \text{otherwise} \end{cases}$$

In fact, α_C^k as just defined equals $(\gamma_C^k)^\flat$.

Partitioning algorithm Our implementation of the algorithm in Fig. 2 will use these depth-k abstractions: each variable value is represented in Val_a^k for some k. It uses the precise abstract interpretation t^k of t on Val_a^k: $t^k(a,b)$ iff $\exists c,d \in V_c \;\; c \in \gamma^k(a)$ & $t(c,d)$ & $\alpha^k(d) = b$. To be precise, given the concrete system $\mathcal{C} = (V_c, I_c, R_c)$ (with $V_c \subseteq \mathbf{N}^n$), define $\mathcal{A}^k = (V_a^k, I_a^k, R_a^k)$ by $V_a^k = (Val_a^k)^n$, $I_a^k = \{\alpha^k(c) \mid c \in I_c\}$ and $R_a^k = \{(\mathbf{a}, \mathbf{a}') \mid t^k(\mathbf{a}, \mathbf{a}')\}$. The algorithm constructs models stable w.r.t. $\mathcal{A}^k$ for ever larger values of k; whence, $\mathsf{pre}^k = \gamma^b_{\mathcal{A}^k} \circ pre_{R_a^k} \circ \gamma_{\mathcal{A}^k}$ is the precondition function used when splitting w.r.t. $\mathcal{A}^k$.

Correctness It works, because not only are the $\mathcal{A}^k$ abstractions of $\mathcal{C}$, but they also form a hierarchy in the sense that $\mathcal{A}^k$ is an abstraction of $\mathcal{A}^l$ if $k \leq l$. This follows from Val_a^k being an abstract interpretation of Val_a^l for $k \leq l$ via the concretization function γ_l^k: $Val_a^k \to 2^{Val_a^l}$ defined as

$$\gamma_l^k(a) = \begin{cases} \{a\}, & \text{if } a < 2^k \\ \{n \mid 2^k \leq n < 2^l,\ n \bmod 2^k = r\} \cup & \\ \quad \{(n, \mathsf{O}) \mid 2^k \leq n < 2^l,\ n \bmod 2^k = r\}, & \text{if } a = (r, \mathsf{O}) \\ \mathsf{T}, & a = \mathsf{T} \end{cases}$$

We also have $\gamma^k = \gamma^l \circ \gamma_l^k$. Hence, an abstraction of $\mathcal{A}^k$ can be transformed into a transition system over V_a^l by replacing every V_a^k state a by the V_a^l states in $\gamma_l^k(a)$. Thus we obtain the initalization for the next iteration of the partitioning algorithm.

As for termination, consider $\mathcal{A}^K$ where K is chosen such that concrete values are at most 2^K. (If $\mathcal{C}$ is finite state then such a K obviously exists.) We have $I_a^K = I_c$, i.e., the abstractions are precise, and R_a^K restricted to the precise abstract values coincides with R_c. Hence, the parts of $\mathcal{C}$ and $\mathcal{A}^K$ that are reachable from the initial nodes are isomorphic. Because furthermore the stable abstraction computed by the partitioning algorithm w.r.t. pre^K is strongly preserving w.r.t. $\mathcal{A}^K$, we have $\mathcal{A} \models \varphi$ iff $\mathcal{C} \models \varphi$.

This seems to prove termination of the algorithm but there is a catch: If $\mathcal{C} \not\models \varphi$, then the algorithm terminates not if $\mathcal{A} \models \varphi$ but if $\mathcal{A}^- \not\models \varphi$ (where $\mathcal{A}$ is the stable abstraction of $\mathcal{A}^K$). In other words, the algorithm terminates if marking satisfies the following property:

> If $\mathcal{A}$ is a stable abstraction of $\mathcal{A}^K$ as computed by the partitioning algorithm, and if the concrete values are at most 2^K, then every edge in the reachable part of $\mathcal{A}$ is marked. (5)

Marking states The stable model that the algorithm attempts to construct is in its turn an abstraction of $\mathcal{A}^k$. Hence, the abstract states A will in fact be subsets of (formally: predicates over) $(Val_a^k)^n$. It is edges between these subsets that are possibly going to be marked. Given two abstract states A and B, there is an edge between them if $A \cap \mathsf{pre}^k(B) \neq$ false. From this we can only conclude that $\exists a \in \gamma_{\mathcal{A}^k}(A)\ \exists b \in \gamma_{\mathcal{A}^k}(B)\ \ a\ R_a^k\ b$, while $a\ R_a^k\ b$ gives that $\exists c \in \gamma^k(a)\ \exists d \in \gamma^k(b)\ \ c\ R_c\ d$.

If $A \subseteq \mathsf{pre}^k(B)$ then we know that

$$\forall a \in \gamma_{\mathcal{A}^k}(A)\ \exists c \in \gamma^k(a)\ \exists b \in \gamma_{\mathcal{A}^k}(B)\ \exists d \in \gamma^k(b)\ \ c\ R_c\ d\ .$$

Comparing this with the definition of exactness of an edge, the relevant question is: 'when can we replace $\exists c \in \gamma^k(a)$ by $\forall c \in \gamma^k(a)$ in this formula?' One obvious answer is when $|\gamma^k(a)| = 1$, i.e., when the concrete values of the variable are represented precisely. Another case in which we can replace it, is when we have a state transformer such as $x' = 0$. If A 'is' the predicate $x = (1, \mathsf{O}) \wedge y = 2$ and B equals $x = 0 \wedge y = 2$, then the edge from A to B is exact although A contains abstract overflow values. Call a variable x a *don't care* for B just in case $\mathsf{pre}^k(B)$ is independent of x (interpreted as a predicate). Then, the criterion for marking an edge (A, B) becomes

> $A \subseteq \mathsf{pre}^k(B)$ and for any variable x, either x is represented precisely in every $a \in \gamma_{\mathcal{A}^k}(A)$ or x is a don't care for $\mathsf{pre}^k(B)$.

It is straightforward to show that under this marking condition every reachable edge will be marked if the depth-k abstraction gives enough precision as expressed in Property (5).

Note that marking as defined here is a 'safe' approximation of exactness. I.e., a marked edge is guaranteed to be exact but not necessarily vice versa. Clearly, marking every exact edge can only be done if the concrete transition relation is known.

Computing abstract relations As for ease of computation, obviously, the fewer bits we use to represent the values, the more efficient computing the abstraction becomes. Also, note that if $t^k(\mathbf{a}, \mathbf{a}')$ holds for 'precise' abstract values, i.e., for values *not* of the form (r, O) or T, then $t^l(\mathbf{a}, \mathbf{a}')$ holds as well for any $l \geq k$. Moreover, for operations like addition or multiplication even if the result of the operation must be represented as (r, O) in Val_a^k, the lower bits r stay the same if the operation is interpreted in a more precise Val_a^l. Hence, it is possible to 'extend' pre^k to pre^l rather than to recompute it.

5 Conclusions and Future Work

We have presented an ACTL state partitioning algorithm that, for a given formula φ, computes the 'coarsest' abstraction that allows the truth of φ to be determined. This algorithm has been extended with a second orthogonal abstraction scheme that allows the automated choice of data abstraction relative to which the states are partitioned.

The algorithm described above is being implemented as part of a VHDL/S verifier developed in ESPRIT project 6128 FORMAT. After the total realization of the algorithm, practical experience has to show how far the existing limits of automatic verification have been pushed ahead. We should also investigate how already computed information can be reused more extensively. We are thinking not only of incrementally extending the ROBDDs that describe the effect of the VHDL/S code fragments when the algorithm changes the depth-k abstraction, but also of integrating model checking the specification φ with the partitioning process: each partitioning step causes an additional subformula of φ to be determined; the model checker needs to do the same. Finally, although the depth-k abstraction seems to be a good candidate for step-wise refinement of abstraction schemes, there may be good alternatives as well. The work done on the field of abstract interpretation of programs for static analysis can help for future applications in the area of automatic verification.

Acknowledgments The first two authors wish to thank OFFIS and the University of Oldenburg for their hospitality. We also thank Werner Damm for the intensive 'bull session' he organized and participated in, during which the ideas on which this paper is based were developed.

References

[BCM+90] J. R. Burch, E. M. Clarke, K. L. Mcmillan, D. L. Dill, and J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Proceedings of the Fifth Anual IEEE Symposium on Logic in Computer Science (LICS)*, 1990.

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

[Bry86] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *Transactions on Computers*, C-35:677–691, 1986.

[Bry92] R. E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.

[CC77] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by constructing or approximation of fixed points. In *Proceedings of the Fourth Annual ACM Symposium on Principles of Programming Languages (POPL)*, pages 238–252. ACM, 1977.

[DGG93] D. Dams, R. Gerth, and O. Grumberg. Generation of reduced models for checking fragments of CTL. In C. Courcoubetis, editor, *Proceedings of the Fifth Conference on Computer-Aided Verification*, volume 697 of *Lecture Notes in Computer Science*. Springer-Verlag, 1993.

[DGG94] D. Dams, O. Grumberg, and R. Gerth. Abstract interpretation of reactive systems: abstractions preserving $\forall$CTL*, $\exists$CTL* and CTL*. In *Proceedings of PROCOMET*, IFIP. North-Holland, 1994. To appear.

[GKPP94] R. Gerth, R. Kuiper, D. Peled, and W. Penczek. A partial order approach to branching time logic model checking, 1994. Submitted.

[GW91] P. Godefroid and P. Wolper. A partial approach to model checking. In *Proceedings of the Sixth Anual IEEE Symposium on Logic in Computer Science (LICS)*, 1991.

[Har87] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8, 1987.

[HK94] J. Helbig and P. Kelb. An OBDD representation of statecharts, 1994. To appear in EDAC94.

[HSD+93] J. Helbig, R. Schlör, W. Damm, G. Döhmen, and P. Kelb. VHDL/S—integrating statecharts, timing diagrams and VHDL. *Microprocessing and Microprogramming*, 38:571–580, 1993.

[Pel93] D. Peled. All from one, one for all, on model-checking using representatives. In *Proceedings of the Fifth International Conference on Computer-Aided Verification*, Lecture Notes in Computer Science, pages 409–423. Springer-Verlag, 1993.

[SD93] R. Schlör and W. Damm. Specification and verification of system-level hardware designs using timing diagrams. In *EDAC93*, 1993.

[Val91] A. Valmari. A stubborn attack on state explosion. In *Proceedings of the Second Conference on Computer-Aided Verification*, Lecture Notes in Computer Science. Springer-Verlag, 1991.