

Partial Model Checking (Extended Abstract)

Henrik Reif Andersen*

Department of Computer Science
Technical University of Denmark
Building 344, DK-2800 Lyngby, Denmark.

Abstract

A major obstacle in applying finite-state model checking to the verification of large systems is the combinatorial explosion of the state space arising when many loosely coupled parallel processes are considered. The problem also known as the state-explosion problem has been attacked from various sides. This paper presents a new approach based on partial model checking: Parts of the concurrent system are gradually removed while transforming the specification accordingly. When the intermediate specifications constructed in this manner can be kept small, the state-explosion problem is avoided. Experimental results with a prototype implemented in Standard ML, shows that for Milner’s Scheduler — an often used benchmark — this approach improves on the published results on Binary Decision Diagrams and is comparable to results obtained using generalized Decision Diagrams.

Specifications are expressed in a variant of the modal μ -calculus.

1 Introduction

Within the last decade temporal logic model checking has turned out to be a useful technique for verifying finite state systems. Since the paper by Clarke and Emerson [13] introduced the idea for the logic CTL*, algorithms performing exhaustive state-space exploration have been found and improved for a variety of logics. However, a major problem in applying model checking on just moderate-sized systems is the combinatorial explosion arising from the many possible combinations of independent states of variables

or processes. Attempts to solve this problem have focussed on algorithms that avoid traversing the complete state space. Either by “symbolic” representations of the state space using Binary Decision Diagrams [10], by collapsing symmetric or otherwise similar states [33, 24, 11, 21] or by abstraction [19, 12].

The most prominent successes on rather large systems have been reported from groups using Binary Decision Diagrams – a heuristic based on compact representations and manipulations of Boolean expressions. We present in this paper a new heuristic that is based on a quotienting operator which allows one process of a parallel composition to be removed by transforming the specification accordingly. The method will be applicable to satisfaction problems of the form

$$(t_1 \mid t_2 \mid \dots \mid t_N) \upharpoonright L \models A, \quad (1)$$

where t_i ($1 \leq i \leq N$) are labelled transition systems modeling processes and L is a restricting set of labels enforcing synchronization between the processes. The specification A is expressed in a variant of the modal μ -calculus.

We have chosen the name *partial model checking* in analogy with *partial evaluation of programs* to reflect that from an initial specification we supply part of the concurrent system and derive a new residual specification specialized to that particular class of systems. In other words: each intermediate specification provides a partial answer to the original satisfaction problem.

2 Logic and Models

Our logic is a version of Kozen’s modal μ -calculus [25]. The main difference will be the use of simultaneous fixed points expressed as extreme solutions to sets of equations. In this respect it follows quite closely Park’s original μ -calculus [29] and the recent trend

*Supported by the Danish Technical Research Council.
E-mail: hra@id.dtu.dk. WWW: <http://www.id.dtu.dk/~hra>.

on model checking (see for example [7], [16], [34], [2], [26]). *Assertions* A are given by the following syntax:

$$A ::= F \mid T \mid X \mid A \vee A \mid A \wedge A \mid \langle \alpha \rangle A \mid [\alpha]A,$$

where X ranges over a set of variables and α over a set of actions Act . From assertions we build sequences of *assertion equations* E , denoting by ϵ the empty sequence of equations:

$$E ::= X =_{\mu} A \ E \mid X =_{\nu} A \ E \mid \epsilon.$$

The first equation is a *minimum fixed-point equation*, the second a *maximum fixed-point equation*. We shall often use σ to range over $\{\mu, \nu\}$ writing for instance $X =_{\sigma} A$. Finally, a *top assertion* is an expression

$$E \downarrow X$$

that projects the simultaneous solution of E onto X . To be well-defined X must be among the left-hand sides of E . To simplify future discussions on assertion equations we assume that all left-hand side variables in E are different. Furthermore, we shall say that E is *simple* if the right-hand sides are of the forms:

$$\begin{aligned} & X, F, T, \\ & \bigwedge \{X_1, \dots, X_k, [a_1]Y_1, \dots, [a_i]Y_i\}, \\ & \bigvee \{X_1, \dots, X_k, \langle a_1 \rangle Y_1, \dots, \langle a_i \rangle Y_i\}, \end{aligned}$$

where we have used $\bigwedge \{A_1, \dots, A_m\}$ to denote $A_1 \wedge A_2 \wedge \dots \wedge A_m$ using the fact that the order is immaterial since \wedge will be associative and commutative.¹

Given a (*pointed*) *labelled transition system* $t = (S, \rightarrow, i)$ with states S , transition relation $\rightarrow \subseteq S \times Act \times S$ and initial state $i \in S$, any assertion A will denote a subset $\llbracket A \rrbracket_i \rho$ of the states of S said to satisfy it. (We often leave out the subscript t when it is implied by the context.) Here, ρ is an environment assigning meanings to free variables. For assertions: $\llbracket F \rrbracket \rho = \emptyset$, $\llbracket T \rrbracket \rho = S$, $\llbracket X \rrbracket \rho = \rho(X)$, \vee is interpreted as set union, \wedge as intersection and the modalities as follows:

$$\begin{aligned} \llbracket \langle \alpha \rangle A \rrbracket \rho &= \{s \in S \mid \exists s'. s \xrightarrow{\alpha} s' \ \& \ s' \in \llbracket A \rrbracket \rho\} \\ \llbracket [\alpha]A \rrbracket \rho &= \{s \in S \mid \forall s'. s \xrightarrow{\alpha} s' \Rightarrow s' \in \llbracket A \rrbracket \rho\} \end{aligned}$$

The unique solution to a set of assertion equations E will be an environment assigning sets of states to each variable. We will make use of the following notation

¹Notice that our notion of a *simple assertion* differs slightly from that introduced in [7] and used in [2], since we allow Boolean operators being applied to modalities.

for environments: \square is the empty environment, $[U/X]$ is the environment that assigns U to X , and for two environments ρ and ρ' with disjoint variable domains, $\rho \star \rho'$ is the union of them. Hence, for example $\square \star [U/X] = [U/X] = [U/X] \star \square$ and $[U/X] \star [V/Y] = [V/Y] \star [U/X]$ which we shall also write as $[V/Y, U/X]$.

For a monotonic function $f(U)$ on the powerset of S we let $\mu U.f(U)$ and $\nu U.f(U)$ denote respectively the minimum and maximum fixed point as given by Tarski's fixed-point theorem [31]. We can now define $\llbracket E \rrbracket \rho$, the solution to E in the environment ρ , by induction on the number of equations:

$$\begin{aligned} \llbracket \epsilon \rrbracket \rho &= \square \\ \llbracket X =_{\sigma} A \ E \rrbracket \rho &= [U'/X] \star (\llbracket E \rrbracket (\rho \star [U'/X])) \\ &\text{where } U' = \sigma U. \llbracket A \rrbracket (\rho \star [U/X] \star \rho'(U)) \\ &\text{and } \rho'(U) = \llbracket E \rrbracket (\rho \star [U/X]) \end{aligned}$$

The first equation defines the solution to the empty set of equations to be the empty environment. The second defines the meaning of $(X =_{\sigma} A \ E)$ in terms of the shorter E . More precisely, we first find the solution for X (called U') and using this value of X , inductively solves E . The solution U' is the extremal σ -fixed point of the function of U , determined by the meaning of A in an environment where the meaning of the free variables of the equation system is given by ρ , the meaning of X is U and the remaining bound variables is given by the inductive solution to E . To be well-defined the function of U must be monotonic, which follows rather directly from the observation that each operator (including μ and ν) are monotonic.

For top assertions which contain no free variables (*free* meaning with no defining equation) we define: $\llbracket E \downarrow X \rrbracket = (\llbracket E \rrbracket \square)(X)$.² With the given semantics it can be shown that the expressive power of our logic is equivalent to that of the modal μ -calculus ([1], [15]).³

A labelled transition system $t = (S, \rightarrow, i)$ is said to *satisfy* a top assertion $E \downarrow X$, if $i \in \llbracket E \downarrow X \rrbracket$. We write $t \models E \downarrow X$ in this case.

To construct models of concurrent systems we shall use three operations on labelled transition systems. To make sense of them we need a little structure on the actions. We assume the presence of a *silent* action

²There are dangerous subtleties here due to the nesting of minimum and maximum fixed points. For instance, the given semantics is different from one where the equation for X is taken as the "starting point" of the recursively defined semantics for $E \downarrow X$.

³The notion of alternation depth for the modal μ -calculus (originally defined in [22]) also has a correlate in our logic relating to the number of alternating sequences of μ - and ν -equations. See for example [1] or [15] for discussions of this point.

$(E \downarrow X) //_{L, M} t$	$=$	$(E //_{L, M} t) \downarrow X_i$
$\epsilon //_{L, M} t$	$=$	ϵ
$(X =_{\sigma} A E) //_{L, M} t$	$=$	$\begin{cases} X_{s_1} =_{\sigma} A //_{L, M} s_1 \\ \dots \\ X_{s_n} =_{\sigma} A //_{L, M} s_n \\ E //_{L, M} t \end{cases}$
$X //_{L, M} s$	$=$	X_s
$\langle \alpha \rangle A //_{L, M} s$	$=$	$\langle \alpha \rangle (A //_{L, M} s) \vee \bigvee_{s \xrightarrow{\alpha} s'} A //_{L, M} s' \quad \text{if } \alpha \neq \tau$
$\langle \tau \rangle A //_{L, M} s$	$=$	$\langle \tau \rangle (A //_{L, M} s) \vee \bigvee_{s \xrightarrow{\tau} s'} A //_{L, M} s' \quad \vee \quad \bigvee_{\beta \in \bar{L} \cap M, s \xrightarrow{\beta} s'} \langle \bar{\beta} \rangle (A //_{L, M} s')$
$(A_1 \vee A_2) //_{L, M} s$	$=$	$(A_1 //_{L, M} s) \vee (A_2 //_{L, M} s)$
$F //_{L, M} s$	$=$	F

Table 1: The quotient $E \downarrow X //_{L, M} t$ on a transition system $t = (\{s_1, \dots, s_n\}, \rightarrow, i)$. We have left out the cases for $\langle \alpha \rangle A, \wedge$ and T . They are immediate duals of $\langle \alpha \rangle A, \vee$ and F .

τ modelling synchronization, and for each non-silent action a a complementary action \bar{a} such that $\bar{\cdot}$ forms a bijection on $Act \setminus \tau$. We will need a *parallel composition* $t_1 \mid t_2$ that allows t_1 and t_2 to synchronize on complementary actions, a *restriction operator* $t \upharpoonright L$ that eliminates from t all non-silent actions not in L , and a *hiding operator* t/M that *hides* all actions in M by renaming them to τ . (All three are taken from CCS although Milner [28] uses a slightly different version of restriction — we do not close up L under $\bar{\cdot}$.)

Formally, $(S_1, \rightarrow_1, i_1) \mid (S_2, \rightarrow_2, i_2)$ is the transition system $(S_1 \times S_2, \rightarrow, (i_1, i_2))$ where

$$\begin{aligned} (s_1, s_2) \xrightarrow{a} (s'_1, s'_2) &\Leftrightarrow_{\text{def}} \\ &s_1 = s'_1 \text{ and } s_2 \xrightarrow{a}_2 s'_2, \\ \text{or } s_1 \xrightarrow{a}_1 s'_1 \text{ and } s_2 = s'_2, \\ \text{or } a = \tau \text{ and } (\exists b. s_1 \xrightarrow{b}_1 s'_1 \text{ and } s_2 \xrightarrow{\bar{b}}_2 s'_2). \end{aligned}$$

Secondly, the restriction $(S, \rightarrow, i) \upharpoonright L$ is the transition system (S, \rightarrow_0, i) where $s \xrightarrow{a}_0 s' \Leftrightarrow_{\text{def}} s \xrightarrow{a} s'$ and $a \in L \cup \{\tau\}$. Thirdly, $(S, \rightarrow, i)/M$ is the transition system (S, \rightarrow_0, i) where

$$\begin{aligned} s \xrightarrow{a}_0 s' &\Leftrightarrow_{\text{def}} \\ &s \xrightarrow{a} s' \text{ and } a \notin M, \\ \text{or } a = \tau \text{ and } \exists b \in M. s \xrightarrow{b} s'. \end{aligned}$$

From the first two operators we define a generalized parallel operator $\parallel_{L, M}$ that restricts its left and right arguments to perform actions in L respectively M :

$$t_1 \parallel_{L, M} t_2 =_{\text{def}} (t_1 \upharpoonright L) \mid (t_2 \upharpoonright M)$$

Let the *sort* of a transition system t be the set of actions appearing in the transition relation. Then, if the sort $L(t_1)$ of t_1 equals L and the sort $L(t_2)$ of t_2 equals M then $t_1 \parallel_{L, M} t_2 = t_1 \mid t_2$, so we can reconstruct the original operators as follows:

$$\begin{aligned} t_1 \mid t_2 &= t_1 \parallel_{L(t_1), L(t_2)} t_2 \\ t \upharpoonright L &= t \parallel_{L, \emptyset} \mathbf{0} \end{aligned} \quad (2)$$

where $\mathbf{0}$ is the transition system with one state and no transitions. The generalized product is used since it makes explicit the actions allowed by the argument processes. This simplifies the definition of quotienting of modalities in the next section.

3 Quotienting

Given a top assertion $E \downarrow X$, we will describe how to find a top assertion $(E \downarrow X) //_{L, M} t$ such that

$$t' \parallel_{L, M} t \models E \downarrow X,$$

if and only if,

$$t' \models (E \downarrow X) //_{L, M} t.$$

This bi-implication justifies the claim that we are moving parts of the concurrent system into the specification. Clearly, if the new top assertion is not much larger than the original, we have simplified the task of model checking.

The top assertion $(E \downarrow X) //_{L, M} t$ is defined in terms of two auxiliary operators: $E //_{L, M} t$ that computes a

Simple Evaluation		
SE1	$X =_{\sigma} \bigwedge \{F, A_1, \dots, A_k\}$	$\Rightarrow X =_{\sigma} F$
SE2	$X =_{\sigma} \bigwedge \{T, A_1, \dots, A_k\}$	$\Rightarrow X =_{\sigma} \bigwedge \{A_1, \dots, A_k\}$
SE3	$X =_{\sigma} \langle \alpha \rangle F$	$\Rightarrow X =_{\sigma} F$
Constant Propagation		
	$X =_{\sigma} A$	$X =_{\sigma} A[F/Y]$
CP	\vdots	\vdots
	$Y =_{\sigma'} F$	$Y =_{\sigma'} F$
Unguardedness Removal		
	$X =_{\sigma} B$	$X =_{\sigma} B[A/Y]$
UR	\vdots	\vdots
	$Y =_{\sigma} A$	$Y =_{\sigma} A$
		(Y unguarded in B)
Trivial Equation Elimination		
TEEμ	$X =_{\mu} \langle \alpha \rangle X$	$\Rightarrow X =_{\mu} F$
TEEν	$X =_{\nu} [\alpha] X$	$\Rightarrow X =_{\nu} T$
Equivalence Reduction		
ERμ	$X =_{\mu} A$ $Y =_{\mu} B$	$\Rightarrow X =_{\mu} A \vee B$ $Y =_{\mu} X$
		($X =_E Y$ and X, Y in same block)
ERν	$X =_{\nu} A$ $Y =_{\nu} B$	$\Rightarrow X =_{\nu} A \wedge B$ $Y =_{\nu} X$
		($X =_E Y$ and X, Y in same block)

Table 2: Valid transformations on assertion equations.

new set of assertion equations, and $A//_{L,M} s$ that computes a new assertion. The first is defined by induction on the number of equations in E , the second by structural induction – see Table 1. (It is an immediate application of work on compositional reasoning [27, 6, 1, 4].)

When E is simple, the size of $E//_{L,M} t$ is easily seen to be bounded by $|E|(|\rightarrow| + |S|)$. Here $|E|$ is the number of operators, variables and constants occurring in E , $|\rightarrow|$ is the number of transitions in \rightarrow and $|S|$ the number of states. Hence, by repeated application of quotienting on systems of the form (1) with transition systems containing n_i states and m_i transitions, we get a resulting assertion of size no larger than $|A| \prod_{i=1}^N (n_i + m_i)$. Of course, as this expression shows there is a risk of a combinatorial explosion. The means to avoid this and make repeated quotienting successful is to *simplify* the quotients.

4 Simplifying Assertions

Deciding validity of μ -calculus formulae is known to be EXPTIME-complete [25, 20]. Hence, for any reasonable notion of size making the constants T and F smallest, the problem of finding a minimal assertion is EXPTIME-hard. To achieve acceptable performance we must therefore rely on heuristics. This takes the form of a collection of few, efficiently implementable strategies for finding smaller but equivalent assertions. The success of applying partial model checking is completely determined by the success of these strategies: The strategies are generally valid but might or might not succeed to decrease the size of the assertion.

Below follows a list of strategies we have found to be useful. They are described as transformations on sequences of assertion equations. They all work on simple sequences of assertion equations inside a top assertion $E \downarrow X$. We later comment on their effect in the benchmark. Table 2 summarizes the transformations.

Simple Evaluation (SE i , $i = 1, 2, 3$).

The simple transformations SE i and three duals (got by interchanging \wedge with \vee , T with F , $\langle a \rangle$ with $[a]$) are applied everywhere.

Reachability Analysis.

Variables that are unreachable from the *root variable* (X in $E \downarrow X$) are removed. This is implemented as a simple graph algorithm.

Constant Propagation (CP).

Variables with right-hand side equal to T or F are replaced by their definition throughout the assertion. All variables that during this process gets a constant right-hand side (after a Simple Evaluation) are propagated in the same manner. This is also implemented as a simple graph algorithm.

Unguardedness Removal (UR).

To facilitate further reduction (e.g. by Equivalence Reduction below) and possible elimination of redundant variables (by Reachability Analysis), we perform an unfolding on unguarded variables. A variable X is *unguarded* in A if it does not appear below a modality. However, due to the occasional presence of cycles of unguarded variables we must take care not to perform infinite unfoldings.⁴ Therefore, we form a graph of unguarded variables that appear in conjunctive forms ($\wedge\{X_1, \dots, X_k, [a_1]Y_1, \dots, [a_i]Y_i\}$) collect all variables that are interconnected in this manner and forms new right-hand sides. An example is:

$$\begin{aligned} X &=_{\nu} X \wedge Y \wedge A \\ Y &=_{\nu} Y \wedge X \wedge B \end{aligned}$$

that will be changed to

$$\begin{aligned} X &=_{\nu} X \wedge A \wedge B \\ Y &=_{\nu} X \end{aligned}$$

This is followed by a substitution that replaces all occurrences of Y in the right-hand sides by X throughout all equations. A similar transformation takes place for disjunctive forms. It is implemented as an application of Tarjan's algorithm for finding strongly connected components [30].

Trivial Equation Elimination (TEE σ).

Equations $X =_{\mu} \langle a \rangle X$ and $Y =_{\nu} [a]Y$ are easily

⁴Although the original assertion contains no such cycles they can get introduced by the quotienting.

seen to have solutions $X = F$ and $Y = T$. In this way we can turn right-hand sides into the constants F or T that can be propagated. This can be done together with Constant Propagation.

Equivalence Reduction (ER σ).

We collapse a set of assertion equations E to a smaller system with equivalent variables identified. When X and Y have the same solution in E (written $X =_E Y$, i.e. $\llbracket E \rrbracket_t(X) = \llbracket E \rrbracket_t(Y)$ for all t), we can use only one of the variables, modify its right-hand side slightly and replace all occurrences of the other by this variable. This makes an equation redundant and it can then be removed. We are approximating the EXPTIME-hard check $X =_E Y$ as explained in the next section. The new right-hand side is formed as the disjunction of the two original right-hand sides if the equations are of type μ and as a conjunction if of type ν . However, when X and Y are judged equal as described below it turns out that these disjunctions and conjunctions will be simple to find: They will be equivalent to any one of the original right-hand sides.

Implicational Reduction.

Another simplification could be to take implications among variables into account. Of course, determining $X \subseteq_E Y$, i.e. $\llbracket E \rrbracket_t(X) \subseteq \llbracket E \rrbracket_t(Y)$ for all t , would also have to be done via safe approximations like for Equivalence Reduction.

5 Equivalence Reduction

A central, non-trivial task is to find and identify equivalent right-hand sides of sequences of assertion equations E . We shall look at an approach that operates on blocks. A *block* is a sequence of assertion equations of the same kind – either μ or ν . A block E can be viewed as a simultaneous fixed point $\vec{X} =_{\sigma} \vec{A}$ in the sense that $\llbracket E \rrbracket \rho$ is equivalent to the environment of solutions found as a *simultaneous* fixed point over a product of powersets ordered pointwise: $[\sigma \vec{U}. \llbracket \vec{A} \rrbracket \rho[\vec{U}/\vec{X}]/\vec{X}]$. This equivalence follows from a straightforward generalization of the Scott-Bekic principle, also known as Bekic's theorem [8]. (The generalized theorem can be found in [1].) In the sequel we assume E and ρ fixed and let $g(\vec{U})$ denote the function that maps \vec{U} to $\llbracket \vec{A} \rrbracket \rho[\vec{U}/\vec{X}]$ where \vec{A} is the tuple of right-hand sides of E . Hence for E a σ -block, $[\sigma \vec{U}. g(\vec{U})/\vec{X}] = \llbracket E \rrbracket \rho$.

As already mentioned the check $X =_E Y$ is EXPTIME-hard to decide. Therefore we approximate

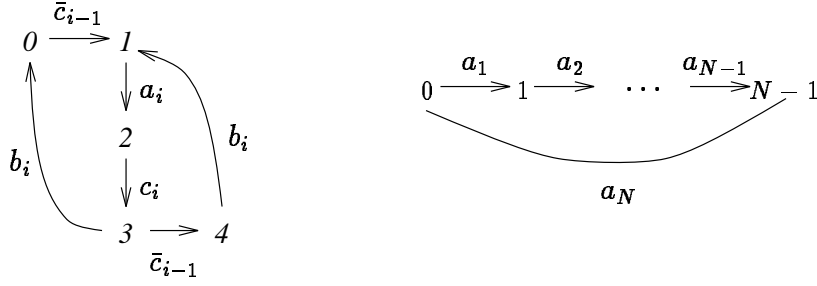


Figure 1: Left: Cycler i ($1 \leq i \leq N$). The initial state is 0 except for cycler 1 which has initial state 1. Right: The specification $spec_N$.

$=_E$ by a relation P that is *safe for E* , i.e. for all left-hand side variables X and Y ,

$$XPY \Rightarrow X =_E Y \quad (3)$$

or more compactly $P \subseteq =_E$. The relation P will be an equivalence relation (reflexive, symmetric and transitive) computed from a sequence of unsafe approximative equivalence relations. The sequence is constructed by repeatedly applying a transformation $G(P)$ on the current approximation P until a fixed point is reached. To state this idea more formally, we notice that any environment ρ induces an equivalence $=_\rho$ on variables defined by $X =_\rho Y \Leftrightarrow_{def} \rho(X) = \rho(Y)$. We shall say that P is safe for ρ if $P \subseteq =_\rho$. We will require the following property of G : For all P and ρ ,

$$\begin{aligned} &\text{if } P \text{ is safe for } \rho \\ &\text{then } G(P) \text{ is safe for } \rho' =_{def} [g(\rho(\vec{X})) / \vec{X}], \end{aligned} \quad (4)$$

where $\rho(\vec{X})$ is the tuple of sets of states $(\rho(X_1), \dots, \rho(X_n))$, and ρ' therefore assigns the variables to the result of applying g to these sets. Now, let \mathcal{U} be the universal relation that relates every variable to every other variable in E , and denote by $G^k(P)$ the k 'th composition of G applied to P . Inductively it is: $G^0(P) = P, G^{k+1}(P) = G(G^k(P))$.

Lemma 1 *If G satisfy (4) and $G^k(\mathcal{U}) = G^{k+1}(\mathcal{U})$ then $G^k(\mathcal{U})$ is safe for E .*

Our technique for judging variables equal is then simply to iterate G over \mathcal{U} until a fixed point P is found. Variables are then collapsed to the equivalence classes of P . The G that was used in the benchmark is defined as follows: $XG(P)Y$, if and only if, one of the following conditions hold:

- X and Y both have conjunctive right-hand sides, $X =_\sigma \bigwedge_{i \in I} A_i, Y =_\sigma \bigwedge_{j \in J} B_j$ satisfying:

$$\begin{aligned} &\forall i \in I. \\ &\quad (A_i \equiv X_i \Rightarrow \\ &\quad \quad \exists j \in J. B_j \equiv Y_j \ \& \ X_i P Y_j) \\ &\& (A_i \equiv [\alpha_i] X_i \Rightarrow \\ &\quad \quad \exists j \in J. B_j \equiv [\beta_j] Y_j \ \& \ \alpha_i = \beta_j \ \& \ X_i P Y_j) \end{aligned}$$

and

$$\begin{aligned} &\forall j \in J. \\ &\quad (B_j \equiv Y_j \Rightarrow \\ &\quad \quad \exists i \in I. A_i \equiv X_i \ \& \ Y_j P X_i) \\ &\& (B_j \equiv [\beta_j] X_j \Rightarrow \\ &\quad \quad \exists i \in I. A_i \equiv [\alpha_i] X_i \ \& \ \beta_j = \alpha_i \ \& \ Y_j P X_i) \end{aligned}$$

- X and Y both have disjunctive right-hand sides and satisfy a condition dual to the condition for conjunctive right-hand sides,
- X and Y both have the same constant right-hand side.

Such a G is enough to identify for example the variables X and Y in the ν -block:

$$\begin{aligned} X &=_\nu [a]X \wedge Z \\ Y &=_\nu [a]Y \wedge [a]X \wedge Z \end{aligned}$$

The sequence tends to converge rather quickly (within 5-6 iterations) in the benchmark, although the upper bound is the number of variables. Each iteration requires in the worst case computation of $XG(P)Y$ for each pair of X and Y , hence on the order of n^2 computations for a block with n variables. This is the dominant factor which, since the reduction is applied on the order of n times (one for each quotienting), results in an n^3 overall running time in the experiments below.

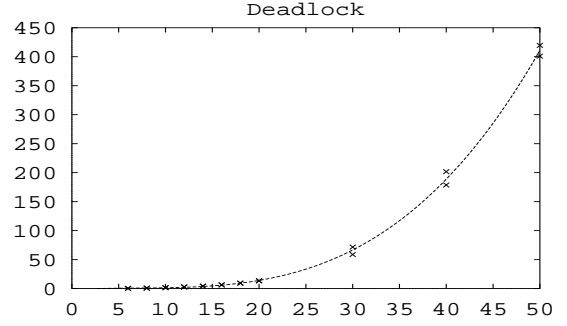
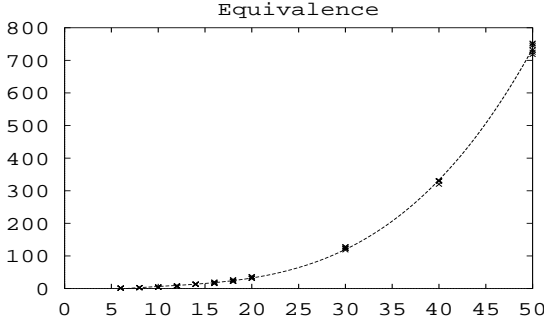


Figure 2: Running times in seconds for the equivalence check and check for absence of deadlock as a function of the number of cyclers N . The crosses are measured times (user and system time including time for garbage collections) the graphs are the third-degree polynomials $0.0106897N^3 - 0.333923N^2 + 5.13771N - 22.331$ respectively $0.00499743N^3 - 0.100896N^2 + 0.79611N - 1.47533$.

6 The Example

Milner's Scheduler [28] consists of N cyclers connected in a ring that co-operates on initiating (a_i) and terminating (b_i) N tasks such that the tasks are always started in sequence $a_1, \dots, a_N, a_1, \dots$ but can terminate in any order.

Figure 1 shows the transition systems for the cyclers and a transition system $spec_N$ expressing the intended behaviour when hiding the b -actions. The full system sys is:

$$(cyc_1 \mid cyc_2 \mid \dots \mid cyc_N) \uparrow \{a_1, \dots, a_N, b_1, \dots, b_N\}$$

The restriction forces the cyclers to synchronize on the actions c_1, \dots, c_N . We first verified that $sys/\{b_1, \dots, b_N\}$ in which the b -actions have been hidden, is weakly bisimilar to $spec_N$ [28]. By a simple algebraic law the hiding can be pushed inside the restriction and the parallel operator since no synchronization takes place on the b_i 's. Thus we verify:

$$(cyc'_1 \mid cyc'_2 \mid \dots \mid cyc'_N) \uparrow \{a_1, \dots, a_N\} \equiv E \downarrow R_1$$

where $cyc'_i = cyc_i/\{b_1, \dots, b_N\}$ and $E \downarrow R_1$ is a top assertion expressing the property of being weakly bisimilar to $spec_N$. The set of equations E is the following:

$$\begin{aligned} R_1 &=_{\nu} [a_1]R_2 \wedge [\tau]R_1 \wedge [-a_1, \tau]F \wedge X_1 \\ &\vdots \\ R_N &=_{\nu} [a_N]R_1 \wedge [\tau]R_N \wedge [-a_N, \tau]F \wedge X_N \\ X_1 &=_{\mu} \langle \tau \rangle X_1 \vee \langle a_1 \rangle Y_1 \\ Y_1 &=_{\mu} \langle \tau \rangle Y_1 \vee R_2 \\ &\vdots \\ X_N &=_{\mu} \langle \tau \rangle X_N \vee \langle a_N \rangle Y_N \\ Y_N &=_{\mu} \langle \tau \rangle Y_N \vee R_1 \end{aligned}$$

(Actually, this assertion was constructed directly from a fixed-point assertion expressing weak bisimilarity and then quotienting with $spec_N$ much like the quotient for the generalized parallel composition. See [3] or [1] for details.) We have used $\langle -a_1, a_2, \dots, a_k \rangle A$ to mean $\bigvee_{a \notin \{a_1, a_2, \dots, a_k\}} \langle a \rangle A$ - a co-finite disjunction over actions. Hence, $\langle - \rangle A$ is $\bigvee_{a \in Act} \langle a \rangle A$. The first quotienting replaces this by a proper finite disjunction of diamond modalities. The presence of the sorts L and M in the parallel operators makes this a simple task.

Secondly, we verified the absence of deadlocks using the top assertion $E' \downarrow X$ where E' is

$$\begin{aligned} X &=_{\nu} [-]X \wedge Y \\ Y &=_{\nu} \langle - \rangle T \end{aligned}$$

In both cases we encoded \mid and \uparrow using the generalized parallel operator (2). We then proceeded by alternating between quotienting out a cycler and simplifying the resulting assertion until none were left and the resulting top assertion turned out to be $(X =_{\nu} T) \downarrow X$.⁵

As Figure 2 clearly shows the running times grow roughly as third-degree polynomials and *not* exponentially like the number of states and transitions (which roughly grows as $20 \cdot 2.2^N$). In other words: We avoided the state explosion.

Table 3 compares the running times with other published results for Milner's Scheduler. The numbers should be taken with a pinch of salt, since they are taken from implementations in different languages and on different machines. The only interesting point is that our running times achieved with a prototype implemented in Standard ML of New Jersey on a

⁵In the final step of the quotienting, modalities are replaced by constants as follows: $\langle \alpha \rangle A$ by F , $[\alpha]A$ by T .

N :	6	8	10	12	14	16	18	20	30	40	50
Transitions:	2016	13824	84480	479232	2580480	13369344	67239936	330301440	748934922240	1352399302164480	2.15e18
Equivalence											
[23](BDD)	21	40	87	145	233	348	569	850			
[9](BDD)	19	39				198	256	333			
[17](DD)	4.8	8.0	12	18	22	29	36	45			
Partial MC	1.4	2.8	4.5	7.5	13	17	24	34	124	329	737
Deadlock											
[17](DD)	0.8	1.4	2.2	3.6	4.7	6.1	7.7	9.7			
Partial MC	0.3	0.7	1.4	2.8	4.2	6.4	9.4	13	65	190	410

Table 3: Results on Milner’s Scheduler. All times are in seconds. The results for partial model checking have been found as an average of several runs. Memory consumption did not exceed 16 MByte for any of the runs. The times for the DD package implemented in C [17] have been found by adding the time for finding the reachable states to the time for verifying equivalence/absence of deadlock in their table p. 188. It is unclear whether their figures include the time for computing the initial DD – which they should for proper comparison. The number of reachable transitions has been found with a BDD-package. (For $N = 50$ a more accurate number is $2.15328357183652e18$.)

DEC 5000 MIPS-station competes seriously with the BDD/DD-packages implemented in Prolog/C.

Most of the reductions mentioned are necessary to avoid the state explosion. Leaving out just one of Reachability Analysis, Constant Propagation or Equivalence Reduction results in assertions that grow exponentially. Only Implicational Reduction was unnecessary.

Another series of experiments were carried out in which the hidden b -actions were not pushed inside the transition systems but treated as a quotient, i.e. it was removed by transforming the assertion equations accordingly. For both the equivalence check and the deadlock check, the same cubic behaviour was observed – although the execution times were up to seven times higher for the equivalence check and up to two times higher for the deadlock check.

7 Conclusion

The idea of partial model checking is based on the hope that if the interfaces between processes are simple, then this will allow assertions expressing requirements to parts of parallel systems to be concise. On the example we discussed in this paper this turned out to be true. To judge the general applicability, further experiments, as with all new heuristics, are needed. The idea of partial model checking is certainly not restricted to Milner’s parallel composition operator and probably not to the modal μ -calculus.

Unguardedness Removal is akin to the Compaction Algorithm in [5] used for inverting fixed points.

Acknowledgements

Thanks to Bart Vergauwen for useful comments and for pointing out a mistake in the presentation of Equivalence Reduction in a preliminary version of this paper.

References

- [1] Henrik R. Andersen. *Verification of Temporal Properties of Concurrent Systems*. PhD thesis, Department of Computer Science, Aarhus University, Denmark, June 1993. PB-445.
- [2] Henrik R. Andersen. Model checking and boolean graphs. *Theoretical Computer Science*, 126(1):3–30, April 1994.
- [3] Henrik R. Andersen. A polyadic modal μ -calculus. Technical Report ID-TR: 1994-145, Department of Computer Science, Technical University of Denmark, August 1994.
- [4] Henrik R. Andersen, Colin Stirling, and Glynn Winskel. A compositional proof system for the modal μ -calculus. In *Proceedings, Ninth Annual IEEE Symposium on Logic in Computer Science*, pages 144–153, Paris, France, 4–7 July 1994.

- IEEE Computer Society Press. Also as BRICS Report RS-94-34.
- [5] Henrik R. Andersen and Bart Vergauwen. Efficient checking of behavioural relations and modal assertions using fixed-point inversion. In *Proceedings of CAV'95, LNCS*. Springer-Verlag, 1995. To appear. Preprint available as ID-TR: 1995-155.
- [6] Henrik R. Andersen and Glynn Winskel. Compositional checking of satisfaction. *Formal Methods In System Design*, 1(4), December 1992.
- [7] André Arnold and Paul Crubille. A linear algorithm to solve fixed-point equations on transition systems. *Information Processing Letters*, 29:57–66, 1988.
- [8] H. Bekič. Definable operations in general algebras, and the theory of automata and flow charts. In C.B.Jones, editor, *Hans Bekič: Programming Languages and Their Definition*, volume 177, pages 30–55. Springer-Verlag, 1984.
- [9] A. Bouali. *Etudes et mises en œuvre d'outils de vérification basée sur la bisimulation*. PhD thesis, Université Paris VII, 1993. In French.
- [10] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Proceedings, Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 428–439. IEEE Computer Society Press, 1990.
- [11] E. M. Clarke, T. Filkorn, and S. Jha. Exploiting symmetry in temporal logic model checking. In Courcoubetis [18].
- [12] Edmund M. Clarke, Orna Grumberg, and David E. Long. Model checking and abstraction. In *Proceedings of the 19'th Annual Symposium on Principles of Programming Languages, Santa Fe, New Mexico*, January 1992.
- [13] E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In Dexter Kozen, editor, *Logics of Programs, Workshop, Yorktown Heights, New York, May 1981*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer-Verlag, 1981.
- [14] E.M. Clarke and R.P. Kurshan, editors. *Proceedings of Workshop on Computer-Aided Verification 1990*, volume 531 of *LNCS*. Springer-Verlag, 1991.
- [15] Rance Cleaveland, Marion Dreimüller, and Bernhard Steffen. Faster model checking for the modal mu-calculus. In v. Bochmann and Probst [32], pages 383–394.
- [16] Rance Cleaveland and Bernhard Steffen. A linear-time model-checking algorithm for the alternation-free modal mu-calculus. In Kim G. Larsen and Arne Skou, editors, *Proceedings of the 3rd Workshop on Computer Aided Verification, July 1991, Aalborg*, volume 575 of *LNCS*. Springer-Verlag, 1992.
- [17] M.-M. Corsini and A. Rauzy. Symbolic model checking and constraint logic programming: a cross-fertilization. In Donald Sannella, editor, *ESOP'94. Proceedings of the 5th European Symposium on Programming, Edinburgh, Scotland, 11-13 April 1994*, volume 788 of *LNCS*, pages 180–194. Springer-Verlag, April 1994.
- [18] Costas Courcoubetis, editor. *Proceedings of the 5th International Conference on Computer Aided Verification, CAV'93*, volume 697 of *LNCS*. Springer-Verlag, 1993.
- [19] Dennis Dams, Orna Grumberg, and Rob Gerth. Generation of reduced models for checking fragments of CTL. In Courcoubetis [18].
- [20] E. A. Emerson and C. S. Jutla. The complexity of tree automata and logics of programs. *IEEE Foundations of Computer Science*, 29:328–337, 1988.
- [21] E. A. Emerson and A. P. Sistla. Symmetry and model checking. In Courcoubetis [18].
- [22] E. Allen Emerson and Chin-Luang Lei. Efficient model checking in fragments of the propositional mu-calculus. In *Symposium on Logic in Computer Science, Proceedings*, pages 267–278. IEEE, 1986.
- [23] R. Enders, T. Filkorn, and D. Taubner. Generating BDDs for symbolic model checking in CCS. *Journal of Distributed Computing*, 6:155–164, June 1993.
- [24] Patrice Godefroid and Pierre Wolper. A partial approach to model checking. In *Proceedings, Sixth Annual IEEE Symposium on Logic in Computer Science*, pages 406–415, Amsterdam, The Netherlands, 15–18 July 1991. IEEE Computer Society Press.
- [25] Dexter Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science*, 27, 1983.

- [26] Kim G. Larsen. Efficient local correctness checking. In v. Bochmann and Probst [32].
- [27] Kim G. Larsen and Liu Xinxin. Compositionality through an operational semantics of contexts. In M.S. Paterson, editor, *Proceedings of ICALP*, volume 443 of *LNCS*, pages 526–539. Springer-Verlag, 1990.
- [28] Robin Milner. *Communication and Concurrency*. Prentice Hall, 1989.
- [29] David Park. Concurrency and automata on infinite sequences. In Peter Deussen, editor, *Proceedings of Theoretical Computer Science, 5th GI-Conference, Karlsruhe, March 23-25, 1981*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer-Verlag, 1981.
- [30] R. Tarjan. *Data Structures and Network Algorithms*. Society for Industrial and Applied Mathematics, 1983.
- [31] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.
- [32] G. v. Bochmann and D. K. Probst, editors. *Proceedings of the 4th Workshop on Computer Aided Verification, CAV'92, June 29 - July 1, 1992, Montreal, Quebec, Canada*, volume 663 of *LNCS*. Springer-Verlag, 1992.
- [33] Valmari. A stubborn attack on state explosion. *CAV'90. AMS DIMACS Series in Discrete Math. & T.C.S.*, 3:25–41, 1990. Also in [14].
- [34] Bart Vergauwen and Johan Lewi. A linear algorithm for solving fixed-point equations on transition systems. In J.-C. Raoult, editor, *Proceedings of 17th Colloquium on Trees in Algebra and Programming, CAAP'92, Rennes, France*, volume 581 of *LNCS*, pages 322–341. Springer-Verlag, 1992.