

Efficient ordering of state variables and transition relation partitions in symbolic model checking

Preliminary version of July 1, 1997

Mathias Block ¹
Clemens Gröpl ²
Harry Preuß ³
Hans Jürgen Prömel ⁴
Anand Srivastav ⁵

Abstract

Among the main algorithmic problems in the verification of sequential circuits are the computation of good orders of state variables and transition relation partitions. Existing model checking packages like SMV from CMU, VIS from Berkeley or Rulebase from IBM Haifa provide variants of Rudell's sifting algorithm for the variable ordering problem and greedy-type algorithms for the partition ordering problem. For both problems, we give new *simulated annealing* based algorithms and we provide a model checking program called VERIFY which is based on SMV. The impact of our approach is demonstrated on industrial arbiter circuits and ISCAS '89 benchmarks. In particular on large industrial circuits our algorithms show a better space/time performance than previously given heuristics.

1 Introduction

A central task in formal verification of sequential circuits is the computation of orders of the state variables that lead to ordered binary decision diagrams (OBDDs) of manageable size. Unfortunately, finding an optimal variable order has shown to be NP-hard [3]. For combinational circuits, empirically the best way to find a good variable order is to construct a start order by heuristic algorithms (which exploit knowledge of the circuit structure) (see [11, 18, 25]) and then to improve the order dynamically by local search algorithms (Rudell [26], Fujita, Matsunaga and Kakuda [13]) or by simulated annealing (Ishiura, Saweda and Yajima [16], Mercer, Kapur and Ross [23], Bollig, Löbbing and Wegener [3]). In symbolic model checking the first and important step is to find a variable order so that the OBDD of the transition relation can be computed. As for combinatorial circuits we need a method to find such a variable order before OBDD computations start. Here a greedy-type heuristic has been proposed by Touati et al. [27]. To

¹e-mail: block@informatik.hu-berlin.de

²Graduate school "Algorithmische Diskrete Mathematik", supported by the Deutsche Forschungsgemeinschaft, grant GRK 219/2-97, e-mail: groepl@informatik.hu-berlin.de

³e-mail: preuss@informatik.hu-berlin.de

⁴supported by Deutsche Forschungsgemeinschaft, grant Pr 296/3-2, e-mail: proemel@informatik.hu-berlin.de

⁵supported by Deutsche Forschungsgemeinschaft, grant Pr 296/3-2, e-mail: srivasta@informatik.hu-berlin.de.
Address of all authors: Institut für Informatik, Humboldt Universität zu Berlin, Unter den Linden 6, 10099 Berlin, Germany.

improve variable orders dynamically, the state-of-the-art model checking programs (SMV [19], VIS [5], Rulebase [2]) provide variants of Rudell’s [26] sifting algorithm. The big advantage of sifting is that it often produces very good orders, but for large circuits it is time expensive.

Another obstacle in model checking has been the computation of the “monolithic” transition relation. Burch, Clarke and Long [6] proposed a method for partitioning the transition relation which drastically reduces space requirements in many cases. Their algorithm computes the transition relation in an iterative fashion by inserting exactly one partition in each iteration step along a given order of the transition relation partitions and takes advantage of early quantification. The sizes of the OBDDs computed in this process heavily depend on the order of the partitions. The complexity of this problem is unknown, but expected to be NP-hard. Current state-of-the-art heuristics are of greedy type (see [15] and [5]). Here again it is desirable to give algorithms which go beyond the greedy approach.

In this paper we give new heuristics for finding good orders of variables and of transition relation partitions based on simulated annealing. We present a fast and efficient model checking program called VERIFY, which is based on SMV [19]. In section 3.1, a greedy algorithm and a simulated annealing algorithm for finding an interleaved start order for sequential circuits is proposed. It works on the communication graph (defined in section 2) and uses an *OBDD-independent* objective function. In section 3.2, we suggest a fast simulated annealing algorithm for dynamic reordering which has been implemented as a new subroutine within SMV. The problem of finding good orders of transition relation partitions is addressed in section 4. We have implemented a new greedy heuristic and a simulated annealing algorithm which is based on a new objective function that can be computed *without* OBDD-operations. Furthermore, it is inevitable in model checking to eliminate all variables and states, which are not required for the verification of a certain specification or become redundant, when certain signals are constant. We suggest linear time algorithms for such eliminations.

The experimental results of our model checking package are given in section 5 and show a good performance of the simulated annealing approach on benchmark as well as on industrial circuits. We used Somenzi’s implementation of sifting with a size growing factor 1.2 (see [22]). It turns out that the combination of simulated annealing reordering with a simulated annealing start ordering was the best algorithm with respect to space/time tradeoff: For the industrial circuits simulated annealing reordering needed only between 7% - 39% of the time of the best sifting. The memory requirement of simulated annealing reordering for the industrial circuit **ibm97** was 7% higher than the requirement of sifting. In the other cases (**ibm95** and **ibm96a**) simulated annealing reordering needed only 64% resp. 48% of the memory required by sifting. For the most complicated industrial circuit (**ibm96b**) only the simulated annealing/simulated annealing approach was successful.

For the partition ordering problem we observed that the simulated annealing approach is clearly better than the greedy algorithm of Geist and Beer [15].

2 Preliminaries

In the context of formal verification, we assume that a sequential design is given as a network of inputs, combinational logic and latches. We define a sequential circuit \mathcal{C} formally as follows. \mathcal{C} consists of

- a set $V = \{v_1, \dots, v_m\}$ of state variables¹ where each v_i is a vector of Boolean state variables, i. e. $v_i = v_i[1], \dots, v_i[n_i]$ for all i . Let $n' = \sum_{i=1}^m n_i$.

¹Variables in SMV are state variables in this sense.

- a set $I = \{w_1, \dots, w_{n''}\}$ of Boolean input variables.
- a set $S_0 \subseteq \{0, 1\}^n$ of initial states ($n = n' + n''$). The points in $\{0, 1\}^n$ are the possible states of \mathcal{C} .
- a transition relation N which is given as a Boolean function $N : \{0, 1\}^n \times \{0, 1\}^{n'} \mapsto \{0, 1\}$.

Throughout this paper we consider synchronous circuits only. In symbolic model checking it is convenient to call a state variable v_i a present state variable v_i and to introduce a copy v_i' of v_i called next state variable. Let V' be the set of next state variables. Note that input variables have unconstrained transition behaviour. For a set of states $T \subseteq \{0, 1\}^n$, let us denote the OBDD of its characteristic function depending on the Boolean present state variables and the input variables by $T(V)$ (in this notation we suppress the dependency on input variables). The set of states reachable from a set $S_k \subseteq \{0, 1\}^{n'}$ within one step is given by the OBDD

$$S_{k+1}(V') := S_k(V') \vee \exists_{v \in V} S_k(V) \wedge N(V, V'). \quad (1)$$

Here we write $\exists_{v \in V}$ as an abbreviation for $\exists v_1 \dots \exists v_m$ where $\exists v_i := \exists v_i[1] \dots \exists v_i[n_i]$ for all i . The set of *reachable states* (starting from the initial set of states S_0) is the least fixed point $S_k = S_{k+1}$ of the above iteration.

The *communication graph* $G = (X, A)$ of a sequential circuit is defined as follows. We assume that a next state variable v_i' depends only on some present state variables, thus only present state variables are required for the determination of next state variables.²

Then G is a directed graph whose vertices are the state variables and whose arcs are defined as follows. Let us identify state variables by their indices. We say that state variable i *depends* on state variable j , if the present state variable v_j is required for the determination of the next state variable v_i' . In this case there is an arc $ji \in A$ in the communication graph G . We also add all loops ii to the arc set A . According to standard notation of graph theory, we put $\Gamma^+(j) := \{i \in X \mid ji \in A\}$ and $\Gamma^-(i) := \{j \in X \mid ji \in A\}$. We denote the set of outgoing arcs by $\partial^+(j) := \{ji \in A \mid i \in \Gamma^+(j)\}$ and the set of incoming arcs by $\partial^-(i) := \{ji \in A \mid j \in \Gamma^-(i)\}$.

The input for symbolic model checking is a description of a sequential circuit \mathcal{C} in high-level description languages like Verilog, VHDL or SMV. Such descriptions express the *circuit complexity* of \mathcal{C} which is defined as the number of gates of \mathcal{C} . In this paper we assume that the length of the SMV file of a circuit \mathcal{C} is proportional to the circuit complexity of \mathcal{C} . On the other hand, in symbolic model checking the complexity of algorithms is measured by a function in the number of state variables. Thus we define the *encoding length* of a sequential circuit \mathcal{C} as the number of its state variables and distinguish between the encoding length and the circuit complexity of a circuit.

The global optimization paradigm we will use in this paper is *simulated annealing*. The general structure of a simulated annealing algorithm is as follows (see [1] for a monograph treatment). Let Π be a combinatorial optimization problem with an objective function Z acting on the solution space of Π . The start solution is π_0 , the start temperature c_0 , and the number of stages in the first phase is L_0 .

Simulated Annealing (Minimization)

```
initialize( $\pi_0, c_0, L_0$ );  {start solution, start temperature, number of stages in first phase}
 $k := 0$ ;  $\pi := \pi_0$ ;
```

²This condition is guaranteed by the syntax of the SMV input language. Note that in general this is not true for the Boolean next state variables.

```

repeat
  for  $l := 1$  to  $L_k$  do {generate/accept}
    generate  $\pi^*$  from  $\pi$ ;
    if  $Z(\pi^*) \leq Z(\pi)$  then
       $\pi := \pi^*$ ;
    else
      choose randomly  $\Theta \in [0..1]$  from the uniform distribution;
      if  $e^{\frac{Z(\pi) - Z(\pi^*)}{c_k}} > \Theta$  then
         $\pi := \pi^*$ ;
      end if
    end if
  end for
   $k := k + 1$ ;
  compute( $L_k$ ); compute( $c_k$ );
until stop criterion satisfied
output  $\pi$ ;

```

The proper choice of L and c is crucial for efficient implementations of simulated annealing. In our implementations, we use the polynomial-time cooling schedule of Aarts and Korst [1], which turned out to be clearly superior to various non-adaptive schedules.

3 The Variable Ordering Problem

In this section, we describe simulated annealing based algorithms for finding initial variable orders and for dynamic reordering.

3.1 Initial Variable Ordering by Simulated Annealing

For $m \in \mathbb{N}$ put $[m] := \{1, \dots, m\}$ and denote by $\mathcal{S}_m := \{\sigma \mid \sigma : V \rightarrow [m]\}$ the set of all variable orders of m state variables. We define $z_\sigma(i)$ to be the latest position of a predecessor of i in G with respect to the variable order σ , or 0, if i is an input. So formally,

$$z_\sigma(i) := \begin{cases} \max\{\sigma(j) \mid j \in \Gamma^-(i)\}, & i \text{ latch;} \\ 0, & i \text{ input.} \end{cases}$$

Let Z be the following objective function

$$Z : \mathcal{S}_m \rightarrow \mathbb{N}; \quad \sigma \mapsto \sum_{i \in V} z_\sigma(i). \quad (2)$$

Good variable orders are related to small values of Z . An explanation for this is the following. Consider the OBDD of the transition relation of a sequential circuit with respect to a given order of state variables. For every state variable i there is at least one computation path in the OBDD which reaches the 0-sink immediately after the state variable $z_\sigma(i)$ has been tested. We argue that an OBDD in which sinks are reached as soon as possible or in other words, in which computation paths for the evaluation of subfunctions are small, should have a small size. This is the same observation, Mercer, Kapur and Ross made. They used it for their concept of partial OBDDs [23]. Minimizing Z means to enforce short computation paths in the OBDD of the transition relation.

The following greedy algorithm is a local minimization heuristic for Z . We denote the set of edges leaving $X \subseteq V$ by $\partial^+(X)$. Define the in-degree of i as $d^-(i) := |\Gamma^-(i)|$. The output is a variable order σ . For disjoint lists of variables σ and τ let $\sigma \frown \tau$ be the list where τ is appended to σ .

Greedy Initial Ordering

```

 $\sigma := \emptyset;$ 
repeat
  choose a node  $i \in V$  with minimal in-degree  $d^-(i)$ ;
  let  $\tau$  be an order of  $\Gamma^-(i) \setminus \{i\}$  with ascending  $d^-$ ;
   $\sigma := \sigma \frown \tau;$ 
  if  $i \notin \sigma$  then
     $\sigma := \sigma \frown (i);$ 
  end if
  in  $H = (V, A)$ , replace  $A$  by  $A \setminus \partial^+(\Gamma^-(i))$ ;
  {remove the edges leaving the predecessors of  $i$ }
   $H := H \setminus i$ ; {remove node  $i$ }
until  $H = \emptyset$ 
output  $\sigma$ ;
```

Note that $\tau = \emptyset$ is possible. The loop terminates, because in each step one node is removed from G . A similar algorithm can be found in Touati et al. [27]. They use the heuristic of Malik et al. [18] to optimize the order within the neighborhood $\Gamma^-(i)$ of i by depth-first search.

The simulated annealing algorithm for finding an initial variable order works as follows.

Initial Ordering by Simulated Annealing

- The objective function is Z as defined in (2).
- Generate new variable order by exchanging two randomly chosen variables.
- Use the cooling schedule of Aarts and Korst [1].

It is straightforward to see that the time for an update of Z is $O(n^2)$. We remark that even for our large example circuits the computation of the start order has been done in less than 2 minutes of cpu time on a an ultra sparc station with 167 MHz cycle.

3.2 Dynamic Variable Ordering by Simulated Annealing

Sifting and its variants like symmetric sifting and group sifting work well but are very time consuming. Furthermore we observed that sometimes sifting with a random start order works better or as good as one with a good start order, because sifting destroys the start order in early reorderings. We want to take advantage of our good start orders, so we looked for a fast alternative, which allows only small local perturbations.

We decided to use a single swap operation only. So the neighborhood of an order π consists of all orders π' arising from π by a swap of two variables and has linear size. The objective function is the OBDD size. Because we use an interleaved variable ordering, here a single swap of two adjacent variables consists of at least four conventional swap operations. First experiments with local search using this neighbourhood result in fast execution of reordering but in large OBDDs. To escape from local minima we used simulated annealing with the described swap neighbourhood.

Our algorithm is not comparable to that of Bollig, Löbbing and Wegener [3]. Bollig et al. obtained good variable orders for several benchmark circuits using an intense simulated annealing algorithm with a jump and exchange neighborhood, which has quadratic size. The execution times were enormous.

Because we use the swap neighbourhood and the cooling schedule of Aarts and Korst [1] for our implementation, we only need n proposed swap operations, before the temperature is reduced again. Furthermore we drastically reduce the temperature ($\delta = 10$ in the cooling schedule of [1]).

The choice of the swap neighbourhood for simulated annealing has one additional advantage. The swap of two variables i and j always causes the same change in the number of nodes, no matter whether or not other variables than i or j have been swapped before. This is because an exchange of two neighboring variables only affects the nodes of the levels corresponding to this variables (see e. g. [3]). Whenever simulated annealing refuses a swap or finds one that reduces the OBDD size, the difference of the number of nodes is stored. If in a later step the same swap is proposed again, we decide acceptance or rejection by looking at the previously stored increment. Thus useless swaps can be avoided. This saves time, because in the simulated annealing algorithm the bad swaps are the most time consuming operations, especially in the final phase of the algorithm.

As can be seen in the experimental results, our algorithm works very well with our start orders but not with random start orders.

4 Ordering Of Transition Relation Partitions

Often the (monolithic) transition relation of a sequential circuit is too large and cannot be computed. Burch, Clarke and Long [6] showed that sometimes it helps to partition the transition relation. Geist and Beer [15] observed that the order of partitions has a strong influence on the space complexity of the computation. They proposed a greedy algorithm for finding good partition orders in the case where no input variables are given and all state variables are Boolean. In this section we will suggest a greedy approach for the general case as well as a simulated annealing algorithm based on the jump neighborhood.

4.1 The Problem

Let \mathcal{C} be a sequential circuit with present (resp. next) state variable set $V = \{v_1, \dots, v_m\}$ (resp. $V' = \{v'_1, \dots, v'_m\}$), input variable set $I = \{w_1, \dots, w_{n'}\}$ and transition relation N . Suppose furthermore that for each state variable $v_i \in V$ there are n_i transition functions $f_{i,j} : I \times V \rightarrow \{0, 1\}$, $j = 1, \dots, n_i$, where $v_i[j] = 1$ iff $f_{i,j} = 1$.

Given the OBDD $S(V)$ of a state set $S \subset \{0, 1\}^n$, the OBDD $S'(V')$ of the next state set in a forward simulation step can be computed by the following formula [6]:

$$S'(V') = \exists_{v \in V} S(V) \wedge N(V, V') \quad (3)$$

The formula for a backward simulation step is

$$S(V) = \exists_{v' \in V'} S'(V') \wedge N(V, V'). \quad (4)$$

Let P_1, \dots, P_q be a partition of the set of next state variables V' . Each partition P_i induces a partition N_i of the transition relation defined by

$$N_i(V, V') = \bigwedge_{v'_i \in P_i} \bigwedge_{j=1}^{n_i} (v'_i[j] \Leftrightarrow f_{l,j}(I, V)).$$

Then

$$N(V, V') = \bigwedge_{i=1}^q N_i(V, V').$$

Let $\pi \in \mathcal{S}_q$ be an order of the partitions N_1, \dots, N_q . The relational product in (3) can now be computed iteratively: For each $1 \leq i \leq q$, let D_i be the set of present state variables and input variables on which N_i depends. Put

$$E_i = D_{\pi(i)} - \bigcup_{l=i+1}^q D_{\pi(l)}.$$

E_i is the set of present state and input variables contained in $D_{\pi(i)}$ that are not contained in $D_{\pi(l)}$ for any l larger than i . Define $S_0(V, V') = S(V)$ and $S_q(V, V') = S(V')$.

$$\begin{aligned} S_1(V, V') &= \bigvee_{v \in E_1} S_0(V, V') \wedge N_{\pi(1)}(V, V') \\ S_2(V, V') &= \bigvee_{v \in E_2} S_1(V, V') \wedge N_{\pi(2)}(V, V') \\ &\vdots \\ S_q(V, V') &= \bigvee_{v \in E_{q-1}} S_{q-1}(V, V') \wedge N_{\pi(q-1)}(V, V') \end{aligned}$$

The computation of the formula in (4) goes similar: Let D'_i be the set of next state variables and input variables on which N_i depends. Put $E'_i = D'_{\pi(i)} - \bigcup_{j=1}^{i-1} D'_{\pi(j)}$, $S'_0(V, V') := S(V)$ and $S'_q(V, V') := S'(V')$. Now $S(V)$ can be computed by the following iteration:

$$\begin{aligned} S'_{q-1}(V, V') &= \bigvee_{v \in E'_q} (S'_q(V, V') \wedge N_{\pi(q)}(V, V')) \\ S'_{q-2}(V, V') &= \bigvee_{v \in E'_{q-1}} (S'_{q-1}(V, V') \wedge N_{\pi(q-1)}(V, V')) \\ &\vdots \\ S'_0(V, V') &= \bigvee_{v \in E'_1} (S'_1(V, V') \wedge N_{\pi(1)}(V, V')). \end{aligned}$$

For an OBDD G let $\text{Var}(G)$ be the set of *Boolean* variables on which G depends and let $\text{var}(G) = |\text{Var}(G)|$.

We would like to choose a partition order π so that each relational product S_i in the iteration above is small. Unfortunately, there is no direct relation known between π and the sizes of the S_i 's. A heuristic measure for the quality of π is the number of variables the relational products S_i depends on. The greedy heuristic as well as the simulated annealing algorithm are based on this observation.

4.2 The Greedy Algorithm

The greedy heuristic of Geist and Beer [15] has been formulated for the special case where $I = \emptyset$ and $m = n = q$. To extend the greedy algorithm to the general case, we are faced with the following problem: Given a subset L of the transition relation partitions, Geist and Beer

call a (Boolean) state variable $v \in N_i \in L$ *unique* with respect to L , if v is not contained in any partition $N_j \in L - N_i$. The main criterion in the greedy heuristic of Geist and Beer is to choose in the i -th step a partition N_j with the maximum number of unique variables among the remaining partitions. This means that a maximum number of variables can be quantified out, so $\text{var}(S_i)$ is minimized. Now, before we can compute S_i , we have to compute the conjunction $S_{i-1} \wedge N_j$. In the case $m = n = q$ the partition N_j adds exactly one new next state variable to $S_{i-1} \wedge N_j$, so we expect that the size of $S_{i-1} \wedge N_j$ increases moderately compared to the sizes of S_{i-1} and N_j . This might not be true in the general case: if $q < n$, a partition N_j corresponding to a large partition set P_j will blow up the number of variables in $S_{i-1} \wedge N_j$, before some variables can be quantified out leading to a small S_i . Thus, the Geist-Beer criterion is not a local optimization strategy in the general case.

We propose a criterion that takes care of both, the conjunction $S_{i-1} \wedge N_j$ and the partial relational product S_i by selecting a partition N_j so that $\text{var}(S_{i-1} \wedge N_j)$ and $\text{var}(S_i)$ are minimized. The greedy algorithm for the forward simulation step is the following procedure.

Greedy Partition Ordering (Forward)

```

 $L := \{N_1, \dots, N_q\};$ 
for  $i = 1$  to  $q$  do
  (a) Choose  $N_j \in L$  that minimizes
       $\text{var}(S_{i-1}(V, V') \wedge N_j(V, V'));$ 
  if there are two or more candidates in (a), then
    (b) Choose among them a partition  $N_j$  that minimizes
         $\text{var}(\exists_{v \in E_{i-1}}(S_{i-1}(V, V') \wedge N_j(V, V')));$ 
    if there are two or more candidates in (b), then
      (c) Choose among them a partition  $N_j$  that maximizes
           $|\text{Var}(N_j) \cap \bigcup_{N_r \in (L - \{N_j\})} \text{Var}(N_r)|;$ 
          {If there are two or more candidates in (c), choose one among them arbitrarily}
    end if
  end if
   $L := L - N_j;$   $\pi(i) := j;$ 
end for
output  $\pi;$ 

```

The counterpart for the backward simulation step is:

Greedy Partition Ordering (Backward)

```

 $L := \{N_1, \dots, N_q\};$ 
for  $i = 1$  to  $q$  do
  (a) Choose  $N_j \in L$  that minimizes
       $\text{var}(S'_{q+i-1}(V, V') \wedge N_j(V, V'));$ 
  if there are two or more candidates in (a), then
    (b) Choose among them a partition  $N_j$  that minimizes
         $\text{var}(\exists_{v \in E'_{q+i-1}}(S'_{q+i-1}(V, V') \wedge N_j(V, V')));$ 
    if there are two or more candidates in (b), then
      (c) Choose among them a partition  $N_j$  that maximizes
           $|\text{Var}(N_j) \cap \bigcup_{N_r \in (L - \{N_j\})} \text{Var}(N_r)|;$ 
          {If there are two or more candidates in (c), choose one among them arbitrarily}
    end if
  end if

```

```

    end if
  end if
   $L := L - N_j; \pi(q + 1 - i) := j;$ 
end for
output  $\pi$ ;

```

4.3 The Simulated Annealing Algorithm

In this section we introduce a global objective function for the assessment of partition orders. Let $\pi \in \mathcal{S}_q$. For $p \in [1, \infty]$ let $\|\cdot\|_p$ be the usual p -norm. Let $X(\pi)$ and $Y(\pi)$ be the vectors

$$\begin{aligned} X(\pi) &:= (\text{var}(S_0(V, V') \wedge N_{\pi(1)}(V, V')), \dots, \text{var}(S_{q-1}(V, V') \wedge N_{\pi(q)}(V, V'))) \\ Y(\pi) &:= (\text{var}(S'_q(V, V') \wedge N_{\pi(q)}(V, V')), \dots, \text{var}(S'_1(V, V') \wedge N_{\pi(1)}(V, V'))) \end{aligned}$$

and define

$$\begin{aligned} c_{p,f}(\pi) &:= \|X(\pi)\|_p, \\ c_{p,b}(\pi) &:= \|Y(\pi)\|_p. \end{aligned}$$

(The index f resp. b indicates forward resp. backward simulation.) For the maximum norm the greedy algorithm in the previous section is a local optimization procedure for the functions $c_{p,f}$ and $c_{p,b}$. We argue that good partition orders should have small $c_{p,f}$ and $c_{p,b}$ values. The idea behind the p -norm for $p < \infty$ is the consideration of an average quality of partition orders. For example, if one component of the vector $X(\pi)$ is inevitable large, but all others are small, the maximum norm is less informative than the L^1 norm. So, p -norms for $p < \infty$ are more sensitive for such situations. The simulated annealing algorithm for forward simulation is:

Initial Partition Ordering by Simulated Annealing (Forward)

- Input: A norm parameter $p \in [1, \infty]$ and a partitioned transition relation $N = N_1 \wedge \dots \wedge N_q$
- Output: An order $\pi \in \mathcal{S}_q$
- Run simulated annealing with cost function $c_{p,f}$, jump neighborhood and the polynomial-time cooling schedule of Aarts and Korst [1].

For the backward simulation step we have

Initial Partition Ordering by Simulated Annealing (Backward)

- Same as for forward step, but with $c_{p,f}$ replaced by $c_{p,b}$

An important property of these objective functions is that they are identical for backward and forward simulation. So we need to use only one of the simulated annealing procedures.

Theorem 4.1 *Assume that $S(V)$ (resp. $S'(V)$) depends on all l present state (resp. next state) variables. Then we have*

$$c_{p,f}(\pi) = c_{p,b}(\pi)$$

for all $\pi \in \mathcal{S}_q$.

Proof: It is sufficient to show for all $1 \leq i \leq q$ that

$$S_{i-1}(V, V') \wedge N_{\pi(i)}(V, V') \quad (5)$$

and

$$S'_i(V, V') \wedge N_{\pi(i)}(V, V') \quad (6)$$

depend on the same variables.

Let $S_{i-1}(V, V')$ depend on a variable x .

- Case 1: x is a present state variable. Then x has not been quantified out. Hence there is a partition $N_{\pi(j)}(V, V')$, $j \geq i$ depending on x . If $j = i$, we are done. If $j > i$, then $S'_i(V, V')$ depends on x , because $S'_{j-1}(V, V')$ depends on x and no present state variables are quantified out in the backward simulation step.
- Case 2: x is a next state variable. Then there is exactly one partition $N_{\pi(j)}(V, V')$, $j < i$, which depends on x . But this means that in the backward simulation x has not been quantified out, so (6) depends on x .
- Case 3: x is an input variable. Then there are partitions $N_{\pi(j)}(V, V')$ and $N_{\pi(j')}(I, V, V')$, $j \geq i > j'$, both depending on x . If $j = i$, (6) trivially depends on x . Otherwise, $S'_{j-1}(V, V')$ depends on x and x has not been quantified out, because $N_{\pi(j')}(V, V')$ depends on x .

All together, $S'_i(V, V') \wedge N_{\pi(i)}(V, V')$ depends on x . In the same fashion we can show that $S_{i-1}(V, V') \wedge N_{\pi(i)}(V, V')$ depends on a variable x , whenever $S'_i(V, V') \wedge N_{\pi(i)}(V, V')$ depends on x . \square

5 Experimental Results

We have chosen test circuits from industry and from the ISCAS'89 benchmarks. The circuits **ibm95**, **ibm96**, and **ibm97** are arbiter circuits designed at the IBM laboratory in Böblingen. **ibm96** is highly symmetric. It activates either a fetch/store signal or a disconnected/sense signal but never both. Thus it suffices to verify **ibm96** by considering the circuit either under the condition that the fetch/store signal is 0 (**ibm96a**) or under the condition that the disconnected/sense signal is 0 (**ibm96b**). This information was provided by the designers. The partial circuit **ibm96a** has 98 Boolean present state variables and **ibm96b** has 166 present state variables.

From the ISCAS'89 benchmark set, we have chosen the circuits **s444**, **s526**, **s713**, **s953**, and **s1238**. Since for these circuits no specifications are available, we decided to define specifications according to Touati et al. [27], where we test the circuits against themselves, i.e. we check whether or not two copies sharing their inputs have the same output behaviour.

The following table shows the number of state variables, input variables, and fixed-point iterations required to compute the set of reachable states for the circuits. Note that in fact, the number of state variables has to be doubled for verification in case of the ISCAS'89 Benchmark Circuits. Nevertheless, these circuits are relatively small compared with the **ibm*** arbiters.

Circuit	s444	s526	s713	s953	s1238	ibm95	ibm96	ibm96a	ibm96b	ibm97
Bool. state var.	21	21	19	29	18	199	234	98	166	94
Bool. input var.	3	3	35	16	14	0	0	0	0	0
Fixed point iter.	151	151	7	11	3	4	–	55	103	32

Our model checking package VERIFY is based on SMV enriched by the simulated annealing procedures discussed in this paper. The tests were performed using the following SMV options.

- **-cp n** This option stands for model checking with a (conjunctive) partitioned transition relation. The integer **n** specifies the OBDD size of the transition relation partitions. In the algorithm, whenever the OBDD size of the actual transition relation partition exceeds **n**, a new partition is generated.
- **-f** (compute reachable state space before evaluating CTL-formulas)
- **-AG** (tests only AG-formulas — this option was re-implemented due to bugs in the original program).

Furthermore, we have introduced a garbage collection option **-gc**, which deletes the OBDD of the relational products computed so far once the new relational product has been computed. Such a clean-up procedure is obviously necessary if there are many partitions, but missing in SMV. Note that the memory requirement of VERIFY itself is already 21 430 272 bytes. This is largely due to the SMV cache options **-c 1046429** and **-k 1046429** which are necessary for large circuits, but were used for all circuits to make the comparison of the various options easier. The real memory requirement was always 20928 KBytes more than the values given in the tables. The tests were performed on an ultra sparc station with 512 MB RAM and 167 MHz cycle.

5.1 Variable Reduction

Variable Reduction was applied for **ibm96**, leading to **ibm96a** and **ibm96b**. Before starting the verification of a specification, we removed variables not required by the specification under consideration. This preprocessing step lead to a drastic reduction of the number of state variables. The corresponding subcircuits can be generated automatically in the following way. First, some signals are fixed to constant values. Then this information is propagated through the whole design, and redundant signals and gates are removed. This can be done by parsing the SMV input and applying a depth-first search, which is linear in the circuit complexity.

5.2 Variable Ordering

Here we give computational results for various initial and dynamic variable ordering heuristics. Initial variable orders were obtained by the greedy heuristic and the simulated annealing described above and compared with a random variable order. Dynamic variable ordering heuristics are sifting and simulated annealing. We used Somenzi's implementation of sifting with a size growing factor 1.2 (see [22]). The parameters we measure are:

- **Kbytes** is the maximal number of Kbytes required in the verification minus 20928
- **time** is the total CPU time seconds
- **trans** is the number of OBDD-nodes of the transition relation before the reachable states computation starts
- **part** is the number of partitions of the transition relation

Reordering was started first, when the number of OBDD-nodes after a garbage collection was 5000 or more. Later, reordering was started, when the number of OBDD-nodes after a garbage collection was more than $4/3$ of the number of OBDD nodes after the last reordering. Furthermore, we used conjunctively partitioned transition relations with partitions size limit 1000.

The experiments show that the combination of simulated annealing reordering with a simulated annealing start ordering was the best algorithm with respect to space/time tradeoff. For the industrial circuits simulated annealing reordering needed only between 7% - 39% of the time of the best sifting. The memory requirement of simulated annealing reordering for **ibm97** was 7% higher than the requirement of sifting. In the other cases (**ibm95** and **ibm96a**) simulated annealing reordering needed only 64% resp. 48% of the memory required by sifting. For the most complicated industrial circuit (**ibm96b**) only the simulated annealing/simulated annealing approach was successful.

All reordering methods except for **ibm95** showed a better performance with a start ordering computed by greedy or simulated annealing than with a random start ordering. The running times of simulated annealing reordering are good only if a good start ordering was given. The tendency we observed is that for large circuits the combination of a good start ordering computed by simulated annealing and reordering by simulated annealing is a highly recommendable approach.

s444		without reorder	sifting reorder	sim.-ann. reorder
random input order	KBytes	960	320	640
	time	75,00	102,93	116,16
	trans	9 576	2 743	3 027
	part	8	5	7
sim.-ann. input order	KBytes	256	192	320
	time	22,07	42,38	38,04
	trans	2 019	2 019	2 019
	part	2	2	2
greedy input order	KBytes	896	320	384
	time	61,34	55,94	56,51
	trans	1 566	1 566	1 566
	part	2	2	2

s526		without reorder	sifting reorder	sim.-ann. reorder
random input order	KBytes	896	320	576
	time	80,04	126,68	129,68
	trans	10 612	3 121	2 763
	part	8	7	7
sim.-ann. input order	KBytes	256	192	320
	time	22,19	69,75	55,07
	trans	3 592	3 592	3 592
	part	3	3	3
greedy input order	KBytes	576	320	320
	time	48,24	67,59	56,09
	trans	2 032	2 032	2 032
	part	2	2	2

s713		without reorder	sifting reorder	sim.-ann. reorder
random input order	KBytes	8 640	640	2 432
	time	85,74	70,11	114,78
	trans	32 052	3 789	21 690
	part	13	7	13
sim.-ann. input order	KBytes	640	640	704
	time	4,75	69,13	14,44
	trans	6 048	4 678	4 829
	part	5	5	5
greedy input order	KBytes	640	640	704
	time	5,18	71,16	13,86
	trans	6 138	4 747	5 090
	part	5	5	5

s953		without reorder	sifting reorder	sim.-ann. reorder
random input order	KBytes	1 536	704	1 152
	time	17,14	182,90	35,52
	trans	30 282	7 034	17 063
	part	17	9	15
sim.-ann. input order	KBytes	768	704	704
	time	6,38	113,57	21,98
	trans	12 103	6 654	8 510
	part	9	8	8
greedy input order	KBytes	704	704	704
	time	6,95	126,86	25,13
	trans	10 885	7 034	7 814
	part	9	9	9

s1238		without reorder	sifting reorder	sim.-ann. reorder
random input order	KBytes	4 032	640	1 088
	time	24,72	48,02	26,93
	trans	8 483	2 164	3 184
	part	6	5	6
sim.-ann. input order	KBytes	704	640	640
	time	4,73	42,81	13,13
	trans	4 347	2 513	3 050
	part	4	4	4
greedy input order	KBytes	640	640	640
	time	4,87	44,77	13,51
	trans	3 941	2 625	3 313
	part	4	3	4

ibm96a		without reorder	sifting reorder	sim.-ann. reorder
random input order	KBytes	spaceout	22 528	12 032
	time		27 338,30	2 260,92
	trans		12 172	31 537
	part		14	25
sim.-ann. input order	KBytes	30 144	9 408	4 544
	time	1 316,78	8 506,65	613,03
	trans	18 428	14 848	14 023
	part	14	12	16
greedy input order	KBytes	49 088	19 904	34 368
	time	6 301,72	26 754,90	9 933,15
	trans	91 884	17 982	40 567
	part	20	16	21

ibm96b		without reorder	sifting reorder	sim.-ann. reorder
random input order	KBytes	spaceout	timeout	timeout
	time			
	trans			
	part			
sim.-ann. input order	KBytes	spaceout	timeout	163 584
	time			88 236,10
	trans			105 976
	part			38
greedy input order	KBytes	spaceout	timeout	timeout
	time			
	trans			
	part			

ibm95		without reorder	sifting reorder	sim.-ann. reorder
random input order	KBytes	1 984	1 472	896
	time	25,54	503,46	60,17
	trans	52 651	15 552	19 432
	part	37	17	18
sim.-ann. input order	KBytes	1 024	1 728	960
	time	23,86	537,30	63,14
	trans	26 901	15 353	20 903
	part	19	13	16
greedy input order	KBytes	1 152	1 408	1 536
	time	22,82	417,08	119,21
	trans	26 612	15 932	16 818
	part	20	14	15

ibm97		without reorder	sifting reorder	sim.-ann. reorder
random input order	KBytes	spaceout	3 008	8 960
	time		1 331,19	1 236,55
	trans		4 760	20 925
	part		8	15
sim.-ann. input order	KBytes	19 776	2 688	5 248
	time	493,60	1 145,61	583,93
	trans	51 294	5 205	10 450
	part	14	7	10
greedy input order	KBytes	15 808	2 944	2 880
	time	483,45	1 594,66	334,38
	trans	61 846	5 080	8 810
	part	13	7	11

5.3 Partition Ordering

Here we compare the natural order given by the SMV input file of the circuit with the orderings obtained by our adapted greedy heuristic (**greedy_part**) and with simulated annealing (**sim_part**). The interesting and relevant parameters are the running time (**time**), the memory (**Kbytes**), the maximum size of the OBDD of a relational product (**max_rel_prod**) and the average size (**mean_rel_prod**) of the OBDDs of all relational products.

For the ISCAS benchmarks we considered the partitioned transition relation have only a few partitions. Here partition ordering does not make any sense. (Larger benchmarks like **s1423** or **s5378** were already untractable.) The following tables contain experimental results for **ibm96a** and **ibm97**. The tests were performed using an input variable ordering obtained by simulated annealing and without reordering. The first table shows the results for a partition size limit of 100, the second is for 1000. Usually the running times are less critical parameters than the memory requirements. The running times for natural orderings are slightly better than the running times of simulated annealing, but both are clearly better than the running times of the greedy algorithm. The interesting observation is that for a simulated annealing ordering the memory requirement was as least as good as the best of the other heuristics and for **ibm97** we have an improvement of 45%.

Partition Size 100

Circuit	Order	time	Kbytes	max_relprod	mean_relprod
ibm96a (49 partitions)	natural order	2672	25984	643050	57122
	greedy	3378	42176	1050625	66485
	simulated ann.	2966	25920	638591	55068
ibm97 (32 partitions)	natural order	833	19712	238980	40103
	greedy	711	16960	315296	33106
	simulated ann.	592	10880	165774	25384

Partition Size 1000

Circuit	Order	time	Kbytes	max_relprod	mean_relprod
ibm96a (14 partitions)	natural order	1173	30144	643050	59798
	greedy	2186	53376	1099513	108001
	simulated ann.	1177	30144	645268	61623
ibm97 (14 partitions)	natural order	492	19776	238982	38874
	greedy	397	15168	235212	32182
	simulated ann.	407	15168	235212	32418

References

- [1] E. Aarts, J. Korst; *Simulated annealing and Boltzmann machines*. Wiley Interscience Series in Discrete Mathematics and Optimization, Tiptree, England (1990).
- [2] I. Beer, S. Ben-David, C. Eisner, A. Landver; *Rulebase: an industry-oriented formal verification tool*. Preprint (1996), IBM Haifa Research Laboratory, Haifa, Israel.
- [3] B. Bollig, M. Löbbing, I. Wegener; *Variable orderings for OBDDs, simulated annealing, and the hidden weighted bit function*. Preprint, Institut für Informatik, Universität Dortmund, 1996.
- [4] K. S. Brace, R. L. Rudell, R. E. Bryant; *Efficient implementation of a BDD package*. In 27. ACM/IEEE Design Automation Conference, 1990, 40–45.
- [5] R. K. Brayton, G. D. Hachtel, A. Sangiovanni-Vincentelli, F. Somenzi, A. Aziz, S.-T. Cheng, S. Edwards, S. Khatri, Y. Kukimoto, A. Pardo, S. Qadeer, R. K. Ranjan, S. Sarwary, T. R. Shiple, G. Swamy, T. Villa; *VIS: A System for Verification and Synthesis*.
- [6] J. R. Burch, E. M. Clarke, D. E. Long; *Symbolic Model Checking with Partitioned Transition Relations*. In International Conference on Very Large Scale Integration, 1991.
- [7] J. R. Burch, E. M. Clarke, D. E. Long; *Representing Circuits More Efficiently in Symbolic Model Checking*. 28th DAC, 1991, 403 – 407.
- [8] J. R. Burch, E. M. Clarke, D. E. Long, K. L. McMillan; *Symbolic Model Checking for Sequential Circuit Verification*. In IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 13, No. 4, 1994, 401–424.
- [9] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, L. J. Hwang; *Symbolic Model Checking: 10²⁰ States and Beyond*, Information and Computation 98 (1992), 142 – 170.
- [10] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill; *Sequential Circuit Verification Using Symbolic Model Checking*. In 27th DAC, 1990, 46 – 51.
- [11] K. M. Butler, D. E. Ross, R. Kapur, M. R. Mercer; *Heuristics to compute variable orderings for efficient manipulation of ordered binary decision diagrams*. In 28. ACM/IEEE Design Automation Conference, 1991, 417–420.
- [12] S. J. Friedman, K. J. Supowit; *Finding the optimal variable ordering for binary decision diagrams*. In IEEE Trans. on Computers 39, 1990, 710–713.
- [13] M. Fujita, H. Fujisawa, N. Kawato; *Evaluation and improvements of Boolean comparison method based on binary decision diagrams*. In IEEE Int. Conf. on Computer Aided Design ICCAD, 1988, 2–5.

- [14] M. Fujita, Y. Matsunaga, T. Kakuda; *On variable ordering of binary decision diagrams for the application of multi-level logic synthesis*. In European Design Automation Conference EDAC, 1991, 50–54.
- [15] D. Geist, I. Beer; *Efficient Model Checking by Automated Ordering of Transition Relation Partitions*. In CAV '94, Lecture Notes in Computer Science, vol. 818, 299–310.
- [16] N. Ishiura, H. Sawada, S. Yajima; *Minimization of binary decision diagrams based on exchange of variables*. In IEEE Int. Conf. on Computer Aided Design ICCAD, 1991, 472–475.
- [17] S.-W. Jeong, B. Plessier, G. D. Hachtel, F. Somenzi; *Variable ordering and selection for FSM traversal*. In IEEE Int. Conf. on Computer Aided Design ICCAD, 1991, 476–479.
- [18] S. Malik, A. R. Wang, R. K. Brayton, A. Sangiovanni-Vicentelli; *Logic verification using binary decision diagrams in a logic synthesis environment*. In IEEE Int. Conf. on Computer Aided Design ICCAD, 1988, 6–9.
- [19] K. L. McMillan; *The SMV System DRAFT*. Carnegie Mellon University, 1992.
- [20] K. L. McMillan, J. Schwalbe; *Formal verification of the Encore Gigamax cache consistency protocol*. In Proceedings of the 1991 International Symposium on Shared Memory Multiprocessors, 1991.
- [21] K. L. McMillan; *Symbolic Model Checking*. Kluwer Academic Press, 1993.
- [22] C. Meinel, A. Slobodová; *Speeding up Variable Reordering of OBDDs*. Forschungsbericht Nr. 96-40, FB-IV Informatik, Universität Trier, 1996.
- [23] M. R. Mercer, R. Kapur, D. E. Ross; *Functional approaches to generating orderings for efficient symbolic representation*. In 29. ACM/IEEE Design Automation Conference, 1992, 614–619.
- [24] R. K. Ranjan, A. Aziz, R. K. Brayton, B. Plessier, C. Pixley; *Efficient BDD Algorithms for FSM Synthesis and Verification*.
- [25] D. E. Ross, K. M. Butler, R. Kapur, M. R. Mercer; *Fast functional evaluation of candidate OBDD variable ordering*. In European Design Automation Conference EDAC, 1991, 4–10.
- [26] R. Rudell; *Dynamic variable ordering for ordered binary decision diagrams*. In Int. Conf. on Computer Aided Design ICCAD, 1993, 42–47.
- [27] H. J. Touati, H. Savoi, B. Lin; *Implicit state enumeration of finite state machines using BDD's*. In Int. Conf. on Computer Aided Design ICCAD, 1991, 130–133.