FORMAL VERIFICATION OF DIGITAL CIRCUITS USING SYMBOLIC MODEL CHECKING

Aleš Časar, Zmago Brezočnik, Tatjana Kapus
University of Maribor, Faculty of Electrical Engineering and Computer Science, Slovenia

Keywords: electrotechnics, computer science, digital circuits, BDD, Binary Decision Diagrams, FSM, Final State Machines, transition relations, characteristic functions, reachable states, CTL, Computation Tree Logic, fairness constraints, formal verification, symbolic model checking, computer software, software packages, experimental results

Abstract: This paper presents efficient algorithms for digital circuit verification. The algorithms are based on symbolic CTL (Computation Tree Logic) model checking. We also present an extension of ordinary CTL with fairness constraints. They enable verification of circuits with symbolic model checking regarding only fair paths in the computation tree. The model checking algorithms were implemented as part of a fully home-made program package for manipulating finite state machine descriptions of digital circuits, represented with Boolean functions. The package is also based on a fully home-made program package for manipulating Boolean functions, represented with binary decision diagrams.

Formalna verifikacija digitalnih vezij s simboličnim preverjanjem modelov

Ključne besede: elektrotehnika, računalništvo, vezja digitalna, BDD grafi odločitveni binarni, FSM stroji stanj končnih, relacije prehajalne, funkcije karakteristične, stanja dosegljiva, CTL logika drevesa izračunavanj, omejitve poštenostne, verifikacija formalna, preverjanje modelov simbolično, oprema programska računalniška, paketi opreme programske, rezultati eksperimentalni

Povzetek: V članku predstavljamo učinkovite algoritme za verifikacijo digitalnih vezij. Algoritmi temeljijo na simboličnem preverjanju modelov z logiko drevesa izvajanj ("Computation Tree Logic" - CTL). Prav tako predstavljamo razširitev navadnega CTL s poštenostnimi omejitvami, ki omogočajo verifikacijo vezij samo vzdolž poštenih poti v drevesu izvajanj. Algoritmi za preverjanje modelov so implementirani kot del povsem domačega programskega paketa za obdelavo opisov digitalnih vezij s končnimi avtomati, predstavljenih z logičnimi funkcijami, ki temelji na prav tako povsem domačem paketu za obdelavo logičnih funkcij, predstavljenih z binarnimi odločitvenimi grafi.

1 Introduction

The manipulation of finite state machines (FSMs) is very common nowadays in the areas of digital circuit design like formal verification, automatic synthesis, and testing [12]. One of the main problems that must be solved is the verification of properties of FSMs representing digital circuits.

The properties are often specified by computation tree logic (CTL) because with a properly written CTL formula it can automatically be verified if certain property is valid in a FSM or not. Everything, CTL formulas, the set of states of the FSM, and its transition relation can be uniquely represented as Boolean functions, which can be very efficiently represented with binary decision diagrams (BDDs). Using symbolic state space traversal and model checking, we can avoid combinatorial explosion, which is one of the main problems with enumeration-based methods.

Unfortunately, it is impossible to specify all properties in CTL. A significant part of such properties are those that could be specified in CTL, but we want them to be verified only in a FSM constrained to normal (fair) paths in order to avoid singularities and similar things. The problem is solved by the introduction of fairness constraints, which constrain the FSM to fair paths, so that the properties can be verified in the usual way.

The purpose of this paper is to present the model checking algorithms which we implemented within fully home-made program package for manipulating finite state machine descriptions of digital circuits, represented with Boolean functions. The algorithms have performed very well.

In Section 2 we briefly review BDDs and show how to represent FSMs with them. Section 3 describes the methods of searching reachable states. The main part of the paper is Section 4 where we present algorithms for symbolic model checking in CTL. Algorithms for symbolic model checking using fairness constraints are described in Section 5. Experimental results for benchmark circuits are given in Section 6. We conclude with some discussion and plans for future work.

2 Preliminaries

Binary decision diagrams (BDDs) are compact canonical representations of Boolean functions [2]. Their size is closely related to the variable ordering. The manipulation of Boolean functions represented with BDDs is very efficient. Hence, BDDs have become widely used in various CAD applications, including state space traversal and model checking.

BDDs can also be used for representing and manipulating sets if we represent sets by means of their characteristic functions. If $\mathcal{A}$ is a subset of $\{0,1\}^n$, its characteristic function $\chi_{\mathcal{A}} : \{0,1\}^n \rightarrow \{0,1\}$ is defined by

$$\chi_{\mathcal{A}}(\underline{y}) = \begin{cases} 0; \underline{y} \notin \mathcal{A} \\ 1; \underline{y} \in \mathcal{A} \end{cases} \quad (1)$$

Thus, set operations can be performed by Boolean operations over the corresponding characteristic functions (e.g., $\chi_{\mathcal{A}^c} = \overline{\chi_{\mathcal{A}}}$, $\chi_{\mathcal{A}\cup\mathcal{B}} = \chi_{\mathcal{A}} + \chi_{\mathcal{B}}$, $\chi_{\mathcal{A}\cap\mathcal{B}} = \chi_{\mathcal{A}} \cdot \chi_{\mathcal{B}}$). For the sake of shorter notation we will denote $\chi_{\mathcal{A}}$ by A in the rest of the paper. Since the characteristic functions are Boolean, we can efficiently manipulate them with BDDs.

A deterministic *finite state machine* (FSM) M is a sixtuple $M = (\Sigma, \mathcal{S}, O, \delta, \lambda, s_0)$, where Σ is a finite set of input symbols, $\mathcal{S}$ a finite set of states, O a finite set of output symbols, $\delta : \mathcal{S} \times \Sigma \rightarrow \mathcal{S}$ a state transition function, $\lambda : \mathcal{S} \times \Sigma \rightarrow O$ an output function, and $s_0 \in \mathcal{S}$ an initial state.

If we want to realize a FSM by a digital circuit, we will have to encode the sets $\mathcal{S}$, Σ, and O by binary symbols (e.g., 0 and 1). States are encoded by state variables. At least $n = \lceil \log_2 |\mathcal{S}| \rceil$ state variables, $m = \lceil \log_2 |\Sigma| \rceil$ input variables, and $l = \lceil \log_2 |O| \rceil$ output variables of the circuit are needed. Let $\mathcal{Y}$, $\mathcal{X}$, and $\mathcal{Z}$ represent the set of state variables, the set of input variables, and the set of output variables, respectively.

Once the states and the input symbols of the circuit are encoded, we denote a state transition function as $\delta : \{0,1\}^n \times \{0,1\}^m \rightarrow \{0,1\}^n$. Then, next state variables are functions of present state variables and input variables. We denote next state variables by an added prime (') and write a transition function of a state variable y_i as

$$y_i' = \delta_i(y_0, y_1, \ldots, y_{n-1}, x_0, x_1, \ldots, x_{m-1}) \quad (2)$$

for $i = 0,1,\ldots,n-1$. We rather introduce transition relations

$$T_i = y_i' \Leftrightarrow \delta_i(y_0, y_1, \ldots, y_{n-1}, x_0, x_1, \ldots, x_{m-1}) \quad (3)$$

Namely, relations have much greater expressive power than functions [3]. With relations it is very easy to handle nondeterminism and to perform backward reachability search that is relatively difficult with functions. Transition relations T_i can be combined by taking their conjunction to form the *monolithic transition relation* $T = T_0 \cdot T_1 \cdot \ldots \cdot T_{n-1}$.

3 Searching Reachable States of FSM

First, let us show how we search reachable states of a FSM. Let $\mathcal{S}_i$ denote a set of states reachable in at most i steps. $\mathcal{S}_0$ represents a set of initial states. In our case we have $\mathcal{S}_0 = \{s_0\}$. A set of states reachable in at most one step is given by

$$\mathcal{S}_1 = \mathcal{S}_0 \cup \{s' \mid \exists a \exists s [a \in \Sigma \wedge s \in \mathcal{S}_0 \wedge \delta(s,a) = s']\} \quad (4)$$

After finding states reachable in at most one step we search for those reachable in at most two steps. In general, a set of states reachable in at most i steps is represented by

$$\mathcal{S}_i = \mathcal{S}_{i-1} \cup \{s' \mid \exists a \exists s [a \in \Sigma \wedge s \in \mathcal{S}_{i-1} \wedge \delta(s,a) = s']\} \quad (5)$$

We continue with this procedure until in a step k no new state is reached. In any case, this happens sooner or later, because we deal with FSMs, where the set of states $\mathcal{S}$ is finite. Then, $\mathcal{S}_k = \mathcal{S}_{k-1}$ is a set of all reachable states.

Introducing the characteristic functions for the sets (e.g., the $\mathcal{S}_i$ is represented by its characteristic function $S_i = \chi_{\mathcal{S}_i}$), we can rewrite (5) into the following logical expression:

$$S_i' = S_{i-1}' + \exists_{y \in \mathcal{Y} \cup \mathcal{X}} (S_{i-1} \cdot T) \quad (6)$$

The function S_i' depends on next state variables y_j'. To obtain S_i we have to replace each y_j' with the corresponding y_j. Therewith, next states are considered to be new present states for the next step. The procedure terminates in a step k, when $S_k = S_{k-1}$.

The most time and space consuming operation in (6) is the computation of $\exists_{y \in \mathcal{Y} \cup \mathcal{X}} (S_{i-1} \cdot T)$. The source of problems is usually very large monolithic transition relation T. In order to avoid this problem, we do not build a single BDD for the whole transition

relation T, but rather partition T in some groups of transition relations of individual state variables and represent each group by a smaller BDD. T is then called a *partitioned transition relation* [3][9][5].

Formula (6) can be written in an extended form as

$$S_i' = S_{i-1}' + \exists x_0 \exists x_1 \ldots \exists x_{m-1} \exists y_0 \exists y_1 \ldots \ldots \exists y_{n-1} (S_{i-1} \cdot T_0 \cdot T_1 \cdot \ldots \cdot T_{n-1}) \quad (7)$$

Although existential quantification does not distribute over conjunction, conjuncts can be moved out of the scope of an existential quantification if they do not depend on any of the variables being quantified. FSMs that represent circuit behaviour exhibit locality, so it is very common that many of the T_j depend on only small number of the input and state variables.

We developed a heuristic algorithm for the partitioning and reordering of functions and variables for existential quantification in (7) that allows us to apply some existential quantifiers before the BDD for the whole transition relation has been built. The function S_{i-1} in (7) represents a set of present states in step $i-1$. Therefore, it may depend on all state variables y_i. So we leave it within the scope of all existential quantifiers of state variables y_j. In contrast to the function S_{i-1}, functions T_j are constant throughout the computation and at the beginning we can determine their place with respect to the existential quantifiers in (7) once and for all.

The dependencies of functions T_j on individual input and state variables are represented by matrix

$$\mathbf{A} = \begin{bmatrix} a_{0,0} & \cdots & a_{0,n-1} \\ \vdots & \ddots & \vdots \\ a_{n-1,0} & \cdots & a_{n-1,n-1} \\ a_{n,0} & \cdots & a_{n,n-1} \\ \vdots & \ddots & \vdots \\ a_{n+m-1,0} & \cdots & a_{n+m-1,n-1} \end{bmatrix} \begin{matrix} \leftarrow & y_0 \\ & \vdots \\ \leftarrow & y_{n-1} \\ \leftarrow & x_0 \\ & \vdots \\ \leftarrow & x_{m-1} \end{matrix}$$
$$\begin{matrix} \uparrow & \cdots & \uparrow \\ T_0 & \cdots & T_{n-1} \end{matrix}$$

which is of size $(m+n) \times n$ and $a_{i,j}$ has value 1 if and only if the function T_j depends on the variable that points to row i from the right side, and value 0 otherwise. First, we fill the matrix with 0s and 1s. Then we choose one of the rows (there might be several) with minimal number of 1s and thus one of the variables that the least number of the functions depend on. We put into the scope of its existential quantifier all the functions that depend on this variable. We will be no more concerned with these functions during the partitioning procedure. Therefore, the selected row and all the columns whose functions were taken are no more needed and we delete them. The whole procedure is then repeated over the reduced matrix as long as we do not run out of functions and variables.

A group of functions which find themselves together in one step of this procedure is called a *block*. We denote it T_{ρ_i} if we got it in the i-th step and ρ_i represents the i-th block of partition ρ over the set $\{0,1,\ldots,n-1\}$. Accordingly, the block T_{ρ_i} can be written as $T_{\rho_i} = \bigwedge_{j \in \rho_i} T_j$. Let us denote a set of variables whose existential quantifiers are immediately in front of block T_{ρ_i} by $\mathcal{Y}_{\rho_i}$, where

$$\mathcal{Y}_{\rho_i} = \{v \mid v \in \mathcal{Y} \cup X \wedge \forall j \in \rho_i : T_j \text{ depends on } v \wedge \forall j \notin \rho_i : T_j \text{ does not depend on } v\} \quad (8)$$

Thus, using partitioned transition relation we got the following formula for a general step in computing reachable states:

$$S_i' = S_{i-1}' + \exists_{y \in \mathcal{Y}_{\rho k-1}} \left(T_{\rho_{k-1}} \cdot \exists_{y \in \mathcal{Y}_{\rho k-2}} \left(T_{\rho_{k-2}} \cdot \ldots \cdot \exists_{y \in \mathcal{Y}_{\rho_0}} (T_{\rho_0} \cdot S_{i-1}) \ldots \right) \right) \quad (9)$$

The algorithm based on (9) has proved itself very well. Just in few cases it was considerably worse (see the example of an up/down counter in Section 6) than the method with monolithic transition relation. However, in many cases it has been shown to be better for many orders of magnitude.

4 Symbolic Model Checking

The logic that we use to specify properties of FSMs is a propositional termporal logic of branching time, called *Computation Tree Logic* — CTL [4]. In this logic each of the usual future time operators of linear temporal logic (**G** — *globally* or *invariantly*, **F** — *sometimes in the future*, **X** — *next time*, **U** — *until*) must be immediately preceded by *path quantifier* **A** (for all computation paths) or **E** (for some computational path). We thus obtain eight different CTL operators: **AG**, **EG**, **AF**, **EF**, **AX**, **EX**, **AU**, **EU**.

CTL formulas are constructed from *atomic propositions* using Boolean connectives and CTL opera-

tors. In our case of verifying FSMs, the set of atomic propositions is equal to the set $\mathcal{Y}$ of state variables of the circuit. If y_j is an atomic proposition in $\mathcal{Y}$, the formula y_j is true in a state s (we say that s satisfies y_j and write $s \models y_j$ if and only if the state variable y_j is true in this state.

The formula $\mathbf{AG}\, f$, for example, will hold in a state provided that f holds in all states along all possible computation paths starting from that state, and $\mathbf{EF}\, f$ will hold in a state provided that there is a computation path from that state to the future state where f holds [1][9][5]. The meaning of all CTL operators is shown in Figure 1 in the form of computation trees. In the figure, full circle represents a state where formula f is true. In states that are represented by full square, formula g is true. We know nothing about the truthfulness of formulas in all other states (empty circles).

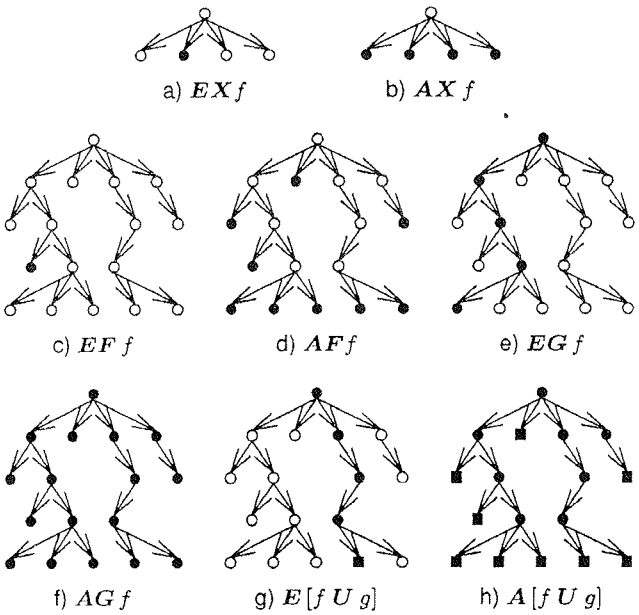

Figure 1: Meaning of CTL operators

Model checking is the problem of determining whether a given CTL formula f is true in a given FSM M. The formula f is true in the FSM M if and only if f is true in all initial states of M. We can say that FSM M is a *model* of f ($M \models f$).

In this section, we present an efficient symbolic model checking algorithm for CTL that recurses over the structure of a given CTL formula f and determines in a bottom up fashion in which states of the model the subformulas of f are true. Finally, it returns the BDD (the function) S that represents exactly the set of states $\mathcal{S}$ of the FSM in which f is true and checks whether $\mathcal{S}_0 \subseteq \mathcal{S}$.

The CTL formulas constructed only from the atomic propositions (state variables) and ordinary Boolean connectives are handled using standard algorithms for computing Boolean connectives with BDDs. The temporal formulas are treated otherwise. Although there are eight different CTL operators, it is sufficient to realize only three of them, since the remaining five can be easily expressed with the three realized. We decided to realize the operators $\mathbf{EX}$, $\mathbf{EU}$, and $\mathbf{EG}$ [7][1].

First, let us have a look at the formula $\mathbf{EX}\, f$, where f represents a characteristic function $S_f = f$ of the set of states $\mathcal{S}_f$ where it is true. Formula $\mathbf{EX}\, f$ is true in all predecessors of states in $\mathcal{S}_f$. The characteristic function of these states is computed by

$$\mathbf{EX}\, f = \exists_{y' \in \mathcal{Y}' \cup X} \left(f' \cdot T\right) \qquad (10)$$

where f' represents the function f with each state variable y_i substituted by the corresponding next state variable y_i' and T is a transition relation of a given FSM. The relational product (10) is quite similar to the relational product (6) defined in Section 3 except that in (10) we compute one step in a backward reachability search instead of a forward reachability search.

Formula $\mathbf{E}[f\ \mathbf{U}\ g]$ means that either g is true in the current state, or f is true in the current state and there exists a successor state where $\mathbf{E}[f\ \mathbf{U}\ g]$ is true. Consequently, the BDD that represents the states where $\mathbf{E}[f\ \mathbf{U}\ g]$ is true can be computed by finding the least fixed point of the predicate transformer F defined by

$$F(S) = g + f \cdot \mathbf{EX}\, S \qquad (11)$$

Formula $\mathbf{EG}\, f$ means that f is true in the current state and there exists a successor state where $\mathbf{EG}\, f$ is true. The BDD that represents the states where $\mathbf{EG}\, f$ is true is computed as the greatest fixed point of the predicate transformer

$$F(S) = f \cdot \mathbf{EX}\, S \qquad (12)$$

After determining the set $\mathcal{S}$ of states that satisfy a given CTL formula f, the algorithm checks

whether this set includes all initial states S_0 ($S_0 \subseteq S$). Checking the condition $S_0 \subseteq S$ is easy in our BDD-based algorithm. It has to be checked whether the Boolean expression $\overline{S_0} + S \Leftrightarrow 1$ is a tautology.

5 Fairness Constraints

Many times we want to check if a given property is true in a given FSM only along the *fair paths* in a computation tree. For example, at checking of a bus arbiter we might be interested only in those paths where none of the devices which the bus was granted to holds the bus to the eternity.[1]

Such properties can not be expressed directly by CTL. To solve the problem, the meaning of CTL should be changed. Therefore, we introduce *fairness constraints*, which are ordinary CTL formulas. A path in the computation tree is *fair* regarding to a set of fairness constraints if every constraint is true *infinitely often* along the path. Path quantifiers in CTL formulas are then constrained to these fair paths [8].

Let the set of CTL formulas $\mathcal{H} = \{h_1, h_2, \ldots, h_r\}$ be fairness constraints. Let us have a look at solving formula $\mathbf{EG}\, f$ with fairness constraints $\mathcal{H}$ (we write $\mathbf{EG}_{\text{fair}}\, f$). Formula f is again characteristic function of set of states where formula f is true, just like with ordinary CTL formulas. Formula $\mathbf{EG}_{\text{fair}}\, f$ says that there exists a path in the computation tree, where f is invariantly true and every formula in $\mathcal{H}$ is true infinitely often. The set of such states S is the largest of the sets for which the following two properties are true:

- f is true in every state from S and
- for every fairness constraint $h_k \in \mathcal{H}$ and every state $s \in S$, there exists a path of length one or more from state s to some state in S, where constraint h_k is true and f is true in every state along that path.

It could be proved that every state in the set S is the beginning of an infinite path in the computation tree, where formula f is always true and every constraint in $\mathcal{H}$ is true infinitely often. By searching for the greatest fixed point of the predicate transformer

$$F(S) = f \cdot \prod_{k=1}^{r} \mathbf{EX}\, \mathbf{E}[f\, \mathbf{U}\, S \cdot h_k] \qquad (13)$$

where CTL operators $\mathbf{EX}$ and $\mathbf{EU}$ are resolved as ordinary CTL operators without considering the fairness constraints, we get a characteristic function of the set of states where formula $\mathbf{EG}_{\text{fair}}\, f$ is true considering the fairness constraints. The fixed point is evaluated in the same manner as without fairness constraints, but note that each computation of the above expression leads to several nested fixed point computations (resolving of operators $\mathbf{EU}$).

Resolving of operators $\mathbf{EX}$ and $\mathbf{EU}$ with fairness constraints is simpler. Characteristic function of the set of states that can be starting states of fair paths in the computation tree is defined as

$$S_{\text{fair}} = \mathbf{EG}_{\text{fair}}\, 1 \qquad (14)$$

There exists a path from every state in this set along which every fairness constraint in $\mathcal{H}$ is true infinitely often. Formula $\mathbf{EX}_{\text{fair}}\, f$ is true in state s if and only if there exists a successor state s' of s such that f is true in s' and s' is the beginning of a fair path. Therefore, the formula $\mathbf{EX}_{\text{fair}}\, f$ considering only fair paths is equal to the formula $\mathbf{EX}(f \cdot S_{\text{fair}})$ over all paths. Thus, let us define

$$\mathbf{EX}_{\text{fair}}\, f = \mathbf{EX}(f \cdot S_{\text{fair}}) \qquad (15)$$

This way we transform the problem from dealing with fair paths to considering all paths in the computation tree, and that is something what we already know how to handle with.

Following analogy the formula $\mathbf{E}[f\, \mathbf{U}\, g]_{\text{fair}}$ is equal to the formula $\mathbf{E}[f\, \mathbf{U}\, g \cdot S_{\text{fair}}]$ considering all computation paths. Therefore, we define

$$\mathbf{E}[f\, \mathbf{U}\, g]_{\text{fair}} = \mathbf{E}[f\, \mathbf{U}\, g \cdot S_{\text{fair}}] \qquad (16)$$

and once again the problem is transformed into something we already know how to solve.

5.1 Example

To illustrate symbolic CTL model checking using fairness constraints we took a parametric bus arbiter with n devices (n is parameter) which grants a bus to the device with the highest priority [11]. An instance of such an arbiter for $n = 3$ is shown in Figure 2. All others are constructed in a similar way. Devices with lower number have higher priority.

We want to know if formulas

$$\mathbf{EF}\, \mathbf{EG}(IN_i \cdot \overline{OUT_i}) \qquad (17)$$

[1] This example shows us, where the name of fair paths comes from. It would be unfair from a device if it held the bus to the eternity and had never released the bus and left it to other devices. By excluding such paths in some way we stay only at fair paths.

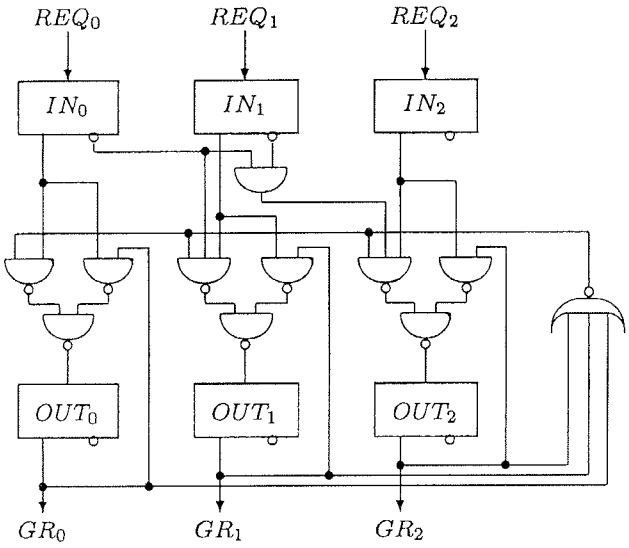

Figure 2: A 3-request bus arbiter

are true, in other words, if there exists a path from the initial state, where at some point device i $(i = 0,1,\ldots,n-1)$ would request a bus and bus would never be granted to it. In general case without any fairness constraints the formulas are true, because every other device could hold the bus for itself and never release it again.

Because such behaviour could be described as unfair, we will try to constrain all possible paths to those where the bus is free infinitely often along the path. This is written by fairness constraint

$$\overline{OUT_0 + OUT_1 + \ldots + OUT_{n-1}} \qquad (18)$$

in the formal language. It could be shown that formula $\mathbf{EF}\,\mathbf{EG}\left(IN_0 \cdot \overline{OUT_0}\right)$ is not true any more. If the device 0 requests a bus and it insists on that request, the bus will be eventually granted to it. All other formulas are still true, because every other device might request the bus, but the bus would never be granted to it. Actually, this is the expected behaviour, since fairness constraint (18) has eliminated only those paths where some device would hold the bus to the eternity. Because nothing stops a device to request a bus again immediately after device has released it, there is a possibility that the bus is always granted to device 0 as device with the highest priority. In that case the bus will never be granted to any other device.

The next level of fairness is achieved by preventing the device 0 such aggressive behaviour in spite of its highest priority. It should be required that infinitely often, after the bus is released, it is not granted to device 0. That is, infinitely often along the path we must reach a state where the bus is released (fairness constraint (18) is true) and in all possible successor states the bus is not granted to device 0. Therefore, formula $\mathbf{AX}\,\overline{OUT_0}$ will be true in that state, too. This leads us to new fairness constraint

$$\overline{OUT_0 + OUT_1 + \ldots + OUT_{n-1}} \cdot \mathbf{AX}\,\overline{OUT_0} \qquad (19)$$

Beside the formula for device 0, now also formula $\mathbf{EF}\,\mathbf{EG}\left(IN_1 \cdot \overline{OUT_1}\right)$ for device 1 is false. Since devices 0 and 1 can interchange in possessing the bus, all other devices can stay without the bus forever. So, other formulas are still true.

To prevent interchanging of the bus holding between two most prioritised devices, the fairness constraint should be extended. Infinitely many times it should happen that after the bus was released it is neither granted to the device 0 *nor* to the device 1. So, we get fairness constraint

$$\overline{OUT_0 + OUT_1 + \ldots + OUT_{n-1}} \cdot \mathbf{AX}\,\overline{OUT_0 + OUT_1} \qquad (20)$$

Now device 2 also gets its time slots where the bus is granted to it. Its CTL formula is now also false, but all others for $i > 2$ are still true.

In the same manner we could proceed, what leads us to fairness constraint

$$\overline{OUT_0 + OUT_1 + \ldots + OUT_{n-1}} \cdot \mathbf{AX}\,\overline{OUT_0 + OUT_1 + \ldots + OUT_{k-1}} \quad \text{for } 0 < k < n \qquad (21)$$

where all CTL formulas from (17) with index $i \le k$ are false, while all others are true.[2]

If a device i, $i = 0,1,\ldots,k$, requests the bus and it insists on that request, the bus will eventually be granted to it. All other devices could stay without the bus to the eternity.

Well, if $n-1$ is chosen for k, we get as fair system as a priority system could be. Not only that none of the devices may hold the bus forever, but also the bus will be granted to each and every device from time to time.

6 Experimental results

Experiments were done on a *Sun Ultra 30 creator* workstation with 300 MHz UltraSPARC II processor, 256 MB of physical memory and 793 MB of virtual memory under *Solaris 2.5.1* operating system. We have used our own BDD package ([6]) that is an efficient *ite*-based implementation of reduced ordered binary decision diagrams with complemented edges.

[2] If $k = 0$, formula (21) becomes formula (18).

Table 1: Computing reachable states for ISCAS'89 circuits and a parametric up/down counter

circuit	# states	# steps	monolithic TR		partitioned TR	
			# nodes	time	# nodes	time
s344	2625	7	11966	0.13	19462	0.19
s349	2625	7	11966	0.14	19462	0.20
s382	8865	151	48580	1.98	46411	0.86
s420.1	65536	65535	278572	7.67	282476	35.86
s444	8865	151	13660	0.14	20863	0.22
s526	8868	151	19610	0.18	36560	0.36
s526n	8868	151	19608	0.18	36558	0.36
s641	1544	7	1299567	170.27	23649	0.36
s713	1544	7	1299622	173.09	23673	0.37
s953	504	11	37187	0.32	11987	0.16
s1196	2616	3	236067	29.87	28788	0.42
s1238	2616	3	236086	29.86	28810	0.40
count3	8	4	66	0.00	56	0.00
count6	64	32	598	0.00	1005	0.00
count9	512	256	3922	0.04	11111	0.11
count12	4096	2048	28558	0.42	44905	1.31
count15	32768	16384	85708	4.32	88920	13.91
count18	262144	131072	427938	36.78	433475	141.48
count21	2097152	1048576	3180807	414.54	3185888	1773.71

Table 1 shows the results obtained by symbolic searching of reachable states on some ISCAS'89 circuits and on the parametric up/down counters. From left to right the columns in Table 1 refer to the circuit name, the number of reachable states, the number of steps needed to compute all reachable states, and the maximal number of BDD nodes whenever generated during the symbolic state space traversal of a given circuit together with the CPU time in seconds both for the search using monolithic transition relation and partitioned transition relation.

Great power of the symbolic state space traversal is demonstrated in the next example. Table 2 shows results on computing reachable states for the previously described parametric arbiter using partitioned transition relation. Although the number of reachable states grows with n extremely fast, the number of BDD nodes and CPU time grow much slower.

Table 2: Computing reachable states for a parametric arbiter

# dev. n	# state variab.	# reachable states	# steps	# BDD nodes	time [s]
100	200	$1.28032710 \cdot 10^{32}$	3	66443	2.39
200	400	$3.22994546 \cdot 10^{62}$	3	203180	13.52
300	600	$6.13147828 \cdot 10^{92}$	3	454743	37.60
400	800	$1.03548220 \cdot 10^{123}$	3	806369	80.66
500	1000	$1.63996869 \cdot 10^{153}$	3	1257932	148.41
600	1200	$2.49385885 \cdot 10^{183}$	3	1809558	246.24
700	1400	$3.68735526 \cdot 10^{213}$	3	2461121	386.41
800	1600	$5.34107956 \cdot 10^{243}$	3	3212747	570.30
900	1800	$7.61589396 \cdot 10^{273}$	3	4064373	809.61
1000	2000	$1.07258011 \cdot 10^{304}$	3	5015936	1100.22

We verified the behavior of the parametric arbiter with symbolic model checking in CTL. The properties being verified together with the corresponding CTL formulas are as follows:

- *Exclusivity.* At most one output is set to '1'. In order to decrease the complexity of the CTL formula we weaken the exclusivity and rather write (as in [11]) that either odd number of outputs is set to '1' or all outputs are set to '0'.

$$\mathbf{AG}\left((OUT_0 \oplus OUT_1 \oplus \ldots \oplus OUT_{n-1}) + \overline{OUT_0} \cdot \overline{OUT_1} \cdot \ldots \cdot \overline{OUT_{n-1}}\right)$$

- *Causality.* If device i $(i = 0,1,\ldots,n-1)$ requests the bus, the access will eventually be granted to device i.

$$\mathbf{AG}\left(IN_i \Rightarrow \mathbf{EF}\, OUT_i\right)$$

- *Starvation.* There is a possibility that a request from device i is never serviced. This behavior will happen if another device being granted the bus never releases its request.

$$\mathbf{EG}\left(IN_i \Rightarrow \mathbf{EG}\, \overline{OUT_i}\right)$$

- Allocation. If a request from device i $(i = 0,1,\ldots,n-1)$ is loaded and no higher priority device requests the bus, then the bus is granted to device i at the next state.

$$\mathbf{AG}\left(\overline{OUT_0} \cdot \ldots \cdot \overline{OUT_{n-1}} \cdot IN_0 \Rightarrow \mathbf{AX}\, OUT_0\right)$$
$$\mathbf{AG}\left(\overline{OUT_0} \cdot \ldots \cdot \overline{OUT_{n-1}} \cdot \overline{IN_0} \cdot IN_1 \Rightarrow \mathbf{AX}\, OUT_1\right)$$
$$\vdots$$
$$\mathbf{AG}\left(\overline{OUT_0} \cdot \ldots \cdot \overline{OUT_{n-1}} \cdot \overline{IN_0} \cdot \ldots \cdot \overline{IN_{n-2}} \cdot IN_{n-1} \Rightarrow \mathbf{AX}\, OUT_{n-1}\right)$$

In Table 3 we collected the results obtained from the verification of the parametric bus arbiter.[3] The columns from left to right represent the number of devices (n), the number of state variables (latches) of the circuit (n), the number of CTL formulas being verified for each n $(3n+1)$, and the maximal number of BDD nodes whenever generated during symbolic model checking of a given circuit together with the CPU time in seconds both for the model checking using monolithic transition relation and partitioned transition relation. An overflow on BDD nodes is denoted by a dash '—'.

We did not perform a lot of experiments with fairness constraints. Well, checking of CTL formulas (17) for $n = 4$ with fairness constraint (21) for $k = 3$ took less than $\frac{1}{100}$ of a second.

7 Conclusions

We reviewed two different algorithms for searching of reachable states in FSMs, using either monolithic transition relation or partitioned transition relation. The introduction of partitioned transition relation may decrease the CPU times for large circuits dra-

[3] Although an arbiter with $n = 1$ is a totally senseless circuit, it is included in Table 3 for the sake of completeness. Anyway, a model checking algorithm does not care whether it handles useful or senseless circuits.

Table 3: Checking of parametric arbiter in CTL

# dev. n	# state var.	# CTL form.	monolithic TR # BDD nodes	monolithic TR time [s]	partitioned TR # BDD nodes	partitioned TR time [s]
1	2	4	18	0.00	11	0.00
2	4	7	174	0.00	76	0.00
3	6	10	947	0.01	298	0.01
4	8	13	4403	0.10	929	0.01
5	10	16	19171	0.64	2299	0.01
6	12	19	78044	4.63	7540	0.11
7	14	22	297614	38.37	15095	0.23
8	16	25	1214392	291.79	40247	0.98
9	18	28	4912359	2019.25	40301	1.77
10	20	31	—	—	100861	9.09
11	22	34	—	—	136629	15.55
12	24	37	—	—	616346	100.33
13	26	40	—	—	745319	184.85
14	28	43	—	—	3827900	1534.16
15	30	46	—	—	4303558	3603.01

matically. Then, we showed methods for verifying synchronous sequential circuits by symbolic model checking [7][9]. Properties to be verified were expressed in CTL.

We illustrated the usage of fairness constraints by verifying a parametric bus arbiter which selects the highest priority device from n devices. We checked if it is possible that some device requests a bus and the bus is never granted to it regarding different fairness constraints.

All algorithms are realized as part of a home-made package for manipulating FSMs which is also based on a fully home-made very efficient package for manipulating Boolean functions represented by BDDs [10].

Our future work will consist in the implementation of searching for counterexamples for false properties and then of automatic generation of testing sequences for digital circuits based on found counterexamples.

8 References

/1/ Zmago Brezočnik, Aleš Časar, and Tatjana Kapus. Efficient Symbolic Traversal Algorithms using Partitioned Transition Relations. In Zmago Brezočnik and Tatjana Kapus, editors, Proceedings of the COST 247 International Workshop on Applied Formal Methods in System Design, pages 146-155, Maribor, Slovenia, June 1996.

/2/ Randal E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. IEEE Transactions on Computers, C-35(8):677-691, August 1986.

/3/ Jerry R. Burch, Edmund M. Clarke, David E. Long, Kenneth L. McMillan, and David L. Dill. Symbolic Model Checking for Sequential Circuit Verification. IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems, 13(4):401-424, April 1994.

/4/ E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic Verification of Finite-State Concurrent Systems using Temporal Logic Specifications. ACM Transactions on Programming Languages and Systems, 8(2):244-263, April 1986.

/5/ Aleš Časar. Verification of Finite State Machines with Symbolic Model Checking. Master's thesis. University of Maribor, Faculty of Electrical Engineering and Computer Science, Maribor, Slovenia, June 1998. In Slovene.

/6/ Aleš Časar and Zmago Brezočnik. Symbolic State Space Traversal of Finite State Automata. In Franc Solina and Baldomir Zajc, editors, Proceedings of the Fourth Electrotechnical and Computer Science Conference ERK'95, volume A, pages 85-88, Portorož, Slovenia, September 1995. In Slovene.

/7/ Aleš Časar, Zmago Brezočnik, and Tatjana Kapus. Exploiting Partitioned Transition Relations for Efficient Symbolic Model Checking in CTL. In Penny Storms, editor, Proceedings of the European Design & Test Conference ED&TC'96, pages 606-606, Paris, France, March 1996. IEEE Computer Society Press.

/8/ Aleš Časar, Zmago Brezočnik, and Tatjana Kapus. Fairness Constraints in Symbolic CTL Model Checking. In Baldomir Zajc and Franc Solina, editors, Proceedings of the seventh Electrotechnical and Computer Science Conference ERK'98, volume B, pages 39-42, Portorož, Slovenia, September 1998. In Slovene.

/9/ Aleš Časar, Zmago Brezočnik, and Tatjana Kapus. Symbolic Model Checking of Finite State Machines with CTL. In Baldomir Zajc and Franc Solina, editors, Proceedings of the fifth Electrotechnical and Computer Science Conference ERK'96, volume A, pages 51-54, Portorož, Slovenia, September 1996. In Slovene.

/10/ Aleš Časar, Robert Meolic, Zmago Brezočnik, and Bogomir Horvat. Representation of Boolean Functions with ROBDDs. Electrotechnical Review, 59(5):299-307, December 1992. In Slovene.

/11/ David Déharbe and Dominique Borrione. Symbolic Model Checking of VHDL Design Entities. Technical report, Atelier de Recherche sur les Techniques Mathématiques et Informatiques des Systemes, Grenoble, France, November 1993.

/12/ Aarti Gupta. Formal Hardware Verification Methods: A survey. Formal Methods in System Design, 1(2/3):151-238, October 1992.

mag. Aleš Časar, univ. dipl. inž. rač.
izr. prof. dr. Zmago Brezočnik, univ. dipl. inž. el.
izr. prof. dr. Tatjana Kapus, univ. dipl. inž. el.
Univerza v Mariboru
Fakulteta za elektrotehniko, računalništvo in informatiko
Smetanova 17
2000 Maribor
tel.: +386-2-22-07-211
fax: +386-2-25-11-178
email: {casar,brezocnik,kapus}@uni-mb.si

Prispelo (Arrived): 10.5.00 *Sprejeto (Accepted): 30.8.00*