CTL may be ambiguous when model checking Moore machines

Cédric Roux[1] and Emmanuelle Encrenaz[1]

UPMC – LIP6 – ASIM
12, rue Cuvier, 75252 Paris CEDEX 5 – France
{Cedric.Roux, Emmanuelle.Encrenaz}@lip6.fr

Abstract. The model checking problem is defined over Kripke structures. However, hardware designers often handle other models, such as Moore machines. When model checking their designs using CTL as a logic, they must translate them into Kripke structures. A given CTL property may be true over the derived Kripke structure when using one translation scheme and false when using another one. This may lead to ambiguities if the designer does not fully understand the translation scheme he uses, which may be the case if he uses automatic tools. We give an example illustrating this, then present iCTL, a logic specifically designed to work with Moore machines, which extends CTL to help the designer removing possible ambiguities when model checking Moore machines. We show it is strictly more expressive than CTL. We give symbolic model checking algorithms for it.

1 Introduction

While developing a symbolic model checker to verify hardware systems described as a composition of synchronous Moore machines, we came across an interesting problem. We use CTL as logic and the formulae we want to verify may include values of input signals of the Moore machines. These input signals do label the transitions of the Moore machine. Since CTL is defined over Kripke structures and not Moore machines, and because the transitions of Kripke structures are not labelled, when translating a Moore machine into a Kripke structure, one has to integrate the input signals in the states of the Kripke structure. Several choices are possible. Depending on the translation chosen, the truth value of a given property may either be true or false over the derived Kripke structure. This introduces an ambiguity that the designer must be aware of when verifying his designs. He has to know how his model is translated into the one used by the model checker, and has to write properties with this in mind, so not to get confused by the answer of the tool. Not doing so could even lead to a counter-intuitive situation, where the designer might view his model as being buggy where in fact he simply wrote wrong formulae, thinking them over Moore machines and not over the derived Kripke structures.

In [7] the authors translate a Moore machine into a Kripke structure by incorporating the input configurations in the source state of the transitions.

And they define the truth value of a CTL property over a Moore machine as being the truth value of this property over the Kripke structure. We present an example illustrating ambiguities appearing in this situation.

In SMV [13], one directly writes Kripke structures and CTL formulae over these structures. It is possible to create Moore machines by not constraining the initial value of some variables neither constraining their next value. These variables may then be seen as input signals. This leads to exactly the same situation as [7].

The VIS model checker [1] accepts, among others, systems described in a Verilog subset, in which collections of Moore machines can be represented. It supports modularity and the concept of input and output signals is present. However, an input signal can appear in a CTL formula only if it is declared of type `reg`, which forces its assignment in guarded blocks. As a consequence, depending on the way this assignment is done, input signals of a Moore machine will be included into the source or target state of the transitions in the Kripke structure, which influences the results of the verification of a given formula.

The purpose of this article is to suggest to add two new operators to CTL to bring together the intuitive idea one can have regarding the truth value of a formula over the Moore machine and the truth value obtained by the verification algorithm over the Kripke structure. These two operators are specifically designed to handle Kripke structures derived from Moore machines and are meaningless in other cases. We hope that their utilization will facilitate the writing of formulae and the understanding of the results produced by the model checker.

The paper is organized as follows. The section 2 presents our target model, which is a collection of synchronously composed Moore machines. The section 3 defines the syntax and semantics of CTL, the logic we use. The section 4 presents different ways to translate a Moore machine into a Kripke structure. The section 5 introduces an example which was the leading stone to define iCTL. We present iCTL in the section 6, and show the interest of the new operators with some other examples ; then we prove that iCTL is more expressive than CTL. Section 7 presents the basic symbolic model checking data structures and algorithms defined for CTL and shows how to adapt them to take into account the new operators. The conclusion appears in section 8.

2 The target model

The models we want to verify are represented as a collection of Moore machines, synchronously composed. This model is sufficient to represent synchronous hardware designs (without asynchronous signals as resets).

Moreover, VHDL programs at delta cycle description level can be seen as a collection of synchronous Moore machines [9, 10, 12].

Each machine is defined by a six tuple as follows.

Definition 1. *A Moore machine is a structure* $\langle S, S_0, I, O, \mathcal{L}, R \rangle$ *where*

1. *S is a finite set of states,*
2. *$S_0 \subseteq S$ is the set of initial states,*
3. *I is the finite set of input symbols,*
4. *O is the finite set of output symbols (we define $n_O = |O|$),*
5. *$\mathcal{L} = \{l_0, \ldots, l_{n_O-1}\}$ is a vector of n_O functions, each function defining the value of exactly one output symbol; for all $0 \le i \le n_O - 1$ we have $l_i : S \to \mathbb{B}$; for all $s \in S$, we have that $l_i(s)$ will be true iff the output symbol associated to l_i is true in the state s,*
6. *$R \subseteq S \times 2^I \times S$ is the transition relation.*

2^I is the set of boolean vectors of size $|I|$. Each of its components represents the value of one input signal. Each element of 2^I is an input configuration.

The Moore machines we handle are complete and deterministic. Complete means that each state has one successor for any input configuration. Deterministic means that for a given input configuration, a state s will always lead to the same state s'.

Their composition is defined as:

Definition 2. *The composition of two Moore machines $M_1 = \langle S_1, S_{1_0}, I_1, O_1, \mathcal{L}_1, R_1 \rangle$, and $M_2 = \langle S_2, S_{2_0}, I_2, O_2, \mathcal{L}_2, R_2 \rangle$, is the Moore machine $M = \langle S, S_0, I, O, \mathcal{L}, R \rangle$, defined by:*

1. *$S \subseteq S_1 \times S_2$,*
2. *$S_0 = S_{1_0} \times S_{2_0}$,*
3. *$I = [I_1 \setminus (O_2 \cap I_1)] \cup [I_2 \setminus (O_1 \cap I_2)]$,*
4. *$O = O_1 \cup O_2$,*
5. *$\mathcal{L} = \{l'_{1_0}, \ldots, l'_{1_{n_{O_1}-1}}\} \cdot \{l'_{2_0}, \ldots, l'_{2_{n_{O_2}-1}}\}$; for all $0 \le i \le n_{O_1} - 1$, we have $l'_{1_i} : S \to \mathbb{B}$; for all i, for all $s = (s_1, s_2) \in S$, we have that $l'_{1_i}(s)$ is true iff $l_{1_i}(s_1)$ is true; for all $0 \le i \le n_{O_2} - 1$, we have $l'_{2_i} : S \to \mathbb{B}$; for all $s = (s_1, s_2) \in S$, we have that $l'_{2_i}(s)$ is true iff $l_{2_i}(s_2)$ is true;*
6. *$R \subseteq S \times 2^I \times S$, such as $\forall x_1 \in S_1, \forall x_2 \in S_2, \forall x'_1 \in S_1, \forall x'_2 \in S_2, \forall c_I \in 2^I$, we have $((x_1, x_2), c_I, (x'_1, x'_2)) \in R$ iff $(x_1, c_{I1}, x'_1) \in R_1$ and $(x_2, c_{I2}, x'_2) \in R_2$ where $c_{I1} \in 2^{I_1}$, $c_{I2} \in 2^{I_2}$ and each component of the vector c_{I1} in $I_1 \setminus (O_2 \cap I_1)$ is the same as the corresponding one in c_I and each component of c_{I1} in $O_2 \cap I_1$ is defined by its value in x_2 (same thing for c_{I2}).*

We must have that $O_1 \cap O_2 = \emptyset$.

3 The logic – CTL

CTL [5, 6] is one of the most widely used logic in the hardware design community since symbolic model checking techniques overcome others temporal logic verification methods due to the use of BDDs [2].

CTL is defined over Kripke structures. We first define Kripke structures, then present the syntax and semantics of CTL over these structures.

3.1 Kripke structures

Definition 3. *A Kripke structure is a five tuple* $\langle S, S_0, P, \mathcal{L}, R\rangle$ *where*

1. S *is a finite set of states,*
2. $S_0 \subseteq S$ *is the set of initial states,*
3. P *is a finite set of atomic propositions (we define* $n_P = |P|$*),*
4. $\mathcal{L} = \{l_0, \ldots, l_{n_P-1}\}$ *is a vector of* n_P *functions, each function defining the value of exactly one atomic proposition; for all* $0 \leq i \leq n_P - 1$ *we have* $l_i : S \rightarrow \mathbb{B}$*; for all* $s \in S$*, we have that* $l_i(s)$ *is true iff the atomic proposition associated to* l_i *is true in* s*,*
5. $R \subseteq S \times S$ *is the transition relation.*

Informally, a Kripke structure is simply a transition system, with states and transitions. States are labelled by atomic propositions, which may be true or false in each state (but not true *and* false, of course). A Kripke structure is used to model a system. States of the model are represented by states of the Kripke structure. Evolution from one state to another one of the system is represented by the transitions of the Kripke structure. In this model, a computation is an infinite path $\sigma = s_0, s_1, \ldots, s_i, s_{i+1}, \ldots$ such that $\forall i \geq 0, (s_i, s_{i+1}) \in R$.

As a remark, transitions in Kripke structures are not labelled, whereas transitions in Moore machines are.

3.2 Syntax and semantics of CTL

The syntax A CTL formula is built upon atomic propositions. We distinguish state and path formulae. State formulae are true in a specific state, while path formulae are true along a specific path.

The syntax of state formulae is the following one :

- if p is an atomic proposition, then p is a state formula,
- if f and g are state formulae, then $\neg f$, $f \vee g$, and $f \wedge g$ are state formulae,
- if f is a path formula, then $\mathbf{A}\, f$ and $\mathbf{E}\, f$ are state formulae.

And for path formulae :

- if f and g are state formulae, then $\mathbf{X}\, f$, $\mathbf{F}\, f$, $\mathbf{G}\, f$, $f\,\mathbf{U}\, g$ and $f\,\mathbf{R}\, g$ are path formulae.

Valid CTL formulae are all the state formulae that can be obtained by using the above rules.

The semantics The CTL semantics is defined over Kripke structures. Let's take a Kripke structure M defined by $\langle S, S_0, P, \mathcal{L}, R\rangle$. Our Kripke structure must be total (i.e., each state has at least one successor). A path in M is an infinite sequence of states, $\sigma = s_0, s_1, \ldots$ such that for every $i \geq 0$, $(s_i, s_{i+1}) \in R$. σ^i will be the suffix of a path σ starting at state s_i. If f is a state formula, the notation $M, s \models f$ means that f holds at state s in the Kripke structure M. If f is a path formula, then $M, \sigma \models f$ means that f holds along path σ in the Kripke structure M. (In the following we assume that f_1 and f_2 are state formulae and g_1 and g_2 are path formulae.)

A Kripke structure M validates a formula f if and only if for all $s_0 \in S_0$, we have $M, s_0 \models f$.

We have :

$M, s \models p$	$\Leftrightarrow$ if $l_j \in \mathcal{L}$ is the function associated to the atomic property p, then we have that $l_j(s)$ is true
$M, s \models \neg f_1$	$\Leftrightarrow M, s \not\models f_1$
$M, s \models f_1 \vee f_2$	$\Leftrightarrow M, s \models f_1$ or $M, s \models f_2$
$M, s \models f_1 \wedge f_2$	$\Leftrightarrow M, s \models f_1$ and $M, s \models f_2$
$M, s \models \mathbf{E}\ g_1$	$\Leftrightarrow$ there is a path σ starting from s, such that $M, \sigma \models g_1$
$M, s \models \mathbf{A}\ g_1$	$\Leftrightarrow$ for every path σ starting from s, $M, \sigma \models g_1$
$M, \sigma \models \mathbf{X}\ f_1$	$\Leftrightarrow M, s_1 \models f_1$
$M, \sigma \models \mathbf{F}\ f_1$	$\Leftrightarrow$ there exists $k \geq 0$ such that $M, s_k \models f_1$
$M, \sigma \models \mathbf{G}\ f_1$	$\Leftrightarrow$ for all $k \geq 0$, $M, s_k \models f_1$
$M, \sigma \models f_1\ \mathbf{U}\ f_2$	$\Leftrightarrow$ there exists a $k \geq 0$ such that $M, s_k \models f_2$ and for all $0 \leq i < k$, $M, s_i \models f_1$
$M, \sigma \models f_1\ \mathbf{R}\ f_2$	$\Leftrightarrow$ for all $j \geq 0$, if for every $i < j$ $M, s_i \not\models f_1$ then $M, s_j \models f_2$

We can express any CTL formula with only $\neg$, $\vee$, **EX**, **EG** and **EU** [6].

4 Translating a Moore machine into a Kripke structure

Kripke structures are the basic model for CTL model checking. But often, a hardware engineer doesn't directly create Kripke structures. In our case, we are interested in Moore machines. So, we have to translate a Moore machine into a Kripke structure in order to apply available model checking algorithms. Several choices are possible. We present three of them.

Figure 1 is a trivial Moore machine which purpose will be to illustrate the different translations discussed here.

4.1 Removing inputs

The main difference between Moore machines and Kripke structures is that transitions are labelled into Moore machines and not into Kripke structures.

So, a basic translation of a Moore machine into a Kripke structure is to remove these labels. In other words, we "forget" the input symbols.

Formally, we have the following.

Definition 4. Translating a Moore machine by removing the inputs *Given a Moore machine* $\langle S_M,\ S_{M0}, I_M,\ O_M,\ \mathcal{L}_M, R_M\rangle$*, we deduce the Kripke structure* $\langle S_K, S_{K0}, P_K, \mathcal{L}_K, R_K\rangle$ *where:*

- $S_K = S_M$,

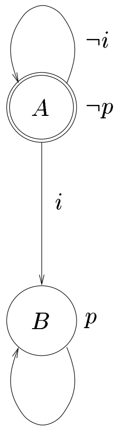

Fig. 1: A little Moore machine. A and B are the states, i is an input, p is an output. A is the initial state.

- $S_{K0} = S_{M0}$,
- $P_K = O_M$,
- $\mathcal{L}_K = \mathcal{L}_M$,
- *for any* $s_1 \in S_K$, $s_2 \in S_K$, $(s_1, s_2) \in R_K$ *if and only if there exist* $c_i \in 2^{I_M}$ *such that* $(s_1, c_i, s_2) \in R_M$.

On the Moore machine of figure 1, this translation would produce the Kripke structure of figure 2.

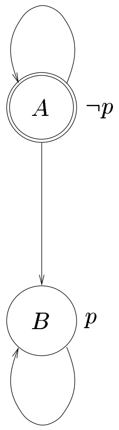

Fig. 2: The Kripke structure corresponding to the Moore machine of figure 1 (inputs are removed).

This approach has the advantage of simplicity, since we don't introduce new states, thus a property expressed only in term of outputs and states will straightforwardly be verified into the Kripke structure. But we can't adopt it, since we need to be able to express properties about the input configurations that the Moore machine will encounter.

So, we absolutely need to keep the input signals. As model checking handles Kripke structures, with labelled states and non-labelled transitions, the input signal must be put into the states.

Several possibilities arise. By starting from the states of the Moore machines, we have two choices. We can put the inputs in the target state of the transition or in the source state.

4.2 Putting inputs in the target state

Informally, if in the Moore machine, a state s_2 is target of n transitions labelled with respectively $c_1, \ldots, c_n$ input configurations, we create n corresponding states in the Kripke structure named $s_2.c_1, \ldots, s_2.c_n$. If in the Moore machine, there exists a transition from s_1 to s_2 labelled by c_i, then there exist transitions from all the states derived from s_1 to $s_2.c_i$ in the Kripke structure. An illustration of this is the figure 3.

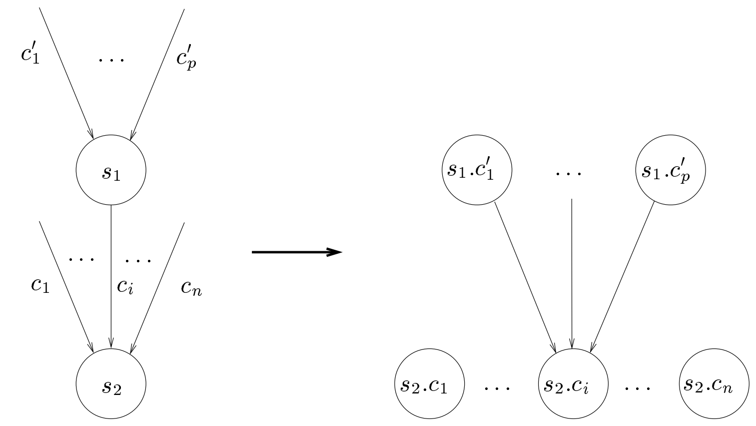

Fig. 3: Relation between Moore machines and Kripke structures when inputs go in target state

The formal definition of this translation comes below.

Definition 5. Translating a Moore machine by putting the inputs in the target state *Given a Moore machine* $\langle S_M, S_{M0}, I_M, O_M, \mathcal{L}_M, R_M \rangle$*, we deduce the Kripke structure* $\langle S_K, S_{K0}, P_K, \mathcal{L}_K, R_K \rangle$ *where:*

– $S_K = S_M \times 2^{I_M}$,

- $S_{K0} = S_{M0} \times 2^{I_M}$,
- $P_K = I_M \cup O_M$ *(we define* $n_{I_M} = |I_M|$ *and* $n_{O_M} = |O_M|$*),*
- $\mathcal{L}_K = \{l_{O0}, \ldots, l_{On_{O_M}-1}\} \cdot \{l_{I0}, \ldots, l_{In_{I_M}-1}\}$*; for all* $0 \leq i \leq n_{O_M} - 1$*, we have* $l_{Oi} : S_K \to \mathbb{B}$*; for all* i*, for all* $s = (s_1, c_1) \in S_K$*, we have that* $l_{Oi}(s)$ *is true iff* $l_{Mi}(s_1)$ *is true; for all* $0 \leq i \leq n_{I_M} - 1$*, we have* $l_{Ii} : S_K \to \mathbb{B}$ *(each* l_{Ii} *is associated to one and only one input signal); for all* i*, for all* $s = (s_1, c_1) \in S_K$*, we have that* $l_{Ii}(s)$ *is true iff the component of* c_1 *corresponding to the input signal associated to* l_{Ii} *is true,*
- $R_K \subseteq S_K \times S_K$ *and* $\forall (s, c_i) \in S_K, \forall (s', c'_i) \in S_K$*, we have* $((s, c_i), (s', c'_i)) \in R_K$ *iff* $(s, c'_i, s') \in R_M$*.*

On our example, this translation would lead to the Kripke structure of figure 4.

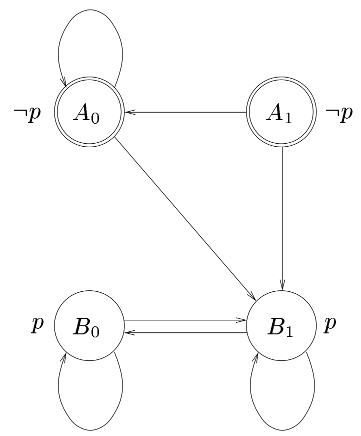

Fig. 4: The Kripke structure corresponding to the Moore machine of figure 1 (inputs get in the target state). In each state, the upper case letter represents the original state and the indice represents the value of signal i.

The symbolic relation is straightforwardly obtained by memorizing each input by adding dedicated registers.

This second translation was abandoned because we plan to do compositional model checking over synchronous Moore machines, hence some outputs of a machine M_0 will be inputs of machine M_1 and conversely. For a machine with primary inputs and inputs coming from other machines, all these inputs have to be taken into account at the same instant. As the outputs are associated to the source state of a transition, and some outputs may be inputs, this imposes the presence of the inputs in the source state.

4.3 Putting inputs in the source state

Informally, if in the Moore machine, a state s_1 is source of n transitions labelled with respectively $c_1, \ldots, c_n$ input configurations, we create n corresponding states in the Kripke structure named $s_1.c_1, \ldots, s_1.c_n$. If in the Moore machine, there exists a transition from s_1 to s_2 labelled by c_i, then there exists transitions from $s_1.c_i$ to all the states derived from s_2 in the Kripke structure, as illustrated by figure 5.

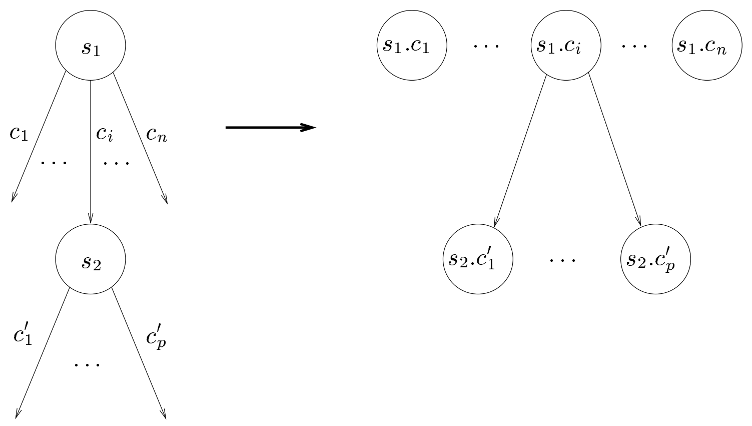

Fig. 5: Relation between Moore and Kripke when inputs go in source states

This approach is the one adopted by [7]. We obtain the following formal definition.

Definition 6. Translating a Moore machine by putting the input in the source state *Given a Moore machine $\langle S_M, S_{M0}, I_M, O_M, \mathcal{L}_M, R_M\rangle$, we deduce the Kripke structure $\langle S_K, S_{K0}, P_K, \mathcal{L}_K, R_K\rangle$ where:*

- *S_K, S_{K0}, P_K and $\mathcal{L}_K$ are the same as in definition 5, and*
- *$R_K \subseteq S_K \times S_K$ and $\forall\,(s, c_i) \in S_K, \forall\,(s', c'_i) \in S_K$, we have $((s, c_i), (s', c'_i)) \in R_K$ iff $(s, c_i, s') \in R_M$.*

On the same example as before, this would give us the Kripke structure of figure 6.

We define the *brotherhood*, which links all the states in a Kripke structure obtained from a unique state in the Moore machine. Formally, we have the following.

Definition 7. *Given a Kripke structure $K = \langle S_K, S_{K0}, P_K, \mathcal{L}_K, R_K\rangle$ obtained from the Moore machine $M = \langle S_M,\ S_{M0}, I_M, O_M,\ \mathcal{L}_M, R_M\rangle$ using the translation of definition 6, we define the relation $B : S_K \times S_K \to \mathbb{B}$. For all $s = (s_1, c_1) \in S_K$, for all $t = (t_1, c_2) \in S_K$, we have that $(s, t) \in B$ iff $s_1 = t_1$. The*

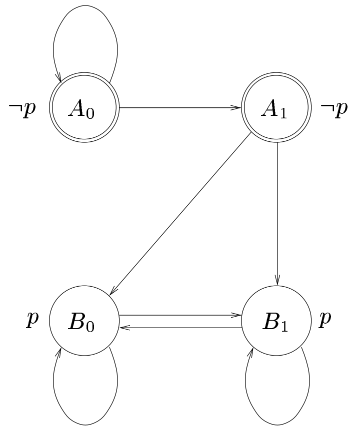

Fig. 6: The Kripke structure corresponding to the Moore machine of figure 1 (inputs get in the source state).

brotherhood *is the function* $B_r : S_K \to \mathcal{P}(S_K)$*, which, given a state* s*, returns the set of all the states* t*, such that* $(s,t) \in B$*.*

Given a state $s_m \in S_M$, we will denote the brotherhood of s_m, $B_r(s_m)$, which should be understood as $B_r((s_m, c_i))$, for some c_i.

The translation of definition 6 is the one we adopted.

A fourth translation may be though of, which consists of creating intermediate states, labelled by inputs. We don't take it into account, since this introduction of intermediate states complicates the definition of the synchronized product and the verification algorithms.

5 A disturbing example

We could simply state that a CTL formula is true in a Moore machine if and only if it is true in the corresponding Kripke structure as done in [7] but the verification results obtained may disturb the designer.

As an illustration, we propose to check the CTL property $(\mathbf{EX}\ p) \wedge (\mathbf{EX}\ \neg p)$ over the Moore machine depicted on figure 1.

Using the first translation for Moore machines, the formula is true on the corresponding Kripke structure of figure 2, as it is true in the state A, because there exists a successor, A, in which $\neg p$ holds, and there exists a successor, B, in which p holds.

This formula is true on the Kripke structure of figure 4 because it is true in A_0 and in A_1, as each of them has a successor where $\neg p$ holds and another one where p holds.

This formula is false on the Kripke structure depicted in figure 6, because neither A_0 nor A_1 has a successor verifying $\neg p$ and a successor verifying p.

In fact, the formula $(\mathbf{EX}\, p) \wedge (\mathbf{EX}\, \neg p)$ is ambiguous over the Moore machine: do we mean that both successors are selected by the same input configuration or by different input configurations?

If what we say over the Moore machine is *"for a given input configuration, there is a successor in which $\neg p$ holds and (for the same input configuration) there is another one in which p holds"* then the model checker must return `false` whereas if what we say over the Moore machine is *"for an input configuration, there is a successor in which $\neg p$ holds and for another input configuration, there is another one in which p holds"* then the model checker must return `true`.

To remove the ambiguity, we incorporate two quantifier modalities "there exists an input configuration for which" and "for all input configurations, we have" defined for states of the Kripke structure (derived from Moore machines; these operators are meaningless with other Kripke structures).

Intuitively, their meaning may be seen as follows. Let $X = \{(s_1, c_1), (s_1, c_2), (s_2, c_1), (s_2, c_2)\}$ be a set of states in the Kripke structure derived from a Moore machine with two states (s_1 and s_2) and one input signal which will lead to two input configurations (c_1 and c_2). Let's have a property p, false in (s_2, c_1), true in all the other states. *"there exists an input configuration for which p holds"* will be true for all the states of X, because it is true on (s_1, c_1), so it will be true for any (s_1, c_x) (the brotherhood of s_1), and it is true on the brotherhood of s_2. *"for all input configurations, we have p"* will be true only for the brotherhood of s_1, because p is false on (s_2, c_1), thus *"for all input configuration…"* cannot be true on the brotherhood of s_2.

On our example of figure 1, we would have the following. Let's write $P_1 =$ *"for a given input configuration, there is a successor in which $\neg p$ holds and (for the same input configuration) there is another one in which p holds"* and $P_2 =$ *"for an input configuration, there is a successor in which $\neg p$ holds and for another input configuration, there is another one in which p holds"*.

On the Kripke structure of figure 2, we can't verify neither P_1 nor P_2 because *"for a given input configuration"* has no meaning, as inputs simply have been removed.

On the Kripke structure of figure 4, P_1 is false, because for one input configuration, we can't have both p and $\neg p$ to hold in the same successor, and this for any of the four available states. P_2 is true on both A_0 and A_1 (they both have as successor A_0 on input configuration $\neg i$ and B_1 on input configuration i), so the Kripke structure validates the formula.

On the Kripke structure of figure 6, P_1 is false, for the same reason as above. *"there is a successor in which $\neg p$ holds"* is true on A_0, so *"for an input configuration, there is a successor in which $\neg p$ holds"* will be true on both A_0 and A_1, as we "existentially abstract" the inputs, so if a property P is true for any of the A_x state, then the property *"for an input configuration, P"* will be true on all the A_x. *"there is another one in which p holds"* is true on A_1, so for the same reason, we will have that *"for another input configuration, there is another one in which p holds"* is true on both A_0 and A_1. So the conjunction of these two sub-properties will be true on both A_0 and A_1, so P_2 is true on the Kripke structure of figure 6.

We see, on this example, that independently of the transition scheme adopted, the properties P1 and P2 have the same truth value over the different Kripke structures obtained.

6 *i*CTL – CTL model checking with input configurations

We introduce two new operators to CTL. Those two operators are $\forall_I$ and $\exists_I$. This defines a new logic, that we call *i*CTL. Given ϕ, an *i*CTL formula (that may contain $\forall_I$ and $\exists_I$ operators), $\forall_I \phi$ stands for "for any input configuration, ϕ holds" and $\exists_I \phi$ stands for "there is an input configuration for which ϕ holds".

Here follows the formal definition of *i*CTL.

6.1 Syntax and semantics of *i*CTL

The syntax is the same as the one of CTL, with the following added rule for state formulae.

- if f is a state formula, then $\forall_I f$ and $\exists_I f$ are state formulae.

The semantics remains the same, with the following added rules.

As the two new operators deal with input configurations, the Kripke structure they apply on are the ones given by our translation from Moore machines. The symbols are thus the same than those from definition 6.

$$M, s \models \forall_I f \Leftrightarrow s = (s_M, c_M) \text{ and for all } c'_M \in 2^{I_M},\ s' = (s_M, c'_M) \text{ and we have that } s' \models f,$$
$$M, s \models \exists_I f \Leftrightarrow s = (s_M, c_M) \text{ and for one } c'_M \in 2^{I_M},\ s' = (s_M, c'_M) \text{ and we have that } s' \models f.$$

Since our Moore machines are complete, for all input configurations, the state s' exists in the Kripke structure, thus $s' \models f$ is sound.

Using *i*CTL, we now can define when a Moore machine validates a logical formula.

Definition 8. *A Moore machine M validates a formula f of iCTL if and only if the formula is true in the corresponding Kripke structure, as given by the transformation of definition 6.*

This definition is the same as in [7], but we expect the designer to remove the ambiguities of CTL by using $\exists_I$ and $\forall_I$ in the places where they are needed.

6.2 Examples

The formula ($\mathbf{EX}\ p$) $\wedge$ ($\mathbf{EX}\ \neg p$) is built upon a conjunction of two sub-formulae. We will now illustrate the use of $\exists_I$ and $\forall_I$ with formulae containing nested temporal operators.

The Moore machine of figure 7 will be used as example.

On the Kripke structure derived from it by the translation of definition 6 (putting the input signals in the source state of transitions), we've got that the formula $\mathbf{AX}\ \mathbf{EX}\ p$ is false in the brotherhood $B_r(s_1)$. If we use the translation of definition 5 (putting the input signals in the target state of transitions), we've got that $\mathbf{AX}\ \mathbf{EX}\ p$ is true in $B_r(s_1)$. Looking at the Moore machine, one might think that this formula is true, since all the successors of s_1 have a successor where p is true (states s_4 and s_6). The formula $\mathbf{AX}\ (\exists_I\ (\mathbf{EX}\ p))$ is true in $B_r(s_1)$, using translation of definition 6. This corresponds to the intuition one might have about the truth value of $\mathbf{AX}\ \mathbf{EX}\ p$ over the Moore machine. We see here that to capture this intuition, $\exists_I$ is necessary.

The same Moore machine can illustrate the use of $\forall_I$. The formula $\mathbf{EX}\ \mathbf{AX}\ p$ is true in $B_r(s_1)$ in the Kripke structure obtained from definition 6. On the opposite, the formula $\mathbf{EX}\ (\forall_I(\mathbf{AX}\ p))$ is false in $B_r(s_1)$. This latest interpretation seems to be consistent with the intuition that one might have for the truth value of $\mathbf{EX}\ \mathbf{AX}\ p$ in the state s_1 of the Moore machine. To capture this intuition, $\forall_I$ is necessary.

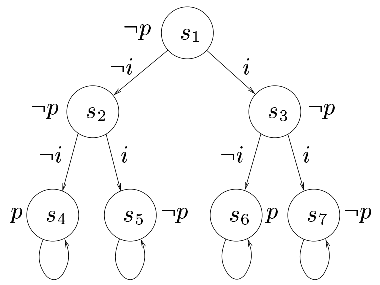

Fig. 7: A Moore machine illustrating the use of $\exists_I$ and $\forall_I$

One interesting point to note is that the ambiguity due to the input configurations is present even with formulae that don't express properties including input configurations.

As a remark, if the engineer wants to use his intuition over the Moore machine when writing iCTL formulae, he simply can think of writing CTL formulae, and each time he writes an $\mathbf{E}$ (conversely $\mathbf{A}$) followed by anything, he would indeed write $\exists_I\ \mathbf{E}$ (conversely $\forall_I\ \mathbf{A}$). An automatic tool might do it for the engineer. But as this semantic is an intuitive one, some people might have another intuition, so to put it in an automatic tool might not be such a good idea.

6.3 iCTL is more expressive than CTL

Given a formula $f \in i$CTL and a formula $g \in$ CTL, we say that the formula f is equivalent to the formula g if and only if for any Kripke structure K derived from a Moore machine M using the translation of definition 6, for any state s of the Kripke structure, we have that $K, s \models f$ iff $K, s \models g$.

In the general case, a formula $f \in i$CTL has no equivalent in CTL, and since each CTL formula is an iCTL formula, this leads us to say that iCTL is more expressive (over the Kripke structures obtained by the translation of definition 6) than CTL.

To prove this, let's take the Moore machine and its derived Kripke structure shown in the figure 8.

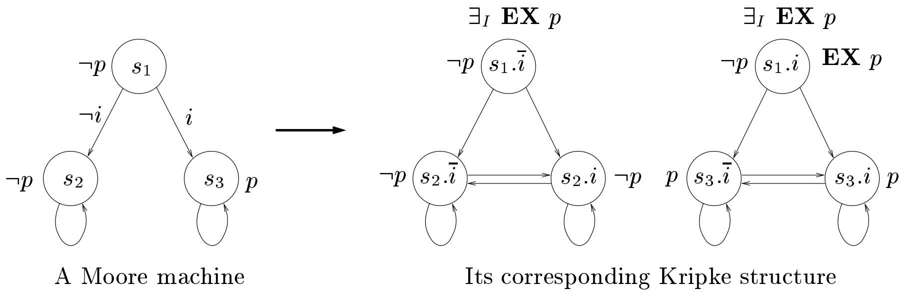

Fig. 8: An example showing the better expressiveness of iCTL

On the Kripke structure, we have that the iCTL property "$\exists_I$ **EX** p" is true in the states $s_1.\bar{i}$ and $s_1.i$, although "**EX** p" is true in $s_1.i$ and false in $s_1.\bar{i}$. If we change the labelling of the states $s_3.\bar{i}$ and $s_3.i$ from p to $\neg p$, we have that "$\exists_I$ **EX** p" is no longer true in $s_1.\bar{i}$ nor $s_1.i$.

If there existed a CTL formula f equivalent to the iCTL formula "$\exists_I$ **EX** p", it would have to have the same truth value for any Kripke structure (as translated from a Moore machine by the definition 6) and any state of this Kripke structure. Thus, when "$\exists_I$ **EX** p" would have been true in the states $s_1.\bar{i}$ and $s_1.i$ of the Kripke structure of figure 8, f would also have to be true, and when "$\exists_I$ **EX** p" would have been false, f would also have to be false.

On the Kripke structure of figure 8, we see that the truth value of "$\exists_I$ **EX** p" in the state $s_1.\bar{i}$ depends of the validity of "**EX** p" in the brotherhood of s_1. If "**EX** p" is true in $s_1.i$, then "$\exists_I$ **EX** p" is true in the brotherhood of s_1; if "**EX** p" is false in all the states of the brotherhood of s_1, then "$\exists_I$ **EX** p" is false in the brotherhood of s_1. But the states in a brotherhood may not be connected by any path, as $s_1.\bar{i}$ and $s_1.i$ in our example. If we change the truth value of "**EX** p" in $s_1.i$ (by changing the labelling of $s_3.\bar{i}$ and $s_3.i$ from p to $\neg p$), this will change the truth value of "$\exists_I$ **EX** p" is $s_1.\bar{i}$. But this change of labelling in $s_3.\bar{i}$ and $s_3.i$ cannot have any repercussion on the truth value of any CTL formula in $s_1.\bar{i}$, since CTL only captures the transition relation between states and not the brotherhood.

We now formally prove that there is no CTL formula f equivalent to "$\exists_I$ **EX** p" on the Kripke structure of figure 8. This can be shown by induction over the size of f. Obviously, if f is made of an atomic property, it won't have its truth value in $s_1.\bar{i}$, $s_2.\bar{i}$ and $s_2.i$ changed by changing the labelling of $s_3.\bar{i}$ and $s_3.i$. Now, let's suppose that any formula f (and g) in CTL of size $\leq n$ doesn't get its truth value in $s_1.\bar{i}$, $s_2.\bar{i}$ and $s_2.i$ changed by changing the labelling of $s_3.\bar{i}$ and $s_3.i$. Obviously $\neg f$, and $f \vee g$ won't have their truth value changed in $s_1.\bar{i}$, $s_2.\bar{i}$ and $s_2.i$, since we suppose by induction that neither f nor g has its truth value changed. **EX** f doesn't see its truth value changed, since, by induction, the truth value of f in $s_1.\bar{i}$, $s_2.\bar{i}$ and $s_2.i$ remains the same, hence the truth value of **EX** f is identical in their predecessor states, which are themselves. The same holds for **EG** f and **E**(f **U** g), since by induction, the sub-formulae f and g don't change, and all the paths starting from $s_1.\bar{i}$, $s_2.\bar{i}$ or $s_2.i$ are only made of the states $s_1.\bar{i}$, $s_2.\bar{i}$ and $s_2.i$, so the truth value of **EG** f and **E**(f **U** g) is not affected by modifying the truth value of p in $s_3.\bar{i}$ and $s_3.i$. As all CTL can be expressed with only the operators we have seen so far, the proof is over.

This proves that the iCTL formula "$\exists_I$ **EX** p'' has no CTL equivalent formula, thus that the power of expression of iCTL (over the Kripke structures derived from Moore machines by the translation of definition 6) is superior to the one of CTL.

One may want to compare iCTL with EQCTL [11] (CTL with existentially quantified atomic propositions). A first difference is that in EQCTL, only the $\exists$ exists, whereas in iCTL both $\exists_I$ and $\forall_I$ exist. A second difference is that $\exists$ in EQCTL is defined over atomic propositions appearing in the following CTL formula, whereas our $\exists_I$ is not defined over atomic propositions, but over internal data of states representing the input configuration. A third difference is that $\exists$ in EQCTL can only appear in front of a formula, whereas in iCTL, our abstractions can be positioned in front of any state sub-formula.

But as our goal was to use CTL, not EQCTL, we didn't investigate further the relative expressive power of EQCTL and iCTL. For our purpose, it is sufficient to know that iCTL is more expressive than CTL.

7 Symbolic model checking

7.1 Symbolic model checking for CTL

Symbolic model checking [4, 8, 13] is a technique designed to handle the main problem of model checking, which is the state explosion. When handling huge models, the number of reachable states of the system becomes very big (10^{20} states and beyond is not uncommon). It is based on the use of BDDs [2], which are data structures representing most of boolean formulae in a compact fashion. Some boolean formulae still require exponential space (for example, multiplication circuits [14]), but in many cases, the gain is sensitive. Using BDDs, we can manipulate set of states, and represent the transition relations of Kripke structures. Explicit model checking algorithms have been transposed to boolean manipulations, which are performed using BDDs.

Sections 7.1 and 7.1 are a short description of what may be found in [13].

Coding states and transition relations Given a Kripke structure $\langle S, S_0, P, \mathcal{L}, R\rangle$, where $|S| = n$, we create a set V of $n_S = \lceil log_2\ n \rceil$ boolean variables v_0, v_1, $\ldots$, v_{n_S-1} (and a set V' of n_S variables v'_0, v'_1, $\ldots$, v_{n_S-1}) to encode the current (and next) states. Each atomic property $p \in P$ gets a characteristic function $f_p : \mathbb{B}^{n_S} \rightarrow \mathbb{B}$ which represents the set of states (encoded with the V variables) where p holds. This function is expressible with a BDD. The transition relation will be represented by its characteristic function $f_R : \mathbb{B}^{n_S} \times \mathbb{B}^{n_S} \rightarrow \mathbb{B}$ which will take two states s and s' (s encoded with V and s' encoded with V') and return true iff there is a transition from s to s' in the Kripke structure (i.e. if $(s, s') \in R$). f_R can be represented using a BDD.

Algorithms for symbolic model checking Using the above encoding, we can devise the following algorithms to do model checking over BDDs. In the following, we will identify the elements of the Kripke structure with their BDD representatives.

The function `Eval` takes as a parameter a CTL property to check and returns the BDD representing the characteristic function of all the states that verify the CTL formula.

```
function Eval(p)
  case
    p: atomic prop ⇒ return bdd(p)
    p = ¬q         ⇒ return bdd_not(Eval(q))
    p = r ∨ s      ⇒ return bdd_or(Eval(r), Eval(s))
    p = EX q       ⇒ return EvalEX(Eval(q))
    p = EG q       ⇒ return EvalEG(Eval(q), true)
    p = EU (r, s)  ⇒ return EvalEU(Eval(r), Eval(s), false)
  end case
end
```

`bdd(p)` returns the bdd of the characteristic function of all the states where the atomic property p is true.

`EvalEX(b)` computes the characteristic function of the set of states having a successor in the set represented by `b`.

```
function EvalEX(b)
  return exist_abstract(R ∧ substitute(b, V, V'), V')
end
```

`exist_abstract` and `substitute` are BDD functions that respectively existentially abstract from the BDD all the variables given by a set (here V') and substitutes one set of variables in a BDD by the other (here V is substituted by V'). Algorithms exist to efficiently handle this relational product (conjunction followed by existential abstraction) [3].

`EvalEG` and `EvalEU` are fixpoints. Here come their algorithms.

```
function EvalEG(b,y)
  y' ← b ∧ EvalEX(y)
  if y = y' then
    return y
  else
    return EvalEG(b,y')
  end if
end

function EvalEU(b,c,y)
  y' ← c ∨ (b ∧ EvalEX(y))
  if y = y' then
    return y
  else
    return EvalEU(b,c,y')
  end if
end
```

We can express any CTL formula with only the symbols for which we just defined model checking functions. We can add other functions too, it's straightforward.

A CTL property p will be true on a Kripke structure if `Eval(`p`)` returns (the characteristic function of) a set containing S_0, the set of init states of the Kripke structure.

7.2 Symbolic model checking for iCTL

Our symbolic model checking algorithms have to handle input configurations and their quantifications.

States representation and transition functions From a Moore machine $M = \langle S_M, S_{M0}, I_M, O_M, \mathcal{L}_M, R_M \rangle$, we deduce the Kripke structure $K = \langle S_K, S_{K0}, P_K, \mathcal{L}_K, R_K \rangle$, as defined by the translation of definition 6.

As we will express properties about states, inputs or outputs, we must include this information in the set of states that the symbolic model checking algorithms will manipulate. So our sets of states will be sets of triplets (s, c_i, c_o) where $s \in S_M$, $c_i \in 2^{I_M}$ and $c_o \in 2^{O_M}$. We will have $n_S = \lceil log_2|S_M| \rceil$, $n_I = \lceil log_2|I_M| \rceil$, and $n_O = \lceil log_2|O_M| \rceil$ variables.

The transition function will be encoded the same way as before.

Algorithms The overall structure of the model checking algorithms remains the same. We have to handle $\forall_I$ and $\exists_I$. And the presence of inputs and outputs in the states imposes some modifications.

`Eval` is slightly modified. We simply add the treatment of $\forall_I$ and $\exists_I$. `univ_abstract` is a BDD function that universally abstracts from the BDD all the variables given by a set (which here, is the set of the inputs). `exist_abstract` does the same, but for existential abstraction[1].

The set of states returned by `Eval` may be constrained by the set of reachable states (assuming that the latest was previously computed).

function `Eval(`p`)`
  case
    p: state, input or output variable $\Rightarrow$ return `bdd(`p`)`
    $p = \neg q$ $\Rightarrow$ return `bdd_not(Eval(`q`))`
    $p = r \vee s$ $\Rightarrow$ return `bdd_or(Eval(`r`), Eval(`s`))`
    $p = \mathbf{EX}\ q$ $\Rightarrow$ return `EvalEX(Eval(`q`))`
    $p = \mathbf{EG}\ q$ $\Rightarrow$ return `EvalEG(Eval(`q`), true)`
    $p = \mathbf{EU}\ (r, s)$ $\Rightarrow$ return `EvalEU(Eval(`r`), Eval(`s`), false)`
    $p = \forall_I\ q$ $\Rightarrow$ return `univ_abstract(Eval(`q`), `I`)`
    $p = \exists_I\ q$ $\Rightarrow$ return `exist_abstract(Eval(`q`), `I`)`
  end case
end

`EvalEX` is a bit more complicated. Because the states are expressed as (s, c_i, c_o) and because the transition function is expressed as (s, c_i, s'), we first have to take away the i and o of the states, then encode s with the future variables, then conjunct it with the transition function, remove future variables, which will gives us states in term of current and input variables. Output variables must be added to this, which will be done by a conjunction with the functional definition of the outputs (named O_f in our algorithm).

function `EvalEX(b)`
  $r_1 \leftarrow$ `exist_abstract(b, `I`)`
  $r_2 \leftarrow$ `exist_abstract(`r_1`, `O`)`
  $r_3 \leftarrow$ `swap(`r_2`, `V`, `V'`)`
  $r_4 \leftarrow$ `bdd_and(`r_3`, `R`)`
  $r_5 \leftarrow$ `exist_abstract(`r_4`, `V'`)`
  return `bdd_and(`r_5`, `O_f`)`
end

`EvalEG` and `EvalEU` remain the same.

[1] Universally abstracting a function f from a variable v is to compute the function $f_{v\leftarrow 0} \wedge f_{v\leftarrow 1}$, where $f_{v\leftarrow 0}$ is the function f where $v = 0$. Having a set of variable instead of a variable is to iterate this process over the variables in the set. For the existential abstraction, we compute $f_{v\leftarrow 0} \vee f_{v\leftarrow 1}$.

8 Conclusion

The paper discusses the consequences of placing input configurations labelling transitions in Moore machines into the source or target states in the derived Kripke structure built to perform CTL symbolic model checking. The choice of such a placing is important since a given CTL formula may either be true or false with regard to the underlying Kripke structure. We provide an example illustrating this case. In the context of verification of a synchronous composition of Moore machines, the input configurations have to be incorporated in the source state of the transitions in the derived Kripke structure. In this model, some CTL properties that the intuition might believe to be true in the Moore machine are false in the Kripke structure. This is due to the lack of expressiveness of CTL that do not take into account labelled transitions, as Moore machines do. To overcome this ambiguity, we introduce two operators, $\exists_I$ and $\forall_I$. We show that the obtained logic, named iCTL, is more expressive than CTL. We present the main modifications introduced in the symbolic model checking algorithms. We have implemented these operators in our model checker and it is our intention to verify complex systems with this logic.

Further research may include the comparison of iCTL with other branching time logics. We also would like to see if the same problems arise when dealing with linear time logics. Another interesting development concerns the verification of Mealy machines.

References

1. 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 and T. Villa, *VIS: a System for Verification and Synthesis,* Proceedings of the Eighth International Conference on Computer Aided Verification CAV, pp. 428–432, 1996.
2. Randal E. Bryant, *Graph-Based Algorithms for Boolean Function Manipulation,* IEEE Transactions on Computers, 35(8):677–691, 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 Trans. Computer Aided Design, vol. 13, No. 4, pp. 401–424, 1994.
4. J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang, *Symbolic Model Checking:* 10^{20} *States and Beyond,* Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science, pp. 428–439, 1990.
5. E. Clarke and E. Emerson, *Design and synthesis of synchronization skeletons using branching-time temporal logic,* Proceedings of the Workshop on Logic of Programs, Yorktown Heights, D. Kozen, ed., vol. 131 of Lecture Notes in Computer Science, Springer-Verlag, pp. 52–71, 1981.
6. Edmund M. Clarke, E. Allen Emerson, and A. Prasad Sistla, *Automatic verification of finite-state concurrent systems using temporal logic specifications,* ACM Trans. on Programming Languages and Systems, 8(2):244–263, 1986.
7. E. M. Clarke, D. E. Long, and K. L. McMillan, *Compositional Model Checking,* Proceedings of the Fourth Annual IEEE Symposium on Logic in Computer Science, pp. 353–362, 1989.

8. O. Coudert, C. Berthet, and J. C. Madre, *Verification of Sequential Machines Using Boolean Functional Vectors,* IFIP International Workshop, Applied Formal Methods for Correct VLSI Design, Leuven, November 1989.
9. D. Deharbe and D. Borrione, *Semantics of a verification-oriented subset of VHDL,* Proceedings of CHARME, Lecture Notes in Computer Science 987, Springer-Verlag, pp. 293–310, 1995.
10. G. Döhmen, R. Herrmann, *A Deterministic Finite-State Model for VHDL,* in Formal Semantics for VHDL, Chapter 6, C. Delgado Kloos and P.T. Breuer editors, Kluwer Academic Publishers, ISBN: 0-7923-9552-2, 1995.
11. E.A. Emerson, *Temporal and modal logic,* in Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics, ed., J. van Leeuwen, chapter 16, pp. 995–1072, Elsevier, 1990.
12. E. Encrenaz, *A Symbolic Relation for a Subset of VHDL'87 Descriptions and its Application to Symbolic Model Checking,* Proceedings of CHARME, Lecture Notes in Computer Science 987, Springer-Verlag, pp. 328–342, 1995.
13. K. L. McMillan, *Symbolic Model Checking: an approach to the state explosion problem,* CMU Technical Report, CMU-CS-92-131, 1992.
14. P. Woelfel, *New Bounds on the OBDD-Size of Integer Multiplication via Universal Hashing*, Proceedings of the 18th Annual Symposium on Theoretical Aspects of Computer Science (STACS), Lecture Notes in Computer Science 2010, Springer-Verlag, pp. 563–574, 2001.