Using CTL formulae as component abstraction in a design and verification flow

Cécile Braunstein
Université Paris 6 - LIP6/SOC
CNRS UMR 7606
12 rue cuvier 75005 Paris France
Phone: +33 (0)1 44 27 70 16
Fax: +33 (0)1 44 27 72 80
cecile.braunstein@lip6.fr

Emmanuelle Encrenaz
Laboratoire Spécification et Vérification
CNRS UMR 8643
61, avenue du Pdt Wilson 94235 Cachan France
Phone: +33 (0)1 47 40 75 63
Fax : +33 (0)1 47 40 75 21
emmanuelle.encrenaz@lsv.ens-cachan.fr

Abstract

The verification of global properties (involving several components) is difficult to achieve, due to combinatorial explosion problem, while the verification of each component is easier to perform. Following the idea of [23], we propose to build an abstraction of a component already verified, starting from a subset of its specification described as CTL formulae. This abstraction replaces the concrete component in the context of global properties verification.

1. Introduction

This work takes place in the context of hardware modular verification by model checking ([3, 11]). Although this latest is not adequate to verify very complex systems, it has been successfully used for medium-sized systems. More precisely, model-checking techniques are well-suited for protocols verification. For instance, successful experiments are described in [9, 20, 14] where the specification is expressed in temporal logic. More recently, the idea of abstracting a component by a subset of its specification properties appears as a new method to alleviate the state space explosion problem. Xie and Browne in [23] proposed a compositional model checking process integrating this idea in the context of software engineering. Temporal properties (LTL) of a component are specified, verified and packed with the component. The whole system is then checked by using abstractions of all its components, each of which being built from already verified temporal properties and some environment assumption (also defined with temporal logic). Afterward, the abstraction refinement is performed with a classical counterexample guided abstraction refinement loop (CEGAR) [7]. But in [23], the details of the algorithm computing the abstraction is not given. Büttner [4] adopts a similar abstraction based on CTL properties in the context of synchronized module composition. Its abstract model of module is well suited to provide a cycle accurate abstraction to be used in micro-architecture verification. The abstraction is then synthesized into hardware and embedded in a simulation environment.

In the present paper, we define a component abstraction *based on the specification of the component*. Components are represented as Kripke structures, specification and environment assumption are expressed as CTL formulas. We propose an algorithm to build a component abstraction relatively to a subset of its specification. The abstraction is thus a direct translation from CTL formulas to an abstract Kripke structure. Our verification process is the following :

- Each component is already specified and its CTL formulae are verified.
- We choose a global property to be checked on the whole system (encompassing several components).
- For each component, we select a subset of its specification, useful for checking the global property.
- For each component, we compute a preliminary abstraction directly from CTL formulas included in its specification.
- We compose all component abstractions and obtain an abstraction of the whole system, that is the seed of a counter-example guided abstraction refinement process (CEGAR).
- After having performed the model checking, if the abstraction needs to be refined, we simply select a new formula from the specification of one (or several) components, add the corresponding abstraction, and re-run the verification.

The main contribution of this work is the definition of an algorithm building automatically abstraction from a subset of the component specification. Peng and Tahar [19] propose to synthesize tableau of ACTL formula ([11]) into Verilog description. Our work goes in the same way but our construction is based on the syntactic analysis of the formula (instead of the fixpoint definition of each subformula, as in Tableau construction). Furthermore, we do not restrict to the universal fragment of CTL, our algorithm synthesizes a structure for all CTL without the next operators.

Several works study the translation of temporal logic into automaton. Kupferman and al. [15] state a linear translation from branching temporal logic into alternating tree automatas. This induces an automata-based model checking algorithm of linear running time for CTL. More recently, Dams and Namjoshi [8] propose to use tree automatas as abstraction for all unwinding of a Kripke structure. Our goal is different : we want to obtain an abstraction that can be plugged into the platform under verification. The model of Abstract Kripke Structure (close to the $\mathcal{L}$-valued Kripke structure in [12]) is well-suited for such a purpose: we can combine concrete and abstract components in a unified way.

The abstract structure we obtain is obviously less precise than the concrete one. We need to represent less information. The idea is to use multi-valued logic as in [1] or [5, 12] to represent the lack of information due to the abstraction. In [1], Bruns and Godefroid assigned a third value for the signal that does not have a truth value **true** or **false** in the abstraction. They interpret it with the value "unknown" (or $\perp$). In our case, during abstraction of a component relative to some properties it verifies, the value of abstracted signals is interpreted as “don’t care”. During the verification of a global property (encompassing several abstracted components), the value of abstracted signal is interpreted as “unknown”. Furthermore, in our approach we do not need a fourth value to represent possible inconsistencies as described in [5]. Instead of inserting the third value of interpretation of atomic proposition in the abstract Kripke structure, we extend the set of atomic propositions and the associated labeling function.

The paper is organized as follows. First, we recall some preliminary definitions on Kripke structure and introduce the abstract Kripke structure model in Section 2. Then we describe, in Section 3, the algorithm which translates a CTL formula into an abstract Kripke structure and the composition of such structures. Section 4 states the link between CTL formulas, abstract and concrete structures. Finally, Section 5 studies the impact of the abstraction in the verification process of a system encompassing: Virtual Component Interface IP’s (VCI[17]), a PI-bus ([18]) and VCI-PI protocol converter.

2. Modeling Abstraction of Component

2.1 Modeling a component

Generally, a specification holds for a component when some assumption about the behaviour of the environment holds. Thus we model a component as a combination of a description of its behaviour, its specification and its assumptions about the environment.

Definition 1 *A* component *is a tuple $C = \langle K,P,A\rangle$ where K is a fair Kripke structure, P and A are disjoint sets of CTL formulas, the set of specifications and the set of assumptions respectively.*

We model the behaviour of one component, or of a composition of components as a *fair Kripke structure* [6].

Definition 2 *A fair Kripke structure is 6-tuple $K = \langle AP,S,S_0,\mathcal{L},R,F\rangle$ where : AP is a finite set of atomic propositions; $S_0 \subseteq S$ is the set of initial states; $\mathcal{L} : S \to 2^{AP}$ is the labeling function that associates each state with the set of atomic propositions that are* **true** *in that state; $R \subseteq S\times S$ is the transition relation : $\forall s \in S\ \exists s' \in s$ s. t. $R(s,s')$; $F \subseteq 2^S$ is a set of fairness constraints (generalized Büchi acceptance condition).*

In this paper the set of fairness constraints are generalized Büchi acceptance condition. This is due to our final goal : to integrate our abstraction in VIS [10]. The model checker VIS only accepts Büchi fairness constraints.

Let s and s' be in S, we write $s \to s'$ as an equivalent notation of $(s,s') \in R$. A *path* in K from s is an infinite sequence of states, $\pi = s_0, s_1, \ldots$ such that $s_0 = s$ and $\forall i\ s_i \to s_{i+1}$.

In this paper we restrict to the positive normal form of CTL\X formulas. Negations only apply to atomic propositions. We define the CTL fragment we consider as follows : p is an atomic proposition, φ,φ_1, φ_2 are state formulas and ψ is a path formula.

State formula : $\varphi ::= \mathbf{true} \mid p \mid \neg p \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid A\psi \mid E\psi$ with $p \in AP$

Path formula : $\psi ::= X\varphi \mid F\varphi \mid G\varphi \mid \varphi_1\, U\, \varphi_2 \mid \varphi_1\, W\, \varphi_2$

The semantic of CTL formulas is defined on the infinite execution tree obtained by unrolling the Kripke structure [6]. We have $K \models \varphi$ iff for all s_0 in S_0 $\langle K,s_0 \models \varphi\rangle$.

2.2 Abstract Kripke Structure

To model a component abstraction we need to represent two additional information about the truth value of the atomic proposition. The first one is *absence of knowledge* about the atomic proposition we do not care about the verification. The second one is *inconsistency* generated by the abstraction itself. Unlike [2] for 3-valued model checking , or [5] with 4-valued model-checking, we do not represent these values with new symbols. Instead, we extend the set of atomic proposition AP of a Kripke structure. We denote Lit the set of atomic propositions and *their negations*. In our abstract Kripke structure (AKS), for all p in Lit, we have $\overline{p}$ also in Lit. The labeling function is not over AP anymore but over Lit. Actually, we represent the knowledge of all atomic propositions in each state. This approach is similar to the *dual-rail encoding* used in asynchronous circuit design ([22]). The difference is that in our case the four values are used to represent a state information. Table 1 summarizes the information contained in each of the four values.

$p \notin \mathcal{L}(s) \wedge \overline{p} \notin \mathcal{L}(s)$	p is unknown in s
$p \notin \mathcal{L}(s) \wedge \overline{p} \in \mathcal{L}(s)$	p is false in s
$p \in \mathcal{L}(s) \wedge \overline{p} \notin \mathcal{L}(s)$	p is true in s
$p \in \mathcal{L}(s) \wedge \overline{p} \in \mathcal{L}(s)$	p is inconsistent in s

Table 1. Knowledge representation for an atomic proposition p in a state s

Definition 3 *An* Abstract Kripke Structure *(AKS) is a tuple $A = \langle AP, S, S_0, \mathcal{L}, R, F \rangle$. S, S_0, R, F are defined as above and $\mathcal{L} : S \rightarrow 2^{Lit}$, Lit is the union of the set of positive atomic propositions and the set of negative atomic propositions.*

We define an inconsistent state as a state where at least one atomic proposition p is true and $\overline{p}$ is also true.

Definition 4 *An* inconsistent state *s is a state where there exists $p \in AP$ such that $p \in \mathcal{L}(s)$ and $\overline{p} \in \mathcal{L}(s)$.*
An inconsistent path *$\overline{\pi}$ contains at least one inconsistent state.*

3. From CTL to an Abstract Kripke Structure

In this section, we present the construction of an AKS K_φ from a CTL formula φ (with the description described above). Then we prove that K_φ is a model of φ. Moreover, this abstracted Kripke structure is less constrained than any concrete component verifying φ.

3.1 Preliminary definitions

We need to characterize a function that extends the labeling function for a structure. During the construction of an AKS K' from an AKS K, this case may occur when we extend the set of atomic propositions, or when we extend the set of states. The two following definitions describe the extension of the labeling function $\mathcal{L}$ when increasing the set of atomic propositions, and when increasing the set of states.

Definition 5 Atomic proposition extension
Let $\mathcal{L}$ be a labeling function $\mathcal{L} : S \rightarrow 2^{Lit}$, let Lit' be a set of atomic propositions ($Lit' \cap Lit = \emptyset$). A labeling function $\mathcal{L}^{+Lit'} : S \rightarrow Lit' \cup Lit$ is defined such that :
$\forall p \in Lit, \forall s \in S\ \mathcal{L}^{+Lit'}(s) = \mathcal{L}(s)$.

The new atomic proprositions have an unkwown value, they do not belong to the labeling funcion of the state s.

Definition 6 State extension
Let $\mathcal{L} : S \rightarrow 2^{Lit}$ and $\mathcal{L}' : S' \rightarrow 2^{Lit}$ be two labeling functions related to the same set of atomic propositions. $\mathcal{L}'' = \mathcal{L} \cdot \mathcal{L}'$ is a labeling function related to Lit for each state $s \in S \cup S'$ such that :

$\mathcal{L}''(s) = \mathcal{L}(s)$ *iff* $s \in S$ *and* $s \notin S'$;

$\mathcal{L}''(s) = \mathcal{L}'(s)$ *iff* $s \in S'$ *and* $s \notin S$;

$\mathcal{L}''(s) = \mathcal{L}(s) \cup \mathcal{L}'(s)$ *iff* $s \in S \cap S'$.

We need two more operations for building the fairness constraints.

Definition 7 *Let $F = \{F_0, \ldots, F_n\}$ be a set of fairness constraints and let S be a set of states :*

- $F \oplus S = \{F_i \cup S, \forall F_i \in F\}$
- $F \ominus S = \{F_i \setminus S, \forall F_i \in F\}$

Synchronous composition is close to the parallel composition of [11] and is defined below. States of the composition are pair of component states. Paths leading to inconsistent states in the AKS have no equivalent path in the concrete Kripke structure, hence *inconsistent states are excluded*. The fairness constraints ensure that a path in $K_1 \parallel K_2$ is fair if and only if its projection to each component results in a fair path.

Definition 8 *Let $K_1 = \langle AP_1, S_1, S_{0_1}, L_1, R_1, F_1 \rangle$ and $K_2 = \langle AP_2, S_2, S_{0_2}, L_2, R_2, F_2 \rangle$ be two AKS. The* parallel composition *of K_1 and K_2 denoted $K_1 \parallel K_2$ is the structure K defined as follows :*

$AP = AP_1 \cup AP_2$

$S \subseteq (S_1 \times S_2) \setminus \{(s_1, s_2) \mid p \in \mathcal{L}((s_1, s_2))$ *and* $\overline{p} \in \mathcal{L}((s_1, s_2))\}$

$S_0 = (S_{0_1} \times S_{0_2})$

$\mathcal{L}((s_1, s_2) = \mathcal{L}_1(s_1) \cup \mathcal{L}_2(s_2)$

$R \subseteq (S_1 \times S_2) \times (S_1 \times S_2)$, *with* $R((s_1, s_2), (t_1, t_2))$ *iff* $R_1(s_1, t_1)$ *and* $R_2(s_2, t_2)$

$F \subseteq \{(P_1 \times S_2) \cap S \mid P_1 \in F_1\} \cup \{(S_1 \times P_2) \cap S \mid P_2 \in F_2\}$

3.2. Building AKS from CTL formulae

We define a function that directly builds an abstract Kripke structure from a CTL$\backslash X$ formula φ that is the less constrained structure where φ holds. We decompose our algorithm in three parts. The first one defines the translation from CTL$\backslash X$ formulas or sub-formulas with a universal quantifier as first quantifier. The second one concerns the formulas or sub-formulas with the existential quantifier as first quantifier. The third part sets some additional rules to combine universal and existential quantifier.

3.2.1 Universal quantifier

CTL formulas are rewritten as follows :

- $AG(f \wedge g)$ in $AG(f) \wedge AG(g)$;
- $A[(f \wedge g)Uh]$ in $A[fUh] \wedge A[gUh]$;

The first step of our algorithm builds the abstract structure of the formula **true**. This structure contains only one state where all formulas may be **true**. We define such special state named s_{TRUE}; it has got a unique outgoing self-loop transition. The execution tree obtained by unrolling transitions from s_T is an abstraction of any execution tree that has no impact on the validity of the formula.

Definition 9 *Let S_T be the set of special states s_T such that : $\forall s_T \in S_T$, $\mathcal{L}(s_T) = \emptyset$ and there exists an unique outgoing transition from s_T to itself.*

Lemma 1 *An abstract Kripke structure reduced to a state $s_T \in S_T$ simulates every non empty Kripke structure (with infinite behaviours).*

PROOF: Let K be any Kripke Structure with infinite behaviours. We define the relation $H = S_K \times S_T$. Obviously, H is a simulation relation since :

- $\mathcal{L}(s) \cap AP' = \mathcal{L}'(s')$ $(AP' = \emptyset)$ and
- for all s, s' such that $H(s,s')$ and for all t s.t $s \rightarrow t$ implies that there exists t' s.t $s' \rightarrow t'$ and $t' = s_T$ and $H(t,t')$.

■

A composed state (s,s') belongs to S_T if and only if s and s' belongs to S_T.

In an abstract Kripke structure obtained by our algorithm, a state s_T is reachable from a state s validating the formula in the initial state, whatever the future of s is. In states s_T the truth value of atomic propositions (and their negation) is undefined. These values can be interpreted as "don't care" values since they do not influence the satisfaction of the formula.

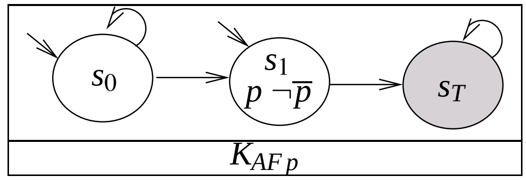

Figure 1. The AKS obtained for the CTL formula $AF\,p$. Initial states are s_0 and $_1$. The labeling function is : $\mathcal{L}(s_0) = \emptyset$; $\mathcal{L}(s_1) = \{p\}$. The fairness constraint is $\{s_T\}$.

Definition 10 *Let S_{predT} the set of predecessor state of a state in S_T : $\forall s \in S_{predT}$, $\exists s_T \in S_T$, such that $s \rightarrow s_T$.*

As an example, Fig. 1 presents the AKS obtained from the CTL formula $AF\,p$. It contains two initial states s_0 and s_1 and the state s_T. In state s_1 p is **true** then $AF\,p$ is **true**, the subsequent execution tree are not relevant for the evaluation of the formula. This is represented by the transition from s_1 to s_T. In this case the formula contains a promise, we need to add some fairness constraints in order to fulfill this promise. In the structure of Fig. 1, the fairness constraints impose to reach p and s_T. At each depth of the execution tree obtained from s_0, state s_0 can reach state s_1, hence it satisfies $AF\,p$.

The following algorithm builds an AKS with respect to a property φ. The basic cases are depicted on Fig.3 and 5. For a state s, all atomic propositions that do not appear in the state label are such that $\mathcal{L}(s) = \emptyset$.

Definition 11 *Let $AKS(\varphi)$ the function mapping a formula φ into an abstract Kripke structure, $AKS(\varphi)$ is inductively defined as follows. AP is a initial set of atomic propositions, p and q are atomic propositions, φ_1, φ_2 are CTL formulae and $K_{\varphi_i} = \langle AP_i, S_i, S_{0_i}, \mathcal{L}_i, R_i, F_i \rangle$ are abstract Kripke structures related to φ_i for $i = 1,2$.*

- $\varphi =$ **true**,
  $K_\varphi = \langle \emptyset, \{s_T\}, \{s_T\}, \emptyset, \{s_T, s_T\}, \{s_T\} \rangle$
- $\varphi = \mathbf{p}$,
  - $K_T = AKS(\mathbf{true}) = \langle \emptyset, S_T, s_T, \emptyset, R_T, F_T \rangle$
  - Let s a new state such that $p \in \mathcal{L}(s)$

  $K_\varphi = \langle \{p\}, \{s\} \cup S_T, \{s\}, \mathcal{L}, \{(s,s') \mid s' \in S_{0_T}\} \cup R_T, F_T \rangle$
- $\varphi = \varphi_1 \vee \varphi_2$,
  - $K_{\varphi_1} = AKS(\varphi_1)$; $K_{\varphi_2} = AKS(\varphi_2)$

  $K_\varphi = \langle AP_{\varphi_1} \cup AP_{\varphi_2}, S_{\varphi_1} \cup S_{\varphi_2}, S_{0_{\varphi_1}} \cup S_{0_{\varphi_2}}, L\varphi_1{}^{+AP_{\varphi_2}} \cdot L_{\varphi_2}^{+AP_{\varphi_1}}, R_{\varphi_1} \cup R_{\varphi_2}, F_{\varphi_1} \cup F_{\varphi_2} \rangle$.

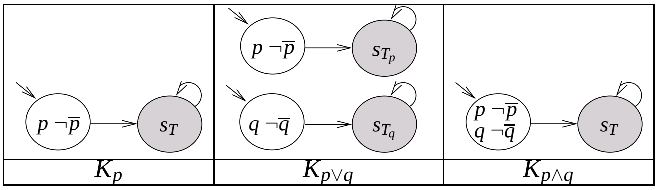

Figure 2. The AKS for p, $p \vee q$ and $p \wedge q$

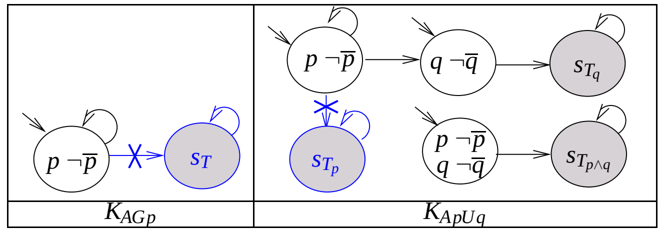

Figure 3. The AKS AGp, $ApUq$

- $\varphi = \varphi_1 \wedge \varphi_2$,
  - $K_{\varphi_1} = AKS(\varphi_1)$; $K_{\varphi_2} = AKS(\varphi_2)$

  $K_\varphi = K_{\varphi_1} \parallel K_{\varphi_2}$

- $\varphi = \mathbf{AF}\varphi_1$,
  - $K_{\varphi_1} = AKS(\varphi_1)$
  - Let s be state s.t $\forall p \in AP_{\varphi_1}$, p and $\overline{p} \notin \mathcal{L}(s)$.

  $K_\varphi = \langle AP_{\varphi_1}, \{s\} \cup S_{\varphi_1}, \{s\} \cup S_{0_{\varphi_1}}, \mathcal{L} \cdot \mathcal{L}_{\varphi_1}, R_{\varphi_1} \cup \{(s,s)\} \cup \{(s,s_i) \mid \forall s_i \in S_{0_{\varphi_1}}\}, F_{\varphi_1}\rangle$

- $\varphi = \mathbf{AG}\varphi_1$,
  - $K_{\varphi_1} = AKS(\varphi_1)$
  - $R = [S_{\varphi_1} \cap S_{predT}] \times S_{0_{\varphi_1}}$: for all states $s \in S_{\varphi_1} \cap S_{predT}$ and for all $s_0 \in S_{0_{\varphi_1}}$, $(s,s_0) \in R$.

  $K_\varphi = \langle AP_{\varphi_1}, S_{\varphi_1} \setminus S_T, S_{0_{\varphi_1}}, L_{\varphi_1}, R \cup (R_{\varphi_1} \setminus \{(s,s_T) \mid s \in S_{predT} \wedge s_T \in S_T\}), (F_{\varphi_1} \ominus S_T) \cup S_{predT_{\varphi_1}}\rangle$

- $\varphi = \mathbf{A}[\varphi_1 \mathbf{U} \varphi_2]$,
  - $K_{\varphi_1} = AKS(\varphi_1)$; $K_{\varphi_2} = AKS(\varphi_2)$; $K_{\varphi_1 \wedge \varphi_2} = AKS(\varphi_1 \wedge \varphi_2)$
  - $R = [S_{\varphi_1} \cap S_{predT}] \times [S_{0_{\varphi_1}} \cup S_{0_{\varphi_2}}]$
  - Let R' be the transition relation defined such that, for all states $s \in S_{0_{\varphi_1}} \setminus S_{predT}$ and for all $(s_{0_1}, s_{0_2}) \in S_{0_{\varphi_1 \wedge \varphi_2}}$, $(s,s_0) \in R'$ iff $(s, s_{0_1}) \in R_{0_{\varphi_1}}$.

  $K_\varphi = \langle AP_{\varphi_1} \cup AP_{\varphi_2}, (S_{\varphi_1} \setminus S_T) \cup S_{\varphi_2} \cup S_{\varphi_1 \wedge \varphi_2}, S_{0_{\varphi_1}} \cup S_{0_{\varphi_2}} \cup S_{0_{\varphi_1 \wedge \varphi_2}}, L_{\varphi_1}^{+AP_2} \cdot L_{\varphi_2}^{+AP_1}, R \cup R' \cup (R_{\varphi_1} \setminus \{(s,s_T) \mid s \in S_{predT} \wedge s_T \in S_T\} \cup R_{\varphi_2} \cup R_{\varphi_1 \wedge \varphi_2}, F_{\varphi_2} \cup F_{\varphi_1 \wedge \varphi_2}\rangle$

- $\varphi = \mathbf{A}[\varphi_1 \mathbf{W} \varphi_2]$ proceeds as $A[\varphi_1 U \varphi_2]$ except that the fairness is $F_\varphi = F_{\varphi_1} \cup F_{\varphi_2} \cup F_{\varphi_1 \wedge \varphi_2}$.

3.2.2 Existential operator

We consider the formula of the following form EGf, $E[fUg]$ where f and g *are not conjunction* of sub-formula

The algorithm for CTL formulae with an existential quantifier as first quantifier is very similar to the one applied on ACTL formulas. The main difference is the representation of all the paths not being model of the path-formula. In the abstract structure, from the initial states we need to represent the path along which the formula holds and the one which diverges. In order to perform this we represent all the diverging parts of the Kripke structure by a state in which no atomic propositions is defined.

Definition 12 *Let S_D be the set of diverging states s_d such that : $\forall s_d \in S_d$, $\mathcal{L}(s_d) = \emptyset$ and there exists an unique outgoing transition from s_d to itself.*

Let S_{predD} the set of predecessor of a state in S_D : $\forall s \in S_{predD}$, $\exists s_D \in S_D$, such that $s \to s_D$.

Lemma 2 *An abstract Kripke structure reduced to a state $s_D \in S_D$ simulates every non empty Kripke structure (with infinite behaviours).*

A composed state (s,s') belongs to S_D if and only if one of the three conditions holds: $s \in S_D$ and $s' \in S_D$; $s \in S_D$ and $s' \in S_T$; $s \in S_T$ and $s' \in S_D$

The following definition has to be added to the definition 11 for the CTL formulas with the universal quantifier as the first quantifier.

Definition 13 *Let $AKS(\varphi)$ the function mapping a formula φ into an abstract Kripke structure, $AKS(\varphi)$ is inductively defined as follows. AP is a set of atomic propositions initially empty, p and q are atomic propositions, φ_1, φ_2 are CTL formulae and $K_{\varphi_i} = \langle AP_i, S_i, S_{0_i}, \mathcal{L}_i, R_i, F_i\rangle$ are abstract Kripke structures related to φ_i.*

- $\varphi = \mathbf{EF}\varphi_1$,
  - $K_{\varphi_1} = AKS(\varphi_1)$
  - Let s be state s.t $\mathcal{L}(s) = \emptyset$.

  $K_\varphi = \langle AP_{\varphi_1}, \{s\} \cup S_{\varphi_1} \cup \{s_d\}, \{s\} \cup S_{0_{\varphi_1}}, \mathcal{L} \cdot \mathcal{L}_{\varphi_1}, R_{\varphi_1} \cup \{(s,s)\} \cup \{(s,s_i) \mid \forall s_i \in S_{0_{\varphi_1}}\} \cup \{(s,s_d)\}, F_{\varphi_1} \oplus \{s_D\}\rangle$

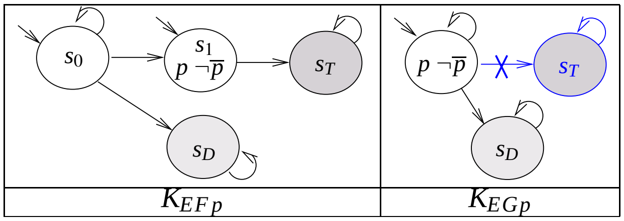

Figure 4. The AKS for EFp, EGp

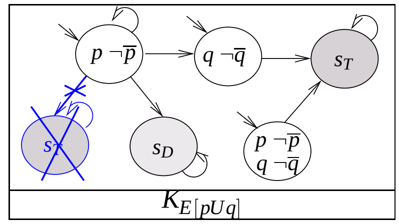

Figure 5. The AKS for $E[pUq]$

- $\varphi = \mathbf{EG}\varphi_1$,
  - $K_{\varphi_1} = AKS(\varphi_1)$
  - $R = [S_{\varphi_1} \cap S_{predT}] \times S_{0_{\varphi_1}}$
  - $R' = [S_{\varphi_1} \cap S_{predT}] \times \{s_D \mid s_D \in S_D\}$

  $K_\varphi = \langle AP_{\varphi_1}, (S_{\varphi_1} \setminus S_T) \cup S_d, S_{0_{\varphi_1}}, L_{\varphi_1}, R \cup R' \cup (R_{\varphi_1} \setminus \{(s, s_T) \mid s \in S_{predT} \wedge s_T \in S_T\}, (F_{\varphi_1} \ominus S_T) \oplus \{s_D\}\rangle$

- $\varphi = \mathbf{E}[\varphi_1 \mathbf{U} \varphi_2]$,
  - $K_{\varphi_1} = AKS(\varphi_1)$; $K_{\varphi_2} = AKS(\varphi_2)$; $K_{\varphi_1 \wedge \varphi_2} = AKS(\varphi_1 \wedge \varphi_2)$
  - $R = [S_{\varphi_1} \cap S_{predT}] \times [S_{0_{\varphi_1}} \cup S_D]$ $(s, s_D) \in R$.
  - Let R' be the transition relation define such that, for all states $s \in S_{0_{\varphi_1}} \setminus S_{predT}$ and for all $(s_{0_1}, s_{0_2}) \in S_{0_{\varphi_1 \wedge \varphi_2}}$, $(s, s_0) \in R'$ iff $(s, s_{0_1}) \in R_{0_{\varphi_1}}$.

  $K_\varphi = \langle AP_{\varphi_1} \cup AP_{\varphi_2}, (S_{\varphi_1} \setminus S_T) \cup S_{\varphi_2} \cup S_{\varphi_1 \wedge \varphi_2} \cup S_D, S_{0_{\varphi_1}} \cup S_{0_{\varphi_2}} \cup S_{0_{\varphi_1 \wedge \varphi_2}}, L_{\varphi_1}^{+AP_2} \cdot L_{\varphi_2}^{+AP_1}, R \cup R' \cup (R_{\varphi_1} \setminus \{(s, s_T) \mid s \in S_{predT} \wedge s_T \in S_T\} \cup R_{\varphi_2} \cup R_{\varphi_1 \wedge \varphi_2}, (F_{\varphi_2} \cup F_{\varphi_1 \wedge \varphi_2}) \oplus \{s_D\}\rangle$

- $\varphi = \mathbf{E}[\varphi_1 \mathbf{W} \varphi_2]$ proceeds as $E[\varphi_1 U \varphi_2]$ except that the fairness is $F_\varphi = (F_{\varphi_1} \cup F_{\varphi_2} \cup F_{\varphi_1 \wedge \varphi_2}) \oplus \{s_D\}$.

In order to obtain a construction for the full CTL with our restriction, we need to add some new rules to the algorithm ACTL especially to manage divergent states.

Definition 14 *Additional rules apply to the AKS build from an ACTL formulae.*

- $\varphi = \mathbf{AG}\varphi_1$,
  - $S_\varphi = S_{\varphi_1} \setminus (S_T \cup S_D)$
  - *Let R the transition relation obtained by definition 11 (item* **AG***), $R_\varphi = (R \setminus \{(s, s_D) \mid s \in S_{predD} \wedge s_D \in S_D) \cup \{(s, s_0) \mid s \in S_{predD} \wedge s_0 \in S_{0_{\varphi_1}}$*
  - $F_\varphi = (F_{\varphi_1} \ominus (S_T \cup S_D)) \cup S_{predD} \cup S_{predD}$
- $\varphi = \mathbf{A}[\varphi_1 \mathbf{U} \varphi_2]$,
  - $S_\varphi = S_{\varphi_1} \setminus (S_T \cup S_D)$
  - *Let R the transition relation obtained by definition 11 (item* **AU***), $R_\varphi = (R \setminus \{(s, s_D) \mid s \in S_{predD} \wedge s_D \in S_D) \cup \{(s, s_0) \mid s \in S_{predD} \wedge s_0 \in S_{0_{\varphi_1}}$*

Example 1 *Figure 6 shows an AKS built from the CTL formula* $\mathbf{EG}(\mathbf{AF}(p \vee q))$*. The different steps of our algorithm are presented below :*

1. *Build $AKS(s_{T_p})$: a state s_T is created;*
2. *Build $AKS(p)$: a state s_p is created such that $p \in \mathcal{L}(s)$ and $\overline{p} \notin \mathcal{L}(s_p)$, a transition from s_p to s_{T_p} is added;*
3. *Build $AKS(s_{T_q})$: a state s_T is created;*
4. *Build $AKS(q)$: a state s_q is created such that $q \in \mathcal{L}(s)$ and $\overline{q} \notin \mathcal{L}(s_q)$, a transition from s_q to s_{T_q} is added;*
5. *Build$AKS(p \vee q)$: union of $AKS(p)$ and $AKS(q)$;*
6. *Build $AKS(\mathbf{AF}(p \vee q))$: a state s is created such that $\mathcal{L}(s) = \emptyset$, a transition to s_p and a transition to s_q are added ;*
7. *Build $AKS(\mathbf{EG}(\mathbf{AF}(p \vee q)))$: all states s_T and all transitions ongoing to these states are removed, a state s_D is created, transitions from s_p and from s_q to the set of initial states and to s_D are added.*

4. AKS's properties

4.1. Properties of the abstract component

The previous section define the construction of an AKS from a CTL formula. In the present section we will state the properties of the abstract component.

Definition 15 *Let $C = \langle K, P, A\rangle$ be a concrete component,* the component abstraction *of C with respect to $\varphi \subseteq P$ is a tuple $C_\varphi = \langle K_\varphi, \varphi, K_A\rangle$ where K_φ and K_A are built following the algorithm of Def. 11,13 and 14.*

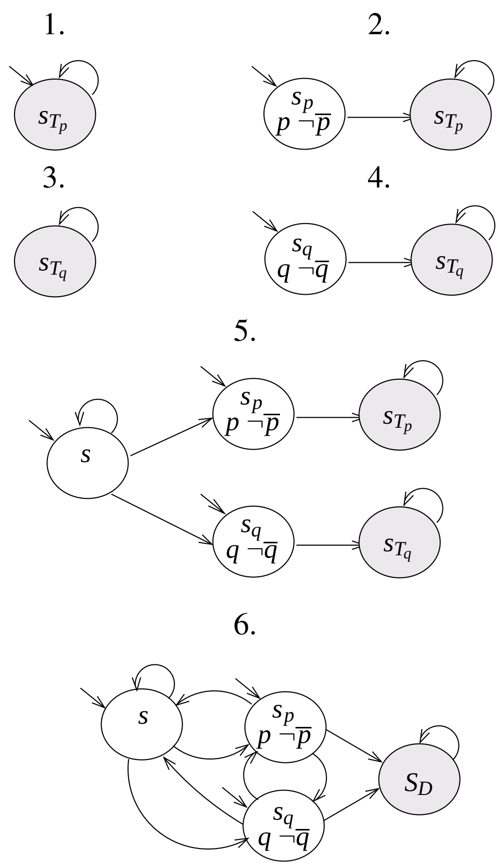

Figure 6. Construction of the AKS representing the formula $\mathbf{EG}(\mathbf{AF}(p \vee q))$

This section deals with the relation between a formula φ expressed in a positive normal form of CTL$\backslash$X, a concrete component C with $\varphi \subseteq P$ and an abstract component C_φ. First, we state that for each formula φ defined as in Section 2, we can construct an AKS K_φ such that $K_\varphi \models \varphi$. Then, we prove that there exists a simulation relation between K the Kripke structure of concrete component satisfying φ, and K_φ the abstract Kripke structure. In order to alleviate the reading, long proofs are given in appendix.

Property 1 *The AKS K_φ built by definitions 11,13 and 14 is such that $\forall s_0 \in S_{0_\varphi}, K_\varphi, s_0 \models \varphi$.*

To ensure concision the proof is not given here. The proof proceeds by induction over the length of the formula.

As in [1, 21], the labeling function $\mathcal{L}$ induces a partial ordering ($\sqsubseteq$) of states according to the information level of each atomic proposition in each state : $s \sqsubseteq s'$ if there exists $p \in AP$ such that p is *less constrained* in s' than in s ($p \in \mathcal{L}(s)$ and $p \notin \mathcal{L}(s') \wedge p \notin \mathcal{L}(s')$) and for all $q \in AP$ and $q \neq p$, q has the same truth value in s and in s'. As in their work we deduce that there exists a simulation relation between K and K_φ, denoted by $K \preceq K_\varphi$, and $\preceq$ is a preorder.

Property 2 *For all Kripke structure K such that $K \models \varphi$ there exists $K_\varphi = AKS(\varphi)$ and there exists a simulation relation $\preceq$ such that $K \preceq K_\varphi$.*

The proof is not given here. It proceeds by induction over the length of the formula, basic cases are formula with no nested operators.

4.2. Composition of abstractions

After having abstracted all components with a subset of their specification, we want to compose all these abstractions in order to obtain an abstraction of the complete system. First of all, we need to ensure that the components to be combined are *compatible*: their output signals sets are disjoint. The systems we consider are hardware components, that can not write simultaneously to same output signal. An arbitration policy guarantees the exclusive driving of shared output signals (as bus). This corresponds to a multiplexer based architecture.

The whole system abstraction is thus obtained by composing all the abstracted components with the parallel composition (Def. 8). By consequence of compatible component the composition of all abstractions can not introduce new inconsistent states. Now, we want to prove that the composition of our abstraction simulates the whole concrete system. Each component is abstracted by some CTL properties, the proof proceeds by applying an assume-guarantee reasoning [13].

Assumption : We can compose a component C_i with other components $C_k \ldots C_m$ if and only if the set of specification of components $C_k \ldots C_m$ implies the assumption of C_i: $\bigcup_{j=k \ldots m} P_j \implies A_i$ [16].

Property 3 *Let C_1, ..., C_n be concrete components $\langle K_i, P_i, A_i \rangle$, and C_{φ_1}, ..., C_{φ_n} be abstract components $\langle K_{\varphi_i}, \varphi_i, K_{A_i} \rangle$ with $\forall i$, $\varphi_i \subseteq P_i$. Let $\Sigma = C_1 || \ldots C_n$ be a concrete system and $\Sigma_A = C_{\varphi_1} || \ldots C_{\varphi_n}$ be the abstract model, then Σ_A simulates Σ.*

PROOF: Directly obtained by assume-guarantee reasoning. Example for two components C_1 and C_2:

$$\begin{array}{cccc}
1 & K_1 \parallel K_{A_1} & \preceq & K_1 \\
2 & K_1 & \preceq & K_{\varphi_1} \\
3 & K_2 \parallel K_{A_2} & \preceq & K_2 \\
4 & K_2 & \preceq & K_{\varphi_2} \\
5 & K_{A_1} & \preceq & K_{\varphi_2} \\
6 & K_{A_2} & \preceq & K_{\varphi_1} \\
\hline
 & K_1 \parallel K_2 & \preceq & K_{\varphi_1} \parallel K_{\varphi_2}
\end{array}$$

$\preceq$ is a simulation relation and preorder. Line 1 and Line 3 come from the property of synchronous composition; line 2 and 4 comes from property 2.

■

We can conclude that all the universal fragment of CTL is preserved from the composition of abstracted components into the composition of concrete components.

5. A case study

In this section, we present an experiment illustrating our verification methodology. Given a *global* property (in ACTL) to be verified on a system encompassing several components, each component is abstracted according to *selected properties* (in CTL) of its specification, and the global property is verified on the *abstract complete system*.

Up to now, during the abstraction of one component with respect to some *local* property φ, signals not influencing the satisfaction of φ, were considered as being assigned with a "don't care" value. During the verification of a *global* property ψ (encompassing several components), the value of such signals are now interpreted as "unknown".

Our experiments use a classical model checker with two truth values. All signals with the "unknown" value are translated with a non deterministic value. If the model checker states that the global property does not hold then we need to determine if the counter-example is :

- a real one (if an corresponding execution over the concrete component exists) or
- a spurious one (no possible execution over the concrete component exist)

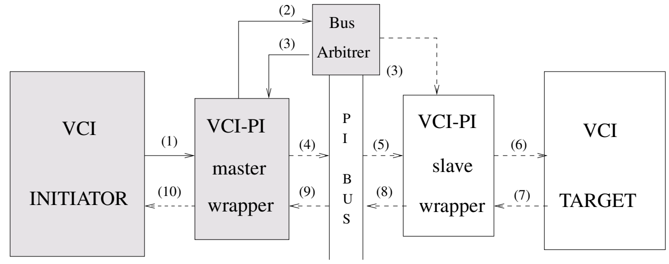

Figure 7. The architecture performing the VCI-PI-VCI translation and illustration of a VCI transfer

Our system interconnects VCI compliant components ([17]) through a physical PI-bus ([18]). Our aim is to verify with model checker VIS ([10]) that the overall architecture correctly transmits messages with successive translations, from VCI to PI and from PI to VCI in both directions.

Using such devices, we are able to connect various VCI compliant-cores through a PI-bus. The simplest architecture is shown in Fig. 7. We have a *VCI initiator* sending requests and a *VCI target* responding to it. The PI protocol distinguishes the component initiating a bus transfer, named *master*, and the component responding to a transfer, named *slave*. A communication from (VCI) initiator to the (VCI) target is shown Fig. 7.

Here we focus on the following three global properties (extended to n initiators and m targets) :

Property 1: **AG**(initiator[i]_state = TRANS ⇒ **AF** (bus_arbiter_signal_gnt[i] = 1))

Property 2: **AG**(initiator[i]_state = TRANS ⇒ **AF** (master_wrapper[i]_state = WRITE))

Property 3: **AG**(initiator[i]_state = TRANS ⇒ **AF** (target[j]_signal_rsp = 1))

Although the atomic propositions of these CTL properties relate two components only, components not appearing in the property may play a role in the action sequence of the communication protocol :

- Property 1: its atomic propositions relate components initiator[i] and bus arbiter. Their communication protocol also involves the master wrapper[i].
- Property 2: its atomic propositions relate components initiator[i] and master wrapper[i]. Their communication protocol also involves the bus arbiter.
- Property 3: its atomic propositions relate components initiator[i] and target[j]. Their communication protocol also involves the bus arbiter, master wrapper[i] and slave wrapper[j].

The verification of property 1 took 3 refinement iterations of the CEGAR verification process. A total of 6 formulas of the specification of the three components (in Grey on Fig. 7) were necessary to terminate the process. The verification of property 2 took 2 iterations and 4 formulas. In the same way, the verification of property 3 took 4 iterations and 10 formulas were needed. Property 3 exhibited a bug in the implementation of VCI protocol between the slave wrapper and the VCI target.

Table 2 summarizes the profits in term of time and state space for three global properties. We note an obvious profit in term of time for the reachability analysis and the model checking of the 3 global properties on the abstracted systems. This is due to the reduction of the tree explored depth. The number of BDD variables did not really fall but, because a lot of signal are free in the abstraction, the BDD structures are smaller (number of nodes).

6. Conclusion

We defined an algorithm for building directly an abstract Kripke structure from a set of CTL formulas. We used a 3-valued logic that nicely models the "don't care" information about the atomic propositions irrelevant for the satisfaction of a CTL formula. We then stated that the abstract Kripke structure, modeling a formula φ, can be used as an abstraction of a concrete component, where φ is in its specification. We showed that this abstraction can be employed in a modular verification process. Furthermore, we determined how our composition of abstractions can be integrated into Bruns and Godefroid's 3-valued model checking framework. We pointed out that a "don't care" value for an atomic proposition of a "local" property φ becomes a "don't know" information when the complete abstract system is checked against a "global" property ψ. We finally exhibited the benefit of our approach in term of time consumption. We applied our method for the verification process of a real architecture dealing with several components that contains VCI compliant components, a PI-bus and VCI-PI wrappers.

We are currently working on an integration of our algorithm into the model checker VIS [10] (automatizing the CEGAR verification process). Another important direction concerns the choice of the set of properties to initiate the abstraction, and to refute a counter-example.

References

[1] G. Bruns and P. Godefroid. Model Checking Partial State Spaces with 3-Valued Temporal Logics. In N. Halbwachs and D. Peled, editors, *CAV'99*, volume 1633 of *LNCS*, pages 274–287. Springer, 1999.

[2] G. Bruns and P. Godefroid. Generalized Model Checking: Reasoning about Partial State Spaces. In N. Halbwachs and D. Peled, editors, *CONCUR'2000*, volume 1877 of *LNCS*, pages 168–182. Springer, 2000.

[3] J. Burch, E. Clarke, K. McMillan, D. Dill, and L. Hwang. Symbolic Model Checking: 10^{20} States and Beyond. *Information and Computation*, 98(2):142–170, 1992. Special issue for best papers from LICS'90.

[4] W. Büttner. Is formal verification bound to remain a junior partner of simulation? volume 3725 of *LNCS*, page 1. Springer, 2005.

[5] M. Chechik, B. Devereux, S. M. Easterbrook, and A. Gurfinkel. Multi-Valued Symbolic Model-Checking. *ACM Trans. Soft. Eng. Meth.*, 12(4):371–408, 2003.

[6] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1999.

[7] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided bstraction refinement for symbolic model checking. *J. of the ACM*, 50(5):752–794, 2003.

[8] D. Dams and K. S. Namjoshi. Automata as Abstractions. volume 3385 of *LNCS*, pages 216–232, Paris, France, 2005. Springer.

[9] A. Goel and W. R. Lee. Formal Verification of an IBM CoreConnect Processor Local Bus Arbiter Core. In ACM, editor, *DAC'00*, pages 196–200, Los Angeles, USA, 2000.

[10] T. V. group. VIS : A System for Verification and Synthesis. In Rajeev Alur and Thomas A. Henzinger, editors, *CAV'96*, volume 1102 of *LNCS*, pages 428–432, New Brunswick, NJ, USA, 1996. Springer-Verlag.

[11] O. Grumberg and D. Long. Model Checking and Modular Verification. In *International Conference on Concurrency Theory*, volume 527 of *LNCS*, pages 250–263. Springer Verlag, 1991.

[12] A. Gurfinkel and M. Chechik. Why Waste a Perfectly Good Abstraction?. volume 3920 of *LNCS*, pages 212–226, Vienna, Austria, 2006. Springer.

[13] T. Henzinger, S. Qadeer, S. Rajamani, and S. Tasiran. An Assume-Guarantee Rule for Checking Simulation. *ACM trans. Prog. Lang. Syst.*, 24(1):51–64, 2002.

Platform name	FSM depth	Number of BDD variables	BDD size (\# of nodes)	Number of Reachable states	Reachable states space analysis time	Property	Checking time
Concrete 1 master 1 slave	475	289	34 578	8,56E+06	50,61s	1	6,75s
						2	6,70s
						3	6,88s
All abstract	7	261	522	2,73E+16	2,1s	1	0,1s
	8	292	796	1.42E+21	5,3s	2	0,2s
	10	302	1 002	1,69E+22	8s	3	0,25s
Concerned component abstracted	12	360	2 296	8,77E+15	4,26s	1	0,31s
	12	320	1 982	5,63E+15	6,22s	2	0,36s
	10	302	1 002	1,69E+22	8s	3 (all)	0,25s
Concrete 2 masters 1 slave	604	436	161 846	3,10E+10	43min	1	46min
						2	19min
						3	48min
All abstract	10	395	1 005	2,83E+20	3,5s	1	0,8s
	11	442	1 079	4,93E+24	3,8s	2	0,52s
	14	501	1564	2,50E+26	12,1s	3	8,25s
Concerned component abstracted	12	532	14 025	6,49E+24	35,21s	1	0,46s
	12	544	14 492	7.49E+24	33,77s	2	0,47s
	14	501	1 564	2,50E+26	12,1s	3	8,25s

Table 2. Comparative results of the concrete models and 3 global properties. Results are obtained with VIS model checker with the reordering algorithm sift. The machine was a Pentium IV, 3.20GHz with 1MB of cache and 1GB of RAM

[14] T. A. Henzinger, X. Liu, S. Qadeer, and S. K. Rajamani. Formal Specification and Verification of a Dataflow Processor Array. In E. S. Jacob K. White, editor, *ICCAD'99*, pages 494–499, San Jose, USA, 1999.

[15] O. Kupferman, M. Vardi, and P. Wolper. An Automata-Theoretic Approach to Branching-Time Model Checking. *J. of the ACM*, 47(2):312–360, 2000.

[16] K. McMillan. A Methodology for Hardware Verification Using Compositional Model Checking. *Science of Computer Programming*, 37(1–3):279–309, 2000.

[17] On-Chip Bus Development Working Group. *Virtual Component Interface Standard (VCI)*. VSI Alliance, 2000.

[18] Open Microprocessors System Initiatives. *OMI324: PI-Bus Standard Specification*. Siemens, Munich, Germany, 1994.

[19] H. Peng, Y. Mokhtari, and S. Tahar. Environment Synthesis for Compositional Model Checking. In *ICCD'92*, pages 70–, Freiburg, Germany, 2002. IEEE Computer Society.

[20] H. Peng, S. Tahar, and F. Khendek. Comparison of SPIN and VIS for Protocol Verification. *STTT*, 4(2):234–245, 2003.

[21] S. Shoham and O. Grumberg. Monotonic Abstraction-Refinement for CTL. In A. P. Kurt Jensen, editor, *TACAS'04*, volume 2988 of *LNCS*, pages 546–560, Barcelona, Spain, 2004. Springer.

[22] J. Sparsø. Asynchronous Circuit Design - A Tutorial. In *Chapters 1-8 in "Principles of asynchronous circuit design - A systems Perspective"*, pages 1–152. Kluwer Academic Publishers, London, dec 2001.

[23] F. Xie and J. Browne. Verified Systems by Composition from Verified Components. In *ESEC/FSE 2003*, pages 277–286, Helsinki, Finland, 2003. ACM Press.