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

Cécile Braunstein[1] and Emmanuelle Encrenaz[2]

[1] University Paris 6 - LIP6 - CNRS – France
cecile.braunstein@lip6.fr
[2] Laboratoire Spécification et Vérification - ENS Cachan - CNRS – France
emmanuelle.encrenaz@lsv.ens-cachan.fr

Abstract. The verification of global properties (involving several components) is difficult or impossible to achieve, due to combinatorial explosion problem, while the verification of each component is easier to perform. Following the idea of Xie and Browne [21], we propose to build an abstraction of a component already verified, starting from a subset of its specification described as CTL formulae. For a component C and a formulae ϕ, the abstraction is a 3-valued Kripke structure, which is smaller than the Kripke structure of C, and both model ϕ. This abstraction replaces the concrete component in the global properties verification, hence it alleviates the state explosion problem. In this paper, we first set out our CTL-based abstraction of a component, and present the algorithm to build an abstract Kripke structure from a component and a subset of its specification CTL. Then we show some results on abstract Kripke structure composition and model-checking. Finally, we exhibit the benefits of this approach in model-checking by applying this abstraction on a hardware system relating VCI components to a PI-bus.

1 Introduction

This work takes place in the context of hardware modular verification by model checking ([3, 12]). 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, 19, 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 [21] 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) [6]. But in [21], 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 in 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 which 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 add a new formula from the specification of one (or several) components.

The main contribution of this work is the definition of an algorithm building automatically abstraction from a subset of the component specification. Several works study the translation of temporal logic into automaton. Kupferman and al. [17] state a linear translation from branching temporal logic into alternating tree automatas. This implies an automata-based model checking algorithm of linear running time for CTL. More recently, Dams and Namjoshi [8] propose to use tree automata 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 [13]) 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, 13] 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 relatively 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 abstract

Kripke structure and the composition of such structures. Section 4 states the link between CTL formulas, abstract and concrete structure. Finally, Section 5 studies the impact of the abstraction in the verification process of a system encompassing: Virtual Component Interface IP's (VCI[10]), a PI-bus ([16]) 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 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 : a fair Kripke structure, P and A are disjoint sets of CTL formulae.*

We model the behaviour of one component, or of a composition of components as a *fair Kripke structure* [7]. A fair Kripke structure is 6-tuple $K = \langle AP, S, S_0, L, R, F \rangle$ where : AP is a finite set of atomic propositions; $S_0 \subseteq S$ is the set of initial states; $L : \{l_0, \ldots, l_{|AP|-1}\}$ is a vector of $|\ AP\ |$ interpretation functions; $R \subseteq S \times S$ 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).

Let s and s' be in S, we write $s \rightarrow s'$ 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 $\forall i\ s_i \rightarrow s_{i+1}$.

Definition 2. *An* interpretation function *l associates a truth value with each atomic proposition for a given state:* $l : S \times AP \rightarrow \{\mathbf{true}, \mathbf{false}\}$.

We call $\mathcal{L} : S \rightarrow 2^{AP}$ *the function that associates each state with the set of atomic propositions that are* **true** *in that state.*

In this paper we restrict to the positive normal form of CTL\X formulae. 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 formulae and ψ is a path formula.

State formula : $\varphi ::= \mathbf{true} \mid p \mid \neg p \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid E\psi \mid A\psi$ 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 formulae is defined on the infinite execution tree obtained by unrolling the Kripke structure [7]. We have $K \models \varphi$ iff for all s in S_0 $K, s_0 \models \varphi$.

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 for 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 new values with new

symbols. Instead, we extend the set of atomic proposition *AP* of a Kripke structure. *AP* is now the set of atomic propositions and *their negations*. In our abstract Kripke structure (AKS), for all p in *AP*, we have $\overline{p}$ also in *AP*. Actually, we represent the knowledge of all atomic propositions in each state.

The "bottom truth value" ($\bot$) in $[2,5]$ is represented here by : $l(s,p) =$ **false** and $l(s,\overline{p}) =$ **false** ; the "top truth value" ($\top$) of the 4-valued logic is expressed here by $l(s,p) =$ **true** and $l(s,\overline{p}) =$ **true**. The case where $p = \neg\overline{p}$ is the traditional one. In the continuation of the paper we will show that fourth value is not useful in our work.

Definition 3. *An* Abstract Kripke Structure *(AKS) is a tuple* $A = \langle AP, S, S_0, \mathcal{L}, R, F\rangle$. $S, S_0, \mathcal{L}, R, F$ *are defined as above and AP 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* $\overline{p} \in \mathcal{L}(s)$.

Definition 5. *An* inconsistent path *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. 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 vector of interpretation functions 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 each function l in the vector of interpretation functions L when increasing the set of atomic propositions, and when increasing the set of states.

Definition 6. Atomic proposition extension
Let L be a vector of $|AP|$ *interpretation functions, let* AP' *be a set of atomic propositions* ($AP' \cap AP = \emptyset$) *and S a set of states.* $L^{+AP'}$ *is a vector of interpretation functions over* $AP' \cup AP$ *such that :*
$\forall p \in AP, \forall s \in S, l(s,p)$ *and* $l(s,\overline{p})$ *are unchanged ;* $\forall p \in AP', \forall s \in S,$ $l(s,p) =$ **false** *and* $l(s,\overline{p}) =$ **false**.

Definition 7. State extension
Let L defined on $S \times AP$ *and* L' *defined on* $S' \times AP$ *be two vectors of interpretation functions related to the same set of atomic propositions.* $L'' = L \cdot L'$ *is a vector of interpretation functions related to AP for each state* $s \in S \cup S'$ *such that* $l''(s,p) = l(s,p)$ *(resp.* $l'(s,p)$*) if* $s \in S$ *(resp* $s \in S'$*) and* $l''(s,\overline{p}) = l(s,\overline{p})$ *(resp.* $l'(s,\overline{p})$*) if* $s \in S$ *(resp* $s \in S'$*)*

Definition 8. Concatenation function
Let K_1 and K_2 be two abstract Kripke structures to be composed, $L = L_1 * L_2$ is a vector of $|\ AP_1 \cup AP_2\ |$ labeling functions for each state $(s_1, s_2) \in S_1 \cup S_2$ and such that :

- $\forall p \in AP_1 \cap AP_2$, $l((s_1,s_2),p) = l_1(s_1,p) \vee l_2(s_2,p)$ *and* $l(s,\overline{p}) = l_1(s_1,\overline{p}) \vee l_2(s_2,\overline{p})$
- $\forall p \in AP_1$ *(resp. AP_2),* $p \notin AP_1 \cap AP_2$, $l((s_1,s_2),p) = l_1(s_1,p)$ *(resp. $l_2(s_2,p)$) and* $l(s,\overline{p}) = l_1(s_1,\overline{p})$ *(resp. $l_2(s_2,\overline{p})$).*

Synchronous composition is closed to the parallel composition of [12] 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 9. *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 :*

$AP = AP_1 \cup AP_2$
$S \subseteq (S_1 \times S_2) \setminus \{(s_1,s_2) \mid l((s_1,s_2),p) = \textbf{true}$ *and* $l((s_1,s_2),\overline{p}) = \textbf{true}\}$
$S_0 = (S_{0_1} \times S_{0_2})$
$L = L_1 * L_2$
$R \subseteq S_1 \times S_2 \rightarrow S_1 \times S_2$, *that is* $R((s_1,s_2),(t_1,t_2))$ *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

In this section, we present the construction algorithm of an abstract Kripke structure from a CTL$\setminus X$ formula. Without loss of generality, CTL formulas are rewritten as follows :

- $AG(f \wedge g) = AG(f) \wedge AG(g)$; $EG(f \wedge g) = EG(f) \wedge EG(g)$
- $A[(f \wedge g) U h] = A[f U h] \wedge A[g U h]$; $E[(f \wedge g) U h] = E[f U h] \wedge E[g U h]$

We need to define two special states : s_{TRUE} and s_{NEG}, each of which having a unique outgoing self-loop transition. The execution tree obtained by unrolling transitions from s_{TRUE} is an abstraction of any execution tree that has no impact on the validity of the formula. The same applies for state s_{NEG}.

State s_{TRUE} is reachable from a state s validating the formula in the initial state, whatever the future of s is. State s_{NEG} is only reachable from a state from which there exists another transition leading to a state where the formula can be evaluated. In states s_{TRUE} and s_{NEG} the truth value of atomic propositions (and their negation) is undefined ($\mathcal{L}(s_{TRUE}) = \emptyset \vee \mathcal{L}(s_{NEG}) = \emptyset$). These values can be interpreted as "don't care" values since they do not influence the satisfaction of the formula.

As an example, Fig. 1 presents the AKS obtained from the CTL formula EFp. It contains two initial states s_0 and s_1 and the states s_{TRUE} and s_{NEG}. In state s_1 p is **true** then EFp is **true**, the subsequent execution tree are not relevant for the evaluation of

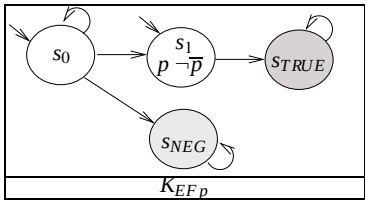

Fig. 1. The AKS obtained for the CTL formula $EF\,p$. Initial states are s_0 and $_1$. The interpretation function is : $l(s_0,p) =$ **false** and $l(s_0,\overline{p}) =$ **false**; $l(s_1,p) =$ **false** and $l(s_1,\overline{p}) =$ **false**

the formula. This is represented by the transition from s_1 to s_{TRUE}. At each depth of the execution tree obtained from s_0, state s_0 can reach state s_1, hence it satisfies $EF\,p$. Other future executions from s_0 are not significant and are abstracted by the execution tree obtained by unrolling state s_{NEG}.

Moreover, we define the sets S_{predT} and S_{predN} the subsets of S such that for all $s \in S_{predT}$, $s \rightarrow s_{TRUE}$, and for all $s \in S_{predN}$, $s \rightarrow s_{NEG}$.

The following definition presents the rules for building an AKS with respect to a property φ. The basic cases are depicted on Fig.2, 3, 4. For a state s, all atomic propositions that do not appear in the state label are such that $l(s,p) =$ **false** and $l(s,\overline{p}) =$ **false**.

Definition 10. *Construction of AKS K_φ is inductively defined as follows. p and q are atomic propositions, φ_1, φ_2 are CTL formulae.*

$\varphi = \mathbf{p}$, we construct
$K_p = \langle \{p,\overline{p}\}, \{s_0, s_{TRUE}\}, \{s_0\}, \{l\}, \{R(s_0,s_{TRUE}), R(s_{TRUE},s_{TRUE})\}, \{s_{TRUE}\}\rangle$ with $l(s_0,p) =$ **true** and $l(s_0,\overline{p}) =$ **false**

$\varphi = \overline{\mathbf{p}}$, we construct
$K_{\overline{p}} = \langle \{p,\overline{p}\}, \{s_0, s_{TRUE}\}, \{s_0\}, \{l\}, \{R(s_0,s_{TRUE}), R(s_{TRUE},s_{TRUE})\}, \{s_{TRUE}\}\rangle$ with $l(s_0,p) =$ **false** and $l(s_0,\overline{p}) =$ **true**

$\varphi = \varphi_1 \vee \varphi_2$, let $K_{\varphi_1} = \langle AP_{\varphi_1}, S_{\varphi_1}, S_{0_{\varphi_1}}, L_{\varphi_1}, R_{\varphi_1}, F_{\varphi_1}\rangle$ and
$K_{\varphi_2} = \langle AP_{\varphi_2}, S_{\varphi_2}, S_{0_{\varphi_2}}, L_{\varphi_2}, R_{\varphi_2}, F_{\varphi_2}\rangle$ be the AKS representing φ_1 and φ_2, we construct $K_\varphi = \langle AP_{\varphi_1} \cup AP_{\varphi_1}, S_{\varphi_1} \cup S_{\varphi_2}, S_{0_{\varphi_1}} \cup S_{0_{\varphi_2}}, L_{\varphi_1}^{+AP_2} \cdot L_{\varphi_2}^{+AP_1}, R_{\varphi_1} \cup R_{\varphi_2}, F_{\varphi_1} \cup F_{\varphi_2}\rangle$.

$\varphi = \varphi_1 \wedge \varphi_2$, let $K_{\varphi_1} = \langle AP_{\varphi_1}, S_{\varphi_1}, S_{0_{\varphi_1}}, L_{\varphi_1}, R_{\varphi_1}, F_{\varphi_1}\rangle$ and
$K_{\varphi_2} = \langle AP_{\varphi_2}, S_{\varphi_2}, S_{0_{\varphi_2}}, L_{\varphi_2}, R_{\varphi_2}, F_{\varphi_2}\rangle$ be the AKS representing φ_1 and φ_2, K_φ is defined as the parallel composition of K_{φ_1} and K_{φ_2} denoted $K_{\varphi_1} \parallel K_{\varphi_2}$. We add the transition rules :
- if $R_{\varphi_1}(s_1, s_{TRUE})$ and $R_{\varphi_2}(s_2, s_{TRUE})$ then $R_\varphi((s_1,s_2), s_{TRUE})$
- if $R_{\varphi_1}(s_1, s_{NEG})$ and $R\varphi_2(s_2, s_{TRUE})$ or $R_{\varphi_2}(s_2, s_{NEG})$ then $R_\varphi(s_\varphi, s_{NEG})$

$\varphi = \mathbf{AF}\varphi_1$, let $K_{\varphi_1} = \langle AP_{\varphi_1}, S_{\varphi_1}, S_{0_{\varphi_1}}, L_{\varphi_1}, R_{\varphi_1}, F_{\varphi_1}\rangle$ be the AKS representing φ_1, we construct K_φ by adding a new state s, whose all outgoing fair paths will eventually reach the initial states of K_{φ_1} (which verify φ_1).
$K_\varphi = \langle AP_{\varphi_1}, \{s\} \cup S_{\varphi_1}, \{s\} \cup S_{0_{\varphi_1}}, L_\varphi, R_{\varphi_1} \cup \{R(s,s)\} \cup \{R(s,s_i) \mid \forall s_i \in S_{0_{\varphi_1}}\}, F_{\varphi_1}\rangle$ where L_φ is the state-extension of L_{φ_1} with $S_\varphi = S_{\varphi_1} \cup \{s\}$.

$\varphi = \mathbf{EF}\varphi_1$, let $K_{\varphi_1} = \langle AP_{\varphi_1}, S_{\varphi_1}, S_{0_{\varphi_1}}, L_{\varphi_1}, R_{\varphi_1}, F_{\varphi_1}\rangle$ be the AKS representing φ_1, we construct K_φ by adding a new state s, from which there exists at least one outgoing

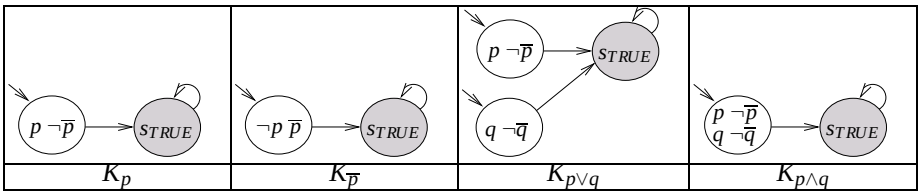

Fig. 2. The four AKS representing K_p, $K_{\overline{p}}$, $K_{p\vee q}$ and $K_{p\wedge q}$ respectively

path eventually reaching the initial states of K_{φ_1} (which verify φ_1), and one outgoing path reaching state s_{NEG} (hence $s \in S_{predN}$). We construct
$K_\varphi = \langle AP_{\varphi_1}, \{s, s_{NEG}\} \cup S_{\varphi_1}, \{s\} \cup S_{0_{\varphi_1}}, L_{\varphi_1}, R_{\varphi_1} \cup \{R(s,s)\} \cup R', F_{\varphi_1} \cup \{s_{NEG}\}\rangle$ such that $R' = \{R(s,s_i) \mid \forall s_i \in S_{0_\varphi}\} \cup \{R(s,s_{NEG}), R(s_{NEG},s_{NEG})\}$ and L_φ is the state-extension of L_{φ_1} with $S_\varphi = S_{\varphi_1} \cup \{s\}$.

$\varphi = \mathbf{AG}\varphi_1$, let $K_{\varphi_1} = \langle AP_{\varphi_1}, S_{\varphi_1}, S_{0_{\varphi_1}}, L_{\varphi_1}, R_{\varphi_1}, F_\varphi\rangle$ be the AKS representing φ_1. We construct $K_\varphi = \langle AP_{\varphi_1}, S_\varphi, S_{0_{\varphi_1}}, L_\varphi, R_\varphi, F_{\varphi_1}\rangle$ such that :

- S_φ is defined as follows: Let S' be the set of new intermediary states : for each $s_0 \in S_{0_{\varphi_1}} \cap S_{predN_{\varphi_1}}$, we associate a new intermediary state s'. We have $S_\varphi = S_{\varphi_1} \cup S' \setminus \{s_{NEG}, s_{TRUE}\}$
- L_φ is the state-extension of L_{φ_1} with $S_\varphi = S_{\varphi_1} \cup S'$, such that $\forall p \in AP_{\varphi_1}, \forall s' \in S'$ corresponding to a state $s_0 \in S_{0_{\varphi_1}}$ $l(s',p) = l(s_0,p)$ and $l(s',\overline{p}) = l(s_0,\overline{p})$
- R_φ is defined such that :
  - for each state $s \in S_{\varphi_1} \cap S_{predT_{\varphi_1}}$, remove the transition from s to s_{TRUE}, and add a new transition from s to each initial state.
  - for each state $s \in S_{0_{\varphi_1}} \cap S_{predN_{\varphi_1}}$, a corresponding state s' has been created. Remove transition from s to s_{NEG} and add a transition from s to s'.
  - for each newly created intermediary state s' from s_0, for each transition $R_{\varphi_1}(s_0,s)$, add a transition from s' to each s.
- $F_\varphi = S_\varphi$.

$\varphi = \mathbf{EG}\varphi_1$, let $K_{\varphi_1} = \langle AP_{\varphi_1}, S_{\varphi_1}, S_{0_{\varphi_1}}, L_{\varphi_1}, R_{\varphi_1}, (S_{0_{\varphi_1}} \cap S_{predT_{\varphi_1}}) \cup \{s_{NEG}\}\rangle$ be the AKS representing φ_1. We construct K_φ by changing S_{φ_1} into S_φ and R_{φ_1} into R_φ as follows : first remove state s_{TRUE}, and if state s_{NEG} were not present in S_{φ_1}, add it. Then for each state $s \in S_{predT_{\varphi_1}}$, remove transition from s to s_{TRUE}, add a transition from s to each initial state (in $S_{0_{\varphi_1}}$) and a transition from s to s_{NEG}.

$\varphi = \mathbf{A}[\varphi_1 \mathbf{U} \varphi_2]$,
let $K_{\varphi_1} = \langle AP_{\varphi_1}, S_{\varphi_1}, S_{0_{\varphi_1}}, L_{\varphi_1}, R_{\varphi_1}, F_{\varphi_1}\rangle$, $K_{\varphi_2} = \langle AP_{\varphi_2}, S_{\varphi_2}, S_{0_{\varphi_2}}, L_{\varphi_2}, R_{\varphi_2}, F_{\varphi_2}\rangle$ be the AKS representing φ_1 and φ_2,
let $K_{\varphi_1\wedge\varphi_2} = \langle AP_{\varphi_1\wedge\varphi_2}, S_{\varphi_1\wedge\varphi_2}, S_{0_{\varphi_1\wedge\varphi_2}}, L_{\varphi_1\wedge\varphi_2}, R_{\varphi_1\wedge\varphi_2}, F_{\varphi_1\wedge\varphi_2}\rangle$ be the AKS representing $\varphi_1 \wedge \varphi_2$. $K_\varphi = \langle AP_\varphi, S_\varphi, S_{0_\varphi}, L_\varphi, R_\varphi, F_\varphi\rangle$ is built such that :

- $AP_\varphi = AP_{\varphi_1} \cup AP_{\varphi_2}$
- S_φ is defined as follows: Let S' be the set of new intermediary states: for each $s_0 \in S_{0_{\varphi_1}} \cap S_{predN_{\varphi_1}}$, we associate a new intermediary state s'. We have $S_\varphi = S_{\varphi_1} \cup S' \cup S_{\varphi_2} \cup S_{\varphi_1\wedge\varphi_2}$.
- $S_{0_\varphi} = S_{0_{\varphi_1}} \cup S_{0_{\varphi_2}} \cup S_{0_{\varphi_1\wedge\varphi_2}}$

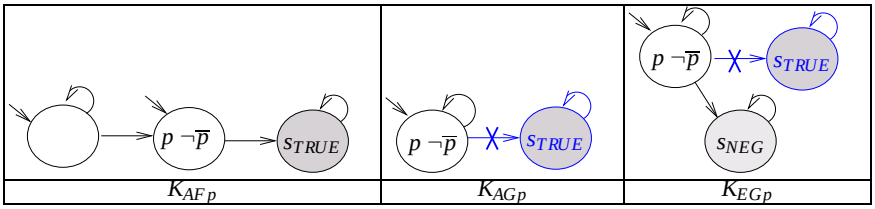

Fig. 3. The three AKS representing K_{AFp}, K_{AGp}, K_{EGp} respectively. State s_{TRUE} is removed in K_{AGp} and K_{EGp}

- $L_\varphi = L_{\varphi_1}^{+AP_2} \cdot L_{\varphi_2}^{+AP_1}$ is the composition of atomic proposition-and state-extensions and all states $s' \in S'$ are such that : $l(s',p) = l(s_0,p)$ and $l(s',\overline{p}) = l(s_0,\overline{p})$
- R_φ is defined as the union of R_{φ_2}, $R_{\varphi_1 \wedge \varphi_2}$ and R'_φ. The latest transition set is obtained by the following transformations of R_{φ_1} :
  1. for each state $s \in S_{\varphi_1} \cap S_{predT_{\varphi_1}}$, remove transition from s to s_{TRUE}, and add a new transition from s to each initial state of $S_{0_{\varphi_1}}$.
  2. for each state $s \in S_{\varphi_1} \cap S_{predN_{\varphi_1}}$, a corresponding state s' has been created. Remove transition from s to s_{NEG} and add a transition from s to s'.
  3. for each newly created intermediary state s', add a transition from s' to each initial state in $S_{0_{\varphi_1}}$.
  4. from each state $s \in S_{\varphi_1} \setminus S_{predT_{\varphi_1}} \cup S'$ add a transition from s to each initial state in $S_{0_{\varphi_1 \wedge \varphi_2}}$
  5. from each states $s \in S_{\varphi_1} \cap S_{predT_{\varphi_1}}$ add a transition from s to each initial state in $S_{0_{\varphi_2}}$.
- $F_\varphi = F_{\varphi_2} \cup F_{\varphi_1 \wedge \varphi_2}$

$\varphi = \mathbf{E}[\varphi_1 \mathbf{U} \varphi_2]$,
let $K_{\varphi_1} = \langle AP_{\varphi_1}, S_{\varphi_1}, S_{0_{\varphi_1}}, L_{\varphi_1}, R_{\varphi_1}, F_{\varphi_1} \rangle$, $K_{\varphi_2} = \langle AP_{\varphi_2}, S_{\varphi_2}, S_{0_{\varphi_2}}, L_{\varphi_2}, R_{\varphi_2}, F_{\varphi_2} \rangle$ be the AKS representing φ_1 and φ_2,
let $K_{\varphi_1 \wedge \varphi_2} = \langle AP_{\varphi_1 \wedge \varphi_2}, S_{\varphi_1 \wedge \varphi_2}, S_{0_{\varphi_1 \wedge \varphi_2}}, L_{\varphi_1 \wedge \varphi_2}, R_{\varphi_1 \wedge \varphi_2}, F_{\varphi_1 \wedge \varphi_2} \rangle$ be the AKS representing $\varphi_1 \wedge \varphi_2$. We built $K_\varphi = \langle AP_\varphi, S_\varphi, S_{0_\varphi}, L_\varphi, R_\varphi, F_\varphi \rangle$ such that

- $AP_\varphi = AP_{\varphi_1} \cup AP_{\varphi_2}$
- $S_\varphi = S_{\varphi_1} \cup S_{\varphi_1 \wedge \varphi_2} \cup S_{\varphi_2} \cup \{s_{NEG}\}$
- $S_{0_\varphi} = S_{0_{\varphi_1}} \cup S_{0_{\varphi_2}} \cup S_{0_{\varphi_1 \wedge \varphi_2}}$
- $L_\varphi = L_{\varphi_1}^{+AP_2} \cdot L_{\varphi_2}^{+AP_1}$
- R_φ is defined as the union of all transitions of R_{φ_2} and $R_{\varphi_1 \wedge f_2}$, $\{R(s_{NEG}, s_{NEG})\}$ and R'_φ. The latest transition set is R_{φ_1} with the following transformations :
  1. $\forall s \in S_{\varphi_1} \cap S_{predT_{\varphi_1}}$, remove transitions from a state s to s_{TRUE} and add a transition from s to all states in $S_{0_{\varphi_1}} \cup S_{0_{\varphi_2}} \cup \{s_{NEG}\}$.
  2. $\forall s \in S_{\varphi_1} \setminus S_{predT_{\varphi_1}}$ add a transition from s to each initial state of $S_{0_{\varphi_1 \wedge \varphi_2}}$
- $F_\varphi = F_{\varphi_2} \cup F_{\varphi_1 \wedge \varphi_2} \cup \{s_{NEG}\}$

$\varphi = \mathbf{A}[\varphi_1 \mathbf{W} \varphi_2]$ proceeds as $A[\varphi_1 U \varphi_2]$ except that the set of states S_{φ_1} is not a part of the fairness of K_φ.

$\varphi = \mathbf{E}[\varphi_1 \mathbf{W} \varphi_2]$ proceeds as $E[\varphi_1 U \varphi_2]$ except that the set of states S_{φ_1} is not a part of the fairness of K_φ.

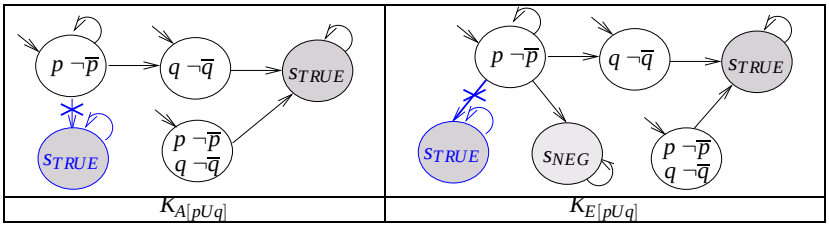

Fig. 4. The two AKS representing $K_{A[pUq]}$ and $K_{E[pUq]}$ respectively

Definition 11. *Let $C = \langle K, P, A\rangle$ be a 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. 10.*

4 AKS's properties

4.1 Properties of AKS_φ

This section deals with the relation between a formula φ expressed in a positive normal form of CTL$\setminus$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 definition 10 is such that $\forall s_0 \in S_{0_\varphi}, K_\varphi, s_0 \models \varphi$.

Proof. (Sketch)
The complete proof is given in appendix The proof proceeds by induction over the length of the formula.

As in [1, 20], interpretation vector 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 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.

4.2 Composition of component abstractions

After having abstracted all components with a subset of 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. 9). 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 [15].

Assumption : We can compose a component C_i with other components $C_k \dots C_m$ if and only if the set of specification of components $C_k \dots C_m$ implies the assumption of C_i: $\bigcup_{j=k\dots m} P_j \implies A_i$ [18].

Property 2. 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 $\Sigma = C_1 || \dots C_n$ be a concrete system and $\Sigma_A = C_{\varphi_1} || \dots C_{\varphi_n}$ be the abstract model, then Σ_A simulates Σ.

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

$$\frac{\begin{array}{rl} K_1 \parallel K_{A_1} \preceq K_1 & \text{(property of synchronous composition)} \\ K_1 \preceq K_{\varphi_1} & (\preceq : \text{simulation relation and preorder}) \\ K_2 \parallel K_{A_2} \preceq K_2 & \\ K_2 \preceq K_{\varphi_2} & \\ K_{A_1} \preceq K_{\varphi_2} & \\ K_{A_2} \preceq K_{\varphi_1} & \end{array}}{K_1 \parallel K_2 \preceq K_{\varphi_1} \parallel K_{\varphi_2}}$$

4.3 Model checking 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".

To this purpose, we extend the possible truth values **true** and **false**, with the values "unknown" (**unkw**) and "inconsistent" (**inct**). The value "unknown" is assigned when in a state s, neither p nor $\overline{p}$ holds. The value "inconsistent" is associated with a state s where both p and $\overline{p}$ hold. In these cases we can not state whether $s \models p$ or $s \not\models p$. We have the following rules :

- $[s \models p] =$ **true** if $l(s,p) =$ **true** and $l(s,\overline{p}) =$ **false**
- $[s \models p] =$ **false** if $l(s,p) =$ **false** and $l(s,\overline{p}) =$ **true**
- $[s \models p] =$ **unkw** if $l(s,p) =$ **false** and $l(s,\overline{p}) =$ **false**
- $[s \models p] =$ **inct** if $l(s,p) =$ **true** and $l(s,\overline{p}) =$ **true**

By definition 9 and 10, the last item does not appear in the AKS.

We choose the Kleene strong 3-valued propositional logic like Bruns and Godefroid, so we can use the model checking algorithm described in [1]. The truth ordering, denoted by $<$, over these values is **false** $<$ **unkw** $<$ **true**.

The model checking algorithm proposed in [1] is based on a classical 2-valued model checker algorithm. The abstract Kripke structure is checked twice. First check the structure with a *pessimistic* choice in which the "unknown" value is understood as **false**. Then check with an *optimistic* interpretation in which the "unknown" value is understood as **true**. The model checker result is a pair of truth valued over 2-valued interpreted in 3-valued sense described as follows. We denote $mc = (x,y)$ the model checker result, x is the pessimist value and y the optimist one.

$$[K,s \models \psi] = \begin{cases} \textbf{true} \ \textit{iff} \ mc = (\textbf{true},\textbf{true}) \\ \textbf{false} \ \textit{iff} \ mc = (\textbf{false},\textbf{false}) \\ \textbf{unkw} \ \textit{iff} \ mc = (\textbf{false},\textbf{true}) \end{cases}$$

Property 3. Model checking ACTL formulas on abstract system is conservative
Let Σ be a concrete system and Σ_A be the abstract system obtained by abstracting some of (or all) components of Σ. For all formulae ψ in ACTL (positive CTL formulae with only the universal path quantifier), $[\Sigma_A \models \psi] = \textbf{true} \Rightarrow [\Sigma \models \psi] = \textbf{true}$

Proof. (Sketch) Directly follows from the existence of a simulation relation between a concrete and abstract Kripke structures, and property 2.

5 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*.

Our system interconnects VCI compliant components ([10]) through a physical PI-bus ([16]). Our aim is to verify with model checker VIS ([11]) that the overall architecture correctly transmits messages with successive translations, from VCI to PI and from PI to VCI in both dierections.

Using such devices, we are able to connect various VCI compliant-cores through a PI-bus. The simplest architecture is shown in Fig. 5. 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. 5.

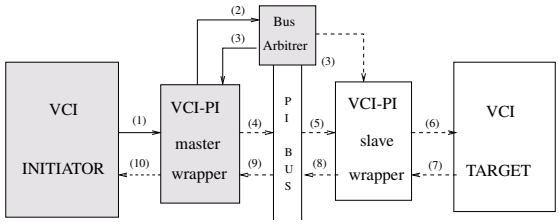

Fig. 5. The architecture performing the VCI-PI-VCI translation and illustration of a VCI transfer

Here we focused 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. 5) 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 1 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.

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	540	308	21 390	3,64E+06	35.82s	1	0,82s
						2	0,74s
						3	0,92s
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	705	453	134 233	2,45E+11	1626,76s	1	423s
						2	597s
						3	331s
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 1. 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 1GO of RAM

We are currently working on an integration of our algorithm into the model checker VIS (automatizing the CEGAR verification process)([11]). 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 Nicolas Halbwachs and Doron Peled, editors, *CAV*, volume 1633 of *LNCS*, pages 274–287. Springer, 1999.
2. G. Bruns and P. Godefroid. Generalized model checking: Reasoning about partial state spaces. In Catuscia Palamidessi, editor, *CONCUR*, volume 1877 of *LNCS*, pages 168–182. Springer, 2000.
3. J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. 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? In Dominique Borrione and Wolfgang J. Paul, editors, *CHARME*, 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. Softw. Eng. Methodol.*, 12(4):371–408, 2003.
6. E.M. Clarke. Counterexample-guided abstraction refinement. In *TIME*, page 7. IEEE Computer Society, 2003.

7. E.M. Clarke, O. Grumberg, and D.A. Peled. *Formal Verifi cation of an IBM CoreConnect Processor Local Bus Arbiter Core*. The MIT Press, Cambridge, Massachusetts, 1999.
8. D. Dams and K. S. Namjoshi. Automata as abstractions. In Radhia Cousot, editor, *VMCAI*, volume 3385 of *LNCS*, pages 216–232. Springer, 2005.
9. A. Goel and W. R. Lee. Formal Verification of an IBM CoreConnect Processor Local Bus Arbiter Core. In *DAC*, pages 196–200, 2000.
10. On-Chip Bus Development Working Group. *Virtual Component Interface Standard (VCI).* VSI Alliance, 2000.
11. The VIS group. VIS : A System for Verification and Synthesis. In Rajeev Alur and Thomas A. Henzinger, editors, *CAV*, volume 1102 of *LNCS*, pages 428–432, New Brunswick, NJ, USA, 1996. Springer-Verlag.
12. O. Grumberg and D.E. Long. Model Checking and Modular Verification. In *International Conference on Concurrency Theory*, volume 527 of *LNCS*, pages 250–263. Springer Verlag, 1991.
13. A. Gurfinkel and M. Chechik. Why waste a perfectly good abstraction?. In Holger Hermanns and Jens Palsberg, editors, *TACAS*, volume 3920 of *LNCS*, pages 212–226. Springer, 2006.
14. T. A. Henzinger, X. Liu, S. Qadeer, and S. K. Rajamani. Formal specification and verification of a dataflow processor array. In *ICCAD*, pages 494–499, 1999.
15. T.A. Henzinger, S. Qadeer, S.K. Rajamani, and S. Tasiran. An assume-guarantee rule for checking simulation. *ACM Trans. Program. Lang. Syst.*, 24(1):51–64, 2002.
16. Open Microprocessors System Initiatives. *OMI324: PI-Bus Standard Specifi cation*. Siemens, Munich, Germany, 1994.
17. O. Kupferman, M. Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *J. ACM*, 47(2):312–360, 2000.
18. K. L. McMillan. A methodology for hardware verification using compositional model checking. *Science of Computer Programming*, 37(1–3):279–309, 2000.
19. H. Peng, S. Tahar, and F. Khendek. Comparison of SPIN and VIS for Protocol Verification. *STTT*, 4(2):234–245, 2003.
20. S. Shoham and O. Grumberg. Monotonic abstraction-refinement for ctl. In Kurt Jensen and Andreas Podelski, editors, *TACAS*, volume 2988 of *Lecture Notes in Computer Science*, pages 546–560. Springer, 2004.
21. F. Xie and J. C. Browne. Verified systems by composition from verified components. In *ESEC/FSE-11*, pages 277–286, New York, NY, USA, 2003. ACM Press.

Appendix (*We do not plan to include all the proof in the final version*)

The proof of Property 1 stating that K_φ is a model of φ needs preliminary properties about the structure we obtain from Definition 10.

Property 4. All AKS without conjunction contains only fair paths from initial states to s_{TRUE} or s_{NEG} or $s \in S_0$.

Proof. Property 4
The proof proceeds by induction over the formula length. Basic cases are CTL formulae with no nested operator.

$\varphi = p$ or $\varphi = \overline{p}$, the resulting AKS contains one initial states with only one transition to s_{TRUE}.

$\varphi = p \vee q$, the resulting AKS contains two initial states and two transitions from an initial states to s_{TRUE}.

$\varphi = AFp$, the resulting AKS contains one state s in $S_{predT} \cap S_{0_{AFp}}$, and another initial state s' such that $R(s,s)$,$R(s,s') \in R_{AFp}$ and s is not in the fairness constraint. The transition from s to s' will be finally taken.

$\varphi = EFp$, the resulting AKS contains only one state s in $S_{predT} \cap S_{0_{AFp}}$, and another initial state s' such that $R(s',s')$,$R(s',s) \in R_{EFp}$ and s is not in the fairness constraint. The transition from s' to s will be finally taken. Moreover there exists a transition from s' to s_{NEG}.

$\varphi = AGp$, the resulting AKS contains a single state which is initial and have a unique transition to itself.

$\varphi = EGp$, the resulting AKS contains two states: an initial one s with a transition to itself and a state s_{NEG} such that $R(s,NEG)$.

$\varphi = A[pUq]$ the resulting AKS is built from K_p and K_q. From $s_p \in S_{0_p}$ there exists a transition to itself, this transition reach an initial state. Moreover the state s_p is not in the fairness constraint of $K_{A[pUq]}$. Thus state $s_q \in S_{0_q}$ is always reachable and there exists a transition from s_{0_q} to s_{TRUE}.

$\varphi = E[pUq]$ the resulting AKS is built from K_p and K_q. In $K_{E[pUq]}$,s_{NEG} is reachable from $s_p \in S_{0_p}$, because $s_p \in S_{predN}$, moreover there exists a transition $R(s_p,s_p)$, this transition reach an initial state. Finally, state s_p is not in the fairness constraint of $K_{E[pUq]}$. Thus state $s_q \in S_{0_q}$ is always reachable and there exists a transition from s_{0_q} to s_{TRUE}.

Induction Hypothesis: all AKS representing formula with no conjunction contains only fair path from initial states to s_{TRUE} or $s \in S_0$.

$\varphi = f_1 \wedge f_2$, the resulting AKS is the parallel composition of K_{f_1} and K_{f_2}. (cf. prop 5)

$\varphi = AFf$, the resulting AKS is build from K_f all fair path in K_f are preserves in K_{AFf}. A new initial states s is added with a transition to itself, one transition to each initial state of K_f and s is not in the fairness constraint of K_{AFf}. Thus from all initial state of K_{AFf} there exists a fair path to all initial state of K_f and by induction hypothesis they verifies the property. No other paths are added.

$\varphi = EFf$, the resulting AKS is build from K_f all fair path in K_f are preserves in K_{EFf}. The behavior added contains only a new state s which do not belong to the fairness constraint of K_{EFf}. The added transition are one to s_{NEG} and one to each initial state of K_f The fairness constraint insure that a path from s will always take a transition to s_{NEG} or an initial state of K_f. Thus by induction hypothesis K_{EFf} verifies the property.

$\varphi = AGf$, f is not a conjunction of formulae. The resulting AKS is build from K_f all fair path in K_f are preserves in K_{AGf}. From all intermediary states s (in S') built from s, s' has got the same outgoing transitions as s and belong to the set of fairness. From all state in S_{predT} the transition is removed and a transition to initial states are added. No other transition are added, thus K_{AGf} verifies the property.

$\varphi = EGf$, f is not a conjunction of formulae. The resulting AKS is build from K_f all fair path in K_f are preserves in K_{EGf}. All state that reach s_{TRUE} in K_f, reach an initial state and s_{NEG} after performing the transformation. No other transition are added, thus K_{EGf} verifies the property.

$\varphi = A[f_1 U f_2]$, f_1 is not a conjunction of formulae. The resulting AKS is build from K_{f_1}, K_{f_2} and $K_{f_1 \wedge f_2}$, all fair path in these AKS are preserves in $K_{A[f_1 U f_2]}$. All states of

S_{f_1} are not in the fairness constraint of $K_{A[f_1 U f_2]}$ and by construction we add a transition from all states of S_{f_1} to initial states of K_{f_2} or $K_{f_1 \wedge f_2}$. By induction hypothesis the property holds in K_{f_2} and $K_{f_1 \wedge f_2}$, thus the property holds in $K_{A[f_1 U f_2]}$.

$\varphi = E[f_1 U f_2]$, f_1 is not a conjunction of formulae. The resulting AKS is build from K_{f_1}, K_{f_2} and $K_{f_1 \wedge f_2}$, all fair path in these AKS are preserves in $K_{E[f_1 U f_2]}$. All states of S_{f_1} are not in the fairness constraint of $K_{E[f_1 U f_2]}$ and by construction we add a transition from all states of S_{f_1} to initial states of K_{f_2} or $K_{f_1 \wedge f_2}$ or s_{NEG}. By induction hypothesis the property holds in K_{f_2} and $K_{f_1 \wedge f_2}$, thus the property holds in $K_{E[f_1 U f_2]}$.

Property 5. Let $K_{f \| g}$, K_f and K_g be three AKS, the have (1) $\Rightarrow$ (2) :

1. All paths $\pi = s_0 \rightarrow s_1 \ldots$ and $\pi' = s'_0 \rightarrow s'_1 \ldots$ are fair paths in K_f and K_g reaching s_{TRUE}, s_{NEG}, and respectively $s \in S_{0_f}$, $s' \in S_{0_g}$.
2. All paths $\pi'' = (s_0, s'_0) \rightarrow (s_1, s'_1) \ldots$ are fair paths in $K_{f \| g}$ reaching s_{TRUE}, s_{NEG}, or a composed state s such that one of its component is in $S_{0_f} \cup S_{0_g}$.

Proof. Property 5
Assume that there exists a path in $K_{f \| g}$, $\pi'' = (s_0, s'_0) \rightarrow (s_1, s'_1) \ldots$ such that $s_i \notin S_{0_f}$ or $s_i \neq s_{TRUE}$ or $s_0 = s_{NEG}$ for all $i \geq 0$. By definition of composition $\pi = s_0 \rightarrow s_1 \ldots$ is a path in K_f and this is a path never reached s_{TRUE}, s_{NEG} or $s_f \in S_{0_f}$. Similarly $\pi' = s'_0 \rightarrow s'_1 \ldots$ and this is a path never reached s_{TRUE}, s_{NEG} or $s_g \in S_{0_g}$. This contradicts (1).

The proof can be easily extended with a composition for any number of AKS.

Property 6. An AKS contains only state s such that

- $s = s_{TRUE}$ or $s = s_{NEG}$.
- $s \in S_0$, s is an initial state.
- $s' \in S'$, where S' is a set of intermediary state such that s' is built from s and has the same labeling function and same outgoing transitions as s.
- s is a state obtained by a composition of states (s_f,s_g) and s_f, s_g are in $\{s_{TRUE}\} \cup \{s_{NEG}\} \cup S_{0_g} \cup S_{0_f} \cup S'_f \cup S'_g$.

Proof. (Sketch) Property 6
The proof proceed by induction over the formula. All operator preserves the previously existing initial state and may add some new or some intermediary state. In case of composition of K_f and K_g one has to prove that all state obtained are composed with state of S_f and S_g that verifies the property.

Proof. Property 1: $\forall s_0 \in S_{0_\varphi}$, $K_\varphi, s_0 \models \varphi$.
The proof proceeds by induction over the formula length. Basic cases are CTL formulae with no nested operator.

$\varphi = p$ or $\varphi = \overline{p}$, the resulting AKS contains one initial state s_0 such that $s_0 \models p$ and $s_0 \not\models \neg p$.

$\varphi = p \wedge q$, the resulting AKS is build from K_p and K_q. The initial state s_0 is composed by s_{0_p} and s_{0_q}. Thus $s_0 \models p$, $s_0 \models q$ and $s_0 \not\models \neg p$, $s_0 \not\models \neg q$. The formula φ holds for the single initial state of $K_{p \wedge q}$.

$\varphi = p \vee q$, the resulting AKS contains two initial states s_0 ans s'_0 such that

- $s_0 \models p$ and $s_0 \not\models \neg p$ thus $s_0 \models p \vee q$.
- $s'_0 \models q$ and $s'_0 \not\models \neg q$ thus $s'_0 \models p \vee q$.

The formula φ holds for all initial state of the resulting AKS.

$\varphi = AFp$, the resulting AKS K_{AFp} contains exactly two initial states s_0 and s'_0 such that

- $s_0 \models p$ and $s_0 \not\models \neg p$ thus $s_0 \models AFp$.
- $s'_0 \not\models p$ and $s_0 \not\models \neg p$, there exists two outgoing transitions from s'_0, one to itself and one to s_0. As s'_0 is not in the fairness F_{AFp} s_0 will always be reach, thus $s'_0 \models AFp$

The formula φ holds for all initial state of K_{AFp}.

$\varphi = EFp$, the resulting AKS K_{EFp} contains two initial states s_0 and s'_0 such that

- $s_0 \models p$ and $s_0 \not\models \neg p$ thus $s_0 \models EFp$.
- $s'_0 \not\models p$ and there exists three outgoing transitions from s'_0, one to itself, one to s_{NEG} and one to s_0. As s'_0 is not in the fairness F_{AFp} and there exists a transition reaching s_0, thus $s'_0 \models EFp$

The formula φ holds for all initial state of K_{EFp}.

$\varphi = AGp$, the resulting AKS is build from K_p such that initial state s_0 is the initial state of K_{AGp}. Thus p hold in the initial state of K_{AGp}. Furthermore s_{TRUE} do not belong to S_{AGp} and there exists only one transition from s_0 to s_0. The formula φ holds for the single initial state of K_{AGp}.

$\varphi = EGp$, the resulting AKS is build from K_p such that initial state s_0 is the initial state of K_{EGp}. Thus p hold in the initial state of K_{EGp}. From s_0 there exists two outgoing transitions: one to itself and one to s_{NEG}. s_0 belong to the fairness constraint, thus there exists an infinite path composed with state s_0 only where p always holds, and another where s_{NEG} is reached where p may does not hold. The formula φ holds for the single initial state of K_{EGp}.

$\varphi = A[pUq]$, the resulting AKS is built from K_p and K_q. The initial state of $K_{A[pUq]}$ is exactly $\{s_0 \in S_p\} \cup \{s'_0 \in S_q\} \cup \{s''_0 \in S_{p \wedge q}\}$.

- $s'_0 \models q$, thus $s'_0 \models A[pUq]$.
- $s''_0 \models p$ and $s''_0 \models q$, thus $s'_0 \models A[pUq]$.
- $s_0 \models p$ and there exists two outgoing transitions from s_0 : one to itself and one to s'_0. Moreover, s_0 is not in the fairness constraint of $K_{A[pUq]}$ thus state s'_0 will always be reached and $s_0 \models A[pUq]$.

The formula φ holds for all initial states of $K_{A[pUq]}$.

$\varphi = E[pUq]$, the resulting AKS is built from K_p and K_q. The initial state of $K_{A[pUq]}$ is exactly $\{s_0 \in S_p\} \cup \{s'_0 \in S_q\} \cup \{s''_0 \in S_{p \wedge q}\}$.

- $s'_0 \models q$, thus $s'_0 \models E[pUq]$.
- $s''_0 \models p$ and $s''_0 \models q$, thus $s'_0 \models E[pUq]$.
- $s_0 \models p$ and there exists three outgoing transitions from s_0 : one to itself, one to s_{NEG} and one to s'_0. s_0 is not in the fairness constraint of $K_{E[pUq]}$ thus there exists one path leading to state s'_0 and one path reaching NEG. We have $s_0 \models E[pUq]$.

The formula φ holds for all initial states of $K_{E[pUq]}$.

Induction Hypothesis : let K_f and K_g be two AKS such that f holds for all initial state in S_{0_f} and g holds for all initial state in S_{0_g} .

$\varphi = f \vee g$, the resulting AKS is build from K_f and K_g such that $S_{0_{f\vee g}} = S_{0_f} \cup S_{0_g}$. By induction hypothesis we have

- $\forall s_f \in S_{0_f}, s_f \models f$, thus $s_f \models f \vee g$
- $\forall s_g \in S_{0_g}, s_g \models g$, thus $s_g \models f \vee g$

f hold for all initial state of $K_{f\vee g}$.

$\varphi = f \wedge g$, the resulting AKS is build from K_f and K_g by parallel composition and $S_{f\vee g} = (S_{0_f} \times S_{0_g})$ By induction hypothesis f holds in all initial state in S_{0_f} and g holds in all initial state in S_{0_g}. By property 5 the parallel composition preserves the fair path in both structure, thus $f \vee g$ holds for all initial state of $K_{f\|g}$. $\varphi = AF\,f$, the resulting AKS is build from K_f in which a new initial state s_0 is added. There exists a transition from s_0 to itself and a transition from s_0 to all initial state of K_f. Moreover s_0 is not in the fairness constraint of $K_{AF\,f}$, thus all path from s_0 will always reach an initial state of K_f. By induction hypothesis f holds for all initial state of K_f thus $s_0 \models AF\,f$. Moreover, f holds for all initial state that belonging to K_f, thus $AF\,f$ holds for all initial state of $K_{AF\,f}$.

$\varphi = EF\,f$, the resulting AKS is build from K_f in which a new initial state s_0 is added. There exists transitions from s_0 to itself, to s_{NEG} and to all initial state of K_f. Moreover s_0 is not in the fairness constraint of $K_{EF\,f}$, thus all path from s_0 will always reach an initial state of K_f or s_{NEG}. By induction hypothesis f holds for all initial state of K_f thus $s_0 \models EF\,f$. Moreover, f holds for all initial state that belonging to K_f, thus $EF\,f$ holds for all initial state of $K_{EF\,f}$.

$\varphi = AGf$, the resulting AKS is build from K_f and $S_{0_{AGf}} = S_{0_f}$. By property 6 all state in S_{AGf} are initial, intermediary state or composed state.

1. By induction hypothesis, f holds in all state $s_0 \in K_{AGf}$.
2. All intermediary states s' introduce in K_{AGf} are obtained from $s_0 \in S_{0_f} \cap S_{predN}$ and have the same transition relation as s_0. Thus f holds in s'.
3. All composed state comes from a nested until operator, the premises of the until formulae holds in all such state, by property 5 all paths in K_{AGf} reach an initial state. Thus f holds in all composed state.

$\varphi = A[fUg]$, the resulting AKS is built from K_f, K_g and $K_{f\|g}$ and $S_{0_{A[fUg]}} = S_{0_f} \cup S_{0_g} \cup S_{0_{f\|g}}$.

- g holds in all initial state $s_g \in S_{0_g}$, thus $A[fUg]$ holds for all s_g.
- g and f holds in all initial states $s_{f\|g} \in S_{0_{f\|g}}$,thus $A[fUg]$ holds for all $s_{f\|g}$.
- f holds in all initial state $s_f \in S_{0_f}$, and all states are not in the fairness constraint of $K_{A[fUg]}$.
  - For all state $s_f \in S_{0_f} \cap S_{predT}$ there exists an outgoing transition to each initial state of S_{0_g}, $A[fUg]$ holds for all s_f.
  - For all state $s'_f \in S_{0_f} \setminus S_{predT}$ there exists an outgoing transition to each initial state of $S_{0_{f\|g}}$, $A[fUg]$ holds for all s'_f.

- For all intermediary state s''_f, we prove that $s'_f \models f$ by applying the same reasoning as operator G. By property 4 there exists a fair path reaching the initial state s_f and s'_f, $A[fUg]$ holds for all s''_f.

$\varphi = E[fUg]$, the resulting AKS is built from K_f, K_g and $K_{f\|g}$ and $S_{0_{E[fUg]}} = S_{0_f} \cup S_{0_g} \cup S_{0_{f\|g}}$.

- g holds in all initial state $s_g \in S_{0_g}$, thus $E[fUg]$ holds for all s_g.
- g and f holds in all initial states $s_{f\|g} \in S_{0_{f\|g}}$,thus $E[fUg]$ holds for all $s_{f\|g}$.
- f holds in all initial state $s_f \in S_{0_f}$, and all states are not in the fairness constraint of $K_{A[fUg]}$.
  - For all state $s_f \in S_{0_f} \cap S_{predT}$ there exists an outgoing transition to each initial state of S_{0_g}, $E[fUg]$ holds for all s_f.
  - For all state $s'_f \in S_{0_f} \setminus S_{predT}$ there exists an outgoing transition to each initial state of $S_{0_{f\|g}}$ and one to s_{NEG}, $E[fUg]$ holds for all s'_f.
  - For all intermediary state s''_f, we prove that $s'_f \models f$ by applying the same reasoning as operator G. By property 4 there exists a fair path reaching the initial state s_f and s'_f, $A[fUg]$ holds for all s''_f.