Symbolic Model Checking of Logics with Actions

Charles Pecheur[1,⋆] and Franco Raimondi[2,⋆⋆]

[1] Université catholique de Louvain
charles.pecheur@uclouvain.be
[2] University College London
f.raimondi@cs.ucl.ac.uk

Abstract. Reasoning about agents and modalities such as knowledge and belief leads to models where different relations over states co-exist, or equivalently, where information (labels, actions) is associated to state transitions. This paper discusses how to augment classical CTL symbolic model-checking to support logics with actions such as A-CTL (action-CTL), and how this can be implemented using BDDs in tools such as the SMV/NuSMV package. Considering general action-state structures, we first propose a natural extension of CTL to actions, called Action-Restricted CTL (ARCTL) and adapt classical results from CTL to express model checking based on three functions eax, eau and eag. On these grounds, we present two different implementations of symbolic model checking with actions. The first approach encodes action-state models and logics into pure state-based models and logics, that can be checked with existing model-checkers. The second approach consists in a native implementation of the three extended operators. We report on our prototype implementation of both approaches based on NuSMV and give an overview of how this is used to model-check the temporal epistemic logic CTLK.

1 Introduction

In the domains of artificial intelligence and multi-agent systems, it is natural to reason about both actions *and* states. Moreover, a number of modalities, such as epistemic or deontic, can be formalized in terms of relations over the states of a system or model. In this setting, it is desirable to have analysis techniques and tools where information can be associated to both states and transitions of the model, or more generally where more than one relation over states can be considered within the same model.

Symbolic model checking, and the SMV tool in particular, have adopted a state-based view of the systems to be verified, expressed mathematically as Kripke structures. Meanwhile, another large body of work has developed based on an observable, action-based view of systems, where the state itself is abstracted away and models are characterized by the visible actions they can perform, and expressed mathematically as Labeled Transition Systems (LTS). These

⋆ With RIACS at NASA Ames while performing this work.
⋆⋆ On internship with MCT at NASA Ames while performing this work.

S. Edelkamp and A. Lomuscio (Eds.): MoChart IV, LNAI 4428, pp. 113–128, 2007.

two views can be, and have been, combined. In this paper, we designate as *mixed* the models and logics that combine state-based and action-based reasoning. However, to the best of our knowledge, there is no widely available tool that allows to apply the power of BDD-based symbolic model checking of branching-time logics to mixed models and logics.

This paper discusses how BDD-based symbolic model checking for mixed logics can be achieved, and presents two different implementations:

1. by reducing the mixed-logic model-checking problem to a state-based model-checking problem that can be solved with existing tools;
2. by extending existing model-checkers to support mixed models and logics natively.

To this end, after reviewing the prominent existing state-based and action-based models (Kripke structures, labeled transition systems) and logics (CTL, A-CTL) in Section 2, we set our formal definition of mixed models, introduce a mixed logic, ARCTL, that cleanly generalizes CTL with actions, and extend symbolic model checking from CTL to ARCTL in Section 3. In Section 4, we describe how mixed models and logics can be reduced to Kripke structures and CTL while preserving validity. In Section 5, we describe a prototype implementation of both approaches based on the NuSMV tool [1]. In Section 6, we give an overview of how a model checker for the temporal epistemic logic CTLK [2] has been built on top of these implementations. Finally, Section 7 discusses related work and Section 8 draws conclusions and perspectives.

2 Background

2.1 State-Based Logics

Computation Tree Logic (CTL) [3,4] is the classical branching-time logic used in symbolic model checking. Given a set of propositional atoms $\mathcal{P}$, a CTL formula is interpreted over a *Kripke Structure* (KS) $\mathcal{K} = \langle \mathcal{S}, \mathcal{S}_0, \mathcal{R}, \mathcal{V} \rangle$, where $\mathcal{S}$ is a non-empty set of states, $\mathcal{S}_0 \subseteq \mathcal{S}$ is a set of possible initial states, $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S}$ is a transition relation, denoted $s \longrightarrow s'$, and $\mathcal{V} : \mathcal{S} \to 2^{\mathcal{P}}$ is an interpretation function.

The syntax of CTL is given by the following grammar, where $p \in \mathcal{P}$ and ϕ and γ range respectively over CTL (state) formulae and path formulae:

$$\phi ::= \mathsf{true} \mid p \mid \neg\phi \mid \phi \wedge \phi \mid \mathsf{E}\gamma \mid \mathsf{A}\gamma$$
$$\gamma ::= \mathsf{X}\,\phi \mid \phi\,\mathsf{U}\,\phi$$

with the usual derived Boolean operators and the following derived temporal operators:

$$\mathsf{EF}\,\phi = \mathsf{E}[\mathsf{true}\,\mathsf{U}\,\phi] \quad \mathsf{AF}\,\phi = \mathsf{A}[\mathsf{true}\,\mathsf{U}\,\phi]$$
$$\mathsf{EG}\,\phi = \neg\mathsf{AF}\,\neg\phi \quad \mathsf{AG}\,\phi = \neg\mathsf{EF}\,\neg\phi$$

Note that temporal logic operators take precedence over Boolean connectives: $\mathsf{EX}\,\phi \vee \phi' = (\mathsf{EX}\,\phi) \vee \phi' \neq \mathsf{EX}\,(\phi \vee \phi')$. The semantics of a CTL formula ϕ is

defined as a satisfaction relation $s \models \phi$ over states $s \in \mathcal{S}$, see for example [4]. We postpone the detailed definition of semantics to mixed logics in Section 3.2.

All CTL operators can be reduced to EX, EU and EG. Symbolic model checking of CTL over finite models, as implemented in the SMV family of tools, works by providing BDD-based evaluation functions $ex(S)$, $eu(S, S')$ and $eg(S)$, where S, S' are Boolean encodings of sets of states, that compute the semantics of the corresponding operators. In the case of eu and eg, this means computing fixpoints over ex (which are guaranteed to converge thanks to finiteness of the model), for example $eg(S) = \nu Z.S \cap ex(Z)$ [4].

CTL symbolic model checking has been extended to support *fairness*, in the form of a set of conditions that characterize fair computations. Fairness is not taken into consideration within the scope of this paper, but the issue is nevertheless discussed in Section 3.4.

2.2 Action-Based Logics

In contrast to state-based logics such as CTL, *action-based logics* focus on the actions that a system can perform. These logics are interpreted over *labeled transition systems* (LTS). A LTS is a structure $\mathcal{L} = \langle \mathcal{S}, \mathcal{S}_0, \mathcal{A}, \mathcal{T} \rangle$, where $\mathcal{S}$ and $\mathcal{S}_0$ are as in Kripke structures, $\mathcal{A}$ is a set of actions and $\mathcal{T} \subseteq \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ is a labeled transition relation. We write $s \xrightarrow{a} s'$ for $(s, a, s') \in \mathcal{T}$, $s \not\xrightarrow{a}$ when no such s' exists and $s \not\longrightarrow$ when no such a and s' exist.

For example, *Action CTL*, or A-CTL [5], is an adaptation of CTL to labeled transition systems.[1] A-CTL extends CTL operators with action formulae α interpreted over actions $a \in \mathcal{A}$. For example, the A-CTL formula $\mathsf{A}[\phi_\alpha \mathsf{U}_{\alpha'} \phi']$ holds if all paths are of the form

$$s_0 \xrightarrow{a_1} s_1 \cdots \xrightarrow{a_k} s_k \xrightarrow{a'} s'$$

for some k, where all s_i, all a_i, s' and a' respectively satisfy ϕ, α, ϕ' and α'. The full syntax and semantics of A-CTL, defined over LTS, can be found in [5]. [6] provides a comprehensive survey of temporal logics with actions, and A-CTL in particular, including fixpoint characterizations of A-CTL operators.

Note that classical action-based models also feature a distinguished internal action. We do not deal explicitly with internal actions in the scope of this paper; our definitions correspond to a "strong" interpretation that treats all actions uniformly.

3 Mixing States and Actions

In this section, we set our formalisation of mixed models and formulae, that combine state-based and model-based reasoning.

[1] Action CTL is usually abbreviated ACTL, but that could be confused with the universal fragment of CTL, unfortunately also referred to as ACTL, for example in [4].

3.1 Mixed Transition Systems

We can generalize both state-based models (Kripke structures, KS) and action-based models (Labeled Transition Systems, or LTS) into a common superstructure that we call *mixed transition system* (MTS). Given two sets of propositional atoms $\mathcal{P}_S$ and $\mathcal{P}_A$, respectively over states and actions, a *mixed transition system* over $\mathcal{P}_S$ and $\mathcal{P}_A$ is a structure $\mathcal{M} = \langle \mathcal{S}, \mathcal{S}_0, \mathcal{A}, \mathcal{T}, \mathcal{V}_S, \mathcal{V}_A \rangle$, where

- $\mathcal{S}$ is a non-empty set of states;
- $\mathcal{S}_0 \subseteq \mathcal{S}$ is the set of possible initial states;
- $\mathcal{A}$ is a non-empty set of actions;
- $\mathcal{T} \subseteq \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ is the transition relation;
- $\mathcal{V}_S : \mathcal{S} \to 2^{\mathcal{P}_S}$ is the interpretation function on states;
- $\mathcal{V}_A : \mathcal{A} \to 2^{\mathcal{P}_A}$ is the interpretation function on actions.

MTS combine actions over transitions from LTS and propositional atoms over states from KS, and add propositional atoms over actions that allow for a generalized and more uniform presentation of logic formulae over MTS models.

An MTS can be projected to a KS sub-structure $\langle \mathcal{S}, \mathcal{S}_0, \mathcal{R}, \mathcal{V}_S \rangle$, where $\mathcal{R} = \{(s, s') \mid (s, a, s') \in \mathcal{T}\}$, or an LTS sub-structure $\langle \mathcal{S}, \mathcal{S}_0, \mathcal{A}, \mathcal{T} \rangle$, and thus both state-based and action-based logics can be interpreted over an MTS.

A *path* π of $\mathcal{M}$ is a finite or infinite sequence of connected transition steps $(s_{i-1}, a_i, s_i) \in \mathcal{T}$, denoted as $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \ldots$. In particular, a zero-length path consists of a single state. Let $\mathcal{T}^*$ (resp. $\mathcal{T}^\omega$) be the set of finite (resp. infinite) paths of $\mathcal{M}$. Given a finite (resp. infinite) path $\pi = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \ldots \xrightarrow{a_n} s_n$ ($\xrightarrow{a_{n+1}} \ldots$), we define:

- $|\pi| = n$ (resp. ω), the length of a path;
- $\pi(i) = s_i$, the i-th state of π $(0 \leq i \leq |\pi|)$;
- $\pi(\bullet i) = a_i$, the i-th action of π $(1 \leq i \leq |\pi|)$.

A *full-path* is a path that is either infinite or ends in a terminal state. We define $\Pi(\mathcal{M})$ (or just Π) as the set of full-paths of $\mathcal{M}$, and $\Pi(\mathcal{M}, s)$ (or $\Pi(s)$) as the set of full-paths from state s.

$$\Pi(\mathcal{M}) := \mathcal{T}^\omega \cup \{\pi \in \mathcal{T}^* \mid (|\pi| = n \wedge \pi(n) \not\longrightarrow)\}$$
$$\Pi(\mathcal{M}, s) := \{\pi \in \Pi(\mathcal{M}) \mid \pi(0) = s\}$$

Note that unlike classical definitions of CTL model-checking, we do not enforce the transition $\mathcal{T}$ to be serial; deadlocks or refused actions are in general possible and full-paths need not be infinite. Even if $\mathcal{T}$ were required to be serial, action-based logics have to consider cases where some action a is not allowed ($s \not\xrightarrow{a}$), so deadlock states where no action is allowed ($s \not\longrightarrow$) arise as a particular case anyway.

3.2 Action-Restricted CTL

As a logic over mixed state-action models, we introduce a generalization of CTL, called *Action-Restricted CTL*, or ARCTL. ARCTL has the same temporal operators as CTL, except that they can be restricted to paths whose actions satisfy a given action formula α. The syntax of ARCTL is given by the following grammar, where $p \in \mathcal{P}_S$, $b \in \mathcal{P}_A$, and ϕ, γ and α range respectively over ARCTL (state) formulae, path formulae and action formulae:

$$\begin{aligned}\phi &::= \mathsf{true} \mid p \mid \neg\phi \mid \phi \wedge \phi \mid \mathsf{E}_\alpha\gamma \mid \mathsf{A}_\alpha\gamma \\ \alpha &::= \mathsf{true} \mid b \mid \neg\alpha \mid \alpha \wedge \alpha \\ \gamma &::= \mathsf{X}\,\phi \mid \phi\,\mathsf{U}\,\phi\end{aligned}$$

Derived forms such as $\mathsf{E}_\alpha\mathsf{F}\,\phi$ are defined as for CTL. Intuitively, given an ARCTL formula $\mathsf{E}_\alpha\gamma$, the path formula γ is evaluated over full α-prefixes of full-paths of the model. To formalize that, we define the α-restriction of a MTS $\mathcal{M} = \langle \mathcal{S}, \mathcal{S}_0, \mathcal{A}, \mathcal{T}, \mathcal{V}_S, \mathcal{V}_A\rangle$ as the structure $\mathcal{M}|_\alpha = \langle \mathcal{S}, \mathcal{S}_0, \mathcal{A}, \mathcal{T}|_\alpha, \mathcal{V}_S, \mathcal{V}_A\rangle$, where $\mathcal{T}|_\alpha = \{(s, a, s') \in \mathcal{T} \mid a \models \alpha\}$. For conciseness we write $\Pi|_\alpha$ for $\Pi(\mathcal{M}|_\alpha)$ and $\Pi|_\alpha(s)$ for $\Pi(\mathcal{M}|_\alpha, s)$. Note that, by construction, any path (or full-path) of $\mathcal{M}|_\alpha$ is a prefix of a path (or full-path) of $\mathcal{M}$. A_α and E_α are interpreted over the full-paths of $\mathcal{M}|_\alpha$, and the path formulae are defined as in standard CTL. We define the semantic relation $(\mathcal{M}, s) \models \phi$, or concisely $s \models \phi$, as follows (we omit the natural semantics of Boolean connectives and propositional atoms):

$$\begin{aligned}s &\models \mathsf{A}_\alpha\gamma \text{ iff } \forall\pi \in \Pi|_\alpha(s) \cdot \pi \models \gamma \\ s &\models \mathsf{E}_\alpha\gamma \text{ iff } \exists\pi \in \Pi|_\alpha(s) \cdot \pi \models \gamma \\ \pi &\models \mathsf{X}\,\phi \text{ iff } \underline{|\pi| \geq 1} \wedge \pi(1) \models \phi \\ \pi &\models \phi\,\mathsf{U}\,\phi' \text{ iff } \exists i \geq 0 \cdot \underline{|\pi| \geq i} \wedge \pi(i) \models \phi' \wedge \forall k \in [0, i-1] \cdot \pi(k) \models \phi\end{aligned}$$

The underlined terms pertain to finite paths. In particular, if $s \overset{\alpha}{\not\longrightarrow}$ (i.e. $s \overset{a}{\not\longrightarrow}$ for any $a \models \alpha$), then $\Pi|_\alpha(s) = \{s\}$, containing a single zero-length trace, and both $\mathsf{E}_\alpha\mathsf{X}\,\phi$ and $\mathsf{A}_\alpha\mathsf{X}\,\neg\phi$ are false for any ϕ, whereas $\neg\mathsf{E}_\alpha\mathsf{X}\,\mathsf{true}$ is true. For any ϕ, α and s, we have that one and only one of $\mathsf{E}_\alpha\mathsf{X}\phi$, $\mathsf{A}_\alpha\mathsf{X}\neg\phi$ and $\neg\mathsf{E}_\alpha\mathsf{X}\,\mathsf{true}$ holds in s. Also note that $\mathsf{E}_\alpha\mathsf{G}\,\phi$ also holds in s if there is a *finite* α-full-path from s where ϕ holds. In contrast, we can define $\mathsf{E}_\alpha\mathsf{G}^\omega\,\phi$ that holds only for infinite α-full-paths, with

$$\mathsf{E}_\alpha\mathsf{G}^\omega\,\phi = \mathsf{E}_\alpha\mathsf{G}\,(\phi \wedge \mathsf{E}_\alpha\mathsf{X}\,\mathsf{true})$$

If we restrict action formulae to $\alpha = \mathsf{true}$ and consider the Kripke substructure in $\mathcal{M}$, we obtain a semantics for CTL with finite and infinite traces. One can easily check that this semantics is consistent with the classical one for infinite traces.

Note that A-CTL can be extended to mixed models in the same manner. We chose instead to introduce ARCTL because it offers a more uniform interpretation of the action conditions: for all path formulae γ, $\mathsf{A}_\alpha\gamma$ means "for all α-full-paths, γ holds". In contrast, in A-CTL, $\mathsf{AF}_\alpha\,\phi$ is co-variant in α ("all paths

do remain α-paths *until* they eventually reach ϕ") while $\mathsf{AG}_\alpha\,\phi$ is contra-variant ("all paths, *as long as* they remain α-paths, maintain ϕ"). Besides its structural simplicity, this also makes ARCTL appropriate for cases where actions denote transition relations of a different nature, such as temporal and epistemic modalities: if some action t is used for temporal transitions, temporal properties can be expressed as t-restricted ARCTL formulae, with their usual CTL interpretation. This was indeed the initial motivation for formulating this logic, as illustrated in section 6. On the other hand, it must be noted that A-CTL is more expressive than ARCTL on pure action models: $\mathsf{A}(\phi\ {}_\alpha\mathsf{U}_{\alpha'}\ \phi')$ cannot be translated to ARCTL (unless α and α' are disjoint).

3.3 Model Checking of ARCTL

Symbolic model-checking can be applied to ARCTL in the same way as to CTL, with two extensions: (i) transitions are constrained by action formulae, and (ii) additional conditions are set to deal with finite paths. For (i), the pre-image computation embodied in the function $ex(S)$ is extended to deal with actions. For (ii), we modify the computations to deal specifically with finite paths.

Given $S, S' \in 2^{\mathcal{S}}$ and $A \in 2^{\mathcal{A}}$, we define functions $eax(A,S)$, $eau(A,S,S')$ and $eag(A,S)$ as follows:

$$\begin{aligned} eax(A,S) &= \{s \mid \exists a, s' \cdot s \xrightarrow{a} s' \wedge a \in A \wedge s' \in S\} \\ eau(A,S,S') &= \mu Z \cdot S' \cup (S \cap eax(A,Z)) \\ eag(A,S) &= \nu Z \cdot S \cap eax(A,Z) \end{aligned}$$

where we write $\mu Z.F(Z)$ (resp. $\nu Z.F(Z)$) for the least (resp. greatest) fixpoint of F. For convenience, we also define $eax(A) = eax(A, \mathcal{S})$. Whereas eax and eau exactly capture the ARCTL operators $\mathsf{E}_\alpha\mathsf{X}$and $\mathsf{E}_\alpha\mathsf{U}$, note that eag accepts infinite paths only; it corresponds to $\mathsf{E}_\alpha\mathsf{G}^\omega$ and not $\mathsf{E}_\alpha\mathsf{G}$. These functions are immediate translations of fixpoint characterizations of the corresponding operators:

$$\begin{aligned} \mathsf{E}_\alpha[\phi\ \mathsf{U}\ \phi'] &= \mu Z \cdot \phi' \vee (\phi \wedge \mathsf{E}_\alpha\mathsf{X}\ Z) \\ \mathsf{E}_\alpha\mathsf{G}^\omega\ \phi &= \nu Z \cdot \phi \wedge \mathsf{E}_\alpha\mathsf{X}\ Z \end{aligned}$$

These characterizations can be proven through a simple adaptation of similar results on CTL, see for example [4]. As an aside, they also imply that ARCTL, like CTL, belongs to the alternation-free fragment of modal μ-calculus, that can be checked in linear time w.r.t. the size of the model and formula.

As in CTL, all ARCTL operators can be expressed in terms of the three primitive computations above, as follows (where $\neg S$ stands for $\mathcal{S} \setminus S$, the complement of S). Note the underlined $eax(A)$ terms, needed for finite traces.

$$\begin{aligned} [\![\mathsf{E}_\alpha\mathsf{X}\ \phi]\!] &= eax([\![\alpha]\!], [\![\phi]\!]) \\ [\![\mathsf{A}_\alpha\mathsf{X}\ \phi]\!] &= \underline{eax([\![\alpha]\!])} \cap \neg eax([\![\alpha]\!], \neg[\![\phi]\!]) \\ [\![\mathsf{E}_\alpha(\phi\ \mathsf{U}\ \phi')]\!] &= eau([\![\alpha]\!], [\![\phi]\!], [\![\phi']\!]) \\ [\![\mathsf{A}_\alpha(\phi\ \mathsf{U}\ \phi')]\!] &= \neg eau([\![\alpha]\!], \neg[\![\phi']\!], \neg[\![\phi']\!] \cap (\neg[\![\phi]\!] \cup \underline{\neg eax([\![\alpha]\!])})) \cap \neg eag([\![\alpha]\!], \neg[\![\phi']\!]) \end{aligned}$$

The evaluation functions eax, eau and eag can be implemented in a BDD-based model checker like NuSMV, based on a Boolean encoding of $\mathcal{S}$ and $\mathcal{A}$, in a very similar way to CTL model-checking as implemented in SMV [7]. A prototype of such an implementation is described in Section 5.3.

3.4 Fairness

CTL symbolic model checking can also handle *fairness conditions*, in the form of a set of sets of states $\mathcal{F} \in 2^{2^S}$. A path is *fair* if it visits every set in $\mathcal{F}$ infinitely often. The functions ex, eg and eu have variants ex_F, eg_F and eu_F restricted to fair paths.

This approach can be extended to mixed models and ARCTL, by considering fair α-full-paths when evaluating α-restricted operators. In that setting, fairness conditions can be extended to sets of states-action pairs $\mathcal{F} \in 2^{2^{\mathcal{S}\times\mathcal{A}}}$ (this is indeed already implemented in NuSMV).

However, by their very definition, fair paths are necessarily infinite, so fair model checking does not work well at all with finite paths. If a state has no infinite path from it, then there is no fair path either, and all E formulae are false and all A formulae are true. As an extreme example, if $s \xrightarrow{a} s' \not\longrightarrow$, then $s \not\models \mathsf{EX}\,\langle\mathsf{true}\rangle$, because there is no fair path from s'.

Extending fair CTL model checking to action-based logics interpreted over finite α-full-paths is an important issue to be further investigated.

4 From Mixed to State-Based Logic

This section presents a transformation *post* from mixed transition systems to Kripke structures and from ARCTL to classical CTL, such that the combined transformation preserves validity. This provides a way to reduce action-based and mixed model-checking to standard CTL model checking, that can be performed using a tool such as SMV.

4.1 Post-Projection of Mixed Models

Given a MTS $\mathcal{M} = \langle \mathcal{S}, \mathcal{S}_0, \mathcal{A}, \mathcal{T}, \mathcal{V}_S, \mathcal{V}_A \rangle$ over $\mathcal{P}_S$ and $\mathcal{P}_A$, we define the *post-projection* as the KS $post(\mathcal{M}) = \langle \mathcal{S}', \mathcal{S}'_0, \mathcal{R}', \mathcal{V}' \rangle$ over $\mathcal{P}' = \mathcal{P}_S \cup \mathcal{P}_A$, where

- $\mathcal{S}' = \mathcal{A} \times \mathcal{S}$,
- $\mathcal{S}'_0 = \mathcal{A} \times \mathcal{S}_0$,
- $\mathcal{R}' = \{((a,s),(a',s')) \mid (s,a',s') \in \mathcal{T} \wedge a \in \mathcal{A}\}$,
- $\mathcal{V}'((a,s)) = \mathcal{V}_A(a) \cup \mathcal{V}_S(s)$.

In essence, transition labels are projected into the post-state, and $s \xrightarrow{a} s'$ becomes $(*,s) \longrightarrow (a,s')$, for any action $*$. By construction, the action atoms in $\mathcal{P}_A$ become state atoms in $\mathcal{P}'$.

4.2 Post-Projection of Action-Based Logics

As action atoms of a MTS $\mathcal{M}$ become state atoms in the KS $post(\mathcal{M})$, action-based formulae can be converted into plain CTL formulae on $post(\mathcal{M})$, with action conditions turning into state conditions. Formally, given an ARCTL state formula ϕ, we define the CTL formula $post(\phi)$ as follows:

$$\begin{aligned}
post(\mathsf{E}_\alpha \mathsf{X}\, \phi) &= \mathsf{EX}\, (\alpha \wedge post(\phi)) \\
post(\mathsf{A}_\alpha \mathsf{X}\, \phi) &= \mathsf{AX}\, (\alpha \Rightarrow post(\phi)) \wedge \mathsf{EX}\, \alpha \\
post(\mathsf{E}_\alpha(\phi\, \mathsf{U}\, \phi')) &= post(\phi') \vee (post(\phi) \wedge \mathsf{EX}\, \mathsf{E}[\alpha \wedge post(\phi)\, \mathsf{U}\, \alpha \wedge post(\phi')]) \\
post(\mathsf{A}_\alpha(\phi\, \mathsf{U}\, \phi')) &= post(\phi') \vee (post(\phi) \wedge \mathsf{EX}\, \alpha \\
&\qquad \wedge \mathsf{AX}\, \mathsf{A}[post(\phi) \wedge \mathsf{EX}\, \alpha\, \mathsf{U}\, \neg\alpha \vee post(\phi')])
\end{aligned}$$

Appropriately, the semantics is preserved by the transformation, in the sense of the generalized semantics of CTL with finite paths:

$$(\mathcal{M}, s) \models \phi \quad \text{iff} \quad (post(\mathcal{M}), s) \models post(\phi)$$

which can be proved by structural induction on ϕ. The details are tedious but the principle is straightforward.

Some sub-formulae get replicated in the transformation, so an exponential increase in the size of the transformed formula may result in the worst case. In practice however, the caching of BDD computation results largely mitigates the impact of this increase when performing symbolic model checking. In any case, this provides additional motivation for using a native implementation of action-based logics in the model checker, that avoids the redundant computations. This is the topic of the next section.

5 Action-Based Model-Checking in SMV

This section discusses how SMV has been extended to support logics with actions. We first give an overview of SMV, then we describe two different implementations of actions in SMV: the first one by implementing the *post* transformation as a pre-processing stage using the macro-processor M4, the second one by modifying the SMV tool itself to support mixed formulae in specifications.

5.1 Overview of SMV

SMV is a symbolic model checker that evaluates CTL specifications on a finite-state model described in a custom language. While SMV was initially developed at Carnegie Mellon [7], we have been using NuSMV, an open-source extended re-implementation of the tool [8]. NuSMV uses an efficient BDD library to perform symbolic model checking of formulae. (NuSMV also supports linear temporal logic and SAT-based bounded model checking.)

The latest version of NuSMV (2.2) provides partial support for action-style constructs, in the form of *input variables*. Input variables (IVARs) are not part

of states, and they are used to represent input values for models (typically, they correspond to the input lines of a circuit). Technically, these variables are existentially quantified out when computing transitions. Input variables can appear in transition relations, but they are not allowed in CTL formulae.

NuSMV uses BDDs to perform model checking of CTL formulae: given encodings of state and input variables into Boolean arrays $\underline{s}$ and $\underline{a}$, respectively, the transition relation $\mathcal{T}$ and initial states $\mathcal{S}_0$ are compiled by NuSMV into BDDs $[\![\mathcal{T}]\!](\underline{s}, \underline{a}, \underline{s}')$ and $[\![\mathcal{S}_0]\!](\underline{s})$. Then, for any CTL formula ϕ, the BDD $[\![\phi]\!](\underline{s})$ corresponding to the set of states of the model in which the formula holds is computed inductively on the formula's structure, based on (fair) implementations of the functions *ex*, *eg* and *eu* described before.

SMV supports two styles for declaring transitions: the *assignment style* is based on non-deterministic assignments of initial and next values of each variable, whereas the *constraint style* allows arbitrary conditions over variable values in consecutive states. The former is safer and more convenient for human-written models, but the latter is more flexible, especially in the context of mechanically generated models.

5.2 Post-Projection to SMV

We have implemented the *post* mapping on SMV models, in the form of a macro library for M4, a generic macro-processor included in most UNIX distributions [9]. The library provides macros supporting the two sides of the *post* transformation: models and logic formulae. Our implementation currently supports ARCTL, but adaptation to similar logics such as A-CTL would be straightforward.

The mapping of logic formulae is a straightforward application of the equations of Section 4. For example, using the macro definition

```
define('EU_A','((($2) & EX E[($1) & ($2) U ($1) & ($3)]) | ($3))')
```

$\mathsf{E}_a[p \,\mathsf{U}\, q]$ can be written as `EU_A(a,p,q)` and is expanded to

```
(((p) & EX E[(a) & (p) U (a) & (q)]) | (q))
```

Since input variables are forbidden in SMV specifications, state variables have to be used instead. It is up to the user to decide which (state) variables in the SMV model represent action and state variables of the mixed model, and consistently use them only in the appropriate parts of the ARCTL operator macros — this is not enforced by the macro package.

The expansion produces SMV constraint-style transition declarations: transitions are declared as `TRANS <tcond>`, where `<tcond>` is a transition condition with sub-terms of the form `next(<cond>)` to refer to the post-state. We provide a macro

```
define('TRANS_A','TRANS next($1) -> ($2)')
```

such that `TRANS_A(a,t)` defines a transition labeled by `a` and constrained by `t`, and expands to `TRANS next(a) -> (t)`. Again, the user must make sure that the action and state parts (`a` and `t`) of `TRANS_A(a,t)` declarations contain only his chosen action and state variables, respectively.

5.3 Action Logics in SMV

We have extended NuSMV to support ARCTL formulae. We use NuSMV's existing input variables as actions, in the sense that any valuation of input variables correspond to a different action. In other words, the action set $\mathcal{A}$ is the cross-product of the ranges of all input variables. Correspondingly, action formulae correspond to conditions over these variables.

In particular, we modified the syntax of formulae accepted by NuSMV to include ARCTL operators, as follows:

$$
\begin{aligned}
\mathit{ctlexpr} ::= {} & \ldots \text{(existing CTL forms)} \\
& \mid \texttt{EAX (} \mathit{simpleexpr} \texttt{) } \mathit{ctlexpr} \\
& \mid \texttt{EAG (} \mathit{simpleexpr} \texttt{) } \mathit{ctlexpr} \\
& \mid \texttt{EA (} \mathit{simpleexpr} \texttt{) [} \mathit{ctlexpr} \texttt{ U } \mathit{ctlexpr} \texttt{]} \\
& \mid \ldots \text{(others defined similarly)}
\end{aligned}
$$

where *simpleexpr* is a conditional expression, further restricted to contain only input variables. For example, `EA(a)[p U q]` is the concrete syntax for $\mathsf{E}_a[p\,\mathsf{U}\,q]$.

Here is an overview of the modifications that were performed on the NuSMV code base to evaluate these new operators:

- Action formulae can readily be evaluated as BDDs, in the same way as standard state formulae, without any code modification.
- We defined a new BDD function $eax(A, S)$ which implements the *eax* function of Section 3.3 over BDD-encoded sets of actions and states (A, S). This function is a fairly simple adaptation of the existing $ex(S)$ function for CTL. Technically, the function merges A and S into a (BDD-encoded) set of action-state pairs.
- Similarly, we defined BDD functions $eau(A, S, S')$ and $eag(A, S)$, implementing functions *eau* and *eag* of Section 3.3 over BDDs, based on fix-point computations using the function *eax*.
- These three BDD functions were used to compute the (BDD corresponding to the) set of states satisfying any ARCTL formula, by providing a corresponding evaluation function for each operator.
- Besides these core changes, support for the new operators had to be folded in several other modules, including of course the SMV model parser and CTL evaluation dispatch functions.

These modifications allow for the evaluation of ARCTL formulae, as illustrated in the example of Figure 1, involving two agents `bob` and `alice` who can non-deterministically select, at each time step, whether to perform an increment of their variable `count` or not. In this case, we have two Boolean input variables `alice.move` and `bob.move`, there are four possible actions corresponding to possible valuations of these variables, and action formulae are conditions on these variables. As illustrated at the end of the example, ARCTL formulae allow to reason about the consequences of actions.

```
MODULE agent
  IVAR move : boolean;
  VAR count : 0..10;
  ASSIGN
    init(count) := 0 ;
    next(count) := case
                     move & count < 10: count + 1;
                     1 : count;
                   esac;
  DEFINE win := (count=10);

MODULE main
  VAR alice : agent;
  VAR bob : agent;

SPEC !EAX (bob.move) bob.count = 0
SPEC AAX (bob.move & alice.move) (bob.count > 0 & alice.count > 0)
SPEC AAF (bob.move) bob.win
```

Fig. 1. NuSMV code with ARCTL specifications

Our implementation is at the prototype stage and still needs some improvements. In particular, the generation of witness traces for unsatisfied specifications is not yet supported for our new ARCTL operators.

Also, our implementation computes one monolithic BDD $[\![T]\!](\underline{s}, \underline{a}, \underline{s}')$ covering all possible actions. In some cases, separate transition relations could be computed for separate actions, potentially resulting in smaller BDDs and thus better scalability. Let us assume a finite action set $\mathcal{A} = \{a_1, \ldots, a_n\}$ (typically, the range of a unique input variable in the SMV model). Then for each $a_i \in \mathcal{A}$ we can define $\mathcal{T}_{a_i}(s, s') = \mathcal{T}(s, a_i, s')$ and pre-compute the BDDs for each action a_i

$$[\![\mathcal{T}_{a_i}]\!] = [\![\mathcal{T}]\!][\underline{a} := a_i]$$

and we have

$$[\![\mathsf{E}_\alpha \mathsf{X}\, \phi]\!](\underline{s}) = \exists \underline{s}'.\ \bigvee_{a \models \alpha} [\![\mathcal{T}_a]\!](\underline{s}, \underline{s}') \wedge [\![\phi]\!](\underline{s}')$$

6 Using ARCTL for Knowledge Logics

The material in this Section is presented in more details in a forthcoming companion article [10].

CTLK is a logic to reason about time and knowledge in a system of agents. Besides the temporal logic operators of CTL, CTLK offers *epistemic* (i.e. knowledge) operators, such as $\mathsf{K}_A\, \phi$, meaning that some agent A *knows* that ϕ holds. Intuitively, A knows ϕ if ϕ holds in all the states that A deems possible. Under certain hypotheses, this is formalized as an *epistemic* (equivalence) relation over states $\sim_A$ that equates states that are indistinguishable by A.

CTLK has been introduced in [2], extending the framework appearing in [11]. A model checking tool for CTLK and various examples have been presented in [12], showing that temporal-epistemic properties may offer a more efficient characterisation than temporal-only formulae.

6.1 From CTLK to ARTCL

The problem of model checking CTLK can be reduced to the problem of model checking ARCTL, as follows. In this setting, each agent Ag_i is associated with a set of "local" variables v_i, so that $s \sim_{Ag_i} s' \equiv (v_i(s) = v_i(s'))$, where $v_i(s)$ is the projection of s on v_i. Given a CTLK model $\mathcal{M}_K$ and a CTLK formula ϕ_K, a MTS $\mathcal{M} = F(\mathcal{M}_K)$ and an ARCTL formula $F(\phi_K)$ can be defined such that $\mathcal{M}_K \models \phi_K$ iff $F(\mathcal{M}_K) \models F(\phi_K)$. The MTS $\mathcal{M}$ includes two kind of action atoms: an atom RUN associated to temporal transitions, and atoms Ag_i (one for each agent) associated to each epistemic relation $\sim_{Ag_i}$. These atoms are used in the definition of two kinds of transitions of $\mathcal{M}$, either temporal or epistemic. $F(\phi_K)$ is generated as follows: standard CTL operators in ϕ_K are translated into their ARCTL extensions restricted to RUN actions, and epistemic operators are translated to ARCTL operators labelled with Ag_i. For instance

$$F(\mathsf{EX}\,\phi) = \mathsf{E}_{RUN}\mathsf{X}\,F(\phi)$$
$$F(\mathsf{K}_{Ag_i}\,\phi) = \mathsf{A}_{Ag_i}\mathsf{X}\,F(\phi)$$

We have implemented this translation as an M4 macro package that allows to write CTLK models and specifications, to be verified with the modified NuSMV presented in Section 5.3.

6.2 Experimental Results

We have conducted some early experiments using action-based model checking for verifying CTLK, in the context of analyzing *diagnosability* properties. Diagnosability is the ability of performing a diagnosis of a given system, knowing which variables (or events) of the system can be observed. Considering these variables as the observations of an agent D (the Diagnoser), diagnosability properties can be phrased as epistemic properties. For example, a fault condition $faulty$ can be detected if and only if the diagnoser always knows whether the system is faulty or not:

$$AG(K_D(faulty) \vee K_D(\neg faulty))$$

We have carried out experiments on a simple cascaded power distribution model, illustrated in Figure 2, for various depths of the cascade. This models features a power source, circuit breakers (CBs) and LEDs (as power sinks), where the CBs can fail in different ways and only the commands applied to the CBs and the LED states are observable. We have been able to verify a fairly large model (240 variables, for a total state space of size $\approx 10^{70}$). Despite its size, this model can be verified in less than 10 minutes thanks to the BDD-based symbolic encodings used in NuSMV. Notice that the modification of the

NuSMV code and the verification of epistemic properties using the reduction to ARCTL do not affect the performance of NuSMV: the non-modified version of SMV obtains similar results in the verification of temporal-only properties for the CB example.

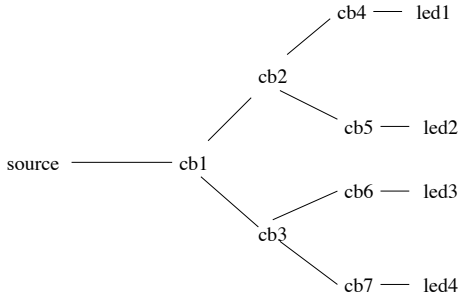

Fig. 2. Sample circuit

7 Related Work

Several authors have already described symbolic model checking algorithms for action-based logics, along very similar lines to what we describe here. In principle, after [7] it is sufficient to have a core set of temporal eventualities expressed as fixpoint formulae to provide the grounds for symbolic model checking.

- In [13,14], the authors present a variant of A-CTL with unless operators EW and AW and slightly different semantics (and syntax). That logic is more expressive than classic A-CTL. They give a fixpoint characterization using extended operators $\mathsf{EX}[\{\alpha\}\phi \vee \{\alpha'\}\phi']$ and $\mathsf{AX}[\{\alpha\}\phi \vee \{\alpha'\}\phi']$, which slightly extend our $\mathsf{E}_\alpha\mathsf{X}$ and $\mathsf{A}_\alpha\mathsf{X}$ and allow the increased expressivity, but otherwise follows the classical approach used as a basis for CTL and ARCTL model-checking. They have implemented their approach in the *Efficient Symbolic Tools* (EST) package [15], which offers symbolic A-CTL model-checking for a process-algebra language.
- [16] describe SAM, a symbolic model checker for μ-A-CTL, an extension of A-CTL with fixpoint operators. SAM uses a BDD-based Boolean system called BSP. SAM is part of the JACK toolset which uses the process algebra CCS/Meije to describe models.

Compared to those two systems, the work presented here is at a more preliminary stage but offers two significant contributions: firstly, it considers mixed systems, reconciling the state-based and action-based schools of formal methods; second, it is implemented in NuSMV, an open-source, feature-rich, efficient and widely distributed system with an expressive modelling language.

In a related topic, [17] discuss the encoding of CCS-style process algebras as BDDs. The authors envision that encoding as a way to enable efficient bisimilarity checking, but action-based temporal logic could equally be applied. [18]

brings this idea one level higher, by proposing a BDD encoding for any process-algebraic language whose structural operational semantics follow a given pattern (so-called *Simple GSOS Systems*).

Regarding the *post* reduction from mixed to state-based models and formulae, a similar reduction from A-CTL to CTL (and more generally from A-CTL* to CTL*) is already presented by Nicola and Vaandrager in [5]. That approach creates additional intermediate states in the KS to represent actions, which complicates the translation of formulae and would double the number of computation steps when computing fixpoints for symbolic model checking. In contrast, our *post* transformation multiplies the state space by the action space, but from a symbolic model checking standpoint, the number of variables and the depth of fixpoint iterations is unchanged.

8 Conclusions and Perspectives

Although symbolic model checking of action-based logics is well-understood in principle and has been implemented in several places, it has so far focused on pure action-based, process-algebra formalisms with a relatively limited distribution. This paper presents a step towards making mixed state-based and action-based reasoning capabilities more widely available, with two complementary contributions: at the theoretical level, a formulation of the model-checking problem for mixed models and an associated mixed logic, called ARCTL, that provides a clean generalization of CTL; at the practical level, two prototype implementations of mixed-logic model-checking in the mainstream symbolic model-checker NuSMV.

The initial motivation for this work arose from a need to perform model checking of logics with modalities for both the temporal evolution of the system and the knowledge (or beliefs) of agents in the system. Under some assumptions, this can be reduced to a mixed-logic model-checking problem. Our initial experiments in checking the epistemic temporal logic CTLK with NuSMV based on that reduction provided successful and encouraging results.

The work presented here can be extended in a number of ways:

- The treatment of internal (invisible) actions needs to be investigated. Essentially, this amounts to considering weak variants of temporal operators that ignore a distinguished τ action. This should not cause major technical issues, as fixpoint characterizations for such operators are well-known (see e.g. [6]), but the precise formalization and implementation need to be worked out.
- Fairness needs to be reconsidered. As we have seen in Section 3.4, the notion of *fair path* used for CTL is not appropriate with finite traces. Instead, action-based theories commonly define fairness in terms of not indefinitely refusing enabled actions [19]. How this can be addressed in symbolic model checking is a matter to be investigated.
- On the implementation side, support for generation of counter-examples needs to be addressed, which should require minimal technical changes in NuSMV. Partitioning the transition relation between a set of actions is a more significant and more involved change that we would also like to address.

- We are currently investigating the feasibility of bounded model checking for action-based logics, which offers a very efficient technique for finding counter-examples, but it is currently supported by NuSMV for the verification of linear temporal logic (LTL) only.
- Support for other action-based logics such as A-CTL could easily be added, either as additional macro packages or as native extensions of SMV. Game-theoretic logics such as ATL [20] would be a very valuable addition but would require a deeper analysis and more involved changes to NuSMV.

A more thorough experimental assessment of the current prototype and its future extensions is also desirable, but, as mentioned in Section 6.2 the first results obtained from the CTLK application are quite encouraging.

Acknowledgements

The authors would like to thank anonymous referees of successive versions of this paper for their useful feedback, as well as the NuSMV development team at IRST for their technical support. The modified version of NuSMV supporting ARCTL operators is available from the authors upon request.

References

1. Cimatti, A., Clarke, E., Giunchiglia, F., Roveri, M.: NuSMV: a new symbolic model verifier. In: Proc. of International Conference on Computer-Aided Verification (1999)
2. Penczek, W., Lomuscio, A.: Verifying epistemic properties of multi-agent systems via bounded model checking. Fundamenta Informaticae 55, 167–185 (2003)
3. Clarke, E.M., Emerson, E.A., Sistla, A.P.: Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Transactions on Programming Languages and Systems 8, 244–263 (1986)
4. Clarke, E.M., Grumberg, O., Peled, D.: Model Checking. MIT Press, Cambridge (1999)
5. Nicola, R.D., Vaandrager, F.: Action versus state based logics for transition systems. In: Guessarian, I. (ed.) Semantics of Systems of Concurrent Processes. LNCS, vol. 469, pp. 407–419. Springer, Heidelberg (1990)
6. Mateescu, R.: Logiques temporelles basées sur actions pour la vérification des systèmes asynchrones. Rapport de recherche 5032, INRIA (2003)
7. Burch, J.R., Clarke, E.M., McMillan, K.L., Dill, D.L., Hwang, J.: Symbolic model checking: 10^{20} states and beyond. Information and Computation 98, 142–170 (1992)
8. Cimatti, A., Clarke, E., Giunchiglia, F., Roveri, M.: NuSMV: a new symbolic model verifier. In: Proc. of International Conference on Computer-Aided Verification (1999)
9. Kernighan, B., Ritchie, D.: The M4 Macro Processor. Bell Laboratories (1977)
10. Lomuscio, A., Pecheur, C., Raimondi, F.: Automatic verification of knowledge and time with NuSMV. In: Proceedings of IJCAI'07, Hyderabad, India (to appear) (2007)
11. Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.Y.: Reasoning about Knowledge. MIT Press, Cambridge (1995)

12. Raimondi, F., Pecheur, C., Lomuscio, A.: Applications of model checking for multi-agent systems: verification of diagnosability and recoverability. In: Proceedings of Concurrency, Specification & Programming (CS&P), Warsaw University, pp. 433–444 (2005)
13. Meolic, R., Kapus, T., Brezocnik, Z.: An action computation tree logic with unless operator. In: Proceedings of the 1st South-East European workshop on formal methods SEEFM 2003, pp. 100–114 (2003)
14. Meolic, R., Kapus, T., Brezocnik, Z.: Verification of concurrent systems using ACTL. In: Hamza, M.H., (ed.) Applied informatics: proceedings of the IASTED international conference AI'2000, IASTED/ACTA Press, pp. 663–669 (2000)
15. Meolic, R., Kapus, T., Brezocnik, Z.: The Efficient Symbolic Tools package. In: 8th International Conference Software, Telecommunications and Computer Networks (SoftCOM 2000), pp. 147–156 (2000)
16. Fantechi, A., Gnesi, S., Mazzanti, F., Pugliese, R., Tronci, E.: A symbolic model checker for ACTL. In: Hutter, D., Traverso, P. (eds.) Applied Formal Methods - FM-Trends 98. LNCS, vol. 1641, pp. 228–242. Springer, Heidelberg (1999)
17. Enders, R., Filkorn, T., Taubner, D.: Generating BDDs for symbolic model checking in CCS. In: Larsen, K.G., Skou, A. (eds.) CAV 1991. LNCS, vol. 575, Springer, Heidelberg (1992)
18. Dsouza, A., Bloom, B.: Generating BDD models for process algebra terms. In: Proceedings of the 7th International Conference on Computer Aided Verification, London, UK, pp. 16–30. Springer, Heidelberg (1995)
19. Brinksma, E., Rensink, A., Vogler, W.: Fair testing. In: Lee, I., Smolka, S.A. (eds.) CONCUR 1995. LNCS, vol. 962, pp. 313–327. Springer, Heidelberg (1995)
20. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. J. ACM 49, 672–713 (2002)