An action based framework for verifying logical and behavioural properties of concurrent systems

R. De Nicola [☼☆], A. Fantechi [☆], S. Gnesi [☆], G. Ristori [▫]
☼ Università La Sapienza Roma
☆ I.E.I. - C.N.R. Pisa
▫ C.P.R. Pisa

Abstract

A system is described which supports proofs of both behavioural and logical properties of concurrent systems; these are specified by means of a process algebra and its associated logics. The logic is an action based version of the branching time logic CTL which we call ACTL; it is interpreted over transition labelled structures while CTL is interpreted over state labelled ones. The core of the system are two existing tools, AUTO and EMC. The first builds the labelled transition system corresponding to a term of a process algebra and permits proof of equivalence and simplification of terms, while the second checks validity of CTL logical formulae. The integration is realized by means of two translation functions from the action based branching time logic ACTL to CTL and from transition-labelled to state-labelled structures. The correctness of the integration is guaranteed by the proof that the two functions when coupled preserve satisfiability of logical formulae.

1. Introduction

Process algebras [Mil89, BW90, Hoa85, Hen88] are generally recognized as a convenient tool for describing concurrent systems at different levels of abstraction. They rely on a small set of basic operators which correspond to primitive notions of concurrent systems and on one or more notions of behavioural equivalence or preorder. The operators are used to build complex systems from more elementary ones. The behavioural equivalences are used to study the relationships between different descriptions (e.g. specification and implementation) of the same system at different levels of abstractions and thus to perform part of the analysis.

In this paper we want to propose a general framework for verifying properties of any process algebra by relying on the fact that they all have a single underlying model: Labelled Transition Systems.

There are already a few verification environments in which properties of concurrent systems specified by means of process algebras can be proved [CPS90, GLZ89, dSV90, BC89, GS90]. All of them provide tools for verifying equivalences or preorders on process algebras specifications. This equivalence-based approach to system verification has a major disadvantage: specifications, even at the most abstract level, tend to be too concrete; they are, anyway, descriptions of system behaviours even when it is assumed that some of the actual actions are invisible.

Logic is a good candidate to provide more abstract specifications; it permits describing systems properties rather than systems behaviours. Indeed, different types of temporal and modal logics have been proposed for the abstract specification of concurrent systems; in particular, modal and temporal logics

Note: The research has been partially supported by the CEC under ESPRIT project 2304 LOTOSPHERE and EBRA project 3011 CEDISYS and by "Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo" of CNR.

have been recognized as suitable for specifying system properties [EH86, HM85, MP89] due to their ability to deal with the notions of *necessity, possibility, eventuality,* etc.. Logics have been equipped with model checkers to prove satisfiability of formulae and thus systems properties: a system is considered as a potential model for the formula expressing the desired property. Actually, very interesting logics like CTL and CTL* which require formulating properties of systems in terms of their states, have been put forward [BCG88, EH86, ES89]; also, sophisticated and efficient model checkers have been developed for them [CES86].

Thus, the behavioural and logical approaches to system specification and verification can be seen as complementary; the first is more fruitfully used to specify abstract properties while the second permits describing more naturally behavioural and structural properties of systems. It would be of great importance to have a uniform setting for reasoning with the support of the tools made available by both methods. Unfortunately, up to now the most successful representatives of the two approaches have been based on different semantic models which take a different standpoint for looking at specifications. State changes and state properties are the base for interpreting logical specifications. Actions causing state changes are the key for interpreting systems behaviours described via process algebras. The semantic models used in the two cases are Kripke Structures and Labelled Transition Systems, respectively. In the first kind of structures, states are labelled to describe how they are modified by the transitions, while in the second, transitions are labelled to describe the actions which cause state changes. Temporal and modal logics and the associated complexity issue have been thoroughly investigated in the setting of Kripke Structures while combinators for transition systems and the issue of behavioural equivalences have received more attention in the setting of Labelled Transition Systems. Due to the success of process algebras, other logics have been proposed (see e.g. [HM85, BGS88, Lar88, Sti89]) which are interpreted over LTS's and tools have been developed to support reasoning with them. However in this case either we have logics, like Hennessy-Milner logic [HM85], that are not sufficiently expressive or logics, like the μ-calculus [Koz83], that require exponential time for model checking.

A recent result from [DV90a] has brought the world of modal logic and process algebras closer. A new logic for process algebras has been defined which is very similar to CTL* but based on actions and interpreted over Labelled Transition Systems. This new logic, ACTL*, is the natural analogue of CTL* but contains relativized modalities like $X_a\varphi$ - to be read "the next transition is labelled by action a and the remaining path satisfies φ"- as demanded by the interpretation model. Like it has been done for CTL*, a purely branching time subset of ACTL*, called ACTL, has been introduced; it is more expressive than Hennessy-Milner Logic and can describe safety and liveness properties [DV90b].

The question is now whether it is possible to have an efficient model checker for ACTL. We show how to use existing model checkers for CTL also to check validity of ACTL formulae. This is done by means of two translation functions, one from ACTL to CTL formulae, the other from Labelled Transition Systems to Kripke Structures. Both translations are linear; this, coupled with the linear algorithm used by the EMC [CES86], guarantees linear model checking.

Indeed, by relying on the translation functions, we define a verification environment that permits both to verify equivalences on systems described by means of a process algebra and properties of such systems expressed in ACTL. The environment consists of two existing tools, AUTO [dSV90] and EMC model checker [CES86], and of two modules, the *model translator* and the *logic translator*, performing the necessary translations. AUTO builds the labelled transition systems corresponding to process algebra terms and permits minimizing and checking equivalences of transition systems. The model translator transforms the transition systems built by AUTO into Kripke structures. The latter are used as models to verify, via EMC model checker, satisfiability of ACTL formulae which have been translated into CTL by the logic translator.

The rest of the paper is organized as follows. In the next section, the relevant definitions of the used logics are summarized. Section 3, the core of the paper, contains the description of our verification environment. In Section 4, a specification and verification example is presented to give a flavour of the potentiality of the verification environment.

2. ACTL: An action based version of CTL

In this section we present the logic CTL [EH86] whose interpretation domains are Kripke Structures and the logic ACTL proposed in [DV90b], based on actions rather than states, whose interpretation domains are Labelled Transition Systems.

A *Kripke Structure* (or *KS*) is a 4-tuple $\mathcal{K} = (S, \mathbf{AP}, L, \rightarrow)$ where:
- S is a set of *states*;
- **AP** is a finite, nonempty set of *atomic proposition names* ranged over by p, q, ...;
- $L: S \rightarrow 2^{\mathbf{AP}}$ is a function that labels each state with a set of atomic propositions true in that state;
- $\rightarrow \subseteq S \times S$ is the *transition relation*; an element $(r, s) \in \rightarrow$ is called a *transition* and is written as $r \rightarrow s$.

A *Labelled Transition System* (or *LTS*) is a structure $\mathcal{A} = (S, A, \rightarrow)$ where:
- S is a set of *states*;
- A is a finite, non-empty set of *actions*; the *silent action* τ is not in A;
- $\rightarrow \subseteq S \times (A \cup \{\tau\}) \times S$ is the *transition relation*; an element $(r,\alpha,s) \in \rightarrow$ is called a *transition*, and is written as $r - \alpha \rightarrow s$.

We let $A_\tau = A \cup \{\tau\}$; $A_\varepsilon = A \cup \{\varepsilon\}$, $\varepsilon \notin A_\tau$. Moreover, we let r, s, ... range over states; a, b, ... over A; α, β, ... over A_τ and k, ... over A_ε.

Let us now introduce the notion of paths and runs over a LTS, $\mathcal{A} = (S, A, \rightarrow)$:
- A sequence $(s_0,\alpha_0,s_1)\ (s_1,\alpha_1,s_2)\ldots \in \rightarrow^\infty$ is called a *path* from s_0; a path that cannot be extended, i.e. is infinite or ends in a state without outgoing transitions, is called a *fullpath*;
- a *run* from $s \in S$ is a pair $\rho = (s,\pi)$, where π is a path from s; we write $first(\rho) = s$ and $path(\rho) = \pi$; if π is finite then $last(\rho)$ denotes the last state of π; a *maximal run* is a run whose second element is a fullpath;
- concatenation of runs is denoted by juxtaposition: $\eta = \rho\theta$; it is only defined if ρ is finite and $last(\rho)=first(\theta)$. When $\eta = \rho\theta$ we say that θ is a suffix of η; it is a proper suffix if $\rho \neq \varepsilon$.

We write run(s) for the set of runs from s and let π, ... range over paths and ρ, σ, η ... over runs.

The notation for runs that we have introduced for LTS's carries over to Kripke Structures $\mathcal{K} = (S, \mathbf{AP}, L, \rightarrow)$ in the obvious way. The only difference is that transitions are no longer triples but pairs.

CTL is a language of state formulae interpreted over Kripke Structures and it is just a subset of CTL* [EH86]; CTL* combines linear and branching time operators; its syntax is given in terms of path formulae that are interpreted over full paths and state formulae that are true or false of a state. CTL is the branching time subset of CTL* in which every linear time operator is immediately preceded by a path quantifier; it is defined as the set of state formulae φ given by the following grammar, where γ ranges on path formulae and p ranges on **AP**:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \exists\gamma \mid \forall\gamma$$
$$\gamma ::= X\varphi \mid \varphi U \varphi$$

We write **true** for $\neg(p_0 \wedge \neg p_0)$ where p_0 is some arbitrarily chosen atomic proposition name.

As usual, a set of logic operators and modalities can be derived from the basic ones: **false**, $\varphi \vee \varphi'$ (*or*), $\varphi \Rightarrow \varphi'$ (*implies*), $F\varphi$ (*eventually*), $G\varphi$ (*always*) [EH86].

Let $\mathcal{K} = (S, \mathbf{AP}, \mathcal{L}, \rightarrow)$ be a Kripke Structure. We give below a maximal interpretation of CTL formulae, that is we suppose that all maximal runs in $\mathcal{K}$ have infinite length. *Satisfaction* of a CTL formula φ (γ) by a state s (run ρ), notation $s \models_{\mathcal{K}} \varphi$ or just $s \models \varphi$ ($\rho \models_{\mathcal{K}} \gamma$, or $\rho \models \gamma$), is defined inductively by:

$s \models p$	iff	$p \in \mathcal{L}(s)$;
$s \models \neg\varphi$	iff	$s \not\models \varphi$;
$s \models \varphi \wedge \varphi'$	iff	$s \models \varphi$ and $s \models \varphi'$;
$s \models \exists\gamma$	iff	there exists a run $\theta \in$ run(s) such that $\theta \models \gamma$;
$s \models \forall\gamma$	iff	for all runs $\theta \in$ run(s) $\theta \models \gamma$;
$\rho \models \varphi U \varphi'$	iff	there exists a suffix θ of ρ such that first(θ) $\models \varphi'$ and for all suffixes η of ρ, which have θ as proper suffix: first(η) $\models \varphi$;
$\rho \models X\varphi$	iff	there exist s, s', θ such that $\rho = (s, (s, s'))\theta$ and $s' \models \varphi$.

In order to define the logic ACTL, in [DV90b], a tiny auxiliary logic of actions is introduced.

The collection *Afor* of *action formulae* over A is defined by the following grammar where χ, χ', range over action formulae, and $a \in A$:

$$\chi ::= a \mid \neg\chi \mid \chi \wedge \chi'$$

We write **true** for $\neg(a_0 \wedge \neg a_0)$, where a_0 is some arbitrarily chosen action, and **false** for $\neg$ **true**.

The *satisfaction* of an action formula χ by an action a, notation $a \models \chi$, is defined inductively by:

$a \models b$	iff	$a = b$;
$a \models \neg\chi$	iff	$a \not\models \chi$;
$a \models \chi \wedge \chi'$	iff	$a \models \chi$ and $a \models \chi'$.

The syntax of the logic ACTL, a subset of ACTL*, is defined by the state formulae generated by the following grammar, where φ, φ', ... range over state-formulae, γ over path formulae and χ and χ' are action formulae:

$$\varphi ::= \mathbf{true} \mid \neg\varphi \mid \varphi \wedge \varphi' \mid \exists\gamma \mid \forall\gamma$$
$$\gamma ::= X_\chi\varphi \mid X_\tau\varphi \mid \varphi\, \chi U_{\chi'} \varphi' \mid \varphi\, \chi U \varphi'$$

We give below the satisfaction relation for ACTL formulae; as above we assume a maximal interpretation, that is, we suppose all maximal runs in the LTS are infinite in length.
Let $\mathcal{A} = (S, A, \rightarrow)$ be a LTS. *Satisfaction* of an ACTL-formula φ (γ) by a state s (run ρ), notation $s \models_{\mathcal{A}} \varphi$ or just $s \models \varphi$ ($\rho \models_{\mathcal{A}} \gamma$, or $\rho \models \gamma$), is given inductively by:

$s \models \mathbf{true}$		always;
$s \models \neg\varphi$	iff	$s \not\models \varphi$;
$s \models \varphi \wedge \varphi'$	iff	$s \models \varphi$ and $s \models \varphi'$;
$s \models \exists\gamma$	iff	there exists a run $\theta \in$ run(s) such that $\theta \models \gamma$;
$s \models \forall\gamma$	iff	for all runs $\theta \in$ run(s) $\theta \models \gamma$;
$\rho \models \varphi\ {}_{\chi}U_{\chi'}\varphi'$	iff	there exists $\theta = (s,(s,a,s'))\theta'$, suffix of ρ, s.t. $s' \models \varphi'$, $a \models \chi'$, $s \models \varphi$ and for all $\eta = (r,(r,\beta,r'))\eta'$, suffixes of ρ, which have θ as proper suffix, we have $r \models \varphi$ and ($\beta \models \chi$ or $\beta = \tau$);
$\rho \models \varphi\ {}_{\chi}U\varphi'$	iff	there exists a suffix θ of ρ s. t. first(θ) $\models \varphi'$ and for all $\eta = (r,(r,\beta,r'))\eta'$ which have θ as proper suffix we have $r \models \varphi$ and ($\beta \models \chi$ or $\beta = \tau$);
$\rho \models X_{\chi}\varphi$	iff	$\rho = (s,(s,a,s'))\theta$ and $s' \models \varphi$ and $a \models \chi$;
$\rho \models X_{\tau}\varphi$	iff	$\rho = (s,(s,\tau,s'))\theta$ and $s' \models \varphi$.

As usual, derived modalities such as **false**, $\varphi \vee \varphi'$, $\varphi \Rightarrow \varphi'$, $F\varphi$, $G\varphi$ are introduced. Other derived modalities similar to those of Hennessy-Milner logic with until defined in [DV90a] are:

φ <a> φ'	for	$\exists(\varphi\ {}_{\mathbf{false}}U_a\ \varphi')$,
φ <ε>φ'	for	$\exists(\varphi\ {}_{\mathbf{false}}U\ \varphi')$,
<k> φ	for	**true** <k> φ,
[k] φ	for	$\neg$ <k>$\neg\varphi$.

The indexed next modalities $X_{\chi}\varphi$, $X_{\tau}\varphi$ say that in the next state of the run, reached respectively with an action in χ or with a τ, the formula φ holds.

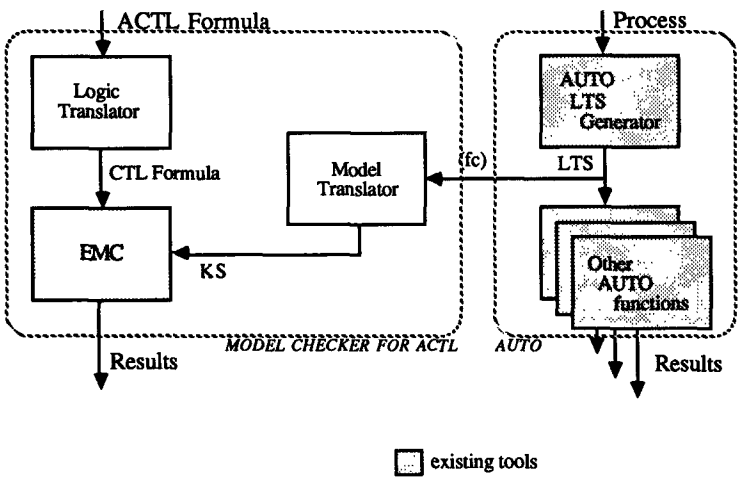
In [DV90b] it has been proved that every formula in ACTL can be translated in one of CTL. In order to define the mapping $\mathfrak{lt}$ between them, they need to introduce a corresponding translation $\mathfrak{mt}$ between their models, that is Labelled Transition Systems and Kripke Structures. In the next section the complete definition of the translation functions $\mathfrak{mt}$ and $\mathfrak{lt}$ is given; these are variants of the original ones $\mathfrak{ks}$ and $\mathfrak{ks}'$.

3. The Verification Tool

To obtain a general verification environment which enables the user to verify both bisimulation based equivalences and ACTL properties of terms of a process algebras, we have chosen to integrate two tools: AUTO [dSV90] and EMC [CES86]. AUTO is able to generate a Labelled Transition System from a CCS [Mil89] or Meije [AB84] specification and permits verification of bisimulation based equivalences and minimization of states. EMC permits to verify the validity of a CTL formula over a Kripke Structure. Indeed, to perform the check of an ACTL formula φ on a Labelled Transition System M, the following steps are needed:

1) Input M and φ;
2) Translate M into the corresponding Kripke Structure M';
3) Translate φ into the corresponding CTL formula φ';
4) Perform the Model Checking of φ' on M'.

The architecture of our environment is summarized by the picture below:



3.1. Generating the Labelled Transition System

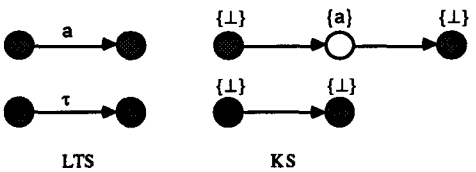
To build the Labelled Transition System corresponding to a term we use AUTO. It must be said that the actual construction phase starts only if a finiteness test is passed. Indeed, some terms might lead to generating an infinite number of states and AUTO; before starting the actual construction of the LTS, takes advantage of the sufficient conditions for finiteness given in [MV90].

The LTS provided by AUTO can be written in a number of different formats; one of these is called *format commun*; it has been proposed as standard format for representing automata.

3.2. The Model Translator

The Model Translator transforms the LTS produced by AUTO into a suitable input for the EMC, i.e. a Kripke Structure. In order to provide this functionality we have implemented the T1 algorithm given below.

Given a Labelled Transition System the corresponding Kripke Structure is obtained by splitting the transitions labelled by visible actions and creating a new state for each of them, labelled by the label of the original transition; the generated system has almost the same structure of the original one. In the picture below, the translations of an observable and of an unobservable transition are shown:



T1 *(From LTS's to KS's)*

Let $\mathcal{A} = (S, A, \rightarrow)$ be a LTS, S_d the subset of states of S without successors, and $\perp$ be a fresh symbol not in A. The KS, $mt(\mathcal{A})$, is defined as $(S', \mathbf{AP}, \mathcal{L}, \rightarrow')$ where:

- $S' = S \cup \{(r,a,s) \mid a \in A \text{ and } r -a\rightarrow s\} \cup N$, with $N = \{s_f\}$ if $S_d \neq \{\}$ and $N = \{\}$ otherwise;

- $\mathbf{AP} = A \cup \{\perp\}$;
- $\rightarrow' = \{(r,s) \mid r -\tau\rightarrow s\} \cup \{(r,(r,a,s)) \mid r -a\rightarrow s\} \cup \{((r,a,s),s) \mid r -a\rightarrow s\} \cup T$, with $T = \{(t,s_f) \mid t \in S_d\} \cup \{(s_f, s_f)\}$ if $S_d \neq \{\}$ and $T = \{\}$ otherwise;
- For $r, s \in S$ and $a \in A$: $\mathcal{L}(s) = \{\perp\}$, $\mathcal{L}((r,a,s)) = \{a\}$
- $\mathcal{L}(s_f) = \{\perp\}$.

Note that the translation produces a Kripke Structure with a larger number of states; new states are labelled with the same label of the corresponding transition while old ones are labelled with a *fresh* symbol, $\perp$, which can be interpreted as: *no visible actions occurred.* To comply with maximal interpretation of ACTL formulae, we have added an additional step to the original translation in [DV90b]: if there exist finite maximal runs in the LTS then in the corresponding KS a self looping new state s_f is created, labelled by $\perp$ and finite paths are extended to relate deadlocked states with s_f.
The size of the sets of states and transitions of the Kripke Structure produced by the above algorithm is given by the following formulae, where $n = |S|$, $d = |S_d|$, $m = |\rightarrow|$, and u is the number of unobservable transitions in $\rightarrow$:

$$|S'| = \begin{cases} n + m - u & \text{if } d = 0 \\ n + m - u + 1 & \text{if } d \neq 0 \end{cases} \quad \text{and} \quad |\rightarrow'| = \begin{cases} 2m - u & \text{if } d = 0 \\ 2m - u + 1 + d & \text{if } d \neq 0 \end{cases}$$

3.3. The Logic-Translator

This module transforms an ACTL formula into a suitable input formula for the EMC, i.e. a CTL formula. This translation step has been implemented by parsing the ACTL formula before presenting it to EMC; once an ACTL formula is parsed, its translation into CTL is given according to the following $\mathfrak{t}$ function. We have modified the original function presented in [DV90b] in order to take into account quantifications over linear time operators.

T2 (*From ACTL to CTL*)
The mapping $\mathfrak{t}$: ACTL $\rightarrow$ CTL is inductively defined by:

- $\mathfrak{t}(\mathbf{true}) = \mathbf{true}$,
- $\mathfrak{t}(\neg\varphi) = \neg\, \mathfrak{t}(\varphi)$,
- $\mathfrak{t}(\varphi \wedge \varphi') = \mathfrak{t}(\varphi) \wedge \mathfrak{t}(\varphi')$,
- $\mathfrak{t}(\exists(\varphi\ \chi U_{\chi'}\ \varphi')) = \exists(((\perp \wedge \mathfrak{t}(\varphi)) \vee (\neg\perp \wedge \chi))\ U\ ((\neg\perp \wedge \chi') \wedge \exists X(\perp \wedge \mathfrak{t}(\varphi'))))$,
- $\mathfrak{t}(\exists(\varphi\ \chi U\ \varphi')) = (\perp \wedge \mathfrak{t}(\varphi')) \vee \exists(((\perp \wedge \mathfrak{t}(\varphi)) \vee (\neg\perp \wedge \chi))\ U\ (\perp \wedge \mathfrak{t}(\varphi')))$,
- $\mathfrak{t}(\exists X_\chi \varphi) = \exists X(\neg\perp \wedge \chi \wedge \exists X(\mathfrak{t}(\varphi)))$,
- $\mathfrak{t}(\exists X_\tau \varphi) = \perp \wedge \exists X(\perp \wedge \mathfrak{t}(\varphi))$,
- $\mathfrak{t}(\forall(\varphi\ \chi U_{\chi'}\ \varphi')) = \forall(((\perp \wedge \mathfrak{t}(\varphi)) \vee (\neg\perp \wedge \chi))\ U\ ((\neg\perp \wedge \chi') \wedge \forall X(\perp \wedge \mathfrak{t}(\varphi'))))$,
- $\mathfrak{t}(\forall(\varphi\ \chi U\ \varphi')) = (\perp \wedge \mathfrak{t}(\varphi')) \vee \forall(((\perp \wedge \mathfrak{t}(\varphi)) \vee (\neg\perp \wedge \chi))\ U\ (\perp \wedge \mathfrak{t}(\varphi')))$,
- $\mathfrak{t}(\forall X_\chi \varphi) = \forall X(\neg\perp \wedge \chi \wedge \forall X(\mathfrak{t}(\varphi)))$,
- $\mathfrak{t}(\forall X_\tau \varphi) = \perp \wedge \forall X(\perp \wedge \mathfrak{t}(\varphi))$.

The key result about $\mathfrak{lt}$ is that it preserves truth, that is if $\mathcal{A}$ is a LTS and φ is an ACTL-formula then $\mathcal{A} \models \varphi$ if and only if $\mathfrak{mt}(\mathcal{A}) \models \mathfrak{lt}(\varphi)$. An interesting property of the T2 algorithm is that the size of $\mathfrak{lt}(\varphi)$ is linear in the size of φ.

Proposition: Let $\mathcal{A}$ an LTS, $\mathcal{K}$ the corresponding KS obtained by means of algorithm T1, and $s \in S$; then $s \models_{\mathcal{A}} \varphi$ if and only if $s \models_{\mathcal{K}} \mathfrak{lt}(\varphi)$.

Sketch of the proof:

This proof is made by induction on the length of the ACTL formulae. We show for every formula φ that $s \not\models_{\mathcal{A}} \varphi$ implies $s \not\models_{\mathcal{K}} \mathfrak{lt}(\varphi)$ and $s \models_{\mathcal{A}} \varphi$ implies $s \models_{\mathcal{K}} \mathfrak{lt}(\varphi)$.

Note that in the construction of $\mathfrak{mt}(\mathcal{A})$ to every state in S there corresponds a state in S' with the same name; moreover, only states corresponding to existing ones are labelled with $\perp$. New states in $\mathfrak{mt}(\mathcal{A})$ are labelled with the action associated to the corresponding transition; then, if the label of the transition satisfies a given action formula χ we have that the corresponding state satisfies the CTL formula χ. We refer the interested reader to the forthcoming version of the paper.

3.4. Model Checking

The output of the Model Translation phase is given as input to EMC. The ACTL expressions to be verified can be given as input to the EMC prompt; any time an ACTL formula is given, EMC calls the Logic Translator providing the corresponding CTL formula; then EMC checks the CTL formula on the Kripke Structure, giving as result true or false.

It is not difficult to see that we can perform model checking for ACTL with time complexity $O((|S|+|\rightarrow|) \times |\varphi|)$. Indeed, if we let $\mathcal{A}$ be a finite LTS, s be a state of $\mathcal{A}$ and φ be an ACTL formula, in order to determine whether $s \models_{\mathcal{A}} \varphi$ it suffices to check whether $s \models_{\mathcal{K}} \mathfrak{lt}(\varphi)$. We can compute $\mathfrak{mt}(\mathcal{A})$ in $O(|S|+|\rightarrow|)$-time and the number of states and transitions of $\mathfrak{mt}(\mathcal{A})$ will be of order $|S|+|\rightarrow|$. The formula $\mathfrak{lt}(\varphi)$ can be computed in $O(|\varphi|)$-time and its size will be of order $|\varphi|$. Model checking of formula φ on $\mathcal{A}$ with the algorithm for CTL of [CES86] can be therefore performed in $O((|S|+|\rightarrow|) \times |\varphi|)$-time.

4. The Crossing example

We present now an example starting from the CCS specification in [BS90], in which a road crossing a railway is specified. This crossing is such that the barriers on the road are usually kept down and lifted when a car approaches and tries to cross; the traffic lights on the railway are usually red and turn green when a train approaches and tries to cross. In the CCS-Meije specification below the actions 'train' and 'car' represent respectively the action of a train and a car approaching, 'tcross' and 'ccross' the passage of a train and the passage of a car. The process Semaphore controls the barriers and the traffic lights. The notation a? represents the complementary action of a.

```
let rec {Rail = train: green: tcross: red?: Rail} in Rail;
let rec {Road = car: up: ccross: down?: Road} in Road;
let rec {Semaphore = green?: red: Semaphore + up?: down: Semaphore} in Semaphore;
let rec {Crossing = (Road // Rail // Semaphore)\green\red\up\down} in Crossing;
```

Our aim is checking wether the process Crossing satisfies the safety property (mutual exclusion):

it never happens that both a car and a train are able to cross, and the following liveness property (fairness):

whenever a train (a car) approaches, it eventually crosses.

To express mutual exclusion, we need describing by means of an ACTL formula that whenever a car can cross then trains cannot cross until the car does it and viceversa:

$$\forall G((\exists X_{ccross}\ \mathbf{true} \rightarrow \forall(\mathbf{true}\ {}_{\neg tcross}U_{ccross}\ \mathbf{true})) \wedge (\exists X_{tcross}\mathbf{true} \rightarrow \forall(\mathbf{true}\ {}_{\neg ccross}U_{tcross}\ \mathbf{true}))) \quad (1)$$

We can express the liveness property with the following ACTL formula:

$$\forall G(([train]\ \forall(\mathbf{true}\ {}_{\neg train}U_{tcross}\ \mathbf{true})) \wedge ([car]\ \forall(\mathbf{true}\ {}_{\neg car}U_{ccross}\ \mathbf{true}))) \quad (2)$$

Below we show the LTS produced by AUTO and the Kripke Structure resulting from the Model Translation phase.

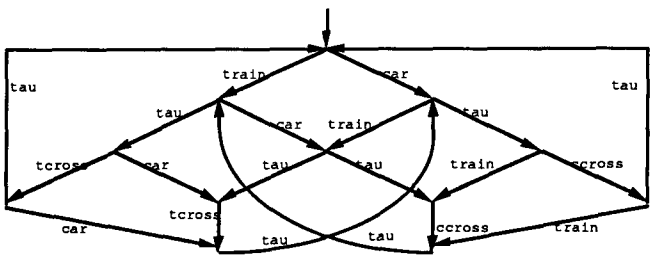


Figure 1: the LTS for Crossing

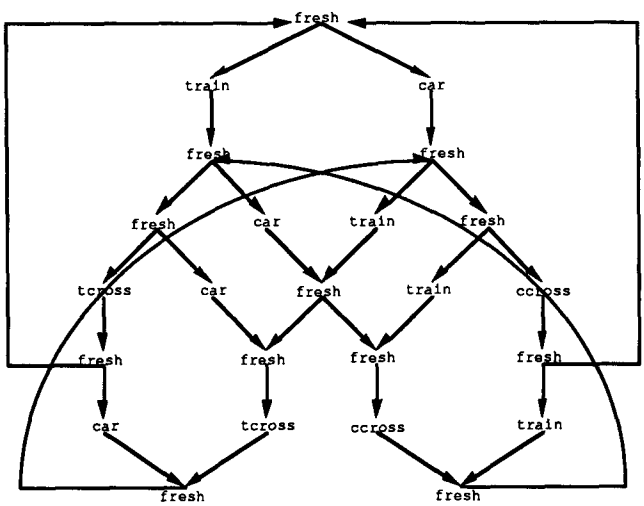


Figure 2: the Kripke Structure in the EMC format.

After the Logic Translation phase, the EMC can be used to check whether the formulae (1) and (2) hold on the Kripke Structure. The result of the model checking phase is that the level crossing does not enjoy the liveness property (2), while it does enjoy the safety property (1).

5. Conclusions

We have presented a uniform environment for the verification of logical and behavioural properties of Process Definition Languages; both classes of properties are interpreted over a single model, namely Labelled Transition Systems. The verification environment is the result of the integration of two existing tools, the EMC model checker [CES86] and AUTO [dSV90], by means of two translation functions from Labelled Transition Systems to Kripke Structures, and from the logic ACTL to the logic CTL. A work similar to ours in this respect is presented in [JKP90]. These authors use CTL as a logic for Labelled Transition Systems but substantially change the satisfaction relation; they have a relativized satisfaction relation $\langle a, s \rangle \models \varphi$ instead of the relativized modality $X_a \varphi$. The expressive power of the two languages is similar, but the satisfaction relation used here is more immediate. Besides, they do not consider invisible actions and we have not yet seen a generalization of their approach to systems with silent steps in a way that would preserve some behavioural equivalence.

We see our tool as an experiment and as a means of assessing the expressivity of the action logic, if it proves fruitful then we would certainly write a direct model checker for ACTL.

In this way we could implement the useful "counterexample" facility provided by EMC: if a formula does not hold in the model, EMC looks for a path in the model which falsifies the given formula. We have considered the possibility of importing this functionality in our framework, but it is not an easy task to reinterprete all the CTL formulae provided by the EMC counterexample facility as ACTL ones. The effort needed to reverse both the logic and the transition system translators appears not smaller than that needed to build a new model checker for ACTL from the scratch.

Acknowledgements

The first author would like to thank Frits Vaandrager for joint work and discussions on the topics of the paper.

References

[AB84] D. Austry, G. Boudol: Algèbre de Processus et Synchronization. *Theoretical Computer Science*, **30**, (1) 1984, pp. 91-131

[BC89] T. Bolognesi, M. Caneve: Squiggles: a Tool for the Analysis of LOTOS Specifications, in "Formal Description Techniques" (K. Turner, ed.), North-Holland, 1989.

[BCG88] M.C. Browne, E.M. Clarke, O. Grümberg: Characterizing Finite Kripke Structures in Propositional Temporal Logic. *Theoretical Computer Science*, **59** (1,2), 1988, pp. 115-131.

[BGS88] A. Boujjani, S. Graf, J. Sifakis: A Logic for the Description of Behaviours and Properties of Concurrent Systems. In *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, (de Bakker, J.et al., eds.) LNCS **354**, Springer-Verlag, 1989, pp. 398-410.

[BS90] J. Bradfield, C. Stirling: Verifying Temporal Properties of Processes. in *Concur 90*(J. C. P. Baeten, J. W. Klop, eds), LNCS **458**, Springer-Verlag, 1990, pp. 115-125.

[BW90] J. C. M. Baeten, W. P. Weijland: Process Algebra. Cambridge Tracts in *Theoretical Computer Science* **18**. Cambridge University Press, 1990.

[CES86] E.M. Clarke, E.A. Emerson, A.P. Sistla: Automatic Verification of Finite State Concurrent Systems using Temporal Logic Specifications. *ACM Toplas*, **8** (2), 1986, pp. 244-263.

[CPS90] R. Cleaveland, J. Parrow, B. Steffen: The Concurrency Workbench. In *Automatic Verification Methods for Finite State Systems* (J. Sifakis, ed.) LNCS **407**, Springer-Verlag, 1990, pp. 24-37.

[dSV90] R. de Simone, D. Vergamini: Aboard AUTO, I.N.R.I.A. Technical Report 111 (1990).

[DV90a] R. De Nicola, F. W. Vaandrager: Three Logics for Branching Bisimulations (Extended Abstract) in *LICS '90*, IEEE Computer Society Press, 1990, pp. 118-129.

[DV90b] R. De Nicola, F. W. Vaandrager: Action versus State based Logics for Transition Systems. In *Semantics of Systems of Concurrent Processes* (I. Guessarian,ed.), LNCS **469**, 1990, pp. 407-419.

[EH86] E. A. Emerson, J. Y. Halpern: "Sometimes" and "Not Never" Revisited: on Branching Time versus Linear Time Temporal Logic. *Journal of ACM*, **33**, 1, 1986, pp. 151-178.

[ES89] E. A. Emerson, J. Srinivasan: Branching Time Temporal Logic. In *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, (de Bakker et al., eds.) LNCS **354**, Springer-Verlag, 1989, pp. 123-172.

[GLZ89] J. C. Godskesen, K. G. Larsen, M. Zeeberg: TAV Users Manual, Internal Report, Aalborg University Center, Denmark, (1989).

[GS90] H. Garavel, J Sifakis: Compilation and Verification of LOTOS Specifications, in *Protocol Specification, Testing and Verification*, X, (L. Logrippo et al., eds.) North Holland , 1990.

[Hen88] M. Hennessy: *An Algebraic Theory of Processes*, MIT Press, Cambridge, 1988.

[HM85] M. Hennessy, R. Milner: Algebraic Laws for Nondeterminism and Concurrency. *Journal of ACM*, **32**, 1985, pp. 137-161.

[Hoa85] C. A. R. Hoare: *Communicating Sequential Processes*. Prentice Hall International, 1985.

[JKP90] B. Jonsson, A.H. Khan, J. Parrow: Implementing a model checking algorithm by adapting existing automated tools. In *Automatic Verification Methods for Finite State Systems* (J. Sifakis, ed.) LNCS **407**, Springer-Verlag, 1990, pp. 179-188.

[Koz83] D. Kozen: Results on the Propositional μ-calculus, *Theoretical Computer Science*, **27**, 1983.

[Lar88] K. G. Larsen: Proof Systems for Hennessy-Milner Logic with Recursion, in Proceedings *CAAP '88* (M. Dauchet & M. Nivat eds) LNCS **299**, Springer-Verlag, 1988.

[Mil89] R. Milner: *Communication and Concurrency*, Prentice Hall International,1989.

[MP89] Z. Manna, A. Pnueli: The Anchored Version of the Temporal Framework, in *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, (de Bakker et al., eds.) LNCS **354**, Springer-Verlag, 1989.

[MV90] E. Madeleine, D. Vergamini: AUTO: A Verification Tool for Distributed Systems Using Reduction of Finite Automata Networks, in *Formal Description Techniques II* (S.T. Vuong, ed.), North-Holland, 1990.

[Sti89] C. Stirling: Temporal Logics for CCS, in *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, (de Bakker et. al., eds.) LNCS **354**, Springer-Verlag, 1989, pp. 660-672.

[vGW89] R. J. van Glabbeek, W. P. Weijland: Branching Time and Abstraction in Bisimulation Semantics. In *Information Processing '89* (G.X. Ritter, ed.), North Holland, 1989, pp. 613-618.