Action versus State based Logics for Transition Systems

Rocco De Nicola
IEI - CNR
Via S. Maria, 46 I-56126 Pisa
ITALY
DENICOLA@ICNUCEVM.CNUCE.CNR.IT

Frits Vaandrager
CWI
P.O. Box 4079, 1009 AB Amsterdam
THE NETHERLANDS
FRITSV@CWI.NL

Abstract

A temporal logic based on actions rather than on states is presented and interpreted over labelled transition systems. It is proved that it has essentially the same power as CTL*, a temporal logic interpreted over Kripke structures. The relationship between the two logics is established by introducing two mappings from Kripke structures to labelled transition systems and viceversa and two transformation functions between the two logics which preserve truth. A branching time version of the action based logic is also introduced. This new logic for transition systems can play an important role as an intermediate between Hennessy-Milner Logic and the modal μ-calculus. It is sufficiently expressive to describe safety and liveness properties but permits model checking in linear time.

1. Introduction

Labelled Transition Systems (LTS's) and Kripke Structures (KS's) are two types of structures which have proven to be basic for many applications in computer science, especially for modelling reactive and concurrent systems. On one hand, LTS's have been more widely used to interpret process algebra languages like CCS and other languages for the description of communicating systems. On the other hand, KS's form the common model for interpreting many temporal and modal logics which are used as tools for specifying properties of communicating systems. The two types of structures are very similar and both can be seen as generalizations of state automata: in LTS's transitions are labelled to describe the actions which cause a state change while in KS's states are labelled to describe how they are modified by the transitions.

In spite of their similarity and, we might say, their complementarity, the two models have mostly been considered as alternative to each other and there are strong advocates standing on each side. For example,

Note: The research has been partially supported by Esprit Basic Research Action Program, Project 3011 CEDISYS and by CNR Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo, project LAMBRUSCO.

due to the "experienced" easiness in formulating properties of systems in terms of their states, Lamport "...decide(d) to base an axiomatic system for describing concurrent programs upon states rather than operations." [Lam83]. Actually, very interesting logics like CTL and CTL* interpreted over KS's have been put forward [EH83, ES89] and have been thoroughly investigated [BCG88]; also, sophisticated and efficient tools have been developed for them [CES86]. For LTS's, on the other hand, Hennessy an Milner which were more interested in properly describing the actual behaviour of communicating systems, did define a new logic, now known as HML [HM85]. More recently, due to the success of process algebras, other, more expressive, logics interpreted over LTS's have been proposed (see e.g. [Lar88, Sti89, DV90]) and tools have been developed to support reasoning with them [CPS90].

Still, one might say that modal and temporal logics for computer science and the associated complexity issue have been more thoroughly investigated in the setting of Kripke structures and that combinators for transition systems and the issue of behavioural equivalence, as the basis for defining process algebras, have received more attention in the setting of Labelled Transition Systems.

The point we want to make with this paper is that there is really no need for taking a definite standing between LTS's and KS's as semantic models. For example, our results will enable one to use ones favorite process algebra to describe system behaviour as an LTS and to use CTL* to specify the requirement the system has to comply with. The model checker for CTL* or (better) for its branching time subset CTL can then be used to check whether a given process satisfies the required properties.

We will introduce an action based version of CTL* interpreted over LTS's (we will call it ACTL*) which is the natural analogue of CTL* in a setting where transitions are labelled. The new logics contains relativized modalities (e.g. $X_a\varphi$ - to be read "the next transition is labelled with an action a and the subsequent path satisfies φ"-) as demanded by the interpretation model and is more expressive than HML. Together with the new logic, we will introduce two transformation functions from KS's to LTS's and viceversa which preserve essential properties of systems. In correspondence of the two transformation functions, we will define two mappings between the logics, one from CTL* to ACTL* and the other in the opposite direction, which, in combination with the functions on the models, preserve truth. We will prove that, if $\mathcal{A}$ is an LTS, $\mathcal{K}$ is a KS , the $\mathfrak{ks}$'s and the $\mathfrak{lts}$'s are the transformation functions, and the two $\models$'s are the satisfaction relation, we have:

- $\mathcal{A}, \rho \models \varphi$ iff $\mathfrak{ks}(\mathcal{A}), \mathfrak{ks}(\rho) \models \mathfrak{ks}(\varphi)$

and

- $\mathcal{K}, \rho \models \varphi$ iff $\mathfrak{lts}(\mathcal{K}), \mathfrak{lts}(\rho) \models \mathfrak{lts}(\varphi)$.

Thus, one might say that the two logics are essentially equivalent.

Like it has been done for CTL*, we will introduce a branching time subset of ACTL* which we will call ACTL. Moreover, we will present a linear time translation from ACTL to CTL. This permits linear model checking for ACTL via reduction to CTL model checking. Also ACTL appears to be rather expressive and we argue that it can be used to express various interesting properties of concurrent systems. We will conclude the paper with a brief discussion on the discriminative power of ACTL without next time operators, be them relativized or not. We will argue that this restricted logic induces on transition systems the same equivalence as the divergence sensitive version of the branching bisimulation equivalence of [GW89] as presented in [DV90].

As mentioned above, our results about ACTL permit equipping verification tools for process algebras with a model checker which is linear in the size of the formula being checked; at the best of our knowledge almost all existing tools rely on model checking variants of the μ-calculus and the algorithm for this is exponential in the size of formulae. A type of work similar to ours in this respect is presented in [JKP90]. These authors do stick to CTL as a logic for LTS's but substantially change its satisfaction relation; in a sense they have a relativized satisfaction relation ($<a, s> \models \varphi$) instead of our relativized modality ($X_a\ \varphi$). The expressive power of the two languages seems similar, but our satisfaction relation is more immediate. It is also worth mentioning that our transformation from LTS's to KS's is linear while theirs is quadratic. Besides, they do not consider invisible actions and we have not been able to generalize their approach to systems with silent steps in a way that would preserve some behavioural equivalence.

We think that our new logic for transition systems can play an important role as an intermediate between HML and the modal μ-calculus. It is well known that HML is not expressive enough and that model checking for the modal μ-calculus requires exponential time. ACTL is sufficiently expressive to describe safety and liveness properties but permits model checking in linear time.

2. ACTL*: A logic for Labelled Transition Systems

In this section, we introduce our action based logic and elaborate upon its expressivity. Firstly, we provide the necessary definitions about labelled transition systems and their runs. Then, we describe the logic and introduce auxiliary modalities which will be useful in the sequel.

Definition 2.1. (*Labelled Transition Systems*)
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 usually written as $r \text{–}\alpha\text{→} s$.

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. ♦

Remark 2.2. (*Finiteness assumptions are not essential*)
The assumption that the set A of actions is finite and non-empty is made for technical reasons. The results of this paper can be generalized to arbitrary sets of actions if either one is willing to use infinitary disjunctions in the logics or to restrict attention to those LTS's for which the set of labels which actually occur in transitions is finite. ♦

Definition 2.3. (*Notation for strings*)
Let K be any set. K^* stands for the set of finite sequences of elements of K; K^ω denotes the set of infinite sequences of elements of K; K^∞ stands for $K^\omega \cup K^*$. Concatenation of sequences is denoted by juxtaposition; λ denotes the empty sequence; $|\pi|$ denotes the length of a sequence π. ♦

Definition 2.4. (*Paths and runs over LTS's*)

Let $\mathcal{A} = (S, A, \rightarrow)$ be a LTS.

- 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; if a path cannot be extended anymore because it is either infinite or ends in a state without outgoing transitions, it 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 $\mathrm{first}(\rho) = s$ and $\mathrm{path}(\rho) = \pi$; moreover, if π is finite then $\mathrm{last}(\rho)$ denotes the last state of π; a *maximal run* is a run whose second element is a fullpath;
- with $\rho < \theta$ and $\rho \leq \theta$ we indicate that run θ is a proper suffix, respectively a suffix, of run ρ;
- concatenation of runs is denoted by juxtaposition; concatenation is a partial operation: $\rho\theta$ is only defined if ρ is a finite run and $\mathrm{last}(\rho)=\mathrm{first}(\theta)$.
- we write $\mathrm{run}_{\mathcal{A}}(s)$, or just $\mathrm{run}(s)$, for the set of runs from s and $\mu\mathrm{run}_{\mathcal{A}}(s)$, or just $\mu\mathrm{run}(s)$, for the set of maximal runs from s;
- we write $\mathrm{run}_{\mathcal{A}}$ and $\mu\mathrm{run}_{\mathcal{A}}$ for the set of runs resp. maximal runs in $\mathcal{A}$.

We let π, ... range over paths and ρ, σ, ... over runs. ♦

Definition 2.5. (*ACTL*: an Action based Computation Tree Logic*)
The syntax of the logic *ACTL* (Action based CTL*)* is defined by the following grammar where we let φ, φ', ... range over ACTL*-formulas:

$$\varphi ::= T \mid \neg\varphi \mid \varphi\wedge\varphi' \mid \exists\varphi \mid \varphi U \varphi' \mid X\varphi \mid X_a\varphi.$$ ♦

Definition 2.6. (*Satisfaction relations for ACTL**)
Let $\mathcal{A} = (S, A, \rightarrow)$ be a LTS. *Satisfaction* of an ACTL*-formula φ by a run ρ, notation $\mathcal{A},\rho \models \varphi$ or just $\rho \models \varphi$, is defined inductively by:

- $\rho \models T$ always;
- $\rho \models \neg\varphi$ iff $\rho \not\models \varphi$;
- $\rho \models \varphi\wedge\varphi'$ iff $\rho \models \varphi$ and $\rho \models \varphi'$;
- $\rho \models \exists\varphi$ iff there exists a run $\theta \in \mu\mathrm{run}(\mathrm{first}(\rho))$ such that $\theta \models \varphi$;
- $\rho \models \varphi U \varphi'$ iff there exists a θ with $\rho \leq \theta$ such that $\theta \models \varphi'$ and for all $\rho \leq \eta < \theta$: $\eta \models \varphi$;
- $\rho \models X\varphi$ iff there exist s, α, s', θ such that $\rho = (s,(s,\alpha,s'))\theta$ and $\theta \models \varphi$;
- $\rho \models X_a\varphi$ iff there exist s, s', θ such that $\rho = (s,(s,a,s'))\theta$ and $\theta \models \varphi$.

For $s \in S$ and $\varphi \in L$ we define $s \models \varphi$ iff $(s, \lambda) \models \varphi$. ♦

Notation 2.7. (*Auxiliary notation for ACTL**)
We write

- F for $\neg T$,
- $\varphi \vee \varphi'$ for $\neg(\neg\varphi \wedge \neg\varphi')$,
- $\vee\{\varphi_i \mid i \in \{i1,..,in\}\}$ for $\varphi_{i1} \vee .. \vee \varphi_{in}$ (by convention $\vee\{\varphi_i \mid i \in \varnothing\} = F$),
- $\varphi \Rightarrow \varphi'$ for $\neg\varphi\vee\varphi'$,
- $\forall\varphi$ for $\neg\exists\neg\varphi$,
- $F\varphi$ for $T\ U\ \varphi$,
- $G\varphi$ for $\neg F \neg\varphi$,
- $X_\tau\ \varphi$ for $X\varphi \wedge \neg(\vee\{X_a\ \varphi \mid a \in A\})$. ♦

In order to define more powerful modalities which will significantly shorten our notation, we introduce a tiny auxiliary logic of actions.

Definition 2.8. *(Action formulas)*
The collection *Afor* of *action formulas* over A is defined by the following grammar where we let χ, χ', range over action formulas:

$$\chi ::= \ a \in A \ \mid \ \neg\chi \ \mid \ \chi\wedge\chi'.$$

We write T for $\neg(a_0\wedge\neg a_0)$ where a_0 is some arbitrarily chosen action. Also, we use the abbreviations F, $\varphi\vee\varphi'$, etc. that were introduced for ACTL*. ♦

Definition 2.9. (*Satisfaction relations for Afor*)
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 $a \models \chi'$. ♦

Definition 2.10. *(Derived modalities)*
By using the notion of action formulas we can introduce a number of very useful modalities. We will write

- $X_\chi\, \varphi$ for $\vee\{X_a\, \varphi \mid a\in A \text{ and } a \models \chi\}$,
- $\varphi\, {}_\chi U_{\chi'}\, \varphi'$ for $(\varphi\wedge(X_\tau T \vee X_\chi T))\ U\ (\varphi \wedge X_{\chi'}\, \varphi')$,
- $\varphi\, {}_\chi U\, \varphi'$ for $(\varphi\wedge(X_\tau T \vee X_\chi\ T))\ U\ \varphi'$,
- $\varphi <a> \varphi'$ for $\exists(\varphi\, {}_F U_a\, \varphi')$,
- $\varphi <\varepsilon>\varphi'$ for $\exists(\varphi\, {}_F U\, \varphi')$,
- $<k> \varphi$ for $T <k> \varphi$,
- $[k]\, \varphi$ for $\neg <k>\neg\varphi$. ♦

Intuitively, a path satisfies $X_\chi\, \varphi$ if it starts with a visible action that satisfies χ and moreover the remainder of the path satisfies φ. A path satisfies $\varphi\, {}_\chi U_{\chi'}\, \varphi'$ if eventually it contains a visible transition whose label satisfies χ' with a remainder satisfying φ', whereas at any moment before this event φ holds and all visible labels satisfy χ. A path satisfies $\varphi\, {}_\chi U\, \varphi'$ if some suffix satisfies φ' and at any moment before φ holds and all visible actions satisfy χ. Please, note that $\varphi\, {}_T U\, \varphi'$ is equivalent to $\varphi\ U\ \varphi'$. The logic ACTL* is more expressive than the Hennessy-Milner logic with until operators that was introduced in [DV90]; these modalities are just the modalities $\varphi <a> \varphi'$ and $\varphi <\varepsilon>\varphi'$ as defined above. The formula $\varphi <a> \varphi'$ holds in a state if it is possible to do some τ-transitions followed by an a-step such that after the a-step φ' holds and at any moment before φ holds. The formula $\varphi <\varepsilon>\varphi'$ is valid if after zero or more τ-steps φ' holds and at any moment before φ holds. Our diamond operator $<a>\varphi$ is slightly different from the diamond operator in the standard Hennessy-Milner Logic (HML) of [HM85]. Our modality requires that there exists a path consisting of a number of τ's followed by an a-transition such that φ holds immediately after the a-step, whereas in standard HML it is allowed to have an additional number of τ-steps between the a-step and the φ-state. The diamond operator $<a>\varphi$ of standard HML is rendered by our $<a><\varepsilon>\varphi$. Finally, we introduce the modality $[k]\, \varphi$, which is the dual of $<k>\, \varphi$.

Example 2.11. (*Expressivity of ACTL**)
ACTL* allows us to express in a concise way interesting properties of reactive systems. For instance, a one bit buffer will satisfy the following property

$$\forall G([in0]\ (\forall\ (T\ {}_{\neg(in0 \vee in1 \vee out1)}U_{out0}\ T)))$$

which expresses that always after a 0 is placed in the buffer eventually the buffer will release it; moreover, as long as this event has not yet occurred, no bit will be accepted by the buffer and also no 1 will be released. ♦

3. CTL*: a logic for Kripke Structures

Those readers who are familiar with the logic CTL* will have realized that the logic ACTL* resembles it very closely. In this section, we will recall the definitions of CTL* and of the Kripke structures which serve as models for it; this will allow us to investigate, later on in the paper, the relationships between the two logics more closely.

Definition 3.1. (*Kripke structures*)
A *Kripke structure* (or *KS*) is a 4-tuple $\mathcal{K} = (S, AP, \mathcal{L}, \rightarrow)$ where:
- S is a set of *states*;
- AP is a finite, nonempty set of *atomic proposition names* ranged over by p, q, ...;
- $\mathcal{L}: S \rightarrow 2^{AP}$ is the *proposition labelling*;
- $\rightarrow\ \subseteq S \times S$ is the *transition relation*; an element $(r,s) \in\ \rightarrow$ is called a *transition* and is usually written as $r \rightarrow s$.

The notations for runs that were introduced for LTS's carry over to Kripke structures in the obvious way. The only difference is that transitions are now no longer triples but pairs. ♦

Definition 3.2. (*CTL**)
The syntax the logic CTL* is defined by the following grammar where we let φ, φ', ... range over CTL* formulas and p over atomic proposition names:

$\varphi ::= p \ \mid\ \neg\varphi \ \mid\ \varphi\wedge\varphi' \ \mid\ \exists\varphi \ \mid\ \varphi U\varphi' \ \mid\ X\varphi.$

We write T for $\neg(p_0\wedge\neg p_0)$ where p_0 is some arbitrarily chosen atomic proposition name. Also, we use the abbreviations F, φ∨φ', etc. that were introduced for the logic ACTL*. ♦

Definition 3.3. (*Satisfaction relation for CTL**)
Let $\mathcal{K} = (S, AP, \mathcal{L}, \rightarrow)$ be a Kripke structure. *Satisfaction* of a CTL* formula φ by a run ρ, notation $\mathcal{K}, \rho \models \varphi$ or just $\rho \models \varphi$, is defined inductively by:
- $\rho \models p$ iff $p \in \mathcal{L}(\mathrm{first}(\rho))$;
- $\rho \models \neg\varphi$ iff $\rho \not\models \varphi$;

- $\rho \models \varphi \wedge \varphi'$ iff $\rho \models \varphi$ and $\rho \models \varphi'$;
- $\rho \models \exists\varphi$ iff there exists a run $\theta \in \mu run(first(\rho))$ such that $\theta \models \varphi$;
- $\rho \models \varphi U \varphi'$ iff there exists a θ with $\rho \leq \theta$ such that $\theta \models \varphi'$ and for all $\rho \leq \eta < \theta$: $\eta \models \varphi$;
- $\rho \models X\varphi$ iff there exist s, s', θ such that $\rho = (s,(s,s'))\theta$ and $\theta \models \varphi$.

For $s \in S$ and $\varphi \in$ CTL* we define $s \models \varphi$ iff $(s, \lambda) \models \varphi$. ♦

For the sake of clarity, please notice that the above relation, which is the standard satisfaction relation for CTL* (see e.g. [ES89]), was called *satisfaction with respect to maximal paths* in [DV90] and it was there written as $\rho \models_{\mu} \varphi$.

4. Actions vs States: relating ACTL* and CTL*

To relate the two logics presented in the previous sections, we will need some preliminary work which allows us to relate the different structures on which they are interpreted, namely Kripke structures and Labelled Transition Systems. We will make use of two (slightly modified) transformation functions introduced in [DV90b]. For both constructions, the generated system has almost the same structure as that of the original one. The first construction builds a Kripke structure from a labelled transition system by splitting transitions labelled by visible actions and creating a new states for each of them, labelled with the label of the original transition. The second construction builds a transition system from a Kripke structure by labellling the original transitions with the set of atomic propositions labelling their target state and by splitting all the original states to avoid that atomic propositions associated to states without incoming transitions be lost. Together with the two tranformation functions on the modelling structures we will present two transformation functions for the two logics and will then prove that truth of logical formula is preserved by them.

Definition 4.1. *(From LTS's to KS's)*
Let $\mathcal{A} = (S, A, \rightarrow)$ be a LTS and $\perp$ be fresh symbol not in A. The KS, $\mathfrak{ks}(\mathcal{A})$, is defined as $(S', \mathbf{AP}, \mathcal{L}, \rightarrow')$ where

- $S' = S \cup \{(r,a,s) \mid a \in A \text{ and } r \xrightarrow{a} s\}$;
- $\mathbf{AP} = A \cup \{\perp\}$;
- $\rightarrow' = \{(r,s) \mid r \xrightarrow{\tau} s\} \cup \{(r,(r,a,s)) \mid r \xrightarrow{a} s\} \cup \{((r,a,s),s) \mid r \xrightarrow{a} s\}$;
- For r, $s \in S$ and $a \in A$: $\mathcal{L}(s) = \{\perp\}$ and $\mathcal{L}((r,a,s)) = \{a\}$.

The mapping can be adapted in the obvious way to runs; it is sufficient to replace each transition (r,a,s) by the pair of transitions (r,(r,a,s)) ((r,a,s),s). ♦

Mapping $\mathfrak{ks}$ is nothing more than the composition of the mappings $\mathbf{tr}_2$ and KS as presented in [DV90b]. Essentially, what $\mathfrak{ks}$ does is to introduce an intermediate state for each visible transition in the LTS, and to label the fresh states with the label of the transition and the old ones with $\{\perp\}$ while forgetting the label of the transitions.

In correspondence of the mapping from LTS's to KS's, we have a mapping from ACTL* formulae to CTL* formulae which preserves truth.

Definition 4.2. (*From ACTL* to CTL**)
The mapping $\mathfrak{ks}$: ACTL*→CTL* is inductively defined by:

- $\mathfrak{ks}(T) = T$,
- $\mathfrak{ks}(\neg\varphi) = \neg\,\mathfrak{ks}(\varphi)$,
- $\mathfrak{ks}(\varphi\wedge\varphi') = \mathfrak{ks}(\varphi)\wedge\mathfrak{ks}(\varphi')$,
- $\mathfrak{ks}(\exists\varphi) = \exists\,\mathfrak{ks}(\varphi)$,
- $\mathfrak{ks}(\varphi U\varphi') = (\bot \Rightarrow \mathfrak{ks}(\varphi))\,U\,(\bot \wedge \mathfrak{ks}(\varphi'))$,
- $\mathfrak{ks}(X\varphi) = X((\bot \wedge \mathfrak{ks}(\varphi)) \vee \bigvee\{(a \wedge X(\mathfrak{ks}(\varphi))) \mid a \in A\})$,
- $\mathfrak{ks}(X_a\,\varphi) = X(a \wedge X(\mathfrak{ks}(\varphi)))$. ♦

Theorem 4.3. (*$\mathfrak{ks}$'s preserve truth*)
Let $\mathcal{A}$ be a LTS, let ρ be a run of $\mathcal{A}$ and let φ be an ACTL*-formula, then:

$\mathcal{A},\rho \models \varphi$ iff $\mathfrak{ks}(\mathcal{A}),\mathfrak{ks}(\rho) \models \mathfrak{ks}(\varphi)$. ♦

The translation $\mathfrak{ks}$ is linear in the size of the formulas, except for the case of the X-modality which can cause an exponential blowup. Note however that by replacing in ACTL* the X-modality by the relativized modality X_τ, which would mean no loss in expressivity since X can be defined in terms of the modalities X_τ and X_a, it becomes easy to give a linear version of $\mathfrak{ks}$ by defining:

$\mathfrak{ks}(X_\tau\,\varphi) = X(\bot \wedge \mathfrak{ks}(\varphi))$.

Somewhat arbitrarily, we have decided to use X in ACTL* instead of X_τ because otherwise the reverse translation $\mathfrak{lts}$: CTL*→ACTL* which we present below would not be linear anymore.

Definition 4.4. (*From KS's to LTS's*)
Let $\mathcal{K} = (S, AP, \mathcal{L}, \rightarrow)$ be a Kripke structure. The LTS $\mathfrak{lts}(\mathcal{K})$ is defined as $(S', A', \rightarrow')$ where

- $S' = S \cup \{\underline{s} \mid s \in S\}$;
- $A' = 2^{AP} \cup \{\bot\}$ (we assume $\bot$ is a fresh symbol);
- $\rightarrow' = \{(s, \bot, \underline{s}) \mid s \in S\} \cup$
  $\{(\underline{s}, \mathcal{L}(s), s) \mid s \in S\} \cup$
  $\{(r, \tau, s) \mid r,s \in S,\ r \rightarrow s \text{ and } \mathcal{L}(r) = \mathcal{L}(s)\} \cup$
  $\{(r, \mathcal{L}(s), s) \mid r,s \in S,\ r \rightarrow s \text{ and } \mathcal{L}(r) \neq \mathcal{L}(s)\}$.

The mapping can be easily adapted to runs by defining:

$\mathfrak{lts}(s0, (s0,s1)(s1,s2)\ \ldots) = (s0, (s0,L1,s1)\ (s1,L2,s2)\ldots)$

where $Ln{+}1 =_{def}$ **if** $\mathcal{L}(sn{+}1) = \mathcal{L}(sn)$ **then** τ **else** $\mathcal{L}(sn{+}1)$. ♦

The above transformation $\mathfrak{lts}$ is the result of a minor modification of the transformation LTS∘**tr** of [DV90b]. We could not directly use transformation **tr** here because in general it does not preserve all the essential information in a Kripke structure. The modification that we presented above uses an idea which was also exploited in the definition of transformation $\mathfrak{ks}$: just like $\mathfrak{ks}$ splits each (visible) transition into two consecutive transitions, the transformation lts splits each state into two adjacent states.

We now present the translation function lts from CTL* to our new logic ACTL*. It is the identity function for all operators but for atomic propositions.

Definition 4.5. (*From CTL* to ACTL**)
The mapping lts: CTL*→ACTL* is inductively defined by:

- $\mathit{lts}(p) \quad = \langle\bot\rangle (\vee \{\langle\alpha\rangle T \mid p \in \alpha \subseteq AP\})$,
- $\mathit{lts}(\neg\varphi) \quad = \neg\, \mathit{lts}(\varphi)$,
- $\mathit{lts}(\varphi\wedge\varphi') \quad = \mathit{lts}(\varphi) \wedge \mathit{lts}(\varphi')$,
- $\mathit{lts}(\exists\varphi) \quad = \exists\, \mathit{lts}(\varphi)$,
- $\mathit{lts}(\varphi U \varphi') \quad = \mathit{lts}(\varphi)\, U\, \mathit{lts}(\varphi')$,
- $\mathit{lts}(X\varphi) \quad = X\, \mathit{lts}(\varphi)$. ♦

We recall from Definition 2.10, that $\langle\bot\rangle\varphi$ means that there exists a path containing any number of silent moves and then a transition labelled by $\bot$ which leads to a state satisfying φ, formally we have: $\langle\bot\rangle\varphi \equiv \exists ((X_\tau T)\ U\ (X_\bot \varphi))$.

Theorem 4.6. (*lts's preserve truth*)
Let $\mathcal{K}$ be a Kripke structure, let ρ be a run of $\mathcal{K}$ and let φ be a CTL*-formula, then:

$\mathcal{K},\rho \models \varphi$ iff $\mathit{lts}(\mathcal{K}),\mathit{lts}(\rho) \models \mathit{lts}(\varphi)$. ♦

The huge disjunction which we need in Definition 4.5 in order to deal with atomic proposition names results from a kind of mismatch between LTS's and Kripke structures: transitions in LTS's are labelled just by actions, whereas states in KS's are labelled with sets of atomic proposition names. Had we used a slightly different type of LTS's which allowed sets of actions to occur as label rather than single actions, it would have been natural to equip the logic ACTL* with modalities X_p for p an element of a transition label. In that case the translation lts could have been simplified even further by defining:

$\mathit{lts}(p) = \langle\bot\rangle \langle p\rangle T$.

The translation ks would not be more complicated in this approach. In this paper we have chosen not to label transitions with sets of actions, but just with actions, in order to preserve more closely the connection with the LTS's semantics of a wide variety of process description languages.

Now we have defined a translation ks from LTS's to KS's and another translation lts from KS's to LTS's, a natural question to ask is what are the relationships between a LTS $\mathcal{A}$ and the LTS $\mathit{lts}(\mathit{ks}(\mathcal{A}))$, or between a KS $\mathcal{K}$ and the KS $\mathit{ks}(\mathit{lts}(\mathcal{K}))$. It is not hard to see that, for instance, $\mathcal{A}$ and $\mathit{lts}(\mathit{ks}(\mathcal{A}))$ are not directly related via some behavioural equivalence like trace equivalence or bisimulation equivalence. Although we still think that there exist interesting behavioural relationships, we have decided not to adress these issues in the present paper.

5. ACTL: a new branching time logic for LTS's

In the previous sections we have introduced the logic ACTL* and shown that it is a very expressive logic which is equivalent to CTL* in the sense that model checking for ACTL* can be reduced to model checking for CTL* and vice versa. However, since model checking for CTL* is in PSPACE [EL87], and because of our polynomial reduction of model checking for CTL* to model checking for ACTL*, this means that model checking for ACTL* is in PSPACE.

The branching time logic CTL is a subset of CTL* which has an efficient model checking algorithm with complexity $O((|S| + |\rightarrow|) \times |\varphi|)$ [CES86]. Moreover an efficient implementation exists in the Extended Model Checker (EMC), developed at CMU. Therefore it becomes interesting to look for subsets of ACTL* which can be translated effieciently to CTL. Of course one should aim at having this subsets as large as possible in order not to loose too much of the expressive power of ACTL*. In this section we present the logic ACTL which is essentially a subset of ACTL* and which has the desired properties mentioned above. However, before we come to ACTL, we will first define the logic CTL to which it is closely related.

Definition 5.1. (*CTL*)
The set of formulas CTL is defined as the smallest set of state formulas such that:
- if $p \in$ **AP**, then p is a state formula;
- if φ and φ' are state formulas, then $\neg\varphi$ and $\varphi\wedge\varphi'$ are state formulas;
- if π is a path formula, then $\exists\pi$ is a state formula;
- if φ and φ' are state formulas, then $X\varphi$ and $\varphi U \varphi'$ are path formulas;
- if π is a path formula, then so is $\neg\pi$.

We let φ,... range over CTL state formulas and π,... over CTL path formulas. ♦

Clearly, CTL is just a subset of CTL*. Thus the definition of the satisfaction relation for CTL* carries over to CTL. Now here comes our proposal for ACTL:

Definition 5.2. (*ACTL*)
The set of formulas ACTL is defined as the smallest set of state formulas such that:
- T is a state formula;
- if φ and φ' are state formulas, then $\neg\varphi$ and $\varphi\wedge\varphi'$ are state formulas;
- if π is a path formula, then $\exists\pi$ is a state formula;
- if φ and φ' are state formulas and χ and χ' are action formulas, then $X_\chi\varphi$, $X_\tau\varphi$, $\varphi\ {}_\chi U_{\chi'}\ \varphi'$ and $\varphi\ {}_\chi U\ \varphi'$ are path formulas;
- if π is a path formula, then so is $\neg\pi$.

Again, we let φ,... range over state formulas and π,... over path formulas. ♦

The modalities $X_\chi\varphi$, $\varphi\ {}_\chi U_{\chi'}\ \varphi'$ and $\varphi\ {}_\chi U\ \varphi'$ used above can be seen as compact notation for ACTL* formulae, thus ACTL is a proper subset of ACTL* and it inherits the satisfaction relation from ACTL*.

For translating ACTL to CTL we cannot just use the mapping ks: ACTL*→CTL*, restricted to ACTL and with the understanding that modalities like $X_\chi\varphi$ are expanded to ACTL*. For instance, consider the

ACTL formula of the form $X_\chi\ \varphi$; if we expand notation, we obtain an ACTL* formula of the form $\vee\{X_a\ \varphi \mid a\in A \text{ and } a \models \chi\}$. Mapping $\mathfrak{ks}$ will translate this to a CTL* formula $\vee\{X(a\wedge X(\mathfrak{ks}(\varphi))) \mid a\in A \text{ and } a \models \chi\}$ which is not a CTL formula; indeed, in CTL no conjunction or disjuntion of X-modalities is allowed. Thus we have to modify the mapping $\mathfrak{ks}$. The reader may check that the mapping $\mathfrak{ks}'$ which is defined below does yield CTL formulas.

Definition 5.3. (*From ACTL to CTL*)
The mapping $\mathfrak{ks}'$: ACTL→CTL is inductively defined by:

- $\mathfrak{ks}'(T) = T$,
- $\mathfrak{ks}'(\neg\varphi) = \neg\,\mathfrak{ks}'(\varphi)$,
- $\mathfrak{ks}'(\varphi\wedge\varphi') = \mathfrak{ks}'(\varphi) \wedge \mathfrak{ks}'(\varphi')$,
- $\mathfrak{ks}'(\exists\pi) = \exists\,\mathfrak{ks}'(\pi)$,
- $\mathfrak{ks}'(\varphi\ {}_\chi U_{\chi'}\ \varphi') = ((\perp \wedge \mathfrak{ks}'(\varphi)) \vee (\neg\perp \wedge \chi))\ U\ (\neg\perp \wedge \exists((\neg\perp \wedge \chi')\ U\ (\perp \wedge \mathfrak{ks}'(\varphi'))))$,
- $\mathfrak{ks}'(\varphi\ {}_\chi U\ \varphi') = ((\perp \wedge \mathfrak{ks}'(\varphi)) \vee (\neg\perp \wedge \chi))\ U\ (\perp \wedge \mathfrak{ks}'(\varphi'))$,
- $\mathfrak{ks}'(X_\chi\varphi) = X(\neg\perp \wedge \chi \wedge \exists X(\mathfrak{ks}'(\varphi)))$,
- $\mathfrak{ks}'(X_\tau\varphi) = X(\perp \wedge \mathfrak{ks}'(\varphi))$,
- $\mathfrak{ks}'(\neg\pi) = \neg\,\mathfrak{ks}'(\pi)$. ♦

The key result about $\mathfrak{ks}'$ is that it preserves truth. An interesting propery is that the size of $\mathfrak{ks}'(\varphi)$ is linear in the size of φ.

Theorem 5.4. (*$\mathfrak{ks}$ and $\mathfrak{ks}'$ together preserve truth*)
Let $\mathcal{A}$ be a LTS, let s be a state of $\mathcal{A}$ and let φ be an ACTL-formula. Then:

$\mathcal{A}, s \models \varphi$ iff $\mathfrak{ks}(\mathcal{A}), s \models \mathfrak{ks}'(\varphi)$. ♦

As a corollary of the above theorem, we have that there exists a model checking algorithm 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, Theorem 5.4 says that in order to determine whether $\mathcal{A}$, s $\models \varphi$ it suffices to check whether $\mathfrak{ks}(\mathcal{A})$, s $\models \mathfrak{ks}'(\varphi)$. We can easily compute $\mathfrak{ks}(\mathcal{A})$ in $O(|S|+|\rightarrow|)$-time and the number of states and transitions of $\mathfrak{ks}(\mathcal{A})$ will be of order $|S|+|\rightarrow|$. The formula $\mathfrak{ks}'(\varphi)$ can be computed in $O(|\varphi|)$-time and its size will be of order $|\varphi|$. Next, we can apply the model checking algorithm for CTL of [CES86] which will terminate in $O((|S|+|\rightarrow|) \times |\varphi|)$-time.

ACTL is still a rather expressive logic in which safety and liveness properties can be formulated. The formula of Example 2.11 for instance is an ACTL formula. Also the next proposition shows that, given our design objective to find a subset of ACTL* which can be translated into CTL, we have still managed to preserve expressiveness: in combination with Theorem 4.6 and Theorem 5.4, the proposition says that ACTL is just as expressive as CTL in the setting with transformations $\mathfrak{lts}$ and $\mathfrak{ks}$ between KS's and LTS's.

Proposition 5.5. (*ACTL has the same expressive power of CTL*)
Let φ be a CTL formula. Then $\mathfrak{lts}(\varphi)$ is an ACTL formula. ♦

An interesting feature of the mapping ks' is that it maps all formulae without the relativized next time operators into formulae of CTL which do not cointain the next operator. This fact allows us to conclude this section and the paper with a few remarks about the relationships between the equivalence induced on LTS's by our new logics and the (divergence sensitive version of the) branching bisimulation equivalence of [GW89].

In fact, by exploiting the results of [DV90] about the correspondence between the equivalence induced by CTL- X and branching bisimulation equivalence and by relying on Theorem 5.4 above, we can deduce that ACTL-$\{X_{\chi,} X_{\tau}\}$ induces on finite LTS's the same identifications as divergence sensitive branching bisimulation equivalence. Due to the way the transformation function ks from ACTL* to CTL* is defined, we cannot use the same chain of reasoning to prove that also the equivalence induced by ACTL*- $\{X, X_a\}$ coincides with branching bisimulation. It is, however, possible to define, in the same vein of ks', a new mapping ks" to CTL*-X from ACTL* without next operators but with the relativized until modalities $_{\chi}U_{\chi'}$; and this would enable us to conclude that also the richer logics is in full agreement with divergence sensitive branching bisimulation equivalence.

6. References

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

[CES89] 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.

[CLM89] E.M. Clarke, D.E. Long & K.L. Macmillan: Compositional Model Checking. In *Proceedings 4th Annual Symposium on Logic in Computer Science (LICS)*, Asilomar, California, IEEE Computer Society Press, Washington, 1989, pp. 353-362.

[CPS88] Cleaveland, R., Parrow, J., Steffen, B. The Concurrency Workbench. In *Automatic Verification Methods for Finite State Systems* (J. Sifakis, ed.) Lecture Notes in Computer Science **407**, Springer-Verlag, 1990, pp. 24-37.

[DV90a] R. De Nicola, & F.W. Vaandrager: Three Logics for Branching Bisimulations (Extended Abstract) in Proc. of the 5th Annual Symposium on Logic in Computer Science (*LICS '90*), Philadelphia, USA, June 1990, IEEE Computer Society Press, Los Alamitos, CA, 1990, pp. 118-129.

[DV90b] R. De Nicola, & F.W. Vaandrager: Three Logics for Branching Bisimulations, CWI Report CS-R9012, 1990.

[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.

[EL87] E.A. Emerson & C.L. Lei: Modalities for Model Checking: Branching Time Strikes Back. *Science of Computer Programming*, **6**, 1987.

[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, de Roever and Rozenberg, eds.) Lecture Notes in Computer Science **354**, Springer-Verlag, 1989, pp. 123-172.

[GV90] J.F. Groote & F.W. Vaandrager: An Efficient Algorithm for Branching Bisimulation and Stuttering Equivalence.In *Proceedings ICALP '90*, Warwick, Lecture Notes in Computer Science, Springer-Verlag, 1990.

[GW89] R.J. van Glabbeek & W.P. Weijland: Branching Time and Abstraction in Bisimulation Semantics (extended abstract). In *Information Processing '89* (G.X. Ritter, ed.), Elsevier Science Publishers B.V. (North Holland), 1989, pp. 613-618.

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

[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.) Lecture Notes in Computer Science **407**, Springer-Verlag, 1990, pp. 179-188.

[Lam83] L. Lamport: What Good Is Temporal Logic? , *Information Processing '83* (R.E.A. Mason, ed.) Elsevier Science Publishers B.V. (North Holland), 1983, pp. 657-668.

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

[Sti89] C. Stirling: Temporal Logics for CCS, in *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, (de Bakker, de Roever and Rozenberg, eds.) Lecture Notes in Computer Science **354**, Springer-Verlag, 1989, pp. 660-672.