Consiglio Nazionale delle Ricerche

ISTITUTO DI ELABORAZIONE DELLA INFORMAZIONE

PISA

T.R.

Modelling Transition Systems within an Action Based Logic

A. Fantechi, S Gnesi, G. Ristori

Nota Interna B4-49

Dicembre-1995

Modelling Transition Systems within an Action Based Logic

Alessandro Fantechi
Dip. di Sistemi e Informatica, Univ. di Firenze

Stefania Gnesi
Istituto di Elaborazione dell'Informazione del C.N.R., Pisa

Gioia Ristori
Dipartimento di Informatica, Univ. di Pisa

Abstract

In this paper we study the ability of a logic to finitely characterize the complete behaviour of a system; given a Labelled Transition System $\mathcal{A}$, we look for a formula of the logic that is satisfied by all and only the Labelled Transition Systems that are equivalent to $\mathcal{A}$. The main result of this paper is that we are able to finitely characterize finite state Labelled Transition Systems, with respect to the classical notion of strong bisimulation equivalence, using a logic without fixed points.

1 Introduction

Temporal logics are widely used in order to specify properties of reactive systems. Different logics have been defined in order to deal with different semantic models for reactive systems [5,16]. An important class of temporal logics is constituted by the so called action-based temporal logics [10], that allow properties of systems specified by means of finite state Labelled Transition Systems (LTSs in short) to be expressed. LTSs are state automata, in which the states (representing agents) are related by action labelled arrows (representing a behavioural relation called transition relation). Several behavioural equivalence/preorder relations on LTSs allow the relationships between descriptions of a reactive system given at a different level of abstraction to be established.

Here we concentrate our attention on the ability of a logic to finitely characterize the complete behaviour of a system; given a LTS $\mathcal{A}$, we look for a formula of the logic that is satisfied by all and only the LTSs that are equivalent to $\mathcal{A}$. A logic formula (named *characteristic formula* in [17]) is hence associated with every finite state LTS.

In the framework of the state-based logics, finite states Kripke Structures have been characterized [2] by using the CTL logic [5]. Since there exist conservative translations from LTSs to Kripke Structures [4,11] (i.e. strong bisimulation [15] is preserved by the translations), the approach in [2] allows LTSs to be characterized in CTL, by computing the characteristic formula of the Kripke Structure associated with the given LTS. In the framework of action-based

logics, a logical characterization of finite states LTSs up to strong bisimulation equivalence can be drawn out from [17], in which the μ-calculus logic [12] is used. The fixed point operator of μ-calculus is necessary, in [17], in order to express recursion.

In this paper we show that finite states LTSs can be faithfully modelled within an action-based logic that is strictly less expressive than CTL and μ-calculus. The logic we choose is the action-based version of CTL, that is ACTL [4]. In [9] it is shown that the ACTL expressive power is less than or equal that of CTL, while in [8] it is shown that the ACTL expressive power is less than or equal that of μ-calculus; counterexamples can be found that show that the inclusions are strict.

Besides the difference between the state-based and the action-based framework and the expressive power of the CTL and ACTL logics, our approach, compared with that in [2], allows to forget any semantic information about the states of the modelled system. Actually, the characteristic formula for a given Kripke Structure is defined by first analysing the structure itself in order to find its characteristic number. The characteristic number is the depth of the tree representing the finite unfolding of the structure. It needs to distinguish all the non equivalent states and this implies a semantic analysis of the structure. Our method is instead based on the association of a Regular CCS expression (involving only action prefixing, nondeterministic choice and recursion operators) with each state of a LTS, and then on the syntactical analysis of the *context degree* shown by the Regular CCS expressions associated with the states. The context degree of a state is the minimum expression unfolding that syntactically distinguishes the Regular CCS expression of a state from the expressions associated with the other states.

As a consequence of our result, we can notice that the ACTL logic, that is not able to express all the properties of finite state LTSs (for instance, it fails to express cyclic properties, whilst μ-calculus can), is however able to describe, in a faithful way, the behaviour of systems modelled by finite states LTSs.

Moreover, it can be noticed that ACTL allows a more efficient model checking than μ-calculus. This is useful if we consider that the characteristic formula of a LTS represents its equivalence class with respect to strong bisimulation, hence the equivalence of two systems can be automatically checked [3] by verifying that the model of a system satisfies the characteristic formula of the other.

2 Background

Labelled Transition Systems (LTSs in short) are 4-tuples $\mathcal{A} = (Q, q_0, Act \cup \{\tau\}, R)$, where Q is a finite set of states, $q_0 \in Q$ is the initial state, Act is a finite set of observable actions and τ is the unobservable action. Moreover, $R \subseteq Q \times (Act \cup \{\tau\}) \times Q$ is the transition relation. Whenever $(q, \alpha, q') \in R$ we will write $q \xrightarrow{\alpha} q'$.
The class of LTSs we take into account is that of systems with a finite set of states and actions, that is *finite states* and *finitely branching* LTSs.
Two LTSs are said *strongly equivalent*, or strongly bisimilar, if there exists a strong bisimulation relation between their initial states [15].

Each finite-state LTS can be described by means of a term of the regular fragment of CCS [14], that is a regular specification language whose semantic models are LTSs. The language of Regular CCS expressions is defined by the following syntax:

$$E ::= Nil \mid a.E \mid E + E \mid x$$

where a ranges over a finite set of action names Act and x ranges over a finite set of variables Var.

A Regular CCS term is a pair $(x, \mathcal{E})$ such that $x \in \text{Var}$ and $\mathcal{E}$ is a finite set of agent definitions $\{x_i \stackrel{\text{def}}{=} E_i\}_{x_i \in \text{Var}}$, where each E_i is a strongly guarded Regular CCS expression over Var and Act (we mean expressions in which every occurrence of variables is guarded by the occurrence of an action).

The semantic models of Regular CCS terms are finite state LTSs. Two terms are said strongly bisimilar if the corresponding LTSs are strongly bisimilar. The semantics of Regular CCS terms is defined by the following rules:

$$a.E \xrightarrow{a} E \qquad \frac{E \xrightarrow{a} E'}{E + F \xrightarrow{a} E',\ F + E \xrightarrow{a} E'} \qquad \frac{E \xrightarrow{a} E',\ (x \stackrel{\text{def}}{=} E) \in \mathcal{E}}{x \xrightarrow{a} E'}$$

The passage from LTSs to Regular CCS expressions can be done defining a system of equations, following [18], by using the choice operator (+) to express nondeterministic (finite) branching, the action prefix operator (.) to describe the labelled transitions, the inactive operator (*Nil*) to describe states without successors, and variables to describe recursion.

Definition 2.1 (Equation System associated with a LTS) *Let $\mathcal{A}$ be a LTS, with $Q = \{s_1, \ldots, s_n\}$. We associate with each state $s_i \in Q = \{s_1, \ldots, s_n\}$ a variable x_i and an equation $x_i \stackrel{\text{def}}{=} E_i$ as follows:*

- *if $s_i \not\rightarrow$ for all $\alpha \in Act_\tau$, then $x_i \stackrel{\text{def}}{=} Nil$*
- *if $s_i \xrightarrow{\alpha_{i,j_k}} s_j$, $j \in J$, $k \in K_j$, then $x_i \stackrel{\text{def}}{=} \sum_{j \in J, k \in K_j} \alpha_{i,j_k}.x_j$*

If s_m is the initial state of $\mathcal{A}$, then we define $(x_m, \mathcal{E})$ be the Regular CCS term associated with $\mathcal{A}$.

Example 2.2 (Equation system associated with a LTS) *Consider the LTS shown in Figure 1. The associated equation system is:*

$$\begin{cases} x_1 \stackrel{\text{def}}{=} a.x_1 + b.x_2 \\ x_2 \stackrel{\text{def}}{=} a.x_3 + b.x_2 \\ x_3 \stackrel{\text{def}}{=} a.x_4 + b.x_2 \\ x_4 \stackrel{\text{def}}{=} Nil \end{cases}$$

From now on, whenever there is no danger of confusion, we will use the variable notation also for the states of the LTSs.

A Regular CCS term can be minimized with respect to the number of the variables. In fact, if x_i does not appear in E_i, and x_i is not the variable corresponding to the initial state, we can substitute x_i with E_i in all the other equations, and eliminate the equation $x_i \stackrel{\text{def}}{=} E_i$ from the set of equation.

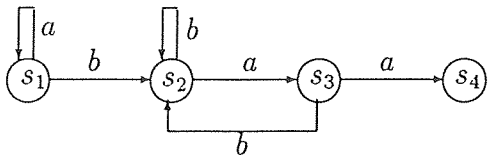

Figure 1: A Labelled Transition System.

Example 2.3 (Reduction of the number of variables) *Consider the system in the Example 2.2. We can substitute* $a.x_4 + b.x_2$ *in place of* x_3 *and* Nil *in place of* x_4*, obtaining:*

$$\begin{cases} x_1 \stackrel{\text{def}}{=} a.x_1 + b.x_2 \\ x_2 \stackrel{\text{def}}{=} a.(a.Nil + b.x_2) + b.x_2 \end{cases}$$

The logic ACTL [4] is the action-based version of the CTL logic [5], and its semantic models are LTSs. ACTL is a temporal logic that is suitable for describing the behaviour of systems that perform actions during their working time. In fact, ACTL embeds the idea of "evolution in time by actions" and is suitable for describing the various possible temporal sequences of actions that characterize a system (i.e. ACTL is a *branching time* temporal logic). The syntax and informal semantics of the ACTL operators is shown in Table 2.

In the table, α is an action of the *finite* set of input and output actions that a given system can perform. An execution (*path*) is a (finite or not) sequence of actions. A *state* represents a time in which a single action has been completed and a new next action may be performed. It is possible that there is more than one action that the system can perform, when its execution reaches a state. Each of these actions represents the beginning of an alternative continuation of the execution. A *state formula* gives a characterization about the possible ways an execution could continue after a state has been reached, while a *path formula* states some properties of a *single* execution.

The ACTL logic is *adequate* with respect to strong bisimulation equivalence on LTSs [4]. Adequacy [16] means that given two LTSs $\mathcal{A}_1$ and $\mathcal{A}_2$, they are strongly bisimilar if and only if $F_1 = F_2$, where $F_i = \{\psi \in \text{ ACTL} : \mathcal{A}_i \text{ satisfies } \psi\}$, $i = 1, 2$.
Notice that adequacy w.r.t. a given equivalence relation does not mean that there exists a finite formula that fully characterizes each equivalence class. For instance, we have that the HML logic [10] is adequate w.r.t. strong equivalence of LTSs, but in general, given an LTS $\mathcal{A}$, it does not exists a finite HML formula that is satisfied by all and only the LTSs strongly equivalent to $\mathcal{A}$.

3 Characteristic ACTL formulae

In this Section we concentrate on the ability of the ACTL logic to completely characterize LTSs, with respect to strong bisimulation equivalence. That is, we associate with each LTS a *characteristic* ACTL formula, in a such a way that two LTSs are strongly equivalent if and only if they have equivalent characteristic formulae[1].

[1] We mean that for each LTS $\mathcal{A}''$, with initial state s_0, the following holds: $s_0 \models [\![\mathcal{A}]\!]$ if and only if $s_0 \models [\![\mathcal{A}]\!]$, where $[\![\mathcal{A}]\!]$ is the characteristic formula of $\mathcal{A}$.

Action formulas

χ	::=	$true$	"any observable action"
		$false$	"no observable action"
		α	"the observable action α"
		$\sim \chi$	"any observable action different from χ"
		$\chi \mid \chi'$	"either χ or χ'"
χ'	::=	χ	

State formulas

ϕ	::=	T	"any behaviour is possible."
		F	"no behaviour is possible."
		$\sim \phi$	" ϕ is impossible"
		$\phi \ \& \ \phi'$	"ϕ and ϕ'"
		$E\gamma$	"there exists a possible execution in which γ"
		$A\gamma$	"for each of the possible executions γ"
		$< action > \phi$	"there exists a next state reachable with *action*, in which ϕ"
		$[action]\phi$	" for all next states reachable with *action*, ϕ is true"
ϕ'	::=	ϕ	

Path formulas

γ	::=	$G\phi$	"at any time ϕ"
		$F\phi$	"there is a time in which ϕ"
		$[\phi\{\chi\}U\{\chi'\}\phi']$	"at any time χ is performed and *also* ϕ, *until* χ' is performed and then ϕ'"
		$X\{\tau\}\phi$	"an unobservable action is immediately performed and, after that, ϕ"
		$X\{\chi\}\phi$	"χ is immediately performed and, after that, ϕ"

Table 1: The ACTL operators

The general method that allows the ACTL characteristic formula of a LTS to be computed consists in the definition of a logic semantics for the term defining the LTS, taking into account the *context degrees* and the ACTL functionals associated with the system variables.

3.1 Context degree

Informally, if E is the expression defining the variable x, the context degree of x is the minimum number of unfoldings of the expression E that suffices in order to semantically distinguish the definition of x from any other variable definition of the equation system.

Example 3.1 (Context degree) *Consider again the system in Example 2.2. We have that the context degree of x_1 is 3, whilst the context degree of the other variables is 1. This informally means that in order to distinguish the behaviour of x_1 from that of x_2, x_3 or x_4 we may have to unfold x_1 three times, whilst the first unfolding suffices for the other variables. Figure 2 informally shows the idea of context degree for the variable x_1. In the four pictures, we have described the original LTS, and the first, the second and the third unfolding of the variable x_1 respectively. The black circles can be thought as denoting states in which any behaviour is*

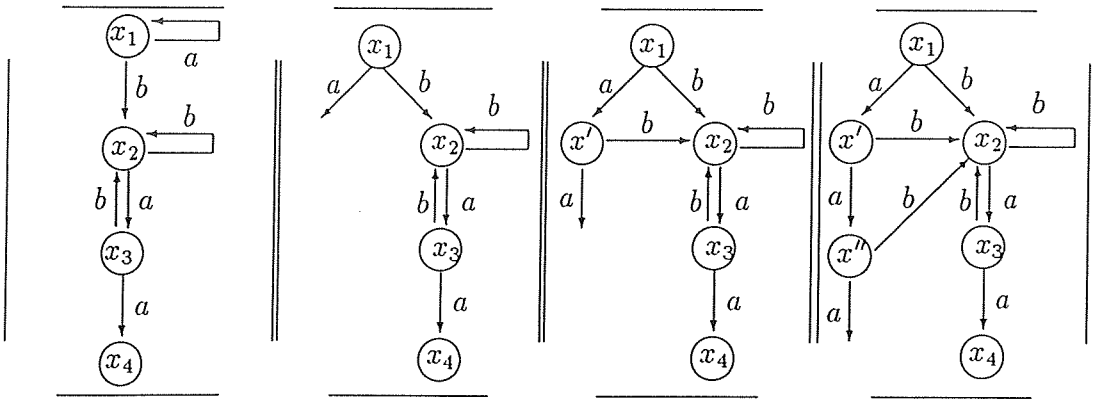

Figure 2: Unfoldings of x_1.

allowed. It is easy to see that (i) if we replace the black circle in the second picture with the state x_3 we get x_1 strongly bisimilar to x_2, (ii) if we replace the black circle in the third picture with the state x_4 we get again x_1 strongly bisimilar to x_2, and finally (iii) there is no way to associate the black circle in the last picture with any behaviour so that x_1 is strongly bisimilar to x_2.

Definition 3.2 (CCS regular context) *We define $\mathcal{C}()$ to be a CCS regular context over $Var = \{x_1, \ldots, x_n\}$ (or simply a context) if it has been obtained from the Regular CCS expression $\mathcal{C}(x_1, \ldots, x_n)$ by replacing all the occurrences of a variable x_i with a "hole", denoted by $()$. When $\mathcal{C}(x_1, \ldots, x_n) = x_1$, the associated context is the expression $()$, constituted by a single hole. This context will be denoted in the following by Ω. The set of the instances generated by $\mathcal{C}()$, that will be denoted by $\Theta(\mathcal{C}())$, is the set of the Regular CCS expressions over Var obtained by replacing each hole in $\mathcal{C}()$ with a Regular CCS expression over Var. We let $\theta, \theta', \ldots$ range over $\Theta(\mathcal{C}())$. Notice that $\Theta(\Omega)$ coincides with the set of Regular CCS expressions over Var.*

Definition 3.3 (Context associated with a variable) *Let $Var = \{x_1, \ldots, x_n\}$ be a set of variables and $\mathcal{E} = \{x_i \stackrel{\text{def}}{=} E_i\}_{x_i \in Var}$ be a finite equation system over Var. We define, for each x_i and each $j \geq 1$, the context $\mathcal{C}_i^j()$ to be:*

$$\mathcal{C}_i^j() = \begin{cases} E_i(x_1, \ldots, x_{i-1}, (), x_{i+1}, \ldots, x_n) & \text{if } j = 1 \\ E_i(x_1, \ldots, x_{i-1}, \mathcal{C}_i^{j-1}(), x_{i+1}, \ldots, x_n) & \text{if } j > 1 \end{cases}$$

Definition 3.4 ($\propto$ relation) *Let E be a Regular CCS expression over Var and $\mathcal{C}()$ be a Regular CCS context over Var. We say that $\mathcal{C}()$ is more general than E, notation $\mathcal{C}() \propto E$, if $\exists \theta \in \Theta(\mathcal{C}())$ such that $\theta \approx E$.*

Example 3.5 (Contexts associated with variables) *Let us consider again the equation system in Example 2.2. We have:*
$\mathcal{C}_1^1() = a.() + b.x_2$
$\mathcal{C}_1^2() = a.(a.() + b.x_2) + b.x_2$
$\mathcal{C}_1^3() = a.(a.(a.() + b.x_2) + b.x_2) + b.x_2$
and so on. Notice that $\mathcal{C}_1^1() \propto x_2$ and $\mathcal{C}_1^1() \propto x_3$; in fact, both $a.x_3 + b.x_2$ and $a.x_4 + b.x_2$

are instances of $C_1^1()$. *On the contrary,* $C_1^1() \not\propto x_4$, *since there are no instances of* $C_1^1()$ *strongly bisimilar to* Nil. *Also, we have that* $C_1^2() \propto x_2$, *because* $a.(a.x_4 + b.x_2) + b.x_2$ *is an instance of* $C_1^2()$ *strongly bisimilar to* x_2. *On the contrary,* $C_1^2() \not\propto x_3$ *because there are no instances of* $a.(a.() + b.x_2) + b.x_2$ *strongly bisimilar to* x_3. *Finally,* $C_1^3() \not\propto x_2$, *because* $a.(a.(a.() + b.x_2) + b.x_2) + b.x_2$ *cannot be strongly bisimilar to* x_2: *in fact,* x_2 *cannot execute a sequence of three* a*'s whilst* $C_1^3()$ *can.*

We will show that whenever we consider equation systems associated with minimized LTSs (that is such that $\forall i \neq j,\ x_i \not\approx x_j$), then for all $i \neq j$ there exists a finite index m such that $C_i^m() \not\propto x_j$. State minimization can be easily achieved by using automated standard state-minimization tools on LTSs [18]. From now on we assume that all the LTSs have been minimized. The same is assumed for equation systems, that is no variables of an equation system are defined by strongly bisimilar expressions.

Theorem 3.6 *Let* $\mathcal{A}$ *be a minimized LTS, with state set* $Var = \{x_1, \ldots, x_n\}$ *and associated equation system* $\mathcal{E} = \{x_i \stackrel{\text{def}}{=} E_i\}_{x_i \in Var}$. *Then* $\forall i \neq j\ \exists m \geq 1$ *such that* $C_i^m() \not\propto x_j$. *Moreover,* $C_i^k() \propto x_j$ *for all* $k \geq 1$ *if and only if* $i = j$.

Proof: The proof is given by contradiction. Suppose that $i \neq j$ and $\forall m \geq 1$ there exists an instance θ of $C_i^m()$ such that $\theta = x_j$. Then x_i and x_j can always simulate each other, thus they are bisimilar. But this contradicts the hypothesis that the system is minimized, that implies that $\forall i \neq j,\ x_i \not\approx x_j$. □

Another advantage of using minimized LTSs is that in this case the verification $C() \propto E$ can be done in a purely syntactical way, that is by verifying the equality between an instance of $C()$ and an expression defining x (possibly with substitutions of defining expressions in place of variable occurrences). Actually, in order to verify whether $C() \propto x$, with $x \stackrel{\text{def}}{=} E$, we can apply the following rules until $C() \propto x$ is obtained or no more rules can be applied successfully (in this last case, we can deduce $C() \not\propto x$):

1. $\Omega \propto E$;
2. $E \propto E$;
3. if $G_1() \propto G$ and $G_2() \propto F$ then $a.G_1() + G_2() \propto a.G + F$ and $a.G_1() + G_2() \propto F + a.G$.

Example 3.7 ($\propto$ relation) *We compute the* $\propto$ *relation for the system in the Example 2.2 by applying the above rules.* $C_1^1() \propto x_2$ *and* $C_1^1() \propto x_3$ *follow from the rule 3 applied to* $C_1^1()$ *and* $a.x_3 + b.x_2$ *and* $a.x_4 + b.x_2$ *respectively, observing that* $\Omega \propto x_3$ *(by rule 1),* $\Omega \propto x_4$ *(by rule 1) and* $x_2 \propto x_2$ *(by rule 2). Also, from the rule 3 we derive that* $C_1^2() \propto x_2$, *because* $C_1^2() = a.C_1^1() + b.x_2$, $C_1^1() \propto x_3$ *and* $x_2 \propto x_2$. *On the contrary, we cannot derive* $C_1^1() \propto x_4$; *actually, no rules can be applied in order to get* $a.() + b.x_2 \propto Nil$. *This also means that* $C_1^2() \not\propto x_3$ *and* $C_1^3() \not\propto x_2$.

Definition 3.8 (Context index and context degree) *Let* $Var = \{x_1, \ldots, x_n\}$ *be a set of variables and* $\mathcal{E} = \{x_i \stackrel{\text{def}}{=} E_i\}_{x_i \in Var}$ *be a finite equation system.*
Let c_{ij}, *with* $j \neq i$, *be the minimum index* m *for which* $C_i^m() \not\propto x_j$ *(it exists by Theorem 3.6). We will call* c_{ij} *the* context index *of* x_i *with respect to* x_j.
We define the context degree *of* x_i *as* $c_i = max\ \{c_{ij} : i \neq j\}$.

3.2 ACTL characteristic formulae

We define now the characteristic functionals associated with the system variables. This is done by associating with each variable a logic functional (an ACTL formula with variables) in such a way that the behaviour expressed by the term defining the variable is fully characterized up to strong bisimulation equivalence. This means that if $\phi_i(x_1,\ldots,x_n)$ is the characteristic functional associated with x_i, then the full ACTL characterization of x_i can be obtained by recursively replacing each occurrence of each variable x_k in $\phi_i(x_1,\ldots,x_n)$ by $\phi_k(x_1,\ldots,x_n)$. Notice that whenever a system present recursive definitions of variables, this task does not terminate. Hence, in order to give a finite characteristic formula, we need another way to logically compose the characteristic functionals associated with the system variables. In [17] this is done by prefixing the characteristic functionals with a fixed point operator. As a main result, we will show that this task can be accomplished in ACTL without using fixed point operators.

Definition 3.9 (ACTL characteristic functional) *Let $Var = \{x_1,\ldots,x_n\}$ be a finite set of variables, and $\mathcal{E} = \{x_i \stackrel{\text{def}}{=} E_i\}_{x_i \in Var}$ be a finite equation system over Var. We define $\mathcal{M}(E)$, where E is a Regular CCS expression, as:*

$$\mathcal{M}(E) = \begin{cases} x & \text{if } E = x,\ x \in Var \\ \delta(E) \wedge \bigwedge_{a \in Act} (A\widetilde{X}_a \lambda_a(E)) & \text{if } E \text{ is not a variable} \end{cases}$$

where:
$A\widetilde{X}_\chi\phi$ stands for $\neg EX_\chi \neg\phi$ (this is called the weak next operator).

$\delta(Nil) = tt$ $\quad$ $\lambda_a(Nil) = \lambda_a(b.E) = ff$
$\delta(a.E) = EX_a\mathcal{M}(E)$ $\quad$ $\lambda_a(a.E) = \mathcal{M}(E)$
$\delta(E' + E'') = \delta(E') \wedge \delta(E'')$ $\quad$ $\lambda_a(E' + E'') = \lambda_a(E') \vee \lambda_a(E'')$

For each $x_i \in Var$, we define the characteristic functional of x_i to be:

$$\phi_i(x_1,\ldots,x_n) = \mathcal{M}(E_i)$$

Moreover, we use the following notation:
$\phi_i^1(x_1,\ldots,x_n) = \phi_i(x_1,\ldots,x_n)$
$\phi_i^{k+1}(x_1,\ldots,x_n) = \phi_i(x_1,\ldots,x_{i-1},\phi_i^k(x_1,\ldots,x_n),x_{i+1},\ldots,x_n)$, *with* $k \geq 1$.

The $\mathcal{M}$ function is given in a syntax driven way. Actually, the functional associated with a given Regular CCS expression is obtained by logically composing the functionals associated with its sub-expressions.

The $\mathcal{M}$ function associates to each Regular CCS expression E an ACTL functional. If E is a variable, then $\mathcal{M}(E)$ is the variable itself. Otherwise, if E is not a variable, then $\mathcal{M}(E)$ is obtained as the conjunction of two parts. The first one, $\delta(E)$, describes what the process E can do; so, if E can execute a and then transforms into E', we have that $EX_a\mathcal{M}(E')$ is a conjunct of $\mathcal{M}(E)$. The other part of the formula, that is the conjunction of the $A\widetilde{X}_a\lambda_a(E)$, establishes what are the correct behaviours of E. Thus, if E cannot execute a, then $A\widetilde{X}_a ff$ will be one of the conjuncts, meaning that there exists no process E' such that E executes the action a and then transforms into E'. On the other hand, if E can execute a and then

transforms into processes $E_1, \ldots, E_k$, $k \geq 1$, then $A\widetilde{X}_a(\mathcal{M}(E_1) \vee \ldots \vee \mathcal{M}(E_k))$ will be one of the conjuncts, meaning that the behaviours allowed after the execution of a are those associated with $E_1, \ldots, E_k$.

Definition 3.10 (ACTL characteristic formulae) *Let $Var = \{x_1, \ldots, x_n\}$ be a finite set of variables, and $\mathcal{E} = \{x_i \stackrel{\text{def}}{=} E_i\}_{x_i \in Var}$ be a finite equation system over Var. We define the* ACTL *semantics of the equation system with respect to the variable x_i to be $[\![x_i, \emptyset]\!]$, where, for $V \subseteq Var$ and $V' = V \cup \{x_i\}$, we define:*

$$[\![x_i, V]\!] = \begin{cases} tt & \text{if } x_i \in V \\ \phi_i^{c_i}(\overline{[\![x, V']\!]}) \wedge AG(\phi_i^{c_i}(\overline{[\![x, V']\!]}) \rightarrow \phi_i^{c_i+1}(\overline{[\![x, V']\!]})) & \text{otherwise} \end{cases}$$

where $\overline{[\![x, V']\!]}$ denotes the argument vector $[\![x_1, V']\!], \ldots, [\![x_n, V']\!]$.

The formula $\phi_i^{c_i}(\overline{[\![x, V']\!]}) \wedge AG(\phi_i^{c_i}(\overline{[\![x, V']\!]}) \rightarrow \phi_i^{c_i+1}(\overline{[\![x, V']\!]}))$ means that the c_j-th unfolding of the functional associated with x_i must be true now, and that whenever c_j-th unfolding of the functional is true then also the $(c_j + 1)$-th unfolding of the functional must be true.

Our method for the computation of the characteristic formula of the LTS is given by five steps. The first step consists in the minimization of the automaton (minimization of equivalent states), that can be done automatically using standard minimization tools [18], and the generation of the associated equation system. The second step deals with the reduction of the number of variables of the equation system associated with the LTS; in fact, if an expression E defining a variable x contains no occurrences of x, then x can be substituted by E in all the other equations. The third step is the computation of the context degree of the variables, that is the number of the expression unfoldings that is needed in order to distinguish an expression from the others. The fourth step consists in the computation of the characteristic functional of each expression defining a variable. The last step is the generation of the characteristic formula of the system, using the functionals and the context degrees associated with the variables.

Example 3.11 (Characteristic formula) *Let us consider again the LTS in Figure 1 and the corresponding equation system in the Example 2.2.*

- Reduction of bisimilar states *In our example, all the states of the LTS are not bisimilar, hence the number of the states cannot be decremented.*
- Reduction of the variables number *The equation system can be minimized with respect to the variables number by observing that the expression defining x_4 does not contain any occurrence of x_4; then, we can also substitute the expression defining x_3 in the equation defining x_2, since x_3 does not appear in its defining expression. As a result, we obtain the two variables equation system:*
  $$\begin{cases} x_1 \stackrel{\text{def}}{=} a.x_1 + b.x_2 \\ x_2 \stackrel{\text{def}}{=} a.(a.Nil + b.x_2) + b.x_2 \end{cases}$$
- Computation of context degrees *For the variable x_1 we have $c_1 = 3$, because:*
  $C_1^1() = a.() + b.x_2$
  $C_1^2() = a.(a.() + b.x_2) + b.x_2$
  $C_1^3() = a.(a.(a.() + b.x_2) + b.x_2) + b.x_2$
  with $C_1^1() \propto x_2$, $C_1^2() \propto x_2$ and $C_1^3() \not\propto x_2$. For the variable x_2, instead, we have $c_2 = 1$, since $C_2^1() = a.(a.Nil + b.()) + b.() \not\propto x_1$.

- Characteristic functionals of variables *For our system we compute the characteristic functionals:*

$$\phi_1(x_1,x_2) = EX_a(x_1) \wedge EX_b(x_2) \wedge A\widetilde{X}_a(x_1) \wedge A\widetilde{X}_b(x_2) \wedge \bigwedge_{\alpha \neq a,b} A\widetilde{X}_\alpha(ff)$$

$$\phi_2(x_1,x_2) = EX_a(\eta) \wedge EX_b(x_2) \wedge A\widetilde{X}_a(\eta) \wedge A\widetilde{X}_b(x_2) \wedge \bigwedge_{\alpha \neq a,b} A\widetilde{X}_\alpha(ff)$$

where

$$\eta = EX_a(\bigwedge_{\alpha \in Act} A\widetilde{X}_\alpha(ff)) \wedge EX_b(x_2) \wedge A\widetilde{X}_a(\bigwedge_{\alpha \in Act} A\widetilde{X}_\alpha(ff)) \wedge A\widetilde{X}_b(x_2) \wedge \bigwedge_{\alpha \neq a,b} A\widetilde{X}_\alpha(ff)$$

- Characteristic formula *Finally, the semantics of the system with respect to the variable x_1 is given by:*

$$[\![x_1,\emptyset]\!] = \phi_1^3([\![x_1,\{x_1\}]\!],[\![x_2,\{x_1\}]\!]) \wedge AG(\phi_1^3([\![x_1,\{x_1\}]\!],[\![x_2,\{x_1\}]\!]) \rightarrow \phi_1^4([\![x_1,\{x_1\}]\!],[\![x_2,\{x_1\}]\!]))$$

where:

$$[\![x_1,\{x_1\}]\!] = tt$$

$$[\![x_2,\{x_1\}]\!] = \phi_2([\![x_1,\{x_1,x_2\}]\!],\ [\![x_2,\{x_1,x_2\}]\!]) \wedge AG(\phi_2([\![x_1,\{x_1,x_2\}]\!],\ [\![x_2,\{x_1,x_2\}]\!]) \rightarrow \phi_2^2([\![x_1,\{x_1,x_2\}]\!],\ [\![x_2,\{x_1,x_2\}]\!]))$$

$$[\![x_1,\{x_1,x_2\}]\!] = [\![x_2,\{x_1,x_2\}]\!] = tt$$

Theorem 3.12 (Full abstractness) *The $[\![\ ]\!]$ semantics we have defined is in full agreement with strong bisimulation. That is, given two LTSs $\mathcal{A}$ and $\mathcal{A}'$, we have that $\mathcal{A} \approx \mathcal{A}'$ if and only if $[\![\mathcal{A}]\!]$ is equivalent to $[\![\mathcal{A}']\!]$, where $[\![\mathcal{A}]\!]$ and $[\![\mathcal{A}']\!]$ are computed with respect to the initial states of $\mathcal{A}$ and $\mathcal{A}'$ respectively.*

The following Section is devoted to the proof of Theorem 3.12.

4 Implicit Fixed Point

In order to prove Theorem 3.12 we will prove that the characteristic ACTL formula we associate with a LTS is equivalent to that defined by using the μ-ACTL logic, that is ACTL augmented by the fixed point operator of μ-calculus. The μ-ACTL logic has the same expressive power as μ-calculus [8]. Once the μ-ACTL characterization of the LTS has been obtained, we will prove that it is equivalent to the ACTL one given in the previous section.

Definition 4.1 (μ-ACTL characteristic formulae) *The syntax of μ-ACTL formulae is given by the grammar below:*

$\phi ::= tt \mid \phi \wedge \phi \mid \neg\phi \mid E\pi \mid A\pi \mid \mu Y.\phi \mid Y$

$\pi ::= X_\chi\phi \mid X_\tau\phi \mid \phi\ {}_\chi U\ \phi \mid \phi\ {}_\chi U_{\chi'}\ \phi$

where Y ranges over a set Var of variables. The satisfaction relation for μ-ACTL formulae is obtained adding to the ACTL semantic the following rule:

$s \models \mu Y.\phi(Y)$ *iff* $s \models \bigvee_{n \geq 0} \phi^n(ff)$, *where* $\phi^0(Y) = Y$ *and* $\phi^{n+1}(Y) = \phi(\phi^n(Y))$.

As usual, $\nu Y.\phi(Y)$ abbreviates $\neg\mu Y.\neg\phi(\neg Y)$.

In order to obtain the μ-ACTL characteristic formula of a LTS we only need to define a new semantic rule for variables: the new semantics is obtained by calculating the maximal solution of the ACTL characteristic functionals with respect to the system variables. Notice that the maximal solution of the characteristic functionals exists, since all the formulae we use are monotonic[2] [1].

Definition 4.2 (μ-ACTL **semantics**) *Let $Var = \{x_1, \ldots, x_n\}$ be a finite set of variables, and $\mathcal{E} = \{x_i \stackrel{\text{def}}{=} E_i\}_{x_i \in Var}$ be a finite equation system over Var. We define the μ-ACTL semantics of the equation system with respect to the variable x_i to be $[\![x_i, \emptyset]\!]_\mu$, where, for $V \subseteq Var$ and $V' = V \cup \{x_i\}$, we define:*

$$[\![x_i, V]\!]_\mu = \begin{cases} x_i & \text{if } x_i \in V \\ \nu\, x_i.\phi_i([\![x_1, V']\!]_\mu, \ldots, [\![x_n, V']\!]_\mu) & \text{otherwise} \end{cases}$$

Theorem 4.3 (Correctness of the μ-ACTL characteristic formula) *The μ-ACTL characterization of LTSs, as given in Definition 4.2, is in agreement with strong bisimulation equivalence.*

Proof sketch: In order to prove the theorem, we consider the μ-calculus characterization of LTSs, as given in [17], that is in agreement with strong bisimulation equivalence. Then, we translate the μ-calculus characteristic functionals defined in [17] into μ-ACTL functionals, by applying the translation from μ-calculus to μ-ACTL given in the Appendix. It comes out that the obtained μ-ACTL formula is proved, in a straightforward way, to be equivalent to that given in Definition 4.2. □

Now we will show that the μ-ACTL characterization of LTSs is equivalent to the ACTL one given in Definition 3.10. In order to do this, we need to show that the maximal solution of the functional associated with a variable is equivalent to the $[\![\ ]\!]$ semantics of the variable.

Theorem 4.4 (Implicit Fixed Point) *Let $\mathcal{A}$ be a LTS with state set Var. Let $x_j \in Var$, with $x_j \stackrel{\text{def}}{=} E_j$. Then for all $x_i \in Var$, for all $V \subseteq Var - \{x_j\}$, we have:*

$$x_i \models [\![x_j, V]\!]_\mu[tt/y, y \in V] \text{ if and only if } x_i \models [\![x_j, V]\!]$$

Proof:
($\rightarrow$): The proof goes by structural induction on $[\![\ ]\!]_\mu$.
(i) $[\![x_j, V]\!]_\mu = x_j$. This means that $x_j \in V$, hence $[\![x_j, V]\!]_\mu[tt/y, y \in V] = tt$ and $[\![x_j, V]\!] = tt$.
(ii) $[\![x_j, V]\!]_\mu = \nu\, x_j.\phi_j(\overline{[\![x, V']\!]_\mu})$. We have that $x_i \models \nu\, x_j.\phi_j(\overline{[\![x, V']\!]_\mu})[tt/y, y \in V]$ if and only if $x_i \models \bigwedge_{k \geq 0} \phi_j^k(\overline{[\![x, V']\!]_\mu})[tt/y, y \in V][tt/x_j]$, i.e. $x_i \models \bigwedge_{k \geq 0} \phi_j^k(\overline{[\![x, V']\!]_\mu})[tt/y, y \in V']$. By induction, we get $x_i \models \phi_j^k(\overline{[\![x, V']\!]})$ for $k \geq 0$. This means that for all $k \geq 1$ there exists an instance of $C_j^k()$ that is bisimilar to x_i, hence, by Theorem 3.6, $x_i = x_j$. The proof follows by contradiction. Suppose

[2]This holds by construction, since only $\wedge$, $\vee$, EX_a and $A\tilde{X}_a$ operators are used, without negation. Monotonicity means that whenever ψ_1 implies ψ_2, then $\psi_3(\psi_1)$ implies $\psi_3(\psi_2)$ for any functional ψ_3.

that there exists a state y reachable from x_i such that $y \models \phi_j^c(\overline{[\![x, V']\!]})$ and $y \not\models \phi_j^{c+1}(\overline{[\![x, V']\!]})$. Notice that $y \neq x_i$ since $x_i \models \phi_j^c(\overline{[\![x, V']\!]})$ and $x_i \models \phi_j^{c+1}(\overline{[\![x, V']\!]})$. Since $y \models \phi_j^c(\overline{[\![x, V']\!]})$ we have that there exists an instance of $\mathcal{C}_j^{c_j}()$ that is bisimilar to y, hence we must have $c_j \geq c_j + 1$, that is a contradiction.[3]

($\leftarrow$): The proof goes by structural induction on $[\![\ ]\!]$.

(i) Let $[\![x_j, V]\!] = tt$. Then we have $x_j \in V$ and $[\![x_j, V]\!]_\mu[tt/y, y \in V] = tt$. *(ii)* Let $[\![x_j, V]\!] = \phi_j^{c_j}(\overline{[\![x, V']\!]}) \wedge AG(\phi_j^{c_j}(\overline{[\![x, V']\!]}) \rightarrow \phi_j^{c_j+1}(\overline{[\![x, V']\!]}))$. Let $x_i \models [\![x_j, V]\!]$. We have to show that $x_i \models \nu\, x_j.\phi_j(\overline{[\![x, V']\!]_\mu})[tt/y, y \in V]$, that is $x_i \models \bigwedge_{k \geq 0}(\phi_j^k(\overline{[\![x, V']\!]_\mu})[tt/y, y \in V])[tt/x_j]$. Since $x_i \models \phi_j^c(\overline{[\![x, V']\!]})$, and $[\![y, V']\!] = tt$ for $y \in V'$, we have that $x_i \models (\phi_j^k(\overline{[\![x, V']\!]_\mu})[tt/y, y \in V])[tt/x_j]$ for all $k \leq c$.

For $k \geq c$, observe that $\phi_j(x_1, \ldots, x_n)$ is always monotonic by definition, and that $AG(\phi_j^c(\overline{[\![x, V']\!]}) \rightarrow \phi_j^{c+1}(\overline{[\![x, V']\!]}))$ means that for all states of the LTS reachable from x_i, $\phi_j^c(\overline{[\![x, V']\!]})$ implies $\phi_j^{c+1}(\overline{[\![x, V']\!]})$. By monotonicity, we have that $\phi_j^k(\overline{[\![x, V']\!]})$ implies $\phi_j^{k+1}(\overline{[\![x, V']\!]})$ for all $k \geq c$. Hence, by applyng the induction, we get $x_i \models \bigwedge_{k \geq c}(\phi_j^k(\overline{[\![x, V']\!]_\mu})[tt/y, y \in V])[tt/x_j]$, and this concludes the proof. □

Now it is easy to prove the Theorem 3.12. In fact, the μ-ACTL solution of the equation system associated with a given LTS is in full agreement with strong bisimulation by Theorem 4.3. On the other hand, by the Theorem 4.4, we have that the ACTL characterization is equivalent to the μ-ACTL one. Hence, the ACTL characteristic formula turns out to be in full agreement with strong bisimulation equivalence.

This result shows that the ACTL logic is able to finitely characterize finite state LTSs, in such a way that equivalence classes from a logical point of view correspond to behavioural equivalence classes. Hence, ACTL shares with μ-ACTL and μ-calculus two important logical properties, i.e. *adequacy* and *expressivity* [16] with respect to finite state LTSs and strong bisimulation. This means that two LTSs are strongly equivalent if and only if they satisfy the same set of ACTL formulae, and that there exists a function from LTSs to ACTL (in our case $\mathcal{M}$) such that a system $\mathcal{A}$ is equivalent to a system $\mathcal{A}'$ if and only if $\mathcal{A}'$ satisfies $\mathcal{M}(\mathcal{A})$. As a consequence of this fact, we have that all the μ-ACTL properties of a system are implied by its ACTL characteristic formula. Nevertheless, ACTL is strictly less expressive than μ-ACTL and μ-calculus. In fact, there are μ-ACTL properties of a LTS that are of course implied by its ACTL characteristic formula, but that are not directly expressible in ACTL. Actually, while safety and liveness properties are naturally expressed in ACTL, cyclic properties, as for instance:

There exists an infinite path in which a *is executed every odd step*

need to be expressed by a fixed point operator, as in the μ-ACTL formula:

$$\nu Y.(EX_a(EX_{tt}Y \vee EX_\tau Y))$$

Notice that Hennessy-Milner Logic [10] does not share the same expressivity property: in fact, Hennessy-Milner Logic, although adequate with respect to finite state LTSs, is expressive

[3] Observe that $\phi_m^k(x_1, \ldots, x_n)$ corresponds to the functional that the function $\mathcal{M}$ associates with the instance $\mathcal{C}_m^k(x_m)$ of the context $\mathcal{C}_m^k()$.

only with respect to cycle-free LTSs, due to the lack of the powerful Until operator; thus recursion [13] must be added to it in order to characterize finite LTSs, getting again the μ-calculus expressive power.

5 Related works and conclusions

We have shown how to give a logic semantics to finite state and finitely branching LTSs. This is done by associating with a LTS an ACTL characteristic formula. The same approach was used in [7] to give a logical characterization to Regular CCS terms by using μ-CTL, that is CTL extended with a fixed point operator. However, we have shown here that the introduction of the fixed point operator is not necessary, the ACTL logic being expressive enough to completely characterize LTSs with respect to strong bisimulation equivalence.
Characteristic formulae for LTSs have already been given in [17], using the μ-calculus logic. With respect to that work, we may point out two advantages of our approach. The first one is *compositionality*, since the method we propose allows the characteristic formula to be computed in a syntax driven (and hence compositional) way. The other one is that we directly characterize recursive processes by means of ACTL formulae, avoiding the use of a fixed point operator that increases the expressive power of the logic and its complexity as well.
Our result presents some analogies also with that in [2], where the CTL logic [5] is used in order to characterize finite states Kripke Structures. Since there exist conservative translations from LTSs to Kripke Structures [4,11] (i.e. strong bisimulation is preserved by the translations), the approach in [2] allows LTSs to be characterized in CTL, by computing the characteristic formula of the Kripke Structure associated with the given LTS. Our approach, instead, is in the action-based framework and allows a characteristic formula to be computed for each finite-state LTS, using the ACTL logic that is less expressive than CTL. Furthermore, we can observe that our method to compute the characteristic formulae is completely syntax driven, differently from the approach proposed in [2], where semantic information on the Kripke Structure are used.

Acknowledgements

We wish to thank Andrea Masini for useful suggestions about the topics of this paper.

References

[1] B. Banieqbal, H. Barringer: *Temporal logic fixed point calculus*, Proceedings Colloquium on temporal Logic and Specification, LNCS 398, 1989, pp. 62-74.

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

[3] R. De Nicola, A. Fantechi, S. Gnesi, G. Ristori: *An action-based framework for verifying logical and behavioural properties of concurrent systems*, Computer Networks and ISDN Systems, Vol. 25, N. 7, North-Holland, pp. 761-778, February 1993.