A Proof Assistant for the Action Based Temporal Logic ACTL

S Gnesi, S. Larosa

Nota Interna B4-50

Dicembre-1995
A proof assistant for the action-based temporal logic ACTL

Stefania Gnesi
gnesi@iei.pi.cnr.it
Salvatore Larosa
larosa@rep1.iei.pi.cnr.it
Istituto di Elaborazione dell'Informazione
Via S. Moria 46, 56100 Pisa, Italy

(Received 31 October 1995)

In this paper we present an HOL-based assistant for the mechanization of proofs in a sound and complete calculus for the branching temporal logic ACTL. This logic has action-based modalities and is interpreted over Labelled Transition Systems, so it is suitable for specifying and studying the behaviour of concurrent systems defined by process algebra terms and modelled on LTSs; moreover, ACTL can naturally describe their safety and liveness properties. The making of the axiom system was the first step in the development of this proof assistant. Our aim was to gradually develop a workbench for a proof-theoretical approach in the use of ACTL, in order to overcome the limitations of model-theoretical approach in verification.

1. Introduction

Different types of temporal and modal logics have been proposed for the abstract specification of concurrent systems. In particular, modal and temporal logics, due to their ability to deal with notions such as necessity, possibility, eventuality, etc., have been recognized as a suitable formalism for specifying properties of such systems (Clarke, Emerson and Sistla, 1986), (Emerson, 1990), (Emerson and Halpern, 1985).

In fact, elegant temporal logics such as CTL' and CTL have been put forward (Emerson and Halpern, 1985). These logics are interpreted on Kripke Structures and permit properties of concurrent systems to be formulated in terms of their states. We qualify them as "state-based" logics.

Process algebras (Hennessy and Milner, 1985) are generally recognized as being a convenient tool for describing concurrent systems at different levels of abstraction. The basic operational semantics of process algebras is usually defined in terms of Labelled Transition Systems, which are then factorized by means of observational equivalences and allow the behaviour of a system to be analysed in relation to the "actions" the specified system may perform.
An apparent contrast arises between the logical models and the behavioural models. In the former, reasoning is based on state properties and on state changes, while in the latter, a central role is played by the actions that cause state changes. To overcome this problem, specific logics for process algebras have been proposed, (see for example Hennessy and Milner, 1985, Stirling, 1995) which are interpreted on Labelled Transition Systems. These logics express properties of systems in terms of the actions they can perform and thus they can be qualified as "action-based."

Two action-based logics, ACTL* (De Nicola and Vaandrager, 1986) and ACTL (De Nicola and Vaandrager, 1990), have recently been proposed, which are more expressive than Hennessy-Milner Logic (Hennessy and Milner, 1985) and can naturally describe safety and liveness properties of concurrent systems.

Indeed, a verification environment, named JACK, has been built around the logic ACTL and its polynomial-time model checker, to verify the satisfiability of formulas on concurrent systems expressed by process algebras terms. Several experiments have been carried out on JACK (Bouali, Gnesi, and Larosa, 1994); among them we recall the study of the properties of a hardware circuit (De Nicola et al., 1995) and of a railway control system (Anselmi et al., 1995).

In this model-theoretical approach to verification (Emerson, 1990), problems arise when real-life industrial case studies are considered, which often have very large size models. To make the verification process possible, various compromises thus have to be made which are related to the configuration of the algebraic processes that describe them.

Solutions to these problems could be found in the compositional model-checking techniques. Unfortunately, these techniques are often related to particular classes of applications, and moreover require the use of specially-tailored logics as for all-CTL (Clarke, Grumberg and Long, 1994).

Another limitation of the model-theoretical approach is related to the difficulty of handling non-finite state systems, which are only in some cases partially solved (De Francesco et al., 1995) (Burkart and Steffen, 1994) (Clarke, Grumberg and Long, 1994) (Hungar, 1994).

On the other hand, the proof-theoretical approach offers interesting means to solve the problems above, and it was our aim to verify this. First of all, we developed a calculus for the logic ACTL in the HOL framework; this was the first step in building a general mechanized workbench for dealing with ACTL. ACTL is thus used not only as a property description language, but as a language to describe a reactive system, too. The logical description of a reactive system becomes a set of hypotheses for the ACTL calculus engine, while the safety and liveness properties are simply theses to be proved by means of the ACTL calculus HOL engine.

We embedded in our workbench a mechanization of the CCS process algebra. We were then able to give an interpretation of ACTL formulas directly over CCS processes. We are currently modifying and developing our ACTL workbench in order to define theorems and proof strategies for handling non-finite state CCS specifications, partially overcoming thus the limitations related to the handling of non-finite specifications by model-checking.

Other works have been done in formalizing modal or temporal logics over HOL: in (von Wright, 1992) the linear time "Temporal Logic of Actions" (Lamport, 1994) has been implemented; in (Nesi, 1992) the Hennessy-Milner logic has been mechanized to prove properties of CCS terms by means of it. What we believe is that our proof assistant for ACTL is interesting because the expressive power of ACTL is greater than HML's (and it
not comparable with TLA). Also, ACTL offers the possibility of studying various kind of process equivalences in a natural way, due to the adequacy of ACTL with respect to the bisimulation equivalence (De Nicola and Vaandrager, 1990). Moreover, ACTL is a rich and complex calculus, so that this mechanizing work could be a good basis for further development and improvement of powerful logic calculi.

This paper is organized as follows: section 2 gives some relevant preliminary definitions; Section 3 contains the definition and the sound and complete axiomatization of ACTL; Section 4 introduces the HOL package; Section 5 and Section 6 go through the HOL implementation and usage of the ACTL workbench using some short examples.

2. Basic Definitions

The structures we will use as models for the branching time logics ACTL (De Nicola, Fantechi, Gnesi and Ristori, 1993) (De Nicola and Vaandrager, 1990) are defined below:

**Definition 2.1.** A Labelled Transition System (LTS) is a 4-tuple

\[ L = (Q, O_Q, \rightarrow, A \cup \{\tau\}), \]

where

- \( Q \) is a set of states, and \( u, v, w, s, t, \ldots \) range over \( Q \).
- \( O_Q \) is the set of initial states.
- \( A \) is a finite and not empty set of visible actions ranged by \( a, b, c, d, \ldots \); \( \tau \) is the silent action, which is not in \( A \). We let \( A_\tau = A \cup \{\tau\} = \{l_1, l_2, \ldots\} \).
- \( \rightarrow \subseteq Q \times A_\tau \times Q \) is the state transition relation. If \( (s, l, t) \in \rightarrow \), we will write \( s \xrightarrow{l} t \).

**Remark:** Hereafter, when we write that a state \( t \) is a successor or son of \( s \), we will mean that \( \exists l \in A_\tau \) such that \( s \xrightarrow{l} t \).

**Definition 2.2.** Let

\[ \xrightarrow{n} = \underbrace{\times \ldots \times}_{n \text{ times}} \rightarrow \]

(\( \xrightarrow{n} \) is the Cartesian product of the \( \rightarrow \) relation defined above). Then:

In an LTS a list of ordered terms

\[ \sigma = (s_0, l_0, s_1)(s_1, l_1, s_2)(s_2, l_2, s_3) \ldots \xrightarrow{\infty} \]

is called path beginning in \( s_0 \). We let \( \sigma, \pi, \delta \) range over paths.

A sequence

\[ \sigma = (s_0, l_0, s_1)(s_1, l_1, s_2) \ldots (s_{k-1}, l_{k-1}, s_k) \xrightarrow{h} \]

is called finite path from \( s_0 \) to \( s_k \).

An infinite path or a finite one that cannot be extended is called fullpath.

\( \sigma(0) \) is the first state of the path \( \sigma \), also denoted by \( \text{first}(\sigma) \).

\( \sigma(n) \) is the \( n \)-th state of the path \( \sigma \).
If $\sigma$ is a finite path, $last(\sigma)$ is its last state. The $n$-th suffix of $\sigma$, denoted by $\sigma^n$, is the sequence that has all the states of $\sigma$ starting from $\sigma(n)$ (included). We have that $\sigma^0 = \sigma$. If $\sigma$ is a finite path and $\delta$ is a path such that $last(\sigma) = first(\delta)$, the path $\pi = \sigma\delta$ is called concatenation of $\sigma$ and $\delta$ (and $\delta$ is a suffix of $\pi$).

3. The temporal logic ACTL

ACTL (De Nicola and Vaandrager, 1990) is a pure branching temporal logic whose operators are action-based and embeds an action calculus to improve the expressivity of its operators.

We begin by presenting the action calculus:

**Definition 3.1.** An action formula $f$ is generated by the grammar

$$f ::= a \mid \neg f \mid f \lor f$$

where $a$ is a generic element of $A$ (and then $a \neq \tau$). We let $A_{\text{for}} = \{\text{action formulae over } A\}$.

Now, we define what is a satisfaction of an action formula $f$ by a single action $a$, and we denote this satisfaction with $a \models f$.

**Definition 3.2.** Let $a \in A$ and $f, g \in A_{\text{for}}$. Then:

- $a \models a$.
- $a \models \neg b$ for each $b \in A$ such that $a \neq b$.
- $a \models \neg f \iff a \not\models f$.
- $a \models f \lor g \iff a \models f$ or $a \models g$.

Usually, $\#$ denotes the ever-satisfied formula of a calculus; also in our action-calculus $A_{\text{for}}$ we define the formula $\#$, choosing an $a \in A$ and letting $\# = a \lor \neg a$.

so, in the action-calculus $\#$ means "all the actions of $A$ are permitted".

Here is the complete ACTL syntax:

**Definition 3.3.** The grammar of the well-formed formulae $\Phi$ of ACTL is:

$$\Phi ::= T \mid \Phi \lor \Phi \mid \neg \Phi \mid Op$$

$$Op ::= \forall \text{ PathOp} \mid \exists \text{ PathOp}$$

$$\text{PathOp} ::= X_f \Phi \mid X_f \Phi \mid \Phi f U \Phi \mid X \Phi \mid F \Phi \mid G \Phi \mid F_f U_g \Phi$$

where $f, g$ are action formula.

We write $T$ to denote the ACTL ever-satisfied formula, which is the terminator of the above grammar, too. $\forall X$ and $\exists X$ are the "next states" operators, $\forall (\_ \_)$ and $\exists (\_ \_)$ are indexed "untils", $\forall$ and $\exists$ are the "eventually" operators, while $\forall G$ and $\exists G$ are the "always" ones. Note that not all the operators are primitive; we could define the whole set of the above operators using just the "untils" and the action-based "nexts". However, the other operators are useful to simplify the task of writing formulas.
3.1. Models for ACTL

Let $L$ be a total LTS (i.e. each state has a successor) and $R_L$ a not-empty and suffix-closed set of paths on $L$, i.e.: $\sigma \in R_L \Rightarrow \forall i \geq 0, \sigma^i \in R_L$.

The couple $(L, R_L)$ is called extended LTS and is a model for ACTL formulae. The reason we only wanted total models was just to simplify the ACTL axiom set. It is not a limitation, because there is a simple way to transform a finite path into a total one by adding a $\tau$ loop to its final state.

Since ACTL has only branching operators, a relation of satisfiability of ACTL formulae over states is suitable to formalize the semantics of the ACTL operators. This is shown below:

**Definition 3.4.** Let $M$ be an extended LTS, $s$ a state of $M$, and

$$S(s) = \{ t \in Q \mid s \xrightarrow{t} l \text{ for some } l \in A_r \}$$

the set of successors of $s$. $\models$ is inductively defined as:

- $M, s \models \neg \phi \iff M, s \not\models \phi$.
- $M, s \models \phi \land \psi \iff M, s \models \phi$ and $M, s \models \psi$.
- $M, s \models \forall X \phi \iff \forall t \in S(s), M, t \models \phi$.
- $M, s \models \forall X_r \phi \iff \forall t \in S(s), M, t \models \phi$ and $s \xrightarrow{\tau} t$.
- $M, s \models \forall (\phi_f U \psi) \iff \forall \sigma$ such that $\sigma(0) = s$.

1) $\exists k \geq 0$ such that $l | M, \sigma(k + 1) \models \psi$,
2) $M, \sigma(k) \models \phi$, $\exists j < k, (M, \sigma(j) \models \phi$ and $\sigma(j) \xrightarrow{\tau} \sigma(j + 1) \Rightarrow (l = \tau$ or $l \models f)$).

- $M, s \models \forall (\phi_f U \psi) \iff \forall \sigma$ such that $\sigma(0) = s$.

1) $\exists k \geq 0$ such that $l | M, \sigma(k) \models \psi$,
2) $\forall 0 \leq j < k, (M, \sigma(j) \models \phi$ and $\sigma(j) \xrightarrow{\tau} \sigma(j + 1) \Rightarrow (l = \tau$ or $l \models f)$.

- $M, s \models \forall F \phi \iff \forall \sigma$ such that $\sigma(0) = s \exists k \geq 0$ such that $M, \sigma(k) \models \phi$.

- $M, s \models \forall G \phi \iff \forall \sigma$ such that $\sigma(0) = s \forall k \geq 0 M, \sigma(k) \models \phi$.

- $M, s \models \exists X \phi \iff \exists t \in S(s)$ such that $M, t \models \phi$.

- $M, s \models \exists X_r \phi \iff \exists t \in S(s)$ such that $M, t \models \phi$ and $s \xrightarrow{\tau} t$.

- $M, s \models \exists (\phi_f U \psi) \iff \exists \sigma$ such that $\sigma(0) = s$ and

1) $\exists k \geq 0$ such that $l | M, \sigma(k + 1) \models \psi$,
2) $M, \sigma(k) \models \phi$, $\exists j < k, (M, \sigma(j) \models \phi$ and $\sigma(j) \xrightarrow{\tau} \sigma(j + 1) \Rightarrow (l = \tau$ or $l \models f)$.

- $M, s \models \exists \phi_f U \psi \iff \exists \sigma$ such that $\sigma(0) = s$ and

1) $\exists k \geq 0$ such that $l | M, \sigma(k + 1) \models \psi$,
2) $M, \sigma(k) \models \phi$, $\exists j < k, (M, \sigma(j) \models \phi$ and $\sigma(j) \xrightarrow{\tau} \sigma(j + 1) \Rightarrow (l = \tau$ or $l \models f)$.

- $M, s \models \exists \phi_f U \psi \iff \exists \sigma$ such that $\sigma(0) = s$ and

1) $\exists k \geq 0$ such that $l | M, \sigma(k + 1) \models \psi$,
2) $M, \sigma(k) \models \phi$, $\exists j < k, (M, \sigma(j) \models \phi$ and $\sigma(j) \xrightarrow{\tau} \sigma(j + 1) \Rightarrow (l = \tau$ or $l \models f)$.
\[ M, s \models \exists f \Upsi \psi \iff \exists \sigma \text{ such that } \sigma(0) = s, \]
\[ \exists k \geq 0 \text{ such that } 1) M, \sigma(k) \models \psi, \]
\[ 2) \forall 0 \leq j < k, (M, \sigma(j) \models \phi \quad \text{and } \sigma(j) \to \sigma(j + 1) \Rightarrow (l = \tau \text{ or } l \models f)). \]
\[ M, s \models \exists F \phi \iff \exists \sigma, k \geq 0 \text{ such that } \sigma(0) = s \text{ and } M, \sigma(k) \models \phi. \]
\[ M, s \models \exists G \phi \iff \exists \sigma \text{ such that } \sigma(0) = s \text{ and } \forall k \geq 0 M, \sigma(k) \models \phi. \]

**Definition 3.5.** Let \( M \) be an extended LTS and \( s \) a state of \( M \); if \( M, s \models \phi \), then we say \( M \) is a model for \( \phi \).

**Notation:** \( \forall X_{f\nu} \phi \) and \( \exists X_{f\nu} \phi \) are not ACTL well-formed formulae; but, to get short formulae, we define the following shortcuts which are used throughout this paper:
\[ \exists X_{f\nu} \phi \overset{df}{=} \exists X_f \phi \lor \exists X_r \phi \]
\[ \forall X_{f\nu} \phi \overset{df}{=} \neg \exists X_f \neg \phi \land \neg \exists X_r \neg \phi \land \neg \exists X_{fT} \]

**3.2. An Axiom System for ACTL**

Here, an axiom system is presented for the logic ACTL. We then sketch the proof that provides a sound and complete axiomatization.

**Axioms:**

- **A0** All tautology instances
- **A1\( \exists X_f \phi \to \forall X_f \phi \)**
- **A2\( \exists X_f (\phi \lor \psi) \to (\exists X_f \phi \lor \exists X_f \psi) \)**
- **A3\( \exists X_r (\phi \lor \psi) \to (\exists X_r \phi \lor \exists X_r \psi) \)**
- **A4\( \neg \forall X_f \phi \to \exists X_{fT} \lor \exists X_r \phi \)**
- **A5\( \neg \forall X_r \phi \to \exists X_{rT} \lor \exists X_f \phi \)**
- **A6\( \forall X_f \phi \to \forall X_f \phi \land \forall X_f T \)**
- **A7\( \forall X_r \phi \to \forall X_r \phi \land \forall X_r T \)**
- **A8\( \exists X_f \phi \to \exists X_f \phi \lor \exists X_r \phi \)**
- **A9\( \forall X_r \phi \to \phi \land \forall (\exists X_f \phi \lor \exists X_{fT}) \)**

**Rules:**

\[ RNA : \frac{\Gamma, \phi \models \phi}{\Gamma \models \forall X \phi}; \quad MP : \frac{\Gamma, \phi \models \phi \land \psi}{\Gamma \models \psi}; \quad RNG : \frac{\Gamma \models \phi}{\Gamma \models \forall \forall \phi}. \]
We now give some explanations about this axiomatization.

A0 is any set of axioms that characterizes the propositional tautologies. A possible choice of A0 could be:

\[
\begin{align*}
A0/1 & \quad (\phi \lor \phi) \rightarrow \phi \\
A0/2 & \quad \phi \rightarrow (\phi \lor \psi) \\
A0/3 & \quad (\phi \lor \psi) \rightarrow (\psi \lor \phi) \\
A0/4 & \quad (\phi \rightarrow \psi) \rightarrow ((\phi \lor \gamma) \rightarrow (\psi \lor \gamma))
\end{align*}
\]

which, together with the MP rule, is a consistent and complete axiomatization of the calculus of the sentences.

A1 defines the $\exists X_f$ operator by single-actions $\exists X_a$ operators.

A2 and A3 are both distribution laws: the former is related to visible actions, the latter to the silent action.

A4 says that if not all the states that are sons of a state $s$ are reachable from $s$ by satisfying $f$, then this means that at least one of them is reachable either by the silent action or by actions that do not satisfy $f$; the reverse holds, too.

A5 states the separation between the visible actions and $\tau$: it says that if not all the state sons of $s$ are reachable from $s$ by $\tau$, then there is one such son that is reachable by a visible action, and vice versa.

A6 and A7 define the $\forall X_f$ and $\forall X_s$ operators. A6 means that if a state $s$ satisfies $\forall X_f \phi$, then all the sons of $s$ satisfy $\phi$ ($\forall X \phi$) and, moreover, they are all reachable by actions that satisfy $f$ ($\forall X_f \psi$). The explanation of A7 is similar.

A8 is simply the definition of $\exists X$, while A9 and A10 are distribution laws. A11 defines $\forall X$ as the dual of $\exists X$, while A12 means that each model for ACTL formulae shall be total.

A13 and A14 define the "eventually" operators by means of the "until" ones. In A15 and A16 the "always" operators are defined as the duals of the $F$ ones.

A17, A18, and A19 simply show the inductive way by which $\exists (\phi_f U \psi)$, $\forall (\phi_f U \psi)$ and $\exists (\phi_f U_\gamma \psi)$ propagate themselves along the states of an LTS. A20 does the same thing for $\forall (\phi_f U_\gamma \psi)$, but in a slightly different way.

Note that A17–A20 do not forbid the infinite unfolding of the "untils": A21–A24 are used to avoid it, while A25 and A26 do the same for the "eventually" operators. The trick is to use a "placeholder" ($\gamma$) to characterize the case of infinite unfolding of an operator $op$ that should have a finite unfolding; then, we say that if $\gamma$ holds in a state (i.e. such a state is the initial one for an infinite unfolding of $op$), then in such a state $op$ cannot hold.

The following theorem ensures the consistence and the completeness of our calculus:

**Theorem 3.1.** (Soundness and completeness) For each well-formed formula $\phi$ of ACTL

\[
\vdash \phi \iff \models \phi
\]

**Proof.** The soundness proof is made by induction on the structure of the derivation of $\phi$, while the completeness proof is made using a technique (Emerson, 1990) (Emerson and Halpern, 1985) based on a decision algorithm for the satisfiability of an ACTL formula.
This technique is a variant of the tableau approach, which was applied to the branching time logics in (Ben-Ari, Pnueli and Manna, 1983) (Emerson and Sistla, 1984).

The full completeness proof is nearly 25 pages long, so we will not show it here: see (Gnesi and Larosa, 1995) for a full explanation of it. □

4. A brief introduction to HOL

Having developed a calculus for ACTL, the natural next step was to build an environment in which theorems can be proved with some mechanical support. The idea was to choose a suitable proof system and extend it with a support for ACTL calculus. We decide to use the HOL '90 package (The HOL system, 1993). HOL is a shell, built on the top of an SML interpreter, that enables one to:

1. define logical theory, theories trees and inheritance in an easy way;
2. define target theorems that the user can interactively try to prove. HOL comes with a variety of built-in ways to handle and transform a logical formula (i.e. a theorem that has to be proved). In addition, the user can define tactics (i.e. standard methods to be used on theorems that have a particular structure) and collections of tactics (tacticals), which are kinds of control structures for tactics.

When users have to prove a theorem, they begin by trying some kind of rewriting or reduction of it. They thus have to choose a tactic (or tactical) and apply it to the theorem. After, when they have finished the proof, the theorem can be inserted into the HOL system and becomes available as a proof tool to work on new theorems. It should be possible to take the claim of a well-known theorem and simply add it to the HOL system as an axiom. However, this is not a safe way to design the bases of a proof assistant: if we let HOL prove a theorem we are sure that it is consistent with the axioms and definitions, while the simple insertion of a theorem into the axiom list does not generally ensure that its consistency will be kept.

Following this methodology for mechanizing the ACTL logic in HOL, we start by defining LTSSs in the HOL syntax, with their related theorems and rules. This creates a related HOL theory. The ACTL syntax and the satisfiability relation of an ACTL formula are then inserted into a new HOL theory. We can then prove the consistency of all the ACTL axioms and rules in the given framework. When we have proved that any ACTL axiom or rule is consistent, it is added into an ACTL theory.

Once the set of theories above have been built, we are ready to use HOL to prove ACTL theorems by means of the axioms and rules of ACTL.

4.1. Mechanizing ACTL in the HOL system

In this section we present the basic HOL definitions on the top of whom a full ACTL workbench has been built. The mechanization of LTSSs is given through the mechanization of the calculus CCS given in (Nesi, 1992) for HOL '88 and reimplemented in HOL'90. The bijective correspondence between CCS agents and LTSSs make possible to define the satisfiability relation of ACTL formulae over CCS agents. In the following we will indifferently use LTSSs and CCS agents. In accordance with this, we will use terms such as "states" and "paths" of LTSSs to match the formal definitions of ACTL but in the HOL implementation we will use "processes" and "process computations" of CCS agents. We
prefer this approach to simplify the mechanization work. Moreover, this choice will allow the verification of ACTL properties on CCS agents to be directly performed in HOL.

The definition of an action formula $A\text{for}$ in the HOL syntax is

```
val Afor_Axiom = define_type {
  name = "Afor_Axiom",
  type_spec = 'Afor = lab of Label | (*simple label*)
  nega of Afor | (*negation of an action formula*)
  conja of Afor ⇒ Afor | (*conjunction of two action formulas*)
  ',
  fixities = [ Prefix, Prefix, Prefix ]
};
```

where the type "Label" contains the definition of what a label (i.e. visible action or tau) is.

The satisfiability of an action formula by a single action, SATa, is then expressed in quite a natural way:

```
;SATa : Label -> Afor -> bool
; is true iff the given label satisfies the given action formula...
val SATa = new_recursive_definition {
  name = "SATa",
  fixity = Prefix,
  rec_axiom = Afor_Axiom,
  def = "'Afor = lab of Label " (Sat of a single label.*
  " \& \& (\&\text{this is the conjunction of the def. items...}*)
  (SATa a \&\text{ nega f} = " (SATa a f) \&\text{ sat of a 'not' formula}*)
  \&
  (SATa (conja f1 f2) = (SATa a f1) \&\text{ sat of an 'and' formula}*)
  '---
};
```

The disjunction of two actions is defined as usual in terms of "nega" and "conja":

```
val disja = new_definition ("disja",
  "'\text{disja f1 f2 = nega (conja (nega f1) (nega f2))'---};
val SATa_disja = store_thm ("SATa_disja",
  "'\text{!a f1 f2. SATa a (disja f1 f2) = SATa a f1 \&/ SATa a f2'}---,
  ...);
```

i.e. we give the satisfiability of "disja" as a theorem stating that for all (the symbol "!") f1 and f2 (the dot "." separates a list of variables from a claim) "SATa a (disja f1 f2)" holds iff "SATa a f1" or "SATa a f1" (or both) holds.

We now define the special action formula "true", which matches all the visible ones. We simply create a special action, "a0", to set "true = a0 or not a0", as in the classical way. We called "tta" the formula "true" in the HOL syntax:
The syntax of the action calculus has now been completely built (we don't show the related tactics and theorems). We are now able to define the ACTL syntax:

```haskell
val ACTL_Axiom = define_type {
    name = "ACTL_Axiom",
    type_spec = 'ACTL = tt!("always true it the T of the ACTL syntax") +
                    neg of ACTL! ("logical not")
                 conj of ACTL => ACTL! ("logical and")
                AXtau of ACTL!
                AXf of Afor => ACTL!
                AX of ACTL!
                AU of ACTL => Afor => ACTL!
                A_Uest of ACTL => Afor => Afor => ACTL!
                AF of ACTL!
                AG of ACTL!
                EXtau of ACTL!
                EXf of Afor => ACTL!
                EX of ACTL!
                EU of ACTL => Afor => ACTL!
                E_Uest of ACTL => Afor => Afor => ACTL!
                EF of ACTL!
                EG of ACTL!
    fixities = [Prefix, Prefix, Prefix, Prefix, Prefix, Prefix, Prefix, Prefix]
};
```

A stands for ∀, E for ∃, Xτ for Xτ, Xf for Xf, where act is an action formula; Uest is the extended version of the until (i.e. -fU_u-).

Before introducing the formal HOL definition of the semantic of the ACTL operators, we present some HOL predicates that are used in such a formalization. We give such definitions using both a CCS–based interpretation (process computations) and an LTS–based one (states of a graph instead of processes).

1. "PATH s L" is a boolean function stating that L is a path beginning in state s (i.e. L is one of the computations of process s).
2. "Sufts s L k" returns the k-th state of the path L that begins in state s (i.e. returns the k-th process in the sequence L of processes from the process s).
3. "INTR k L" (boolean) states that k is the index of some state (process) of path L.
4. "TRANSIT s t" (boolean) states that there is a path from state s to state t (i.e. there is one of the sequences of actions that process s could perform and, after such a sequence, s behaves as process t).
5. "Suflact L j i" returns the label of the transition (of the path L) from the state of index j to the state t.

The satisfaction of an ACTL formula by a state s of a LTS, M, is then formalized as follows:
val SAT = new_recursive_definition {
  name = "SAT",
  rec_axiom = ACTL_Axiom,
  fixity = Prefix,
  def = "\n  (*sentence calculus part*)
  (SAT (M:CCS) (s:CCS) (tt) = T )
  (* i.e. tt is always satisfied*)
  /
  (SAT M (s:CCS) (neg fi) = "(SAT M s fi))
  (*sat. of logical not*)
  /
  (SAT M s (conj fi1 fi2) = (SAT M s fi1) /
   (SAT M s fi2))
  (*sat. of logical and*)
  /
  (*branching time temporal operators part*)

  ...
}

Notice that both M and s are objects of type "CCS process": an LTS (M) is just a
CCS process and saying that "s is a state of M" really means, in the HOL mechanization
we performed, that "there is some execution of M that will lead it to behave like s".

We don't show all the definition items related to the temporal operators, just someone
to represent the style in which HOL allows the satisfaction relation to be defined:

(SAT M s (AXtau fi) = "(TERMINAL s) /
 (\exists u.TRANS s u s' =>
  (u = tau) /
  (SAT (M s' fi)) ),

means that ∀Xϕ (i.e. AXtau fi in the HOL syntax) is satisfied by s just if s is not a
terminal state of M and, for all the successors s' of s, the label associated to the transition
(s,u,s') is tau and s' satisfies fi;

(SAT M s (EU fi fi') = (!L.PATH s L =>
 (\forall k. (INF k L) /
 SAT M (Suffs s L k) fi') /
 (!\forall j. (0 <= j) /
   (j < k) =>
   (!\forall (M (Suffs s L j) fi) /
   ([Sufact L j = tau] /
    (?L.SATa l f /
     (Sufact L j = label l)))))),

means that ϕUϕ' (i.e. AU fi fi' in the HOL syntax) is satisfied by s just if for all the
paths L starting from s (L.PATH s L) exists a state (the symbol "?" means "exists"),
indexed by k, in the path L (INF k L) such that:

1 It satisfies ϕ' and it is in the position k of the path L starting from s, (SAT M
(Suffs s L k) fi') and
2 for all the indexes i less than k (i.e. (0 <= j) /
 (j < k)), their related states
in L satisfy ϕ (SAT M (Suffs s L j) fi) and they are the first states of transi-
tions labelled either by r (Sufact L j = tau) or by a label l satisfying the action
formula f (L.SATa l f /
 (Sufact L j = label l)).

We did not show all the rules, lemmas and theorems related to the definitions above,
in order to not enter in deeper technical level of discussion about the ACTL implement-
mentation. However, the definitions we presented are sufficient for introducing the nextsections.
5. Proving the ACTL axioms consistency by a proof engine

In this section we will to see how, after we have mechanized the notion of satisfiability of an ACTL formula, we can verify the consistency of the axiom set of ACTL.

To do this we use an HOL mechanism that guides a user in extending a theory in a "safe" way: a user is not allowed to extend a theory by adding some fact to it that would lead to an inconsistency.

Using this feature, we thus check the consistency of each of the axioms and rules of ACTL, and then we add each of them to the ACTL, HOL, theory.

We will now see an example of how the above strategy, to create the ACTL theory, is implemented.

To this purpose, the HOL code which is used to prove the consistency of the $\forall G$ rule is presented. We have to prove that:

$$|- \phi \implies |- AG \phi$$

i.e. " $\phi$ is valid implies (implies) that $\forall G \phi$ is valid, too".

We give the next command to the system to declare our target theorem, expanding the HOL definition of $\vdash$:

```haskell
set_goal ([], '!', fi.VALIDT fi => VALIDT (AG fi), '!', fi);
```

This means "For each $\phi$, if $\phi$ is valid then $\forall G \phi$ is, too". What the system does is to try to divide a goal into subgoals, and so on, until it reaches a list of "elementary" subgoals that can immediately proved. Tactics and tacticals are used to perform such a division, while the HOL "e(...)" function is used to go through the process of successive divisions above by applying of a tactic or tactical. For example,

```haskell
e (REWRITE_TAC[VALIDT]);
```

is used to rewrite the goal by expanding the definition of VALIDiT. It gives the result:

1 subgoal:

```haskell
(\!\!fi.
 (\!\!s. ~TERMINAL s) => (\!\!s. TRANSIT M s => SAT M s fi)) =>
 (\!\!s. ~TERMINAL s) =>
 (\!\!s. TRANSIT M s => SAT M s (AG fi))
```

because the definition of VALIDiT is:

```haskell
VALIDiT fi = (\!\!s. ~TERMINAL s) => (\!\!s. TRANSIT M s => SAT M s fi)
```

The hypotheses of the main implication state that each state should be a non-terminal (i.e. each LTS is total), while the thesis is that for each model $M$ and each of its states $s$, $s$ has to satisfy $\phi$.

We now want to apply the tactic STRIP_TAC, which keeps trying to perform some standard reduction until it is successful. To achieve this, we use the tactical "REPEAT STRIP_TAC" which keeps trying to strip the goal until it is successful. Before continuing, we say that when we have $\phi \rightarrow \gamma$ as a subgoal and we try to perform a subgoal division of it, we get $\gamma$ as a new subgoal and $\phi$ is added to the list of terms (below the subgoal) that
constitute the hypotheses a user has to deal with to prove $\gamma$. Moreover, in such a process the quantifier "$\forall$" disappears, and the universally quantified variables are instantiated to generic ones.

So, the iteration of STRIP_TAC on the subgoal above causes the splitting of all the above $\Rightarrow$ parts into theses (subgoals) and hypotheses (that accumulate under the double line; at the end of this process, the current subgoal is just the rightmost thesis:

```
e (REPEAT STRIP_TAC);
OK..
1 subgoal:
(---'SAT M s (AG fi)'

＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝
(---'!(s. "(TERMINAL s)" Расsat M s => SAT M s fi)'
(---'!(s. "(TERMINAL s)")'
(---'TRANSIT M s''--

Notice that two free (i.e. not quantified) occurrences of a variable name over and/or under the double line are bound. Now we work on the new subgoal to explicitly write what "SAT M s (AG fi)" means; we use the tactic "SAT_AG_TAC", that tries to expand a subgoal by applying the HOL definition of satisfiability:

```
e (SAT_AG_TAC);
OK..
1 subgoal:
(---'!l. PATH s L => (!k. INTR k L => SAT M (Sufst s s L k) fi)'

＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝
(---'!(s. "(TERMINAL s)" => (!m. TRANSIT M s => SAT M s fi)'
(---'!(s. "(TERMINAL s)")'
(---'TRANSIT M s''--

In the first line we have thus explicitly defined the satisfaction of the formula $\forall G \phi$ on a model M. We perform again a set of STRIP_TACs to simplify the goal by eliminating the deduction symbol $\Rightarrow$ (and the shifting of its hypothesis under the line) and instantiating the quantified variables.

```
e (REPEAT STRIP_TAC);
OK..
1 subgoal:
(---'SAT M (Sufst s s L k) fi''--

＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝
(---'!(s. "(TERMINAL s)" => (!m. TRANSIT M s => SAT M s fi)'

A

B

C

The hypotheses of the deduction have shifted under the hypothesis list. What we have so far obtained, was to get a subgoal that is just a term without any boolean operators. We labelled the three terms that compose the first items of the hypothesis list by A, B and C, to make our proof strategy clear. In fact it is possible to that the subgoal is simply an instance of the term C (substituting s by Sufst s s L k).
To get C alone and instantiated, we proceed using a tactic that adds to the subgoals a deduction from it; we use a theorem, PATH_TRANSIT, that is part of the HOL CCS/LTS theory and states that "for each state s and LTS L whose initial state is s, we have that a path exists from s to each of the states of L". The formalization of such a theorem in HOL is:

\[ \text{| |- \forall s L. \text{PATH} s L \implies (\forall k. \text{INTR} k L \implies \text{TRANSIT} s (Sufsts s L k))} \]

We obtain:

```

  e (IMP_RES_TAC PATH_TRANSIT);
  OK..
  1 subgoal:
  (--"SAT M (Sufsts s L k) fi'--
  ===============
  (--"(s. "(TERMINAL s)) \implies (\forall M s. TRANSIT M s \implies SAT M s fi')--
  (--"s. "(TERMINAL s)"--
  (--'TRANSIT M s'--
  (--'PATH s L'--
  (--'INTR k L'--
  (--'TRANSIT s (Sufsts s L k)"--

In the HOL definition of the LTS rules there is a rule called TRANSIT_TRANS which says that if there is a path from x to s, and one from s to t, then it is possible to go from x to t. Because of such a rule states the transitivity of TRANSIT, we are now able to deduce the new item TRANSIT M (Sufst s L k), i.e. B, by applying IMP_RES_TAC to TRANSIT_TRANS:

  e (IMP_RES_TAC TRANSIT_TRANS);
  OK..
  1 subgoal:
  (--"SAT M (Sufsts s L k) fi'--
  ===============
  9 (--"(s. "(TERMINAL s)) \implies (\forall M s. TRANSIT M s \implies SAT M s fi')--
  8 (--"s. "(TERMINAL s)"--
  7 (--'TRANSIT M s'--
  6 (--'PATH s L'--
  5 (--'INTR k L'--
  4 (--'TRANSIT s (Sufsts s L k)"--
  3 (--'s'". TRANSIT (Sufsts s L k) s'' \implies TRANSIT s s''')"--
  2 (--'TRANSIT M (Sufsts s L k)"--
  1 (--'s'''". TRANSIT s''' M \implies TRANSIT s''' s''')"--

From the items 7 and 4 we got the new item 2, while the new items 1 and 3 have been deduced by IMP_RES_TAC but are not necessary for our proof. Our next task is to prove the current subgoal simply by instantiating state s, in the hypothesis 9, with the state "Sufst s L k". Before doing so, we need to isolate, by applying the Modus Ponens tactic, the thesis of implication 9. This is achieved by:

MP (el 9 thl) (el 8 thl).

Then, we will instantiate what we have previously got by replacing s by Sufsts s L k.
To specialize (instantiate) a term with another, we will use the tactic SPECL, that takes a list containing the old and the new value of a variable and a term that contains such a variable, returning the instance of such a term (without quantifications related to the substituted variable).

Finally, we will perform a new MP between the term we get by SPECL and the item 2 of the list, so getting an instance of C that is equal to the subgoal. By means of STRIP_ASSUME_TAC we will assume the term that we will deduce by the steps described above. The tactic ASSUM_LIST, taking the full list of hypotheses assumed as theorems and a function that handles them, will be used to deduce the subgoal. So, the following step completes our proof:

e (ASSUM_LIST (fn thl => STRIP_ASSUME_TAC
 (MP (SPECL["'M':CCS'--','Sufsts s L k'--'] (MP (el 9 thl) (el 8 thl))))
 (el 2 thl)))
OK...

Goal proved.
... |- SAT M (Sufsts s L k) fi

Goal proved.
... ... |- SAT M (Sufsts s L k) fi

Goal proved.
... ... ... |- SAT M (Sufsts s L k) fi

Goal proved.
... |-> \l. PATH s L ==> (!k. INTR k L ==> SAT M (Sufsts s L k) fi)

Goal proved.
... |- SAT M s (AG fi)

Goal proved.
... |- TRANSIT M s ==> SAT M s (AG fi)

Goal proved.
... |- !s. TRANSIT M s ==> SAT M s (AG fi)

Goal proved.
... |- !M s. TRANSIT M s ==> SAT M s (AG fi)

Goal proved.
. |- (!s. "(TERMINAL s)) ==> (!M s. TRANSIT M s ==> SAT M s (AG fi))

Goal proved.
|-> ((!s. "(TERMINAL s)) ==> (!M s. TRANSIT M s ==> SAT M s i)) ==> 
  (is. "(TERMINAL s)) ==>
  (!M s. TRANSIT M s ==> SAT M s (AG fi))

Goal proved.
|-> \fi. 
  (is. "(TERMINAL s)) ==> (!M s. TRANSIT M s ==> SAT M fi) ==>
  (is. "(TERMINAL s)) ==>
  (!M s. TRANSIT M s ==> SAT M s (AG fi))

Goal proved.
l- !fi. VALIDT fi \implies VALIDT (AG fi)

Top goal proved.

We tried to make explicit all of the passages of the last call to the "e(...)" function. In fact HOL can implicitly perform all the passages of the last step: we could simply have given the command
e(RESE_TAC)

to resolve the goal by applying some internal standard reductions, instead of explicitly doing the MP and SPECL steps on the list of hypotheses, and thus prove the theorem.

6. Assisted theorem proving in the HOL workbench

After we have built a mechanization of the ACTL calculus, verifying in the meantime its consistency by means of the HOL features, we are now ready to use it to assist the process of proving an ACTL theorem. This task cannot be completely automatic, because a proof tree generally has infinite-length paths and so they cannot be explored by means of an algorithm in a way that always guarantees the generation of a proof.

The main task of HOL is thus to assist the proof process, offering the user a structured approach to the proof task and a large set of tactics and tacticals to treat (sub)formulas. We saw an example of this in the previous section. We now show how HOL can be used as a theorem prover for the ACTL logic. Let us look at the following ACTL theorem:

\[ \vdash \phi \rightarrow \psi \]

\[ \vdash \exists X \phi \rightarrow \exists X \psi \]

We can easily prove it in the classic way by the following steps:

1. \[ \vdash \phi \rightarrow \psi \] Hypothesis
2. \[ \vdash \forall X(\phi \rightarrow \psi) \] by \( \forall X \)
3. \[ \vdash \forall X(\phi \rightarrow \psi) \rightarrow (\exists X, \phi \rightarrow \exists X, \psi) \] A10
4. \[ \vdash \forall X(\phi \rightarrow \psi) \rightarrow (\exists X_{true} \phi \rightarrow \exists X_{true} \psi) \] Instance of A9
5. \[ \vdash \exists X, \phi \rightarrow \exists X, \psi \] MP of 2 and 3
6. \[ \vdash \exists X_{true}, \phi \rightarrow \exists X_{true}, \psi \] MP of 2 and 4
7. \[ \vdash (\exists X_{true}, \phi \lor \exists X, \phi) \rightarrow (\exists X_{true}, \psi \lor \exists X, \psi) \] Tautology by 5 and 6
8. \[ \vdash \exists X \phi \rightarrow \exists X \psi \] By A8 and 7

Below, we automatize the proof above by the support of HOL.

First of all, we insert our thesis into the HOL workbench:

set_goal([],--'!fi gi.VALIDT (imp fi gi) \implies VALIDT (imp (EX fi) (EX gi))'--);

We now instantiate the universally quantified variables to generic values; we then assume the hypothesis of the theorem and we apply rule \( \forall X \) to it:

e (REPEAT STRIP_TAC THEN IMP_RES_TAC RAX);

OK.
1 subgoal
VALIDT (imp (EX fi) EX (gi))

2) VALIDT (imp fi gi)
1) VALIDT (AX (imp fi gi))

We then use a function, VAL_MP, defined as follows:

VAL_MP (T1 |- VALIDT (imp fi gi)) (T2 |- VALIDT gi) = (T1 U T2 |- VALIDT gi),

which is a version of Modus Ponens that merges the hypotheses of the theorems that are given as its arguments. The next step is to apply VAL_MP first to (1) and A10, and then to (1) and A9:

\[
\begin{align*}
e & (ASSUME_TAC (VAL_MP (SPEC_ALL V_AXIOM10) \\
& (ASSUME (\neg\neg VALIDT (AX (imp fi gi)))) THEN \\
& ASSUME_TAC (VAL_MP (SPEC[\neg\neg fi:ACTL', \neg\neg gi:ACTL', \neg\neg tta:Afor']) \\
& V_AXIOM9) \\
& (ASSUME (\neg\neg VALIDT (AX (imp fi gi)))\); \\
\end{align*}
\]

OK..
1 subgoal
\[
\begin{align*}
(\neg\neg VALIDT (imp (EX fi) (EX gi))\); \\
\end{align*}
\]

2) (\neg\neg VALIDT (imp (EX tau fi) (EX tau gi))\)
1) (\neg\neg VALIDT (imp (EX fi tta fi) (EX fi tta gi))\)

We thus reach the step 7 in the hand-proof. We use the following tautology from the ACTL theory:

\[
V_imp_disj : \vdash ((\phi_a \rightarrow \phi_b) \land (\phi_c \rightarrow \phi_d)) \rightarrow ((\phi_a \lor \phi_c) \rightarrow (\phi_b \lor \phi_d)).
\]

In HOL syntax, \(V\_imp\_disj\) becomes:

\[
V\_imp\_disj = \vdash \psi fd fc fb fa. VALIDT (imp (conj (imp fa fb) (imp fc fd)) \\
(imp (disj fa fc) (disj fb fd)));
\]

Step 7 is mechanized in HOL as follows:

\[
\begin{align*}
e & (ASSUME_TAC (VAL_MP \\
& (SPEC [\neg\neg EX tau gi', \neg\neg EX tau fi', \neg\neg EX fi tta gi', \\
& EX fi tta fi']\) V\_imp\_disj) \\
& (VAL_conj espos \\
& (ASSUME (\neg\neg VALIDT (imp (EX fi tta fi) (EX fi tta gi))))\); \\
& (ASSUME (\neg\neg VALIDT (imp (EX tau fi) (EX tau gi)))\)); \\
\end{align*}
\]

OK..
\[
(\neg\neg VALIDT (imp (EXfi) (EX gi))\); \\
\]

1) VALIDT (imp (disj (EX fi tta fi) (EX tau fi)) \\
(imp (disj (EX fi tta gi) (EX tau gi))

In our ACTL theory we previously proved and inserted the straightforward theorem:
\[
V_{eq\_imp\_trans1} : \quad \vdash \phi \rightarrow \gamma \quad \vdash \phi = \psi \\
\quad \frac{}{\vdash \psi \rightarrow \gamma}
\]

and its symmetric one:

\[
V_{eq\_imp\_trans2} : \quad \vdash \phi \rightarrow \gamma \quad \vdash \gamma = \psi \\
\quad \frac{}{\vdash \phi \rightarrow \psi}
\]

In HOL syntax they become:

\[
V_{eq\_imp\_trans\_1} = \exists ! fi \, gi \, di. \quad VALIDT (imp \, fi \, gi) \land VALIDT (equal \, fi \, di) \implies \quad VALIDT (imp \, di \, gi);
\]

\[
V_{eq\_imp\_trans\_2} = \exists ! fi \, gi \, di. \quad VALIDT (imp \, fi \, gi) \land VALIDT (equal \, gi \, di) \implies \quad VALIDT (imp \, fi \, di);
\]

Moreover, we use the function VAL\_eq\_sym, defined as:

\[
VAL_{eq\_sym} (T \not\vdash VALIDT (equal \, fi \, gi)) = (T \not\vdash VALIDT (equal \, gi \, fi));
\]

which is simply used to exchange the members of an equivalence.

The last step in our assisted proof is:

\[
e \quad (ASSUME_TAC (VAL\_eq\_imp\_trans1 \\
\qquad (ASSUME (\neg 'VALIDT (imp (disj (EXf tta fi) (EXTau fi)) \\
\qquad \quad (disj (EXf tta gi) (EXTau gi)))\sim)) \\
\qquad (ASSUME (\neg 'VALIDT (imp (EX f) (disj (EXf tta gi) (EXTau gi)))\sim)) \\
\qquad (VAL\_eq\_sym (SPECL [\neg 'gi:ACTL\sim] VAXIOM))) ;
\]

OK..

top goal proved

\[
\exists ! fi \, gi. \quad VALIDT (imp \, fi \, gi) \implies VALIDT (imp \, (EX \, fi) \, (E \, gi))
\]

Finally, if we wish to add our freshly-proved theorem to our ACTL theory, calling it “RXimp”, we perform the following HOL command:

\[
val REXimp = save_top_thm "RXimp"
\]

7. Conclusions

In this paper we have presented the mechanization in the HOL system of the branching temporal logic ACTL. This logic has action-based modalities and is interpreted over Labelled Transition Systems, so it is suitable to study/specify the behavior of concurrent systems defined by process algebras and modelled on LTSs.

There are two approaches to the use of logic in the study of concurrent systems: the model-theoretical approach and the proof-theoretical one. Each of them has advantages and drawbacks. The model-theoretical approach is almost intuitive. In fact, given a system the satisfiability of a formula, expressing a desired property, is checked against its model.
The proof-theoretical approach is in general less intuitive than the previous one, but it can overcome some of its drawbacks, especially those related to state explosion problems in dealing with real systems.

However, if we have available both a proof assistant for the logic ACTL and a model checker for the same logic we could study a hybrid approach to the formal specification and validation of real systems. The ACTL model checker can be used to verify some properties of the "low-level" entities of a specification, while the proof assistant could be used to prove other properties related to the interaction of such entities when their model-checked properties were given as axioms.

Another interesting extension of this work, could be the mechanization of the logic ACTL*. This logic, as happen for CTL*, includes both linear and branching time operators. It is well known that the model-checking algorithms for this class of logics have an exponential time complexity, making difficult the verification problem. So, a proof-theoretical approach in verification for ACTL* could partially solve this for at least some classes of system properties.

Acknowledgements

The authors want to thank Alessandro Fantechi, of Istituto di Elaborazione dell'Informazione, for interesting discussions on the topics of this paper and Maurizio La Bella, who has developed the ACTL proof assistant.

References


