B4-38

B4-38

Model Checking of non-finite state processes by Finite Approximations

N. De Francesco, A. Fantechi, S. Gnesi, P. Inverardi

October 13, 1994

Abstract

In this paper we present a verification methodology, using an action-based logic, able to check logical properties for full CCS terms, thus allowing complete generality of the class of reactive systems that can be specified. Obviously, for some properties we are only able to give a semidecision procedure. The idea is to use (a sequence of) finite state transition systems which approximate the, possibly infinite state, transition system corresponding to a term. To this end we define a particular notion of approximation, which is stronger than simulation and is very expressive with respect to liveness and safety properties. We show some examples of chains based on this notion and built using different operational semantics. In particular, we define an approximation chain which is very expressive with respect to liveness and safety properties. In order to reason on the properties that we are able to prove with approximation chains, we also give a syntactic characterization of different kinds of properties, which allows us to to prove a set of interesting results. Moreover, we define a criterion, based on the set of checkable properties, to compare the suitability of approximation chains to prove properties. We show how the approach has been implemented in the JACK environment, thus extending its model checking functionalities to the verification of ACTL formulae on non-finite state LTS's.

1 Introduction

Many verification environments are presently available which can be used to automatically verify properties of reactive systems specified by means of process algebras, with respect to behavioural relations and logical properties. Most of these environments [9,20,18,28,17] are based on the hypothesis that the system can be modelled as a finite state Labelled Transition Systems (LTS) and that the logic properties are regular properties. That is, no means are provided to deal with non-finite state LTS's. In general, given a process term p, they try to generate a finite LTS corresponding to p, the verification phase can then start only if this construction has been successfully accomplished. Usually, to avoid the nontermination of the generation phase, a term is required to satisfy some finiteness syntactic conditions: in the case of CCS, for example, terms where a constant x occurs as operand of a parallel composition belonging to the expansion of x are not handled. Although approaches have been proposed to deal with non finite-state systems, which are not based on LTS's [2,?,22,23,24], here we are interested in LTS based environments.

Relevant properties of reactive systems can be expressed by using an action-based logic, for example ACTL [16]. In fact, ACTL is sufficiently powerful to express liveness and safety properties without introducing the overhead of formulae with fixed points, as happens in μ-calculus [26].

In this paper we present a verification methodology able to check ACTL properties for terms of full CCS with no syntactic restriction, thus allowing complete generality of the class of reactive systems that can be specified. We are able to carry on the verification even though the "usual" LTS generation fails. Obviously, for some properties we are able to give only a semidecision procedure. The idea is to use a sequence of finite state transition systems approximating the, possibly infinite state, transition system corresponding to a term by the standard CCS semantics. In order to characterize these approximation chains, we define a particular notion of approximation, which is stronger than simulation and is suitable to define and prove liveness and safety properties of the process terms. We show some examples of chains based on this notion. These chains are built using different operational semantics of *CCS* and this ensures their correctness. In particular, we define an approximation chain, denoted as $\{N_i\}$, which is very expressive with respect to liveness and safety properties.

In order to reason on the properties that we are able to prove with approximation chains, we also give a syntactic characterization of different kinds of properties, which allows us to to prove a set of interesting results.

Moreover, we define a criterion to compare the suitability of approximation chains to prove properties. The criterion is based on the set of properties for which we have a semidecision procedure which uses an approximation chain. Following this notion, we can formalize the fact that a chain is "better" than another one.

Actually, we differ from the Abstract Interpretation approaches [1,10] for model checking of transition systems [4,8] because we do not build an abstract transition system on which the properties are proved, but a chain of finite transition systems: when we manage infinite systems, this allows us to choose the approximation level case by case. Moreover, we are interested in applying the method in practice: in the paper we show how our approach has been used to extend the model checking functionalities of the JACK environment [13] to cover the verification of ACTL formulae on non-finite state LTS's. JACK is an integrated general verification environment that offers a large spectrum of functionalities. In particular JACK integrates the graph and behavioural capabilities of the tool AUTO [28] and the logic facilities of an ACTL model checker [15]. We have added to JACK a tool able to generate the approximation chain $\{N_i\}$ cited above. We remark that the method is non-complete in the sense that in some cases the result of the model checker will give an undefined answer on the validity of a formula. Then, interactively, a user can try again, with a more refined approximation.

2 Background

2.1 CCS

We assume that the reader is familiar with the basic concepts of process algebras and CCS. We summarize the most relevant definitions below, and refer to [30] and to Appendix 1 for more details. The CCS syntax we consider is the following:

$$p ::= \mu.p \mid nil \mid p+p \mid p|p \mid p\backslash A \mid x \mid p[f]$$

Terms generated by p ($Terms$) are called *process terms* (called also *processes* or *terms*); x ranges over a set $\{X, Y, ..\}$, of constants. A constant is defined by a constant definition $x \overset{def}{=} p$, (p is called the expansion of x). Constants may be mutually recursive. As usual, there is a set of visible actions $Vis = \{a, \overline{a}, b, \overline{b}, ...\}$ over which α ranges, while μ, ν range over $Act = Vis \cup \{\tau\}$, where τ denotes the so-called *internal action*. We denote by $\overline{\alpha}$ the action complement: if $\alpha = a$, then $\overline{\alpha} = \overline{a}$, while if $\alpha = \overline{a}$, then $\overline{\alpha} = a$. By nil we denote the empty process. The operators to build process terms are prefixing ($\mu.p$), summation ($p+p$), parallel composition ($p|p$), restriction ($p\backslash A$) and relabelling ($p[f]$), where $A \subseteq Vis$ and $f : Vis \longrightarrow Vis$. Given a term p, an occurrence of a constant x is *guarded in p* if it is within some sub-term of the form $\mu.q$. Moreover, we say that x is *guarded in p* if each occurrence of x in p is guarded. Notice that this definition of guarded terms is more general than the usual one since we accept also constants guarded by the τ action. We assume that

- Vis is finite;
- each constant is guarded in all constant expansions, i.e. for each definition $x \overset{def}{=} p$, each constant occurring in p is guarded in p;
- all terms are closed, i.e. all constants occurring in a term are defined.

Below we list some known definitions regarding the operational semantics of CCS.

An operational semantics OP is a set of inference rules defining a relation $D \subseteq Terms \times Act \times Terms$. The relation is the least relation satisfying the rules. If $(p, \mu, q) \in D$, we write $p \xrightarrow{\mu}_{OP} q$. The rules defining the semantics of CCS [30], from now on referred to as SOS, are recalled in Appendix 1.

A *labelled transition system* (or simply *transition system*) TS is a quadruple (S, T, D, s_0), where S is a set of states, T is a set of transition labels, $s_0 \in S$ is the initial state, and $D \subseteq S \times T \times S$. A transition system is finite if D is finite.

A finite computation of a transition system is a sequence $\mu_1\mu_2..\mu_n$ of labels such that $s_0 \xrightarrow{\mu_1}_{OP} .. \xrightarrow{\mu_n}_{OP} s_n$.

Given a term p (and a set of constant definitions) and an operational semantics OP, $OP(p)$ is the transition system $(Terms, Act, D, p)$, where D is the relation defined by OP.

Let $TS_1 = (S_1, T_1, D_1, s_{0_1})$ and $TS_2 = (S_2, T_2, D_2, s_{0_2})$ be transition systems and let $s_1 \in S_1$ and $s_2 \in S_2$.

s_1 and s_2 are *strongly equivalent* (or simply *equivalent*) ($s_1 \sim s_2$) if there exists a *strong bisimulation* that relates s_1 and s_2. $\mathcal{B} \subseteq S_1 \times S_2$ is a strong bisimulation if $\forall (r, s) \in \mathcal{B}$ (where $\mu \in T_1 \cap T_2$),

- $r \xrightarrow{\mu} r'$ implies $\exists s' : s \xrightarrow{\mu} s'$ and $(r', s') \in \mathcal{B}$;

- $s \xrightarrow{\mu} s'$ implies $\exists r' : r \xrightarrow{\mu} r'$ and $(r', s') \in \mathcal{B}$.

s_2 ***simulates*** s_1 if there exists a ***strong simulation*** that relates s_1 and s_2. $\mathcal{R} \subseteq S_1 \times S_2$ is a strong simulation if $\forall (r, s) \in \mathcal{R}$ (where $\mu \in T_1 \cap T_2$),

- $r \xrightarrow{\mu} r'$ implies $\exists s' : s \xrightarrow{\mu} s'$ and $(r', s') \in \mathcal{R}$.

TS_1 and TS_2 are said to be ***equivalent*** ($TS_1 \sim TS_2$) if a strong bisimulation exists for s_{0_1} and s_{0_2}.

TS_2 simulates TS_1 ($TS_1 \leq TS_2$) if a strong simulation exists that relates s_{01} and s_{02}.

CCS can be used to define a wide class of systems, that ranges from Turing machines to finite systems [31]; therefore, in general, CCS terms cannot be represented as finite states systems.

2.2 The action based logic ACTL

We introduce now the action based branching temporal logic ACTL defined in [16]. This logic is suitable to express properties of reactive systems defined by means of TS's. ACTL is in agreement with the notion of bisimulation defined above. Before defining syntax and semantics of ACTL operators, let us introduce some notions and definitions which will be used in the sequel.

For $A \subseteq Act$, we let $D_A(s)$ denote the set $\{s'$: there exists $\alpha \in A$ such that $(s, \alpha, s') \in D\}$. We will also use the action name, instead of the corresponding singleton denotation, as subscript. Moreover, we let $D(s)$ denote in short $D_{Act}(s)$ and $D_{A_\tau}(s)$ denote $D_{A\cup\{\tau\}}(s)$.
For $A, B \subseteq Act$, we let A/B denote the set $A - (A \cap B)$.

Given a LTS TS=(S,T,D,s_0), we define:

- a *path* in TS is a finite or infinite sequence $s_1, s_2, \ldots$ of states, such that $s_{i+1} \in D(s_i)$. The set of the paths starting from a state s is denoted by $\Pi(s)$. We let $\sigma, \sigma' \ldots$ range over paths;
- a path $\sigma \in \Pi(s)$ is called maximal if it is infinite or if it is finite and its last state s' has no successor states (i.e. $D(s') = \emptyset$);
- if σ is infinite, then we define $|\sigma| = \omega$; if $\sigma = s_1, s_2, \ldots, s_n$, then we define $|\sigma| = n - 1$.
  Moreover, if $|\sigma| \geq i - 1$, we will denote by $\sigma(i)$ the i^{th} state in the sequence. □

To define the logic ACTL [16], an auxiliary logic of actions is introduced. The collection *Afor* of *action formulae* over A is defined by the following grammar where χ, χ', range over action formulae, and $\alpha \in Vis$:

$$\chi ::= \alpha | \neg\chi | \chi \wedge \chi \qquad \wedge$$

We write ff for $\alpha_0 \vee \neg\alpha_0$, where a_0 is some chosen action, and tt stands for $\neg\mathit{ff}$. Moreover, we will write $\chi \wedge \chi'$ for $\neg(\neg\chi \vee \chi')$. An action formula permits the expression of constraints on the actions that can be observed (along a path or after next step); for instance, $\alpha \wedge \beta$ says that the only possible observations are α and β, while tt stands for "all actions are allowed" and ff for "no actions can be observed", that is only silent actions can be performed.

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

- $\alpha \models \beta$ iff $\alpha = \beta$;
- $\alpha \models \neg\chi$ iff not $\overset{\alpha}{a} \models \chi$;
- $\alpha \models \chi \wedge \chi'$ iff $\overset{\alpha}{a} \models \chi$ and $\overset{\alpha}{a} \models \chi'$

Given an action formula χ, the set of the actions satisfying χ can be given by the function $\kappa : \mathcal{AF}(Act) \rightarrow 2^{Act}$ as follows:

- $\kappa(\mathit{tt}) = Act$;
- $\kappa(b) = \{b\}$;
- $\kappa(\neg\chi) = Act/\kappa(\chi)$;
- $\kappa(\chi \vee \chi') = \kappa(\chi) \cup \kappa(\chi')$.

The syntax of ACTL is defined by the state formulae generated by the following grammar:

$$\phi ::= \mathit{tt} \mid \phi \wedge \phi \mid \neg\phi \mid E\gamma \mid A\gamma$$
$$\gamma ::= X_\chi\phi \mid X_\tau\phi \mid \phi\,{}_\chi U\,\phi \mid \phi\,{}_\chi U_{\chi'}\,\phi$$

where χ, χ' range over action formulae, E and A are path quantifiers, X and U are *next* and *until* operators respectively.

Let $TS = (S, Act, D, s_0)$ be a LTS. *Satisfaction* of a state formula ϕ (path formula γ) by a state s (path σ), notation $s \models_{TS} \phi$ ($\sigma \models_{TS} \gamma$) is given inductively by :

$s \models_{TS} tt$		always;
$s \models_{TS} \phi \wedge \phi'$	iff	$s \models_{TS} \phi$ and $s \models_{TS} \phi'$;
$s \models_{TS} \neg\phi$	iff	not $s \models_{TS} \phi$;
$s \models_{TS} E\gamma$	iff	there exists a path $\sigma \in \Pi(s)$ such that $\sigma \models_{TS} \gamma$;
$s \models_{TS} A\gamma$	iff	for all maximal paths $\sigma \in \Pi(s)$, $\sigma \models_{TS} \gamma$;
$\sigma \models_{TS} X_{\chi}\phi$	iff	$\lvert\sigma\rvert \geq 1$ and $\sigma(2) \in D_{\kappa(\chi)}(\sigma(1))$ and $\sigma(2) \models_{TS} \phi$;
$\sigma \models_{TS} X_{\tau}\phi$	iff	$\lvert\sigma\rvert \geq 1$ and $\sigma(2) \in D_{\{\tau\}}(\sigma(1))$ and $\sigma(2) \models_{TS} \phi$;
$\sigma \models_{TS} \phi\,{}_{\chi}U\phi'$	iff	there exists $i \geq 1$ such that $\sigma(i) \models_{TS} \phi'$, and for all $1 \leq j \leq i-1$: $\sigma(j) \models_{TS} \phi$ and $\sigma(j+1) \in D_{\kappa(\chi)_{\tau}}(\sigma(j))$;
$\sigma \models_{TS} \phi\,{}_{\chi}U_{\chi'}\phi'$	iff	there exists $i \geq 2$ such that $\sigma(i) \models_{TS} \phi'$ and $\sigma(i) \in D_{\kappa(\chi')}(\sigma(i-1))$, and for all $1 \leq j \leq i-2$: $\sigma(j) \models_{TS} \phi$ and $\sigma(j+1) \in D_{\kappa(\chi)_{\tau}}(\sigma(j))$.

Several useful modalities can be defined, starting from the basic ones. In particular, we will write:

- $EF\phi$ for $E(tt\ {}_{tt}U\ \phi)$, and $AF\phi$ for $A(tt\ {}_{tt}U\ \phi)$; these are called the *eventually* operators.

- $EG\phi$ for $\neg AF\neg\phi$, and $AG\phi$ for $\neg EF\neg\phi$; these are called the *always* operators.

ACTL can be used to define *liveness* (something good eventually happen) and *safety* (nothing bad can happen) properties of reactive systems. Informally, liveness properties can be classified as in the following [25,29]:
Termination properties: They state that a good thing happens at some states of a computation.
Persistence properties: They state that a good thing happens at infinitely many states of a computation.
Recurrence property: They state that a good thing happens at all but finitely many states of a computation.
Safety properties state that a good thing happens at all states of a computation path.

In a branching time logic both termination and safety properties could be divided into two classes: *universal* liveness (safety) properties and *existential* liveness (safety) properties. The former state that a condition holds at some (all) states of all computation paths. The latter state that a condition holds at some (all) states of one computation path.

For the logic ACTL a linear model checker, AMC, was developed [15], which allows the satisfiability of ACTL formulae to be checked on finite transition systems.

3 Verification by approximations

In this section we present general results which support our approach. Our aim is to provide a methodology to prove (some) logical properties of non finite state transition systems. To this end we first define a notion of chain of finite approximations of the transition system of a term p; then, we present a syntactic characterization, as ACTL formulae, of the logical properties we will deal with. We end the section with a bunch of results about the provability of such formulae by using as models suitable approximation chains of the intended TS.

3.1 Branching complete simulation and Approximation chains

Given a CCS term p, we are going to define suitable approximations of SOS(p) on which logical properties can be checked. In particular, we define chains of finite transition systems which more and more accurately simulate the behaviour of SOS(p). Since each transition system in a chain is finite this allows us to use proof checking methodologies for finite transition systems. In order to do this, we introduce a notion of simulation between transition systems which is stronger than usual simulation. This notion, in contrast with to simulation, permits the definition of TS approximation chains that *preserve* the branching structure, that is, for each approximation, if a node has been exploded all its branches are developed.

Definition 3.1 (Branching Complete Simulation) *Let* $TS_1 = (S_1, T_1, D_1, s_{0_1})$ *and* $TS_2 = (S_2, T_2, D_2, s_{0_2})$ *be transition systems and let* $s_1 \in S_1$ *and* $s_2 \in S_2$.

s_2 BC-simulates s_1 *if there exists a* strong BC-simulation *that relates* s_1 *and* s_2. $\mathcal{R} \subseteq S_1 \times S2$ *is a strong BC-simulation if* $\forall (r,s) \in \mathcal{R}$, ~~(*where* $\mu \in T_1 \cap T_2$)~~

- $r \xrightarrow{\mu} r'$ *implies* $\exists s' : s \xrightarrow{\mu} s'$ *and* $(r', s') \in \mathcal{R}$.

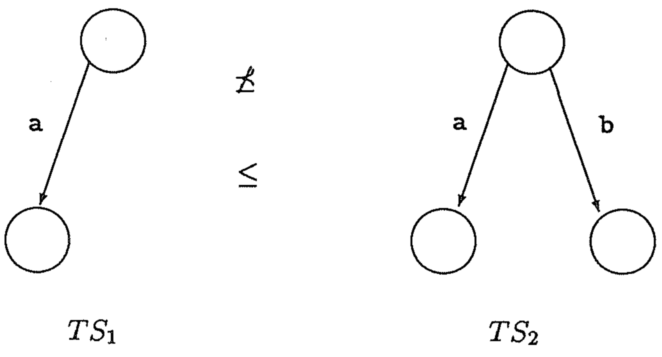

Figure 1: Simulation vs. BC-simulation.

- *$s \xrightarrow{\mu} s'$ implies either $r \sim nil$ or $r \xrightarrow{\mu} r'$ and $(r', s') \in \mathcal{R}$.*

TS_2 BC-simulates *TS_1 ($TS_1 \preceq TS_2$) if a branching complete simulation exists that relates s_{01} and s_{02}.*

It is easy to see that $TS_1 \preceq TS_2$ implies $TS_1 \leq TS_2$, but the converse is not true in general. For example, TS_2 does not $BC - simulate$ TS_1 in Figure 1.

Analogously to the simulation relation, BC-simulation is a preorder. By standard construction [21], it can be shown [14] that $\preceq$ induces a complete partial order on the set of transition systems quotiented by the equivalence relation, $[TS_i] = \{\cancel{(TS_i,} TS_j\cancel{)} | TS_i \preceq TS_j \wedge TS_j \preceq TS_i\}$ and this ensures the existence of the least upper bound of the chains of this complete partial order. It is therefore possible to introduce the following notion of approximation chain of transition systems:

Definition 3.2 (Approximation chain, total and partial TS) *Given a process p, a chain $\{T_i | i \geq 0\}$ of transition systems is called* approximation chain for *p iff:*

- *for each i, T_i is finite;*
- *for each i, $T_i \preceq T_{i+1}$;*

- $\sqcup\{T_i\} \sim SOS(p)$.

Each T_i in the chain is said to be total *if it is strongly equivalent to $SOS(p)$; otherwise it is said to be* partial.

Informally, partial transition systems are those that can be furtherly developed expanding some states. Note that, if we have a finite approximation chain $\{T_i | r \geq i \geq 0\}$, then $T_r \sim SOS(p)$. As we have already noted this notion of approximation chain based on BC-simulation preserves the branching structure of the transition systems all along the chain. This will allow us to prove properties not provable with an approximation chain based only on simulation.

3.2 Temporal properties and approximation chains

In order to formally relate the notion of approximation chains to temporal properties of reactive systems we need to precisely characterize the class of properties we deal with. We start by giving a syntactical presentation of liveness and safety properties by means of ACTL formulae.

Definition 3.3 (Finite property) *We say that σ is a finite property if it can be expressed by an ACTL formula defined by the following grammar:*
$\sigma ::= tt \mid \sigma \wedge \sigma \mid \sigma \vee \sigma \mid \neg\sigma \mid E\gamma \mid A\gamma$
$\gamma ::= X_\chi\sigma \mid X_\tau\sigma$

Definition 3.4 (Positive finite property) *We say that π is a positive finite property if it is a finite property without negations.*

Definition 3.5 (Liveness property) *We say that ψ is a liveness property if one of the following holds, where π is a positive finite property:*

- $\psi = AF\pi$ *or* $\psi = EF\pi$
  *(*termination property*)*
- $\psi = AFAG\pi$, $\psi = EFAG\pi$, $\psi = AFEG\pi$ *or* $\psi = EFEG\pi$
  *(*persistence property*)*
- $\psi = AGAF\pi$, $\psi = EGAF\pi$, $\psi = AGEF\pi$ *or* $\psi = EGEF\pi$
  *(*recurrence property*)*

Definition 3.6 (Safety property) *We say that θ is a safety property if $\theta = AG\pi$ or $\theta = EG\pi$ and π is a positive finite property.*

We remark that the given syntactical presentation of liveness and safety properties does not obviously cover all the liveness and safety properties expressible by means of all the ACTL operators (e.g. $\neg\phi, \phi\,_{\chi}U\,\phi', \phi\,_{\chi}U_{\chi'}\,\phi$). We will comment later on the extension of the following results to such properties.

We have that finite, liveness and safety properties are decidable on a finite state transition system. In general, while finite properties can be provable, liveness (including termination, persistence and recurrence) and safety properties can be undecidable for a non-finite state term p.

We now state two propositions relating approximation chains with liveness and safety properties. Let p be a term and $\{T_i\}$ an approximation chain for p:

Proposition 3.1 *If ϕ is a liveness property, we have that:*

- *if $s_0 \models_{T_i} \phi$ for some i, then $s_0 \models_{SOS(p)} \phi$*

Proof sketch *By structural induction on the structure of the liveness formulae and taking into account that with BC-simulation the simulating transition system is forced to exactly mantain all (and only) the branches of the simulated one, if any.*

PROOF SKETCH ALTERNATIVO. By structural induction on the structure of the liveness formulae:

- *termination property ($AF\pi, EF\pi$): if a positive finite property π is found to hold at a (future) point on the n-th approximation, then it will be true at the same point on the n+1-th approximation, since approximations maintain the computations and do not add branches.*
- *recurrence, persistence: these are infinite properties, that can hold only on infinite transition systems, that is, transition systems with infinite paths. Therefore, if one of these properties is found true on an approximation, this approximation does contain infinite paths and, in particular:*
  1. *either the property required the existence of a path satisfying a condition, and therefore such path has been found on the approximation and will be present also in SOS(p),*
  2. *or it requires a condition to hold on all paths, and hence all paths of the approximation are infinite: in this case, for the definition of BC-simulation and of the approximation chains, the approximation is total.*

Obviously, if $s_0 \not\models_{T_j} \phi$, then nothing can be said about the satisfiability of ϕ on $SOS(p)$.

It is important to remark that liveness properties are not preserved by simulation, while they are preserved by BC-simulation. Consider, for example, the following liveness property:

(1) Each path contains a state from which all the outcoming arcs are labelled by a $(AFAX_a tt)$

If we consider the transition systems TS_1 and TS_2 in Figure 1 we have that $TS_1 \leq TS_2$, but TS_1 verifies the property and TS_2 does not.

Let us consider a liveness property ϕ and an approximation chain $\{T_i\}$ for a term p. Proposition 3.1 above ensures that, if we are able to prove ϕ on an element of the chain, we can assert the validity of ϕ on $SOS(p)$. Thus an algorithm to check the validity of a liveness property is that of checking it on the elements of the chain, starting from the first one, until we find that the property is verified. But the converse of proposition 3.1 is not true in general: if a liveness property ϕ is verified on $SOS(p)$, this does not imply that it is true on an element of $\{T_i\}$. Thus, given an approximation chain, the above algorithm (which checks a liveness property on the elements of the chain) is not in general a semidecision procedure.

Moreover, different approximation chains for the same term can be used to check different sets of properties, in the sense that, given a property ϕ, it is possible that the above algorithm is a semidecision procedure for ϕ if using a chain, while it cannot be used to semidecide ϕ with another chain (we shall see an example in the following section). This suggests a comparison criterion on the suitability of approximation chains for proving properties.

Definition 3.7 (Checkable properties) *Let be given a term p and an approximation chain* $\{T_i\}$ *for p. We say that a liveness property* ϕ *is checkable by* $\{T_i\}$ *if*

- *either* ϕ *is not verified by* $SOS(p)$ *or*
- $T_r \in \{T_i\}$ *exists such that* $s_0 \models_{T_r} \phi$.

The set of checkable properties of p by $\{T_i\}$ *is denoted as* $\mathcal{P}_{T_i}(p)$.

By proposition 3.1, we have that, for each approximation chain $\{T_i\}$ for p,if a property in $\mathcal{P}_{T_i}(p)$ is verified by an element of $\{T_i\}$, then it is verified by $SOS(p)$. Thus $\mathcal{P}_{T_i}(p)$ includes the properties for which there is a semidecision procedure using $\{T_i\}$.

Definition 3.8 (Suitability of approximation chains) *Let be given a term p and two approximations chains $\{T_i\}$ and $\{S_i\}$ for p. We say that $\{T_i\}$ is* more suitable or equal for p *than $\{S_i\}$ if $\mathcal{P}_{S_i}(p) \subseteq \mathcal{P}_{T_i}(p)$. Moreover, $\{T_i\}$ is* strictly more suitable for p *than $\{S_i\}$ if $\mathcal{P}_{S_i}(p) \subset \mathcal{P}_{T_i}(p)$*

Note that the notion of suitability of approximation chains is different from a notion considering the "growing rate" of the chains. Given, for example, an approximation chain $\{T_i\}$, let us consider the chain containing a subset of the elements of $\{T_i\}$, for example the elements of even position, i.e. $\{S_i\} = \{T_0, T_2, T_4, \cdots\}$. We have that $\{S_i\}$ grows faster than $\{T_i\}$, but it is not more suitable than $\{T_i\}$.

Proposition 3.2 *If ψ is a safety property, we have that:*

- *if $s_0 \not\models_{T_i} \psi$ for some i, then $s_0 \not\models_{SOS(p)} \psi$*

Proof sketch *For duality from Prop. 3.1*

Prop. 3.2 corresponds, according to the definition of safety properties in [29,25], to say that if a finite approximation of a term p violates the property, then p itself violates the property. If there exists a T_i such that $s_0 \models_{T_i} \psi$, then nothing can be said about the satisfiability of ψ on SOS(p).

The definitions of checkable properties and of suitability of approximation chains have been given only for liveness properties: dual definitions could be given for safety property as well.

Following propositions 3.1 and 3.2, we note that there is no distinction in proving existential (E...) or universal (A...) properties, due to the fact that BC-simulation preserves the branching structure of the transition systems.[1]

Let us now consider finite properties. Obviously, we have a semidecision procedure for them by using approximation chains. In order to have a decision procedure, we must furtherly constraint our chains. Let us consider,

[1]If the notion of approximation chains were based on simulation instead of BC-simulation, Proposition 3.1 would hold only for existential termination properties, where this subclass of properties is so defined: A ***positive existential finite property*** is a positive finite property containing only E quantifiers; an ***existential termination property*** is a termination property composed by an EF or operator and a positive existential finite property.

for example, the following finite property for $SOS(p)$ for some p:

(2) All paths start with the action b and contain at least an action a as a second action (AX_bEX_att).

Approximation chains are not suitable to give a positive or negative answer if $SOS(p)$ is infinite: in fact a new path of length 2 may appear in whatever element of the chain. The property is decidable if, instead, each transition system T_i of the chain grows on all possible paths with respect to T_{i-1}. This suggests the following notion:

Definition 3.9 (Transition system path-approximation) *Let TS_1 and TS_2 be transition systems. We say that TS_1 is an* n-path-approximation *of TS_2 ($TS_1 \preceq_n TS_2$) if $TS_1 \preceq TS_2$ and the computations of length $\leq n$ of TS_1 and TS_2 coincide.*

Definition 3.10 (Strong Approximation chain) *An approximation chain $\{T_i\}$ is a strong approximation chain if for each i, $T_i \preceq_i T_{i+1}$*

Thus the n-th element of a strong approximation chain have all the paths of length less or equal to n that are in SOS(p). We can now state the following

Proposition 3.3 *Let π be a finite property of depth n, that is with n nested next operators, and $\{T_i\}$ a strong approximation chain for a term p. We have that:*

- $s_0 \models_{SOS(p)} \pi$ *iff* $s_0 \models_{T_n} \pi$.

Proof sketch *For the guardedness of the CCS terms we have that $T_n \preceq_n SOS(p)$.*

~~[illegible]~~ T_n ha tutti i path lunghi n di SOS(p)
By definition π

4 How to build approximations

In this section, we present two ways of constructing approximation chains. In order to obtain correct approximations for a term p, the idea is to derive p using the operational semantics until some stopping condition, thus obtaining a partial transition system, which is furtherly expanded to obtain the successive elements of the chain. The first chain we present is based on the standard SOS semantics. In order to obtain better approximations, we then introduce a second chain, which is based on a different semantics, able to produce "more expressive" transition systems.

4.1 SOS-approximations

In the following, we give an example of a simple approximation chain, based on the SOS semantics, by means of which some properties can be proved. Given a CCS term p, a simple way of approximating SOS(p) is the following chain:

Definition 4.1 ($\{M_i\}$) *The chain $\{M_i = (S_i, Act, D_i, s_0)\}$ is inductively defined as follows:*

- $M_0 = (\{p\}, Act, \{\}, s_0)$
- $M_{i+1} = (S_{i+1}, Act, D_{i+1}, s_0)$ *where*
  - $S_{i+1} = S_i \cup \{q | p \in S_i$ *and* $\mu \in Act$ *exist such that* $p \xrightarrow{\mu}_{SOS} q\}$*;*
  - $D_{i+1} = D_i \cup \{(p, \mu, q) | p \in S_i$ *and* $\mu \in Act$ *exist such that* $p \xrightarrow{\mu}_{SOS} q\}$.

Informally, M_{i+1} is obtained from M_i by adding to the states of M_i all those states reachable from them with only one action and the related transitions. The following proposition holds:

Proposition 4.1 *Given a term p, the chain $\{M_i\}$ defined above is a strong approximation chain for p.*
Proof sketch *By induction on the length of $\{M_i\}$, we prove that, for each M_i it holds $M_i \preceq_i M_{i+1}$. Moreover, by construction, each M_i is finite and the least upper bound of the chain is $SOS(p)$.*

As an example, let us consider the recursive definition of a bag containing two kinds of elements:

$$X = p1.(g1.nil|X) + p2.(g2.nil|X)$$

where p_1 and p_2 represent insertions and g_1 and g_2 deletions of the two kinds of elements, respectively. It is known [3] that X is not finite state and moreover not even context-free. Some properties can be checked on the bag:
Liveness properties:
1) *Termination* The bag is not a set, that is, it is possible to have a multiple

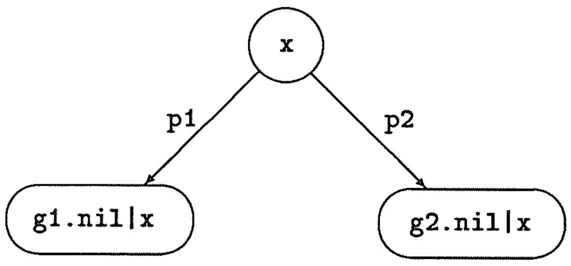

Figure 2: M1 approximation.

occurrence of the same value in the bag. ($AFAX_{p_1}EX_{p_1}tt$).
2) *Persistence property.* There exists a state in a computation path from which it is possible to do infinitely often a *put* action immediately followed by a *get* action ($EFEG(EX_{p_1}EX_{g_1})tt$).
3) *Recurrence property.* There exists a computation path on which on all (but finitely many) states it is possible to do *put* actions ($EGAF(EX_{p_1}tt \vee EX_{p_1}tt)$).
Finite property:
4) At the beginning, only put actions are possible on the empty bag ($AX_{p_1 \vee p_2}tt$).
Safety property:
5) It is always possible to perform a put action ($AGEX_{p_1}tt$).

If we consider the chain $\{M_i\}$ for X, we have that M_0 is given by X itself, while M_1 and M_2 are represented in Figures 2 and 3 respectively.

If we consider M_1 we have the following results:

- $s_0 \not\models_{M_1}$ 1)
- $s_0 \not\models_{M_1}$ 2)
- $s_0 \not\models_{M_1}$ 3)
- $s_0 \models_{M_1}$ 4)
- $s_0 \models_{M_1}$ 5)

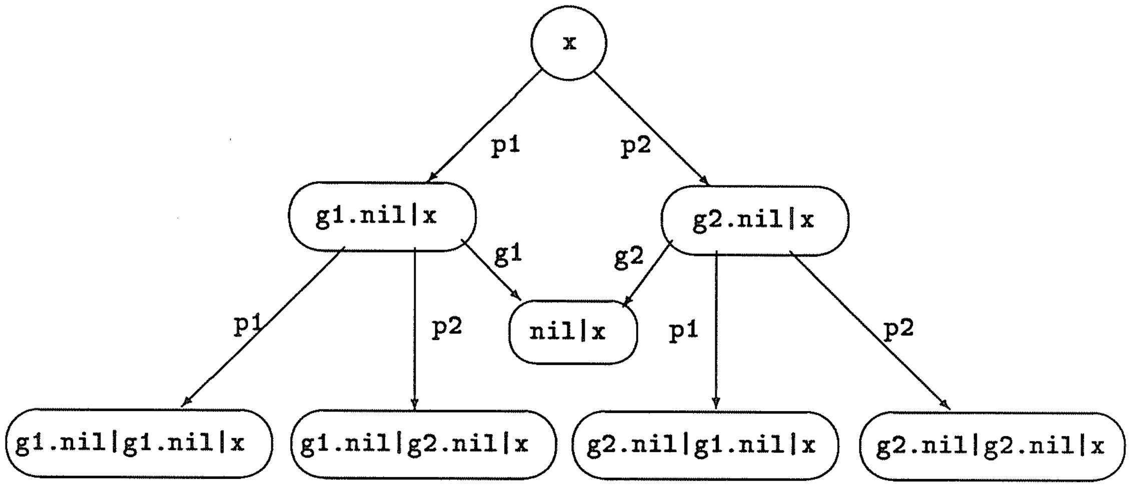

Figure 3: M2 approximation.

This means that property 4) is verified by M_1, and thus is true for the bag, as one should have expected, from proposition 3.3. Moreover property 5) is also verified by M_1, but this does not mean that it is true for the bag, since it is a safety property. The other properties do not hold for M_1. If we consider M_2, we have:

- $s_0 \models_{M_2}$ 1)
- $s_0 \not\models_{M_2}$ 2)
- $s_0 \not\models_{M_2}$ 3)
- $s_0 \models_{M_2}$ 5)

This means that property 1) is true for M_2 and thus is true for the bag, as one should have expected, for proposition 3.1. Instead, properties 2) and 3) are not verified by M_2. It is easy to see that these properties are not verified by any M_i, for each i, since their satisfiability implies detecting a cycle in the transition system: this cycle will never appear in the chain $\{M_i\}$ for p. Thus, if we use this chain to approximate $SOS(p)$, these properties are not semidecidable. Nothing can be asserted instead about property 5), following proposition 3.2.

4.2 SS Approximations

In this section we present a way of approximating $SOS(p)$ which is more suitable than $\{M_i\}$ and so it allows us to prove a greater set of properties. It is based both on a different operational semantics and on a different notion of chain. In [12] the semantics SS was defined, which is more abstract than SOS, since the SS rules have built in some behavioural equivalence axioms, i.e. they accomplish some simplifications on the terms during the derivations, with the purpose of obtaining, if possible, a finite-state transition system for p. The rules of SS are such that SS(p) is strongly equivalent to SOS(p). The definition of SS, whose rules are shown in Appendix 2, is based on the following considerations. Given the CCS syntax, those operators that, in presence of recursion, would give rise to the derivation of growing terms (and therefore to an infinite number of derivations) are parallel composition, restriction and relabelling. For restriction and relabelling, in a language with finite action set, the unlimited growth of terms can be prevented by using suitable inference rules. In fact, successive, possibly intermixed, occurrences of restriction and relabelling can be reduced to only one restriction, followed by only one relabelling. Moreover, the parallel operator can be deleted as soon as one of the two arguments terminates, i.e. is equivalent to *nil*. For example, it is easy to see that two terms $(p|nil)\backslash a\backslash c$ and $p\backslash\{a,c\}$ are equivalent. The SS inference rules accomplish these strong equivalence preserving simplifications during the derivation.

In [12] it is proved that, if we use the SS rules, the only terms that can lead to an infinite transition system are those which allow a new unfolding of a constant x to be performed while some sub-term belonging to the preceding unfolding of x is still present in the term being derived. This only occurs when a term of the kind $x|p$ is reached from x. These terms and the corresponding states are denoted as *dangerous terms* (*dangerous states*) (the precise definition is given in Appendix 3). This suggests to consider as approximation chain the chain $\{N_i\}$ of transition systems, each obtained from the previous one by unfolding dangerous states, until other dangerous states are derived. More formally:

Definition 4.2 ($\{N_i\}$) *The chain $\{N_i = (S_i, Act, D_i, s_0)\}$ is inductively defined as follows:*

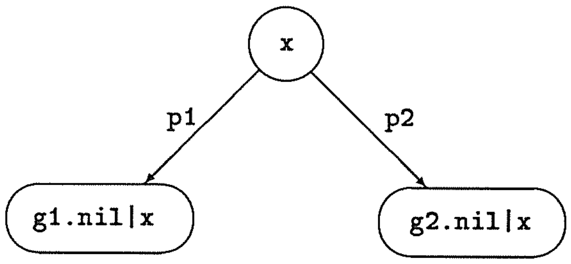

Figure 4: N1 approximation.

- $N_0 = (\{p\}, Act, \{\}, s_0)$ $q_1, \cdots q_n$
- $N_{i+1} = (S_{i+1}, Act, D_{i+1}, s_0)$ *where*
  - $S_{i+1} = S_i \cup \{q | p \in S_i$ *and* $\{\mu_1, \cdots \mu_n\} \subseteq Act$ *exist*, $n \geq 1$, *such that* $p \xrightarrow{\mu_1}_{SS} q_1 \cdots q_n \xrightarrow{\mu_n}_{SS} q$ *and* $q_1, \cdots q_n$ *are not dangerous*$\}$
  - $D_{i+1} = D_i \cup \{$~~(p, μ, q)~~$| p \in S_i$ *and* $\{\mu_1, \cdots \mu_n\} \subseteq Act$ *exist*, $n \geq 1$, *such that* $p \xrightarrow{\mu_1}_{SS} q_1 \cdots q_n \xrightarrow{\mu_n}_{SS} q$ *and* $q_1, \cdots q_n$ *are not dangerous*$\}$
    $(p, \mu_1, q_1)(q_1, \mu_2, q_2) \cdots (q_n, \mu_n, q)$

We can prove, by the results shown in [12], that each N_i is finite state. We remark that the definition of approximation chain based on stopping the derivation of dangerous states cannot be applied if we use SOS to generate the chain (as in the case of $\{M_i\}$), since in this case the finiteness of the approximations is not ensured. If we reconsider the bag example, Figure 4 shows N_1: we have the two dangerous states $g_1.nil|X$ and $g_2.nil|X$. In order to obtain N_2, shown in Figure 5, we expand these states and we stop the new four dangerous states.

The following proposition holds:

Proposition 4.2 *Given a term* p, *the chain* $\{N_i\}$ *defined above is a strong approximation chain for* p.
Proof sketch *By induction on the length of* $\{N_i\}$ *and using the* SS *semantics definition, we prove that, for each* N_i *it holds* $N_i \preceq_i N_{i+1}$. *Moreover, in [12] it is proved that each* M_i *is finite. Finally, the least upper bound of the chain is* $SS(p)$, *which is stronly equivalent to* $SOS(p)$, *as shown in [12].*

Let us now consider the relations between the chains $\{M_i\}$ and $\{N_i\}$.

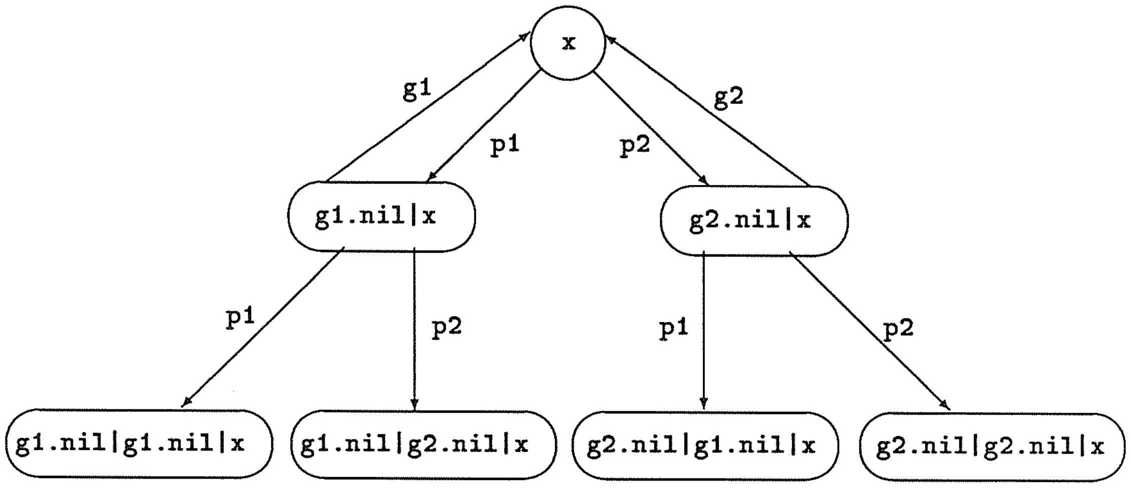

Figure 5: N2 approximation.

Proposition 4.3 *Given a term p, for each i we have $M_i \preceq_i N_i$.*
Sketch of the proof. *By induction on the length of the chains. Note that each state s of N_{i+1}, obtained by a dangerous state of N_i has the same outputs with SS as with SOS, since all constants are guarded in all constant expansions.*

As a consequence of the above proposition, we can state the following propositions:

Proposition 4.4 *For each term p, $\mathcal{P}_{M_i}(p) \subseteq \mathcal{P}_{N_i}(p)$, i.e. $\{N_i\}$ is more suitable or equal than $\{M_i\}$.*
Proof sketch. *For each i, if a property ϕ holds on M_i, a j exists, $j \leq i$, such that ϕ holds on N_j.*

The following proposition states that the converse of proposition 4.4 is not true in general.

Proposition 4.5 *Given a term p, $\mathcal{P}_{M_i}(p) \subset \mathcal{P}_{N_i}(p)$, i.e. $\{N_i\}$ is strictly more suitable than $\{M_i\}$.*

Proof sketch There exist some properties not provable on $\{M_i\}$ which are provable on $\{N_i\}$. Let us consider again the recursive definition of the bag. If we consider N_2, we have the following result:

- $s_0 \models_{N_2} 2)$
- $s_0 \models_{N_2} 3)$
- $s_0 \models_{N_2} 5)$

Thus we can assert the validity of 2) and 3) on the bag. We remark that the validity of 2) and 3) is not provable on the chain $\{M_i\}$ of the previous section because it was not able to detect the cycle of *put* and *get* actions detected instead by $\{N_i\}$. Again nothing can be said on property 5). From proposition 4.4, we have that, for what concerns the properties 1) and 4), we have the same results as with $\{M_i\}$.

5 Implementation in the JACK environment

The JACK system [5] is a verification environment for process algebra description languages. It is able to cover a large extent of the formal software development process, such as rewriting techniques, behavioural equivalence proofs, graph transformations, and (ACTL) logic verification. In JACK a particular description format is used to represent TSs, the so called *format commun fc2*, that has been proposed as standard format for automata [27]. The ACTL model checker was built on the basis of an algorithm similar to that of the EMC model checker [7], so it guarantees model checking of an ACTL formula on a TS in a linear time complexity [15].

5.1 The construction of $\{N_i\}$

The JACK environment has been extended with a tool to build the chain $\{N_i\}$: the tool uses a non-standard semantics, denoted as NSS, shown in Appendix 3, whose rules block dangerous terms when they are derived from the constants. This is done by using an extended language which takes into account whether a parallel operator is inside or outside the recursive unfolding of a constant x: given a non-empty set of constants C, $rec.C.p$ keeps the information that p belongs to the unfolding of the constants in C ($(rec.C)$ is denoted as *recursive prefix*). Thus the term $rec.\{x\}.(x|b.nil)$ is no longer derived, while $(x|b.nil)$ can be derived in the same way of SOS.

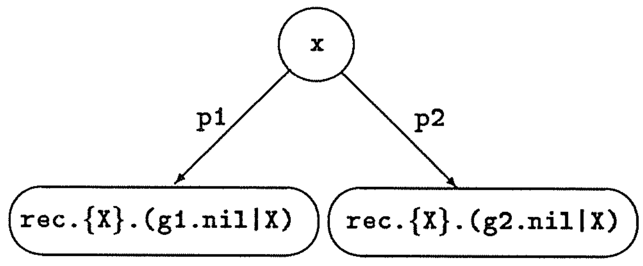

Figure 6: First approximation built by NSS.

The behaviour of NSS on a term p is the following: p is derived (using the same simplifications on $q\backslash A$, $q[f]$ and $r|q$ as with SS) without using terms prefixed by $rec.C$, until a constant x is reached; in this case we derive the x expansion, but prefixed by $rec.\{x\}$. If we reach another constant y, we add y to the prefix, which thus becomes $rec.\{x,y\}$. When a constant terminates an unfolding safely, it is deleted from the prefix. A constant x terminates an unfolding safely only when either x is reached, possibly restricted and/or relabelled, or a term is reached from which x is no longer reachable. When a dangerous term is reached, it is no longer derived. Once N_1 is obtained in this way, the successive approximations are built by deleting the $rec.\{x\}$ prefixes and deriving all dangerous states. For example, the transition system produced for N_1 and N_2 are shown in Figures 6 and 7, respectively.

5.2 Verification of logical properties on non-finite state TSs

In section 4, Propositions 3.1, 3.2 and 3.3 have given a formal support to a verification methodology for non-finite state transition systems: given a CCS term p,

first we define an approximation chain for p and then we check properties on it. We now describe this methodology in proving properties in the JACK environment. Let be given a CCS term p and a list of ACTL formulae to be checked on it. A verification session has the following steps:

1. The term is input to JACK. If the term satisfies the finiteness condition of the transition system generator inside JACK, a corresponding transition system TS is built and the list of ACTL formulae is checked

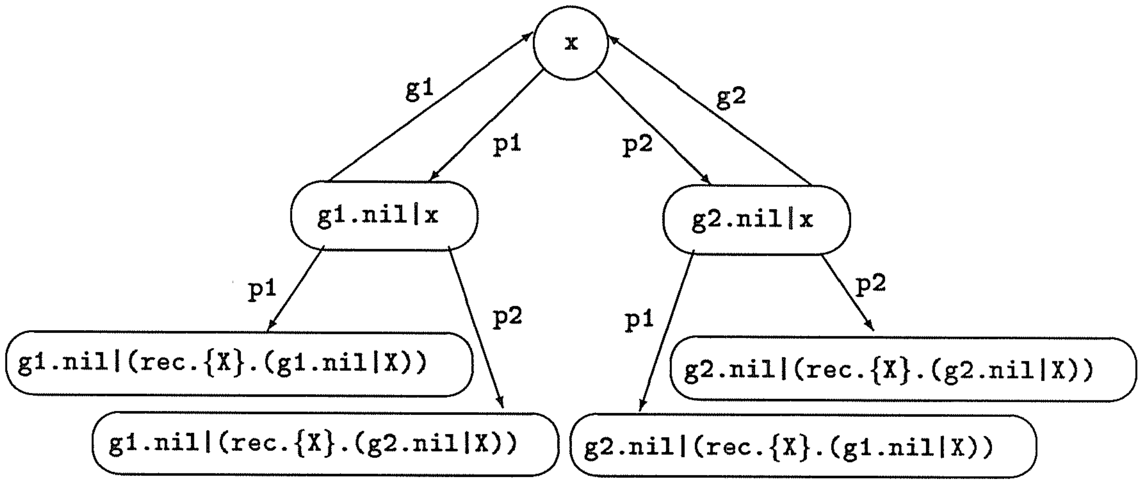
x
g1
g2
p1
p2
g1.nil|x
g2.nil|x
p1
p2
p1
p2
g1.nil|(rec.{X}.(g1.nil|X))
g2.nil|(rec.{X}.(g2.nil|X))
g1.nil|(rec.{X}.(g2.nil|X))
g2.nil|(rec.{X}.(g1.nil|X))

Figure 7: Second approximation built by NSS.

on it. The session terminates.

2. If the syntactic finiteness conditions are not satisfied, then we call the chain generator of JACK. Once obtained the first approximation N_1, we put $TS := N_1$.

3. The list of ACTL formulae is input to the model checker which checks them on TS. If TS is total, the session terminates. If TS is partial, the results of the model checker are analyzed according to propositions 3.1, 3.2 and 3.3. This means that, possibly, a new approximation is built, i.e. $TS := N_{i+1}$ and we repeat step 2.

6 Conclusions

We have presented a method to check logic properties of non-finite state processes. The method is based on checking the validity of an ACTL formula on finite approximations of the infinite state transition system of the processes. The method has been implemented in the JACK environment

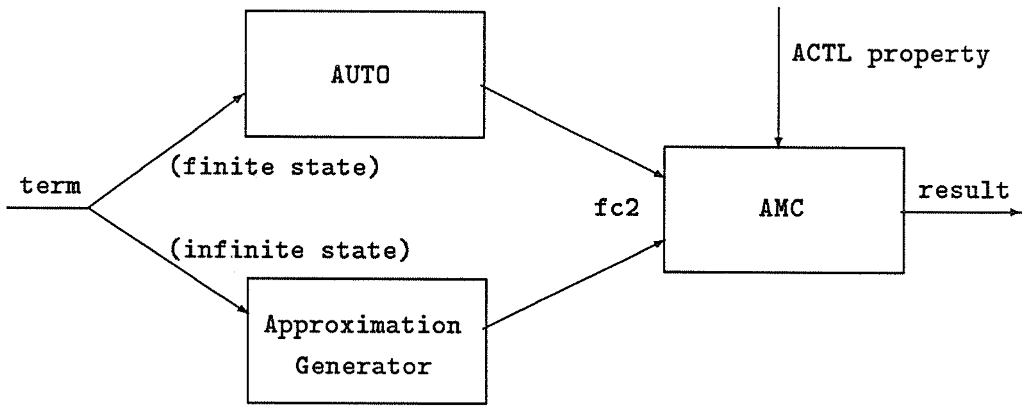

Figure 8: Verification by approximation.

thus providing a uniform framework in which analysing finite and non-finite state systems. We have shown that a particular chain of approximations, namely, that based on the non-standard semantics NSS, makes it possible to verify some properties that cannot be verified with other approximation chains. Moreover, this semantics is able to build a finte-state representation of some processes which are not recognized as such by the usual verification environments.

In the definition of the verification method we have given a classification of safety and liveness properties based on their syntactic appearance. This classification leaves out a number of properties expressed by formulae using not or until operators of the logic. Nevertheless, we argue that the results given to support our verification method are general enough to cover all liveness and safety properties, even if they have been proved only on the well defined classes: for properties outside these classes, the same results and the same verification method apply, whenever the user is able to classify them as safety or liveness ones.

Future work will concern a more deep analisys of classes of logic properties for which our approach provides a verification method on infinite state systems.

Acknowledgement

We wish to acknowledge Luigi Polverini for its work on the implementation of the NSS approximation generator.

References

[1] S. Abramsky, C. Hankin. Abstract Interpretation of Declarative Languages. Ellis Horwood Ltd, 1987.

[2] J. C. M. Baeten, J. A. Bergstra, J. W. Klop. Decidability of bisimulation equivalence for processes generating context-free languages. Journal of ACM 40,3,1993, pp. 653-682.

[3] J. A. Bergstra, J. W. Klop. The Algebra of Recursively Defined Processes and the Algebra of Regular Processes. ICALP '84, LNCS vol. 172, pp. 82-94, 1984.

[4] G. Bruns. A practical technique for process abstraction. CONCUR'93, LNCS Vol. 715, pp. 37-49, 1993.

[5] A. Bouali, S. Gnesi, S. Larosa. The integration Project for the JACK Environment. CWI Report CS-R9443, Amsterdam, 1994.

[6] O. Burkart, B. Steffen. Pushdown processes: Parallel Composition and Model Checking. Proceedings, CONCUR 94, LNCS 836, 1994, pp.98-113.

[7] 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.

[8] E.M.Clarke, O.Grumberg, D.E.Long. Model Checking and Abstraction. POPL, Albuquerque, New Mexico, January 1992.

[9] R. Cleaveland, J. Parrow, B. Steffen. Model Checking and Abstraction. The Concurrency Workbench. Proceedings of Automatic Verification Methods for Finite State Systems. Lecture Notes in Computer Science 407, Springer-Verlag, 1990, pp. 24-37.

[10] P. Cousot, R. Cousot, Systematic Design of Program Analysis Frameworks. Proc. 6th ACM Symp. Principles of Programming Languages (1979), pp. 269-282.

[11] D.Dams, O.Grumberg, R.Gerth. Automatic Verification of Abstract Interpretation of Reactive Systems: Abstractions Preserving ∀CTL*, ∃CTL*, CTL*. IFIP working conference on Programming Concepts, Methods and Calculi (PROCOMET'94),1994.

[12] N. De Francesco, P. Inverardi. Proving Finiteness of CCS Processes by Non-standard Semantics. Acta informatica, vol.31, n.1, 1994, pp. 55-80.

[13] R. De Nicola, A. Fantechi, S. Gnesi, P. Inverardi. JACK: Just Another Concurrency toolKit. IEI Technical Report, 1993.

[14] N. De Francesco, A. Fantechi, S. Gnesi, P. Inverardi. Model Checking of non-finite state processes by Finite Approximations. IEI Internal Report, October 1994.

[15] R. De Nicola, A. Fantechi, S. Gnesi, G. Ristori. An action-based framework for verifying logical and behavioural properties of concurrent systems. Computer Network and ISDN systems, Vol. 25, No.7, 1993, pp 761-778.

[16] R. De Nicola, F. W. Vaandrager. Action versus State based Logics for Transition Systems. Proceedings Ecole de Printemps on Semantics of Concurrency. Lecture Notes in Computer Science 469, Springer-Verlag, 1990, pp. 407-419.

[17] P. van Eijk. The Lotosphere Integrated Tool Environment. Proceedings of the 4th International Conference on Formal Description Techniques (FORTE '91), North-Holland, 1991, pp. 473-476.

[18] J.C. Fernandez, H. Garavel, L. Mounier, A. Rasse, C. Rodriguez, J. Sifakis. A Toolbox for the Verification of LOTOS Programs. 14th ICSE, Melbourne, 1992, pp. 246-261.

[19] S. Gnesi, E. Madelaine, G. Ristori. An Exercise in Protocol Verification. Proceedings of the 2nd Lotosphere Workshop, Pisa, September 1992.

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

[21] M. Hennessy. Algebraic Theory of processes. The MIT Press, 1988.

[22] H. Hungar, B. Steffen. Local Model Checking for Context-Free Processes. Proceedings, ICALP 93, LNCS 700, 1993, pp.593-605.

[23] H. Hungar. Local Model Checking for Parallel Composition of Context-Free Processes. Proceedings, CONCUR 94, LNCS 836, 1994, pp.114-128.

[24] H. Huttel, C Stirling. Actions speak louder than words: Proving Bisimilarity for Context Free Processes. LICS 91, IEEE Computer Society Press, 1991, pp. 376-386.

[25] D. Kozen. Safety and Liveness Properties: A Survey. Bullettin of the European Association for Theoretical Computer Science, 53, 1994, pp. 268-272.

[26] D. Kozen. Results on the Propositional μ-calculus. Theoretical Computer Science, 27, 1983, pp. 333-354.

[27] E. Madelaine. Verification Tools from the Concur Project. Bulletin of EATCS 47, 1992, pp. 110-120.

[28] E. Madelaine, D. Vergamini. AUTO: A Verification Tool for Distributed Systems Using Reduction of Finite Automata Networks. Proceedings FORTE '89, North-Holland, 1990, pp. 61-66.

[29] Z. Manna, A. Pnueli. The Anchored Version of the Temporal Framework, Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency. Lecture Notes in Computer Science 354, Springer-Verlag, 1989, pp. 201-284.

[30] R. Milner. Communication and Concurrency. Prentice Hall,1989.

[31] D. Taubner. Finite Representations of CCS and TCSP Programs by Automata and Petri Nets. LNCS vol. 369, 1989.

Appendix 1. The SOS semantics.

The *SOS* semantics is shown in figure 9.
bf Appendix 2. The SS semantics

The *SS* semantics contains the rules shown in figure 10. The following notation is used in the rules:

$term(p) =$

- $true$, if p has no derivative with SOS
- $false$, otherwise

$p\backslash\backslash A =$

- $p\backslash A$, if $p \neq q\backslash B, p \neq q[f]$
- $q\backslash A \cup B$, if $p = q\backslash B$
- $q\backslash f^1(A)[f]$, if $p = q[f], q \neq r\backslash B$
- $q\backslash f^1(A) \cup B[f]$, if $p = q\backslash B[f]$

$p[[f]]=$

- $p[f]$, if $p \neq q[g]$
- $q[f \circ g]$, if $p = q[g]$

Appendix 3. The NSS semantics

For each rule R belonging to the SS semantics, except $S-Con$, there is a rule in the NSS semantics, which is equal to R, except that it adds to the premise the condition "p well-formed". Moreover, NSS also contains rules:
The following notation is used in the rules above :

Extended CCS syntax:
$p ::= b \mid rec.C.p \mid p|p \mid p\backslash A \mid p[f]$,

where b is a CCS term and C is a non-empty set of constants.

Given a CCS term p, we indicate by $p(\backslash A[f])$ the set of the terms $\{p, p\backslash A, p[f], p\backslash A[f] | A \subseteq Vis, f : Vis \rightarrow Vis\}$.

Given a term p of the extended language, we denote by $F(p)$ the CCS term obtained by deleting from p all the recursive prefixes.

Given a CCS term p, a constant x is potentially-reachable ($P-reachable$) by p if:

- $p \notin x(\backslash A[f])$; and

- either x is a sub-term of p or a constant y is a sub-term of p and x is P-reachable by the expansion of y.

$[p]_C$ *definition.* Given a non-empty set C of constants and an ECCS term p:

- If $p \neq p\backslash A, p \neq q[f]$ and $p \neq rec.E.q$, let $D = \{x | x \in C$ and x is P-reachable by $F(p)\}$. We define $[p]_C = p$ if $D = \emptyset$, $[p]_C = rec.D.p$ otherwise;
- If$p \neq q\backslash A, p \neq q[f]$ and $p = rec.E.q$, then $[p]_C = [q]_{C \cup E}$;
- $[p\backslash A]_C = [p]_C\backslash A$;
- $[p[f]]_C = [p]_C[f]$.

Dangerous terms, Safe terms. Let be given a constant x. The set of $x - dangerous$ terms is the least set defined by the following rules:

- a term $p|q$, where x is either non-guarded in p or/and non-guarded in q, is x-dangerous;
- p x-dangerous implies $p + q, q + p, p|q, q|p$ and $p(\backslash A[f])$ x-dangerous, for any q.

A term p is $x - safe$ if it is not x-dangerous. Given a set C of constants, we say that a term p is $C - dangerous$ if it is x-dangerous for at least one $x \in C$, and a term q is $C - safe$ if it is x-safe for all $x \in C$.

Well-formedness. A term p of the extended language is $well - formed$ if, for each sub-term of the form $rec.C.q$, either $F(q)$ is C-safe or $term(F(q)) = true$.

$$\textbf{Act}\ \frac{}{\mu.p \xrightarrow{\mu} p}$$

$$\textbf{Con}\ \frac{p \xrightarrow{\mu} p',\ x \overset{def}{=} p}{x \xrightarrow{\mu} p'}$$

$$\textbf{Sum}\ \frac{p \xrightarrow{\mu} p'}{p + q \xrightarrow{\mu} p' \text{ and } q + p \xrightarrow{\mu} p'}$$

$$\textbf{Par}\ \frac{p \xrightarrow{\mu} p'}{p|q \xrightarrow{\mu} p'|q \text{ and } q|p \xrightarrow{\mu} q|p'}$$

$$\textbf{Com}\ \frac{p \xrightarrow{\alpha} p',\ q \xrightarrow{\overline{\alpha}} q'}{p|q \xrightarrow{\tau} p'|q'}$$

$$\textbf{Res}\ \frac{p \xrightarrow{\mu} p',\ \mu, \overline{\mu} \notin A}{p\backslash A \xrightarrow{\mu} p'\backslash A}$$

$$\textbf{Rel}\ \frac{p \xrightarrow{\mu} p'}{p[f] \xrightarrow{f(\mu)} p'[f]}$$

Figure 9: The SOS rules