Model Checking of non-finite state processes by Finite Approximations

N. De Francesco ♣, A. Fantechi ♣ ◇, S. Gnesi ◇ and P. Inverardi ◇ ♡

♣ Dipartimento di Ingegneria dell'Informazione, Univ. di Pisa, Italy, e-mail: nico@iet.unipi.it, fantechi@iet.unipi.it

◇ Istituto di Elaborazione dell'Informazione, C.N.R. Pisa, Italy, e-mail: gnesi@iei.pi.cnr.it

♡ Dip. di Matematica Applicata, Univ. dell'Aquila, Italy, e-mail:inverard@iei.pi.cnr.it

Abstract

In this paper we present a verification methodology, using an action-based logic, able to check properties for full CCS terms, allowing also verification on infinite state systems. 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, suitable to define and prove liveness and safety properties of the process terms.

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 [7, 12, 13, 19] 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. Usually, in these environments, to avoid the nontermination of the generation phase a term must satisfy some finiteness syntactic conditions: in the case of CCS, for example, terms where a process variable x occurs in a parallel composition belonging to the definition of x are not handled [22].

We are interested here to deal with non finite-state systems; approaches have been proposed to this aim, which are not based on LTS's [1, 4, 14, 15, 16]; we consider instead LTS based verification. The idea is to use, for proving a logical property, a sequence of finite state LTSs approximating the, possibly infinite state, LTS corresponding to a term by the standard CCS semantics.

In this paper we present a verification methodology to check properties expressed in ACTL, an action based logic [11], on full CCS terms (with no syntactic restriction), thus allowing complete generality of the class of reactive systems to be specified. We are able to carry on the verification even though the "usual" LTS generation fails. Obviously, for some of the properties, we are able to give only a semidecision procedure. This procedure is based on a notion of approximation and on the study of the ACTL properties *preserved* by the approximation. In this way, we can infer the satisfaction of a property by the whole system from the satisfaction of the property by a chain of approximations. In particular, we define an approximation chain, denoted as $\{N_i\}$, which is very expressive with respect to liveness properties.

In order to reason on the properties that we are able to prove with approximation chains, we start giving a syntactic characterization of different kinds of properties. Moreover, we define a

$$\textbf{Act}\ \frac{}{\mu.p \xrightarrow{\mu} p} \qquad \textbf{Con}\ \frac{p \xrightarrow{\mu} p',\ x \stackrel{def}{=} p}{x \xrightarrow{\mu} p'}$$

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

$$\textbf{Com}\ \frac{p \xrightarrow{\alpha} p',\ q \xrightarrow{\overline{\alpha}} q'}{p|q \xrightarrow{\tau} p'|q'} \qquad \textbf{Res}\ \frac{p \xrightarrow{\mu} p',\ \mu, \overline{\mu} \notin A}{p\backslash A \xrightarrow{\mu} p'\backslash A}$$

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

Figure 1: The SOS rules

criterion to compare the suitability of approximation chains to prove properties. Following this notion, we formalize the fact that a chain is "better" than another one, if its set of provable properties is greater. Our work differs from the abstract interpretation approaches for model checking of transition systems [2, 6, 8] since we do not build an abstract (with respect to values) model on which the properties are proved, but a suitable chain of finite labelled transition systems based on the operational semantics: when dealing with infinite systems, this allows us to choose the approximation level case by case.

2 Background

2.1 CCS

We summarize the most relevant definitions regarding CCS, and refer to [21] for more details. The CCS syntax 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 process variables. A process variable is defined by a process definition $x \stackrel{def}{=} p$, (p is called the expansion of x). 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 $f: Vis \to Vis$. Given a term p, an occurrence of a process variable x is *guarded in* p if it is within some sub-term of the form $\mu.q$. We assume that (i) Vis is finite; (ii) for each definition $x \stackrel{def}{=} p$, each occurrence of each process variable is guarded in p; (iii) all terms are closed, i.e. all variables occurring in a term are defined.

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 [21], from now on referred to as SOS, are recalled in Figure 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 process variable 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 $\forall(s_1, s_2) \in \mathcal{B}$ (where $\mu \in T_1 \cup T_2$),

- $s_1 \xrightarrow{\mu}_1 s_1'$ implies $\exists s_2' : s_2 \xrightarrow{\mu}_2 s_2'$ and $(s_1', s_2') \in \mathcal{B}$; $s_2 \xrightarrow{\mu}_2 s_2'$ implies $s_1 \xrightarrow{\mu}_1 s_1'$ and $(s_1', s_2') \in \mathcal{B}$

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 $\forall(s_1, s_2) \in \mathcal{R}$ (where $\mu \in T_1 \cup T_2$): $s_1 \xrightarrow{\mu}_1 s_1'$ implies $\exists s_2' : s_2 \xrightarrow{\mu}_2 s_2'$ and $(s_1', s_2') \in \mathcal{R}$.

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}. Two CCS terms p and q are *equivalent* ($p \sim q$) if $SOS(p) \sim SOS(q)$.

TS_2 simulates TS_1 if a strong simulation $\mathcal{R}$ exists such that $(s_{01}, s_{02}) \in \mathcal{R}$.

Given a state s of a transition system $TS = (S, T, D, s_0)$, we say that $s \not\rightarrow$ if no $s' \in S$ and $\mu \in T$ exist such that $(s, \mu, s') \in D$.

CCS can be used to define a wide class of systems, that ranges from Turing machines to finite systems [22]; therefore, in general, CCS terms cannot be represented as finite state systems.

2.2 ACTL

We introduce now the action based branching temporal logic ACTL defined in [11]. 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:

- σ is a path from $r_0 \in S$ if either $\sigma = r_0$ (the empty path from r_0) or σ is a (possibly infinite) sequence $(r_0, \alpha_1, r_1)(r_1, \alpha_2, r_2)\ldots$ such that $(r_i, \alpha_{i+1}, r_{i+1}) \in D$ for each $i \geq 0$.
- A path σ is called maximal if either it is infinite or it is finite and its last state r has no successor states ($D(r) = \emptyset$). The set of maximal paths from r_0 will be denoted by $\Pi(r_0)$.
- If σ is infinite, then $|\sigma| = \omega$.
  If $\sigma = r_0$, then $|\sigma| = 0$.
  If $\sigma = (r_0, \alpha_1, r_1)(r_1, \alpha_2, r_2)\ldots(r_n, \alpha_{n+1}, r_{n+1})$, $n \geq 0$, then $|\sigma| = n + 1$. Moreover, we will denote the i^{th} state in the sequence, i.e. r_i, by $\sigma(i)$. □

To define the logic ACTL [11], an auxiliary logic of actions is introduced. The collection $\mathcal{AF}$ of *action formulae* over Vis is defined by the following grammar where χ, χ', range over action formulae, and $\alpha \in Vis$:

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

We write ff for $\alpha_0 \wedge \neg\alpha_0$, where a_0 is some chosen action, and tt stands for $\neg \mathit{ff}$. Moreover, we will write $\chi \vee \chi'$ for $\neg(\neg\chi \wedge \chi')$. An action formula permits the expression of constraints on the actions that can be observed (along a path or after next step); for instance, $\alpha \vee \beta$ says that the only possible observations are α or 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, $\alpha \models \chi$, is defined inductively by:

$\bullet \alpha \models \beta$ iff $\alpha = \beta$; $\bullet \alpha \models \neg\chi$ iff not $\alpha \models \chi$; $\bullet \alpha \models \chi \wedge \chi'$ iff $\alpha \models \chi$ and $\alpha \models \chi'$

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

$\bullet \kappa(\mathit{tt}) = Vis$; $\bullet \kappa(\alpha) = \{\alpha\}$; $\bullet \kappa(\neg\chi) = Vis/\kappa(\chi)$; $\bullet \kappa(\chi \vee \chi') = \kappa(\chi) \cup \kappa(\chi')$.

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 :

$s \models_{TS} \mathit{tt}$		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 $1 \leq j \leq i-1$: $\sigma(j) \models_{TS} \phi$ and $\sigma(j) \in D_{\kappa(\chi)_\tau}(\sigma(j-1))$.

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

- $EF\phi$ for $E(\mathit{tt}\,_{\mathit{tt}}U\,\phi)$, and $AF\phi$ for $A(\mathit{tt}\,_{\mathit{tt}}U\,\phi)$; 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. In a branching time logic both liveness 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. Moreover liveness properties can be better classified as in the following [17, 20]:

Termination properties: "a good thing happens at some states of a (all) computation(s)".
Recurrence properties: "a good thing happens at infinitely many states of a (all) computation(s)".
Persistence property: "a good thing happens at all but finitely many states of a (all) computation(s)".
We can also talk of *finite properties*, that state some condition on the finite initial part of the behaviour of the system.

2.3 Infinite state systems and logical properties

We know that all ACTL formulae are decidable on finite state transition systems and the linear time ACTL model checker [10] can be used to do this job. Hence, when we have a CCS description of a system and we want to prove on it ACTL properties, the labeled transition system associated to it needs to be built. This will be the model on which the satisfiability of the formulae will be checked. Problems, obviously, arise when the system to be modelled has an infinite state representation, due for example to the interplay between parallel composition and recursion operators.

As an example, let us consider the CCS 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 that X is neither finite state nor context-free. Some typical properties of a bag could be requested to be checked on this specification, in order to validate it:
1) The bag is not a set, therefore it is possible to put twice the same value in the bag consecutively: $AFAX_{p_1}EX_{p_1}tt$.
2) It is possible, on all (but finitely many) states to do a *put* action immediately followed by a *get* action: $EFEG(EX_{p_1}EX_{g_1}tt)$.
3) There exists a computation path on which it is possible to do infinitely often *put* actions: $EGAF(EX_{p_1 \vee p_2}tt)$.
4) It is always possible to perform a put action: $AGEX_{p_1 \vee p_2}tt$.

3 Verification by approximations

Let us first present a syntactic characterization, as ACTL formulae, of the logical properties we will deal with. We then introduce the general notion of chain of finite approximations of the transition system of a term p. Finally, we introduce a notion of approximation suitable to prove liveness properties.

3.1 Temporal properties

Definition 3.1 (Positive formula) *We say that π' is a positive formula if it is an ACTL formula without negations.*

Definition 3.2 (Liveness property) *We say that ψ is a liveness property if one of the following holds, where π' is a positive formula:*

- $\psi = AF\pi'$ *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 3.3 (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 (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.*

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 as the negation operator. Indeed, negation makes the syntactic classification of formulae difficult. Following this classification, we have that properties 1) to 3) of the bag example are liveness properties, while 4) is a safety one.

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

3.2 Approximation chains

Given a CCS term p, we define chains of finite LTSs which more and more accurately simulate the behaviour of SOS(p). Since each LTS in a chain is finite proof checking methodologies for finite LTSs can be used. First we define in the most general way the concept of approximation chain. In the following we denote, with $\mathcal{T}$ and T, the set of all LTSs and a generic LTS, respectively.

Definition 3.6 (Approximation chain) *Let $\preceq$ a preorder over $\mathcal{T}$. We say that T_1 approximates by $\preceq$ ($\preceq$-approximates) T_2 iff $T_1 \preceq T_2$. Given a term p, a chain $\{T_i(p) | i \geq 0\}$ on $(\mathcal{T}, \preceq)$ is called* approximation chain for p by $\preceq$ *($\preceq$-approximation chain) iff:*

- *for each i, $T_i(p)$ is finite;*
- *for each i, $T_i(p) \preceq T_{i+1}(p)$;*
- $\sqcup\{T_i(p)\} \sim SOS(p)$.

Note that, if we have a finite approximation chain $\{T_i(p) | r \geq i \geq 0\}$, then $T_r(p) \sim SOS(p)$.

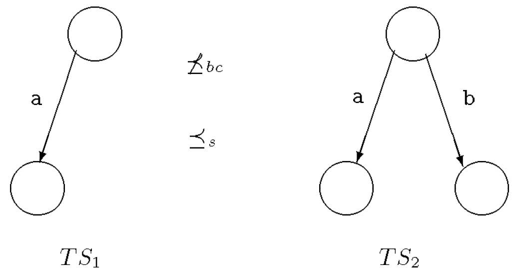

Figure 2: Simulation vs. BC-simulation.

Definition 3.7 (Properties preserved by $\preceq$) *A preorder $\preceq$ preserves a property ϕ if whenever T_1 verifies ϕ and $T_1 \preceq T_2$ then T_2 verifies ϕ.*

The above definitions allow us to define a procedure for proving the validity of a property on an infinite state-system, by checking the property on the elements of an approximation chain, starting from the first one, until we find that the property is verified. The procedure is sound if the chain preserves the property, i.e. it must happen that, if we are able to prove ϕ on an element of the chain, we can assert the validity of ϕ on $SOS(p)$. This means that the property must be monotonic on the preorder. The first result we show is that simulation, from now on denoted by $\preceq_s$, is not suitable to prove all liveness properties.

Proposition 3.1 *$\preceq_s$ does not preserve all liveness properties.*
Proof *Let us consider the following liveness property:*
Each path contains a state from which all the outcoming arcs are labelled by *a, expressed by $(AFAX_a tt)$ and the transition systems TS_1 and TS_2 in Figure 2.*
We have that $TS_1 \preceq_s TS_2$, but TS_1 verifies the property and TS_2 does not.

In order to manage all liveness properties, we now introduce a stronger notion of simulation between transition systems. This notion, in contrast to simulation, permits the definition of approximation chains that *preserve* the branching structure, that is, for each approximation, if a node has been exploded, all its branches have been developed.

Definition 3.8 (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. $\mathcal{R} \subseteq S_1 \times S_2$ is a strong BC-simulation if $\forall (s_1, s_2) \in \mathcal{R}$, $\mu \in T_1 \cup T_2$,*

- $s_1 \xrightarrow{\mu}_1 s'_1$ *implies* $\exists s'_2 : s_2 \xrightarrow{\mu}_2 s'_2$ *and* $(s'_1, s'_2) \in \mathcal{R}$.
- $s_2 \xrightarrow{\mu}_2 s'_2$ *implies either* $s_1 \not\rightarrow_1$ *or* $s_1 \xrightarrow{\mu}_1 s'_1$ *and* $(s'_1, s'_2) \in \mathcal{R}$.

TS_2 BC-simulates *TS_1 ($TS_1 \preceq_{bc} TS_2$) if a branching complete simulation $\mathcal{R}$ exists such that $(s_{0_1}, s_{0_2}) \in \mathcal{R}$.*

It is easy to see that $TS_1 \preceq_{bc} TS_2$ implies $TS_1 \preceq_s TS_2$, but the converse is not true in general. For example, TS_2 does not BC-simulate TS_1 in Figure 2.

The following proposition holds.

Proposition 3.2 *$\preceq_{bc}$ is a preorder.*
Proof Sketch. *It is reflexive and transitive.*

The notion of approximation chain based on BC-simulation preserves the branching structure of the transition systems all along the chain. This allow us to prove properties not provable on a chain based on simulation. One of the main results of the paper is the following:

Proposition 3.3 *$\preceq_{bc}$ preserves liveness properties.*
Proof sketch *By structural induction on the structure of the liveness formulae and taking into account that the liveness properties are defined on a positive fragment of ACTL and that the BC-simulation forces the simulating transition system to exactly mantain all the (bisimilar) branches of the simulated one, if any.*

It is now easy to relate approximation chains, based on BC-simulation, with liveness properties. The following proposition is the basis of our verification method.

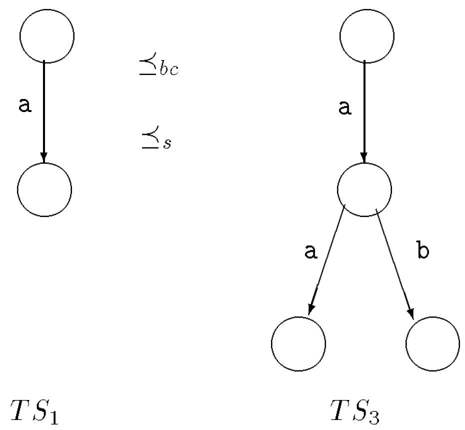

Figure 3: Simulation and BC-simulation.

Proposition 3.4 *Let p be a term and $\{T_i(p)\}$ a $\preceq_{bc}$-approximation chain for p. If ϕ is a liveness property, it holds that: if $s_0 \models_{T_i(p)} \phi$ for some i, then $s_0 \models_{SOS(p)} \phi$.*
Proof. *It follows by proposition 3.3.*

For safety properties, the following proposition holds :

Proposition 3.5 *$\preceq_s$ and $\preceq_{bc}$ do not preserve safety properties.*
Proof *Let us consider the safety property* "It is always possible to perform an action *a* ", *expressed by $AGAX_a tt$, and the transition systems TS_1 and TS_3 in Figure 3. Then $TS_1 \preceq_{bc} TS_3$ and $TS_1 \preceq_s TS_3$, but TS_1 verifies the property and TS_3 does not.*

This means that we cannot define a semidecision procedure based on approximation chains for the satisfaction of safety properties; on the converse, the following proposition gives a method to prove that a safety property does not hold for $SOS(p)$.

Proposition 3.6 *Let p be a term and $\{T_i(p)\}$ a $\preceq_{bc}$-approximation chain for p. If ψ is a safety property, it holds that: if $s_0 \not\models_{T_i(p)} \psi$ for some i, then $s_0 \not\models_{SOS(p)} \psi$.*
Proof sketch *For duality from Prop. 3.4*

Prop. 3.6 corresponds, according to the definition of safety properties in [20, 17], to say that if a finite approximation of a term p violates the property, then p itself violates the property.

Let us now consider finite properties. The following holds:

Proposition 3.7 *$\preceq_s$ does not preserve all finite properties.*
Proof sketch *Consider the finite property:* Each path starts with an action *a ($AX_a tt$), with TS_1 and TS_2 of Figure 2. We have that $TS_1 \preceq_s TS_2$, but TS_1 verifies the property and TS_2 does not.*

Since finite properties represent a particular class of liveness properties we have a semidecision procedure for testing the validity of these properties by using approximation chains based on $\preceq_{bc}$. We can do more, as one should have expected, and provide a decision procedure for finite properties. To this end, we furtherly constrain our chains. Let us consider, for example, the following finite property for $SOS(p)$ for some p:
All paths start with the action b and contain at least an action a as a second action ($AX_b EX_a tt$).

Approximation chains based on $\preceq_{bc}$ 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 $T_i(p)$ of the chain grows on all possible paths with respect to $T_{i-1}(p)$. 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*

- $TS_1 \preceq_{bc} TS_2$*;*
- *either $TS_1 \sim TS_2$ or the paths of length $\leq n$ of TS_1 and TS_2 coincide.*

We can now state the following:

Proposition 3.8 *Let π be a finite property of depth n, that is with only n nested next operators, and $\{T_i(p)\}$ a $\preceq_{bc}$-approximation chain for a term p such that $T_i(p) \preceq_i SOS(p)$ for each i. Then $s_0 \models_{SOS(p)} \pi$ iff $s_0 \models_{T_n(p)} \pi$.*
Proof sketch *We have that $T_n(p)$ has all the paths of length n of $SOS(p)$.*

4 How to build approximations

In this section, we present some 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, described in the following sub-secton, 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

Definition 4.1 ($\{M_i(p)\}$) *Given a term p, the chain $\{M_i(p) = (S_{M_i}, Act, D_{M_i}, s_0)\}$ is inductively defined as follows:*

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

Informally, $M_0(p)$ has the only state p without transitions and $M_{i+1}(p)$, $i \geq 0$, is obtained from $M_i(p)$, by adding to the states (and the related transitions) of $M_i(p)$ all those states reachable from them with only one action. The following proposition holds:

Proposition 4.1 *Given a term p, the chain $\{M_i(p)\}$ is a $\preceq_{bc}$-approximation chain for p.*
Proof sketch. *By induction on the length of the chain and by definig suitable BC-simulations.*

Actually, the chain $\{M_i(p)\}$ is the simplest chain derivable from SOS(p) which is a $\preceq_{bc}$-approximation chain. In fact the simpler approximation chain which at any step adds a single new transition to the previous element of the chain, is not a $\preceq_{bc}$-approximation chain.

Example 4.1 Let us now reconsider the bag example of section 2.3, and try to prove the properties on the chain $\{M_i(X)\}$. Since $\{M_i(X)\}$ is a $\preceq_{bc}$-approximation chain, it preserves all properties from 1) to 3) and does not preserve the safety property 4). Thus, if we find that an approximation $M_i(X)$ verifies a property among 1) and 3), we prove that the property holds for the bag (i.e. $SOS(X)$). $M_0(X)$ is given by a transition system with only one state, i.e. X itself, while $M_1(X)$ and $M_2(X)$ are represented in Figures 4 and 5 respectively.

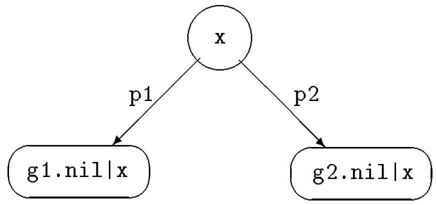

Figure 4: $M_1(X)$

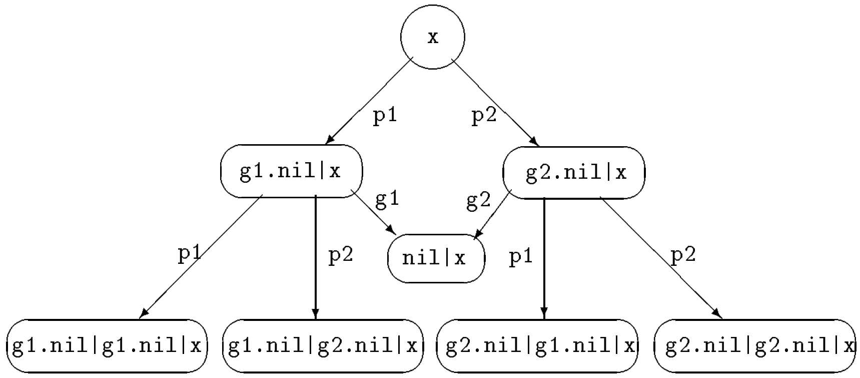

Figure 5: $M_2(X)$

We have that property 1) is not satisfied by $M_1(X)$, it is satisfied by $M_2(X)$ and thus it is true for the bag. Moreover, property 4) is verified by $M_1(X)$, but this does not mean that it is true for the bag, since it is a safety property. Properties 2) and 3) are not verified by $M_1(X)$ neither by $M_2(X)$. It is easy to see that these properties are not verified by any $M_i(X)$, for each i. In fact their satisfiability implies detecting a cycle in the transition system: this cycle will never appear in the chain $\{M_i(X)\}$.

Thus, if we use this chain to approximate $SOS(X)$, these properties are not provable, while they hold for $SOS(X)$. Nothing can instead be asserted about property 4), following proposition 3.6. The following proposition states that each M_i is a $\preceq_i$-approximation of $SOS(p)$, i.e. the size of the transition system grows.

Proposition 4.2 *Given a term p, for each $i \geq 0$, $M_i(p) \preceq_i SOS(p)$.*
Proof. *By proposition 4.1 it holds that $M_i(p) \preceq_{bc} SOS(p)$. Moreover, by induction on the length of $\{M_i(p)\}$, we have by definition that $M_i(p)$ has all the paths of length less or equal to i.*

As a consequence, using $\{M_i(p)\}$ we can decide any finite property of depth n of a term p: it sufficies to check the property on $M_n(p)$.

4.2 SS Approximations

In this section, we present a way of approximating $SOS(p)$ based on a different operational semantics, which allows us to prove a greater set of properties than those proved by $\{M_i(p)\}$. In [9] 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 figure 6, 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*. The SS inference rules accomplish these strong equivalence preserving simplifications during the derivation. The following notation is used in the rules:

$p\backslash\backslash A =$	$p\backslash A$, 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 $p = q[g]$

S-Act=Act S-Con=Con S-Sum=Sum

S-Par$_1$ $\dfrac{p \xrightarrow{\mu} p', \textit{not } p' \not\rightarrow}{p|q \xrightarrow{\mu} p'|q \text{ and } q|p \xrightarrow{\mu} q|p'}$ **S-Par$_2$** $\dfrac{p \xrightarrow{\mu} p',\ p' \not\rightarrow}{q|p \xrightarrow{\mu} q \text{ and } q|p \xrightarrow{\mu} q}$

S-Com$_1$ $\dfrac{p \xrightarrow{\alpha} p',\ q \xrightarrow{\overline{\alpha}} q',\ \textit{not } p' \not\rightarrow \ \textit{and not } q' \not\rightarrow}{p|q \xrightarrow{\tau} p'|q'}$ **S-Com$_2$** $\dfrac{p \xrightarrow{\alpha} p',\ q \xrightarrow{\overline{\alpha}} q',\ p' \not\rightarrow}{p|q \xrightarrow{\tau} q' \text{ and } q|p \xrightarrow{\tau} q'}$

S-Res $\dfrac{p \xrightarrow{\mu} p', \mu, \overline{\mu} \notin A}{p\backslash A \xrightarrow{\mu} q\backslash\backslash A}$ **S-Rel** $\dfrac{p \xrightarrow{\mu} q}{p[f] \xrightarrow{f(\mu)} q[[f]]}$

Figure 6: The SS rules

The chain $\{N_i(p)\}$ is defined in a similar way to $\{M_i(p)\}$, but using the above rules:

Definition 4.2 ($\{N_i(p)\}$) *Given a term p, the chain $\{N_i(p) = (S_{N_i}, Act, D_{N_i}, s_0)\}$ is inductively defined as follows:*

- $N_0(p) = (\{p\}, Act, \{\}, p)$

- $S_{N_{i+1}}(p) = (S_{N_{i+1}}, Act, D_{N_{i+1}}, p)$ *where*
  - $S_{N_{i+1}} = S_{N_i} \cup \{q | p \in S_{N_i}$ *and* $\mu \in Act$ *exist such that* $p \xrightarrow{\mu}_{SS} q\}$;
  - $D_{N_{i+1}} = D_{N_i} \cup \{(p, \mu, q) | p \in S_{N_i}$ *and* $\mu \in Act$ *exist such that* $p \xrightarrow{\mu}_{SS} q\}$.

If we reconsider the bag example, Figures 7, 8 show $N_1(X)$ and $N_2(X)$, respectively.

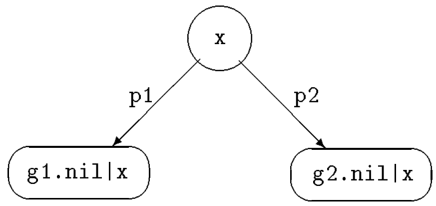

Figure 7: $N_1(X)$

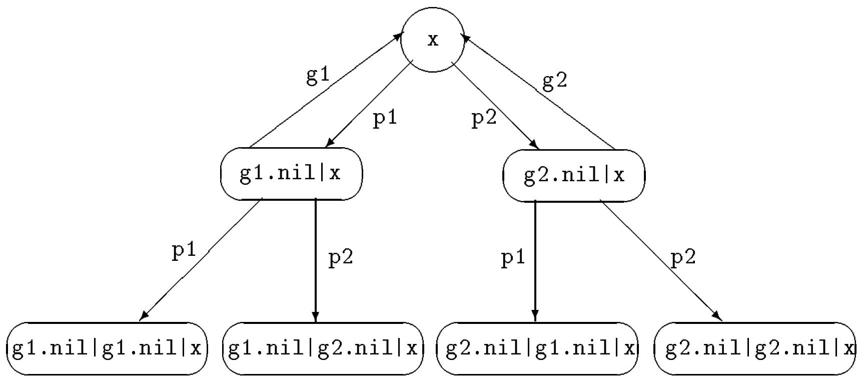

Figure 8: $N_2(X)$

The following proposition holds:

Proposition 4.3 *Given a term p,*

- *the chain* $\{N_i(p)\}$ *is a* $\preceq_{bc}$*-approximation chain for p;*
- *for each* $i \geq 0$, $N_i(p) \preceq_i SOS(p)$

Proof sketch *Analogous to the proof of proposition 4.1 and 4.2 and since* $SOS(p) \sim SS(p)$.

If we check the properties 1) ... 4) on the chain $\{N_i(p)\}$, we have the same results as with $\{M_i(p)\}$ for 1) and 4), but $N_2(X)$ satisfies properties 2) and 3), which are then true for the bag, while their validity is not provable on the chain $\{M_i(X)\}$.

The following proposition relates the two chains we have introduced.

Proposition 4.4 *Given a CCS term p, for each* $i \geq 0$, $\exists j$ *such that* $M_i(p) \preceq_{bc} N_j(p)$.
Proof. *The finite paths are equal in* $M_i(p)$ *and* $N_i(p)$, *since they are both* $\preceq_i SOS(p)$. *Moreover,*

it holds that: $\forall s \in S_{M_i}, \exists s' \in S_{N_i}$ such that $s \sim s'$ and $length(s') \leq lengh(s)$, where $length(t)$ denotes the number of operators occurring in the term t. This holds since terms generated by SS are "shorter" than terms generated by SOS. Consider an infinite path in $M_i(p)$, i.e. a path leading from a state $s \in S_{M_i}$ to itself and take n equal to the number of terms t equivalent to s and such that $length(t) \leq lengh(s)$. Take $j = i + n$.

Note that the converse of the above proposition is not true: if we consider the bag example, no $M_i(X)$ exists which is $\preceq_{bc} N_2(X)$.

5 Suitability of approximation chains

Let us consider a liveness property ϕ and a $\preceq_{bc}$-approximation chain $\{T_i(p)\}$ for a term p. Proposition 3.4 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 3.4 is not true in general: if a liveness property ϕ is verified on $SOS(p)$, this does not imply that it is true for some $\{T_i(p)\}$. 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 for the validity of a formula. This is the case of the chain $\{M_i(p)\}$ and the properties 2) and 3) of our example above. 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 the validity ϕ with another chain. This suggests a comparison criterion on the suitability of approximation chains for proving liveness properties.

Definition 5.1 (Checkable properties) *Let be given a term p and a $\preceq_{bc}$ approximation chain $\{T_i(p)\}$. We say that a liveness property ϕ is checkable by $\{T_i(p)\}$ if*

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

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

Thus $\mathcal{P}_{T_i}(p)$ includes the properties for whose validity there is a semidecision procedure using $\{T_i(p)\}$.

Definition 5.2 (Suitability of approximation chains) *Let be given a term p and two $\preceq_{bc}$ approximations chains $\{T_i(p)\}$ and $\{S_i(p)\}$. We say that $\{T_i(p)\}$ is* more suitable or equal for p *than $\{S_i(p)\}$ if $\mathcal{P}_{S_i}(p) \subseteq \mathcal{P}_{T_i}(p)$. Moreover, $\{T_i(p)\}$ is* strictly more suitable for p *than $\{S_i(p)\}$ 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 $\{T_i(p)\}$, let us consider the chain containing a subset of the elements of $\{T_i(p)\}$, for example the elements of even position, i.e. $\{S_i(p)\} = \{T_0(p), T_2(p), T_4(p), \cdots\}$. We have that $\{S_i(p)\}$ grows faster than $\{T_i(p)\}$, but it is not more suitable. As a consequence of the above definitions and propositions 4.4, we can state the following propositions:

Proposition 5.1 *For each term p, $\mathcal{P}_{M_i}(p) \subseteq \mathcal{P}_{N_i}(p)$.*
Proof sketch. *By proposition 4.4.*

The following proposition states that the converse of proposition 5.1 is not true in general.

Proposition 5.2 *Given a term* p*,* $\mathcal{P}_{M_i}(p) \subset \mathcal{P}_{N_i}(p)$*, i.e.* $\{N_i(p)\}$ *is strictly more suitable than* $\{M_i(p)\}$*.*
Proof sketch *Properties 2) and 3) in the bag example are checkable by* $\{N_i(p)\}$ *but not by* $\{M_i(p)\}$*.*

6 Implementation in the JACK environment

The JACK system [3] 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 [18]. The ACTL model checker was built on the basis of an algorithm similar to that of the EMC model checker [5], so it guarantees model checking of an ACTL formula on a TS in a linear time complexity [10].

The JACK environment has been extended with a tool to build the chain $\{N_i(p)\}$. We now describe the methodology for proving properties. 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 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 $N_1(p)$, we put $TS := N_1(p)$.

3. The list of ACTL formulae is input to the model checker which checks them on TS. If $N_{i+1}(p) = TS$, the session terminates, since $TS \sim SOS(p)$. Otherwise, the results of the model checker are analyzed according to propositions 3.4, 3.6 and 3.8. This means that, possibly, a new approximation is built, i.e. $TS := N_{i+1}(p)$ and we repeat step 3.

Acknowledgement

We wish to acknowledge Luigi Polverini for its work on the implementation of the NSS approximation generator, Rocco De Nicola and Gioia Ristori for interesting discussion about the topics of this paper.

References

[1] 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.

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

[3] A. Bouali, S. Gnesi, S. Larosa. The integration Project for the JACK Environment. Bulletin of the EATCS, n.54, October 1994, pp.207-223.

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

[5] 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.

[6] E.M.Clarke, O.Grumberg, D.E.Long. Model Checking and Abstraction. ACM Toplas, 16 (5), 1994, pp.1512-1542.

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

[8] 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.

[9] N. De Francesco, P. Inverardi. Proving Finiteness of CCS Processes by Non-standard Semantics. Acta Informatica, 31 (1), 1994, pp. 55-80.

[10] 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.

[11] R. De Nicola, F. W. Vaandrager. Action versus State based Logics for Transition Systems. Proceedings Ecole de Printemps on Semantics of Concurrency. LNCS 469, 1990, pp. 407-419.

[12] 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.

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

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

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

[16] 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.

[17] E. Kindler. Safety and Liveness Properties: A Survey. Bulletin of the EATCS, 53, 1994, pp.268-272.

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

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

[20] 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.

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

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