Testing Semantics for Unbounded Nondeterminism*

Luis Fdo. Llana Díaz and Manuel Núñez

Dpto. Sistemas Informáticos y Programación
Universidad Complutense de Madrid.
Ciudad Universitaria. 28040 Madrid. Spain.
e-mail: {llana,manuelnu}@eucmax.sim.ucm.es

Abstract We present an extension of the classical testing semantics for the case when nondeterminism is unbounded. We define the corresponding *may* and *must* preorders in the new framework. As in the bounded setting the *may* preorder can be characterized by using the set of finite traces of processes. On the contrary, in order to characterize the *must* preorder is necessary to record some additional information about the infinite behavior of processes. This characterization will be an extension of *acceptance sets*, considering not only the finite traces a process can execute but also its *infinite* traces.
Keywords: Process algebra, CSP, unbounded nondeterminism, operational semantics, testing semantics.

1 Introduction

Over the last years process algebras have been widely studied. Most models are restricted to consider *bounded* nondeterminism. This leads to some restrictions on the language, which are necessary for some algebraic rules to be true.

In [RB89,Bar91,Ros93,Mis95] extensions of the failures model coping with unbounded nondeterminism have been presented. The main problem, in order to give a denotational semantics to a language supporting unbounded nondeterminism, is the lack of continuity in the model. In fact, fixpoints may be reached, in general, after more than ω iterations.

We present a testing semantics for a language supporting unbounded nondeterminism. Besides, we define an operational characterization of this testing semantics which is based on an extension of acceptance sets [Hen88]. The lack of continuity in the denotational semantics of the models cited above is also reflected in our model. Actually, we need infinite tests to fully characterize the semantics of processes, while, as it is well known, only finite tests are necessary in classical testing semantics.

2 Syntax and Operational Semantics

In this section we describe the syntax and operational semantics of our language. In order to concentrate on the main characteristics and problems of unbounded nondeterminism, we introduce a simple process algebra based on CSP [Hoa85] which, however, contains the main operators characterizing such an algebra. We consider

* Research partially support by the CYCIT project TIC 94-0851-C02-02.

an infinite set of *visible* actions Act, an *invisible* action $\tau \notin Act$, and then we consider the set of actions $Act_\tau = Act \cup \{\tau\}$. As usual, the syntax of the language can be described by means of a BNF expression:

$$P ::= x \mid \mathtt{STOP} \mid \mathtt{DIV} \mid \mathrm{rec}\, x.P \mid P \,\Box\, Q \mid \tau \to P \mid P \parallel_A Q \mid P \setminus A \mid P[\phi] \mid$$
$$a : A \to g(a) \mid \prod_{i \in I} P_i$$

where $A \subseteq Act$, g is a function from A to processes, I is a non empty set of indices, ϕ is a renaming function ($\phi : Act \mapsto Act$), and x ranges over the set VAR of process variables. We denote by Term(VAR) the class of syntactic terms defined by this BNF expression, and by Proc the set of closed terms.

When a syntax contains infinitary operators, the idea of *what* is defined by this syntax is not obvious, thus, we will briefly discuss it here. BNF expressions are regarded as fixpoint equations defining the smallest class of terms closed under the various operations appearing at the right hand side of the equation. If the syntax does not contain infinitary operators, then this fixpoint is reached after (at most) ω iterations (every term is *born at a finite time*). But, for instance, in our language we have the term

$$\prod_{n \in \mathbb{N}} P_n$$

where each P_n is born at time n, and so P is born at time $\omega+1$. Now, let us consider the term

$$\prod_{P \in \mathsf{Term(VAR)}} P$$

Should we allow it as a valid term? The answer must be negative because if we would consider it as a valid term, then it is easy to prove that the set of terms is as *big* as its power set. So, to fully formalize the set of valid expressions, we begin by bounding the size of the possible sets of indices I, and that of the set of actions Act by some infinite cardinal κ. The functional governing the right hand side of the equation is clearly monotone, but it is not so obvious whether it has any fixpoint. Fortunately it has. Besides, it is guaranteed that it is reached after (at most) λ iterations, where λ is the smallest regular cardinal bigger than κ. Then, the principle of structural induction is valid and corresponds to the principle of transfinite induction.

In order to simplify the notation we will use $a \to P$ as a shorthand of $a : A \to g(a)$ when $A = \{a\}$ and $g(a) = P$.

The operational semantics of a process P is given as a labeled transition system $\langle \mathsf{Proc}, \longrightarrow, P \rangle$ where $\longrightarrow \subseteq \mathsf{Proc} \times Act_\tau \times \mathsf{Proc}$. As usual, we write $P \xrightarrow{e} Q$ instead of $(P, e, Q) \in \longrightarrow$. Transitions are generated by using the usual "natural deduction" style presented in Table 1. A transition of the form $P \xrightarrow{a} Q$ is called *visible* if $a \in Act$, and *invisible* otherwise (i.e. if $a = \tau$). We will use the convention that $P \xrightarrow{a}$ stands for $\exists P' : P \xrightarrow{a} P'$.

3 Testing Semantics

In this section we define a testing semantics for our language. First, we define the *convergence* predicate, denoted by $P \Downarrow$, and its dual predicate, the *divergence* predicate, denoted by $P \Uparrow$. Intuitively, $P \Uparrow$ if an infinite sequence of internal actions can be executed from P. In order to define these predicates we also need an auxiliary predicate called the *weak convergence* predicate.

$$\mathtt{DIV} \xrightarrow{\tau} \mathtt{DIV} \qquad \mathrm{rec}\,x.P \xrightarrow{\tau} P[\mathrm{rec}\,x.P/x]$$

$$b : A \to g(a) \xrightarrow{a} g(a), \quad a \in A \qquad \tau \to P \xrightarrow{\tau} P$$

$$\prod_{i \in I} P_i \xrightarrow{\tau} P_j, \quad j \in I$$

$$\frac{P \xrightarrow{\tau} P'}{P \,\Box\, Q \xrightarrow{\tau} P' \,\Box\, Q, \quad Q \,\Box\, P \xrightarrow{\tau} Q \,\Box\, P'} \qquad \frac{P \xrightarrow{a} P'}{P \,\Box\, Q \xrightarrow{a} P', \quad Q \,\Box\, P \xrightarrow{a} P'}$$

$$\frac{P \xrightarrow{e} P'}{P \parallel_A Q \xrightarrow{e} P' \parallel_A Q, \quad Q \parallel_A P \xrightarrow{e} Q \parallel_A P'}, \quad e \notin A \qquad \frac{P \xrightarrow{a} P', \quad Q \xrightarrow{a} Q'}{P \parallel_A Q \xrightarrow{a} P' \parallel_A Q'}, \quad a \in A$$

$$\frac{P \xrightarrow{e} P'}{P \setminus A \xrightarrow{e} P' \setminus A}, \quad e \notin A \qquad \frac{P \xrightarrow{a} P'}{P \setminus A \xrightarrow{\tau} P' \setminus A}, \quad a \in A$$

$$\frac{P \xrightarrow{a} P'}{P[\phi] \xrightarrow{\phi(a)} P'[\phi]} \qquad \frac{P \xrightarrow{\tau} P'}{P[\phi] \xrightarrow{\tau} P'[\phi]}$$

Table1. Rules of the Operational Semantics.

Definition 1. We define the *weak convergence* predicate over terms, denoted by $P \downarrow$, as the least predicate satisfying:

- $\mathtt{STOP} \downarrow$, $a : B \to g(a) \downarrow$, and $\tau \to P \downarrow$.
- If $P \downarrow$ and $Q \downarrow$ then $P \,\Box\, Q \downarrow, P \parallel_B Q \downarrow, P \setminus B \downarrow$ and $P[\phi] \downarrow$.
- If for all $i \in I$ $P_i \downarrow$ then $\prod_{i \in I} P_i \downarrow$.
- If $P \downarrow$ then $\mathrm{rec}\,x.P \downarrow$.

Finally, we define the *convergence* predicate as

$$P \Downarrow \iff P \downarrow \wedge \left(\forall P' : P \xrightarrow{\tau} P' \Rightarrow P' \Downarrow\right)$$

and we write $P \Uparrow$ if $P \Downarrow$ does not hold.

Tests are just processes where the syntax has been extended adding a new process, OK, which expresses that the test has been (successfully) passed. Thus, they are defined by the following BNF expression:

$$T ::= \mathtt{STOP} \mid \mathtt{OK} \mid x \mid \mathrm{rec}\,x.T \mid T_1 \,\Box\, T_2 \mid \tau \to T \mid b : B \to g(b) \mid \prod_{i \in I} T_i \mid T_1 \parallel_B T_2 \mid T \setminus B$$

The same comments we did for the syntax of processes are valid here, and the set of tests is that of closed terms generated by the grammar above. The operational semantics of tests is defined in the same way as for plain processes, but including a new rule for the test OK[1]:

$$\mathtt{OK} \xrightarrow{\mathtt{OK}} \mathtt{STOP}$$

Finally, we define the composition of a test and a process as

$$P \mid T = (P \parallel_{Act} T) \setminus Act$$

[1] To be exact, we should extend the definition of the operational semantics of both processes and tests to mixed terms defining their compositions. Formally, these mixed terms are neither processes nor tests, but since this extension is immediate we have preferred to avoid this formal definition.

Definition 2. Let P be a process and T be a test. For any computation of $P \mid T$

$$P \mid T = P_1 \mid T_1 \xrightarrow{\tau} P_2 \mid T_2 \cdots P_k \mid T_k \xrightarrow{\tau} P_{k+1} \mid T_{k+1} \cdots$$

we say that it is *complete* if it is finite and blocked (no more steps are allowed) or infinite, and we say that it is *successful* if there exists some k such that $T_k \xrightarrow{\mathtt{OK}}$ and for any $j < k$ $P_j \Downarrow$ holds.

We say P *may pass* the test T, denoted by P may T, iff there exists a successful computation of $P \mid T$. We say P *must pass* the test T, denoted by P must T, iff any complete computation of $P \mid T$ is successful.

Now, we can define the usual testing preorders relating processes.

Definition 3. Let P and Q be processes. We write:

- $P \sqsubseteq_{\text{may}} Q$ iff for any test T, P may T implies Q may T.
- $P \sqsubseteq_{\text{must}} Q$ iff for any test T, P must T implies Q must T.

4 Alternative Characterization

Although we have given two relations between processes whose definitions are very simple, it is rather complicated to find out whether two processes are related by $\sqsubseteq_{\text{may}}$ or $\sqsubseteq_{\text{must}}$, so it would be very desirable to provide alternative characterizations of these testing preorders. In [Hen88] alternative characterizations are given in terms of (finite) traces and in terms of acceptances sets, respectively. We will show that, as in the bounded case, the *may* preorder can be characterized just by using finite traces, but in order to provide an alternative characterization for the *must* preorder we need some additional information. The extra information we need will be the set of *infinite* traces of a process, where an infinite trace is just an element $u = \langle a_k \mid k \in \mathbb{N} \rangle \in Act^\omega$.

First we extend the transition relation to deal with sequences of visible actions. As usual, $\xrightarrow{\tau}{}^*$ stands for the reflexive and transitive closure of $\xrightarrow{\tau}$.

Definition 4. Let P and Q be processes, $s \in Act^*$, and $u \in Act^\omega$.

We write $P \stackrel{s}{\Longrightarrow} Q$ if there exists a finite computation

$$P = P_0 \xrightarrow{\tau}{}^* P'_0 \xrightarrow{a_1} P_1 \xrightarrow{\tau}{}^* P'_1 \xrightarrow{a_2} P_2 \cdots \xrightarrow{a_n} P_n \xrightarrow{\tau}{}^* P'_n = Q$$

such that $s = a_1 a_2 \cdots a_n$ (if $n = 0$ the transition is denoted by $P \stackrel{\epsilon}{\Longrightarrow} Q$).

We write $P \stackrel{u}{\Longrightarrow}$ if there exists an infinite computation

$$P = P_0 \xrightarrow{\tau}{}^* P'_0 \xrightarrow{a_1} P_1 \xrightarrow{\tau}{}^* P'_1 \xrightarrow{a_2} \cdots P_{k-1} \xrightarrow{\tau}{}^* P'_{k-1} \xrightarrow{a_k} P_k \cdots$$

such that $u = a_1 a_2 \cdots a_k \cdots$.

As in the bounded case, a *state* represents the set of actions that a process is ready to execute; more exactly, a state A is a (possibly empty) set of visible actions or an element Ω representing an undefined state (i.e. the state of a divergent process), that is $A \in \mathcal{P}(Act) \cup \{\Omega\}$. We will denote the set of states by $\mathcal{ST}$. An *acceptance* will represent any reached state A of a process after the execution of a sequence of actions s; that is, an element $(s, A) \in Act^* \times \mathcal{ST}$, but we will write sA instead of (s, A).

Definition 5. Let P be a process.

- We define the *finite traces* of P as $\mathsf{fins}(P) = \{s \mid \exists P' : P \stackrel{s}{\Longrightarrow} P' \wedge s \in Act^*\}$, and the *infinite traces* of P as $\mathsf{infs}(P) = \{u \mid P \stackrel{u}{\Longrightarrow} \wedge u \in Act^\omega\}$.
- We define the *state* of P, denoted by $\mathsf{S}(P)$, as
$$\mathsf{S}(P) = \begin{cases} \{a \in Act \mid P \stackrel{a}{\Longrightarrow}\} & \text{if } P \Downarrow \\ \Omega & \text{if } P \Uparrow \end{cases}$$
- We define the *acceptances set* of P as
$$\mathcal{A}(P) = \{sA \mid \exists P' : P \stackrel{s}{\Longrightarrow} P' \wedge S(P') = A\}$$

Definition 6. We define the relation $\ll$ between acceptances as the least relation satisfying:

- If $A, A' \in \mathcal{P}(Act)$ are such that $A \subseteq A'$ then $\epsilon A \ll \epsilon A'$.
- If sA is an acceptance then $\epsilon\,\Omega \ll sA$.
- If $a \in Act$ and $s'A' \ll sA$ then $(as')A' \ll (as)A$.

It could be helpful to mention that $sA \ll s'A'$ iff either $s = s'$ and $A \subseteq A' \subseteq Act$, or s is a prefix of s' and $A = \Omega$. The previous relation can be extended to acceptances sets as follows

$$\mathcal{A}_1 \ll \mathcal{A}_2 \iff \forall s_2A_2 \in \mathcal{A}_2\ \exists s_1A_1 \in \mathcal{A}_1 : s_1A_1 \ll s_2A_2$$

As in the bounded framework, the may testing preorder can be characterized by the set of *finite* traces of processes.

Theorem 7. *Let P and Q be processes. Then, $P \sqsubseteq_{may} Q \iff \mathsf{fins}(P) \subseteq \mathsf{fins}(Q)$.*
Proof. Similar to that in [Hen88].

So, infinite traces do not play any role in order to characterize the may preorder. We will show that this is not the case for the must preorder.

In [Hen88], where the nondeterminism is bounded, we have $P \sqsubseteq_{\text{must}} Q$ iff $\mathcal{A}(P) \ll \mathcal{A}(Q)$, but this is no longer true in our setting, as the following example shows

Example 8. Let us consider the family of processes $\{P_n\}_{n\in\mathbb{N}}$ defined as
$$P_1 = \mathtt{STOP}, \quad P_{n+1} = a \to P_n$$
and the processes $P = \prod_{i\in\mathbb{N}} P_i$ and $Q = (\mathrm{rec}\, x.a \to x) \sqcap P$. It is easy to show that $\mathcal{A}(P) = \mathcal{A}(Q)$, but if we consider the test $T = \mathrm{rec}\, x.((a \to x) \,\Box\, (\tau \to \mathtt{OK}))$, we have P must T, while Q must T because of the infinite computation

$$\begin{aligned} Q \parallel_{Act} T &\xrightarrow{\tau} (\mathrm{rec}\, x.a \to x) \parallel_{Act} T \\ &\xrightarrow{\tau} (\mathrm{rec}\, x.a \to x) \parallel_{Act} ((a \to \mathrm{rec}\, x.((a \to x) \,\Box\, (\tau \to \mathtt{OK}))) \,\Box\, (\tau \to \mathtt{OK})) \\ &\xrightarrow{\tau} (a \to (\mathrm{rec}\, x.a \to x)) \parallel_{Act} ((a \to \mathrm{rec}\, x.((a \to x) \,\Box\, (\tau \to \mathtt{OK}))) \,\Box\, (\tau \to \mathtt{OK})) \\ &\xrightarrow{a} (\mathrm{rec}\, x.a \to x) \parallel_{Act} T \\ &\ldots \end{aligned}$$

Thus, the above characterization does not hold. □

The rest of this section is devoted to give the desired characterization of the must testing preorder. Looking at the previous example, we observe that the only difference between P and Q is that Q has the infinite trace of a's, while P has not this infinite trace. When nondeterminism is bounded, infinite traces are determined

by their finite prefixes; that is, a process has an infinite trace iff it has all its finite prefixes. As the previous example shows this does not hold any more when dealing with unbounded nondeterminism: P has as traces all the finite prefixes of the infinite trace $u = aaaaa\cdots$, but u is not an infinite trace of P. So, infinite traces seem to be the key in order to define the adequate alternative characterization.

Definition 9. Let P and Q be processes. We write $P \ll_{unb} Q$ iff the following conditions hold:

- $\mathcal{A}(P) \ll \mathcal{A}(Q)$, and
- For all $u \in \mathsf{infs}(Q)$ either $u \in \mathsf{infs}(P)$ or there exists a finite prefix s of u such that $s\Omega \in \mathcal{A}(P)$.

Theorem 10. *Let P and Q be processes. Then, $P \ll_{unb} Q \Longrightarrow P \sqsubseteq_{must} Q$.*

Proof. Let us consider a test T such that P must T. We have to prove that any complete computation of $Q \mid T$ is successful. Any computation of $Q \mid T$ of the form

$$Q \mid T = Q_1 \mid T_1 \xrightarrow{\tau} Q_2 \mid T_2 \cdots Q_{k-1} \mid T_{k-1} \xrightarrow{\tau} Q_k \mid T_k \cdots$$

may be unzipped into a computation of Q and a computation of T:

$$Q \stackrel{s}{\Longrightarrow} Q_k \qquad \wedge \qquad T \stackrel{s}{\Longrightarrow} T_k$$

If this computation is finite, then there exists a *last* process Q_k in it, i.e. $Q \stackrel{s}{\Longrightarrow} Q_k$. We can suppose that $Q_i \Downarrow$ for all $i < k$, because if there exists $i < k$ such that $Q_i \Uparrow$, then we could take Q_i as the last process. So, we get an acceptance $sA \in \mathcal{A}(Q)$ such that $Q \stackrel{s}{\Longrightarrow} Q_k$ and $A = \mathsf{S}(Q_k)$. Given that $P \ll_{unb} Q$, there must exist an acceptance $s'A' \in \mathcal{A}(P)$ such that $s'A' \ll sA$. So, there exists a process P' such that $P \stackrel{s'}{\Longrightarrow} P'$ and $A' = \mathsf{S}(P')$. By the definition of $\ll_{unb}$ we have that s' is a prefix of s, and so the computation of P can be joined with the corresponding computation of T, and, since P must T, we have that the computation of $P \mid T$ is successful, and then so is the one of $Q \mid T$.

Now let us suppose that the computation is not finite. If there exists a process Q_k such that $Q_k \Uparrow$, we could take the computation as finite. Otherwise, we have two possibilities:

- $\exists k \in \mathbb{N}\, \forall l \geq k :\ Q_l \xrightarrow{\tau} Q_{l+1}$. Then, there exists a trace s such that $Q \stackrel{s}{\Longrightarrow} Q_k$ and $Q_k \Uparrow$, and we can construct the successful computation as in the previous case.
- $\forall k \in \mathbb{N}\, \exists l_k \geq k :\ Q_{l_k} \xrightarrow{a_k} Q_{l_k+1} \wedge a_k \in Act$. If we consider the infinite trace $u = \langle a_k \mid k \in \mathbb{N} \rangle$, then we obtain $u \in \mathsf{infs}(Q)$. By the definition of $\ll_{unb}$ we have either $u \in \mathsf{infs}(P)$ or there exists a finite prefix s of u such that $s\Omega \in \mathcal{A}(P)$. If $u \in \mathsf{infs}(P)$ then we have $P \stackrel{u}{\Longrightarrow}$; in the second case there exists some P' such that $P \stackrel{s}{\Longrightarrow} P'$. In both cases, the computation of P can be joined with the corresponding computation of T, and, since P must T, we have that the computation of $P \mid T$ is successful, and then so is the one of $Q \mid T$.

Before proving the converse of the previous theorem, we will define the so called family of *basic* tests:

Definition 11.

- Let s be a finite trace and $A \subseteq Act$. We inductively define the tests $T(s, A)$ as
  - $T(\epsilon, \varnothing) = \mathtt{STOP}$.

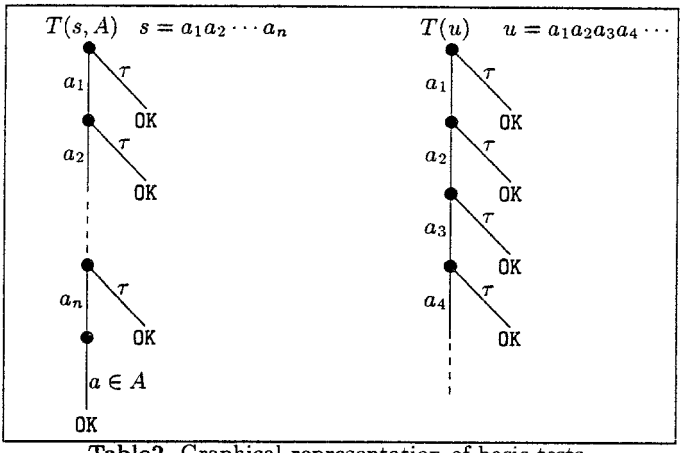

Table2. Graphical representation of basic tests.

- If $A \neq \varnothing$ then $T(\epsilon, A) = a : A \to g(a)$, where $g(a) = \mathtt{OK}$ for all $a \in A$.
- $T(a \cdot s', A) = (\tau \to \mathtt{OK}) \,\Box\, (a \to T(s', A))$.

– Let $u = \langle a_k \mid k \in \mathbb{N} \rangle$ be an infinite trace. Let $B = \{b_i \mid i \in \mathbb{N}\} \subseteq Act$ be a numerable infinite set of actions (note that we can get such a set since Act is infinite). Let us consider the renaming functions ϕ_B^1 and ϕ_B^2 defined as

$$\phi_B^1(c) = \begin{cases} b_{i+1} \text{ if } c = b_i \in B \\ c \quad \text{if } c \notin B \end{cases} \qquad \phi_B^2(c) = \begin{cases} a_i \text{ if } c = b_i \in B \\ c \text{ if } c \notin B \end{cases}$$

Then, we define the test $T(u)$ taking

$$T(u) = \Big(\mathrm{rec}\, x.(\tau \to \mathtt{OK}) \,\Box\, (b_1 \to x[\phi_B^1])\Big)[\phi_B^2]$$

In Table 2 we show a graphical representation of the tests defined before[2]. While the meaning of finite tests is clear, it is not so evident why we have defined the tests $T(u)$ in such a way. The idea is that we want a syntactically finite test offering the infinite sequence $a_1, a_2, \ldots, a_n, \ldots$ The first attempt to get such a test would be to consider a renaming function such that $\phi(a_i) = a_{i+1}$. But this does not work because it could exist $i \neq j$ with $a_i = a_j$ but $a_{i+1} \neq a_{j+1}$. Then, it is necessary to consider an infinite set of actions playing the same role. First, we define a renaming function such that $\phi_B^1(b_i) = b_{i+1}$. Then, a renaming function such that $\phi_B^2(b_i) = a_i$ works as desired. Finally, note that $\mathsf{infs}(T(u)) = \{u\}$.

The next result follows easily from the previous definition.

Lemma 12. *Let P be a process, s be a trace, $A \subseteq Act$, and u be an infinite trace. Then, the following conditions hold:*

– P *must* $T(s, A) \iff \forall\, s'A' \in \mathcal{A}(P) : \begin{cases} s' = s \wedge A' \cap A \neq \varnothing \wedge A' \neq \Omega \\ \quad \vee \\ s'A' \not\ll s\Omega \end{cases}$

[2] Formally, these are not exactly our tests because after the action τ is performed the corresponding action a_i should be available (composed in external choice with the process $\mathtt{OK}$). But if the test executes any τ, then the test is successfully passed, so we can omit this detail in the graphical representation.

– P *must* $T(u) \iff u \notin \mathsf{infs}(P) \wedge \nexists s', u' : u = s'u' \wedge s'\Omega \in \mathcal{A}(P)$

Theorem 13. *Let P and Q be processes such that $P \not\ll_{unb} Q$. Then, there exists a test T such that P must T but Q must T.*

Proof. (Sketch) If $P \not\ll_{unb} Q$ then there are two possible cases:

– There exists some $sA \in \mathcal{A}(Q)$ such that there is no $s'A' \in \mathcal{A}(P)$ with $s'A' \ll sA$. Then we consider the set of actions

$$A'' = \bigcup_{sA' \in \mathcal{A}(P)} A' \setminus A$$

Note that, because of the previous assumptions, if $sA' \in \mathcal{A}(P)$ then $A' \neq \Omega$. From this set of actions we can consider the associated test $T(s, A'')$, and we obtain P must $T(s, A'')$ while Q must $T(s, A'')$.

– There exists some $u \in \mathsf{infs}(Q)$ such that neither $u \in \mathsf{infs}(P)$ nor there exists any finite prefix s of u with $s\Omega \in \mathcal{A}(P)$. Then, by the previous lemma we have P must $T(u)$ and Q must $T(u)$.

As an immediate corollary of this theorem we have that the family of tests given by Definition 11 constitutes a set of *essential* tests. By essential we mean that in order to show that two processes are not equivalent, it is enough to find a test from this family that distinguishes the processes; that is, a test such that one of the processes must pass it while the other one does not.

5 Conclusions and future Work

We have extended the classical testing semantics developed in [Hen88] to a framework where internal choice is not restricted to a finite number of arguments. First, we extended CSP with an internal choice operator having an arbitrary number of arguments (in particular, an infinite number of arguments). We have defined the corresponding *may* and *must* testing semantics for the new framework. Then, we have given alternative characterizations of these preorders: in the may case, everything remains the same as in the case of bounded nondeterminism, but in order to characterize the must preorder we need to extend the usual acceptances of processes with infinite traces.

As future work we plan to define fully abstract denotational semantics of the must testing preorders by using appropriate extensions of acceptance trees [Hen85].

References

[Bar91] G. Barret. The fixed point theory of unbounded non-determinism. *Formal Aspects of Computing*, 3:110–128, 1991.

[Hen85] M. Hennessy. Acceptance trees. *Journal of the ACM*, 32(4):896–928, 1985.

[Hen88] M. Hennessy. *Algebraic Theory of Processes.* MIT Press, 1988.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice Hall, 1985.

[Mis95] M.W. Mislove. Denotational models for unbounded nondeterminism. In *Proceedings of 11th Mathematical Foundations of Programming Semantics.* Electronic Notes in Theoretical Computer Science 1, 1995.

[RB89] A. W. Roscoe and G. Barret. Unbounded non-determinism in CSP. In *Mathematical Foundations of Programming Semantics, LCNS 442*, 1989.

[Ros93] A.W. Roscoe. Unbounded non-determinism in CSP. *Journal of Logic and Computation*, 3(2):131–172, 1993.