Testing Theories for Asynchronous Languages *

Ilaria Castellani[1] and Matthew Hennessy[2]

[1] INRIA, BP 93, 06902 Sophia-Antipolis Cedex, France
[2] COGS, University of Sussex, Brighton BN1 9QH, UK

Abstract. We study testing preorders for an asynchronous version of *CCS* called *TACCS*, where message emission is non blocking. We first give a labelled transition system semantics for this language, which includes both external and internal choice operators. By applying the standard definitions of may and must testing to this semantics we obtain two behavioural preorders based on asynchronous observations, $\underset{\sim}{\sqsubset}_{may}$ and $\underset{\sim}{\sqsubset}_{must}$. We present alternative behavioural characterisations of these preorders, which are subsequently used to obtain equational theories for the finite fragment of the language.

1 Introduction

The asynchronous π-calculus [5, 9] is a very simple but powerful formalism for describing distributed and mobile processes, based on the asynchronous exchange of names. Its descriptive power as a programming calculus has been demonstrated theoretically in papers such as [5, 9, 10, 13] and practically in [14], where a minor variant takes the role of the target implementation language for the sophisticated distributed and higher-order programming language *Pict*.

However a calculus should not only be computationally expressive. The most successful process calculi, such as *CCS* [12], on which the asynchronous π-calculus is based, *CSP* [8] and *ACP* [2], also come equipped with a powerful equational theory with which one may reason about the behaviour of processes. For the asynchronous π-calculus, much work still remains to be done in developing suitable equational theories; to our knowledge the only proposed axiomatisations concern a variation of strong bisimulation equivalence called *strong asynchronous bisimulation*[1] and an asynchronous version of *may testing* [4].

A major attraction of the asynchronous π-calculus is its simplicity. In spite of its expressive power, it consists of very few process combinators: blocking input and non blocking output on a named channel, parallelism, recursion, and the powerful scoping mechanism for channels from the standard π-calculus. But operators which may be of limited attraction from a computational point of view can play a significant role in an equational theory. The choice operator $+$ of *CCS* is a typical example. In this paper we demonstrate that such operators can also help in providing interesting behavioural theories for asynchronous languages.

* Research partially funded by the EU Working Group CONFER II and the EU HCM Network EXPRESS. The second author was also supported by the EPSRC project GR/K60701.

We examine a simplification of the asynchronous π-calculus, an asynchronous variant of *CCS* equipped with internal and external choice operators, which we call *TACCS* ; essentially an asynchronous version of the language studied in [7]. In Section 2 we define an operational semantics for *TACCS* . In the following section we apply the standard definitions of *testing* to this operational semantics to obtain two preorders on asynchronous processes [6, 7], the *may* testing preorder $\sqsubseteq_{may}$ and the *must* testing preorder $\sqsubseteq_{must}$. The main result of this section is an alternative characterisation of these contextual preorders, defined in terms of properties of the operational semantics of processes. These characterisations are similar in style to those for the corresponding preorders on the synchronous version of the language, studied in [7], but at least for the *must* case, are considerably more subtle. In Section 4 we present our main result, namely an equational characterisation of the preorder $\sqsubseteq_{must}$ for the finitary part of *TACCS*; the extension to recursion can be handled using standard techniques. In the full paper we also give an axiomatisation of $\sqsubseteq_{may}$, along the lines of that proposed in [4] for a variant of our language. Here we outline the equational characterisation of the must preorder: this requires the standard theory of internal and external choice [7], the axioms for asynchrony used in [1], a law accounting for the interplay between output and choice, and three new conditional equations.

The paper ends with a brief conclusion and comparison with related work. In this extended abstract all proofs are omitted. They may be found in the full version of the paper.

2 The language and operational semantics

Given the set Names ranged over by a, b, c, we use $\mathsf{Act} = \{\, a, \bar{a} \mid a \in \mathsf{Names}\,\}$ to denote the set of visible actions, and $\mathsf{Act}_\tau = \mathsf{Act} \cup \{\tau\}$ for the set of all actions, ranged over by α, β, γ. The syntax of the language *TACCS* is given by

$$\begin{aligned} t ::= {} & \mathsf{0} \mid a.\, t \mid \widehat{a}.\, t \mid \overline{a} \mid t \,\|\, t \mid t \oplus t \mid t + t \\ & x \in \mathsf{Var} \mid \mathsf{rec}\, x.\, t \end{aligned}$$

We have the empty process $\mathsf{0}$, two choice operators, external choice $+$ and internal choice $\oplus$, parallelism $\|$ and recursive definitions $\mathsf{rec}\, x.\, t$ from [7]. We also have input prefixing $a.\, t$ and the new construct of *asynchronous output prefixing* $\widehat{a}.\, t$, where the emission on channel a is non blocking for the process t. The atom $\overline{a}$ is introduced to facilitate the semantics of asynchronous output.

We have not included the hiding and renaming constructs from *CCS*; they can be handled in a straighforward manner in our theory and would simply add uninteresting detail to the various definitions. Note however that unlike the version of asynchronous *CCS* considered in [15] or the presentation of the asynchronous π-calculus in [1] we do not require a separate syntactic class of guarded terms; the external choice operator may be applied to arbitrary terms.

As usual the recursion operator binds variables and we shall use p, q, r to denote *closed* terms of *TACCS*, which we often refer to as processes. The usual abbreviations from *CCS* will also be employed, for example omitting trailing occurrences of $\mathsf{0}$; so e.g. we abbreviate $a.\,\mathsf{0}$ to a.

The operational semantics of processes in *TACCS* is given in Figure 1 (where some obvious symmetric rules are omitted) in terms of three transition relations:

- $p \xrightarrow{a} q$, meaning that p can receive a signal on channel a to become q
- $p \xrightarrow{\bar{a}} q$, meaning that p can asynchronously transmit a signal on channel a to become q
- $p \xrightarrow{\tau} q$, meaning that p can reduce to q after an internal step, possibly a communication

This semantics differs from the standard operational semantics for synchronous *CCS*, given in [7], only for the asynchronous behaviour of outputs. More precisely:

- An asynchronous output prefix $\widehat{a}.\,p$ can only perform an internal action, whose effect is to spawn off an atom $\overline{a}$ which will then run in parallel with p

$$\widehat{a}.\,p \xrightarrow{\tau} \overline{a} \parallel p$$

- Only a spawned off atom $\overline{a}$ can emit a signal on channel a

$$\overline{a} \xrightarrow{\bar{a}} \mathsf{0}$$

- Atoms $\overline{a}$ behave asynchronously with respect to external choice $+$: in particular they are not consumed when the choice is resolved. We have for instance

$$\overline{a} + b \xrightarrow{\tau} \overline{a} \parallel \mathsf{0} \xrightarrow{\bar{a}} \mathsf{0} \parallel \mathsf{0}$$

We feel that the first two properties correspond to a natural behavioural interpretation of the intuitive idea of non blocking output. The third ensures the asynchrony of output, even in the presence of external choice. Intuitively, asynchronous actions should not affect the behaviour of the rest of the system, e.g. by preventing other actions as would be the case with the usual rule for $+$.

3 Testing asynchronous processes

In this section we apply the standard testing scenario to *TACCS* to obtain asynchronous versions of the *may* and *must* testing preorders. After discussing the difference between synchronous and asynchronous testing we give alternative behavioural characterisations of the preorders.

We first recall the standard definition of testing from [7]. Let ω be a new action representing "success". Then a *test* or *observer* e is simply a process which may contain occurrences of ω. The definitions of the *may* and *must* testing preorders are just the standard ones.

$$\text{Input} \quad \frac{}{a.\,p \xrightarrow{a} p}$$

$$\text{Output} \quad \frac{}{\widehat{a}.\,p \xrightarrow{\tau} \overline{a} \parallel p} \qquad\qquad \text{Atom} \quad \frac{}{\overline{a} \xrightarrow{\bar{a}} \mathsf{0}}$$

$$\text{Par} \quad \frac{p \xrightarrow{\alpha} p'}{p \parallel q \xrightarrow{\alpha} p' \parallel q}$$

$$\text{Com} \quad \frac{p \xrightarrow{a} p', \qquad q \xrightarrow{\bar{a}} q'}{p \parallel q \xrightarrow{\tau} p' \parallel q'}$$

$$\text{Int} \quad \frac{}{p \oplus q \xrightarrow{\tau} p} \qquad\qquad \frac{}{p \oplus q \xrightarrow{\tau} q}$$

$$\text{Ext} \quad \frac{p \xrightarrow{a} p'}{p + q \xrightarrow{a} p'} \qquad\qquad \frac{p \xrightarrow{\tau} p'}{p + q \xrightarrow{\tau} p' + q} \qquad\qquad \frac{p \xrightarrow{\bar{a}} p'}{p + q \xrightarrow{\tau} \overline{a} \parallel p'}$$

$$\text{Rec} \quad \frac{}{\mathsf{rec}\, x.\, t \xrightarrow{\tau} t[\mathsf{rec}\, x.\, t/x]}$$

Fig. 1. Operational Semantics

For any process p and test e let

- p may e if there exists a maximal computation

$$e \parallel p = e_0 \parallel p_0 \xrightarrow{\tau} \cdots e_k \parallel p_k \xrightarrow{\tau} \cdots$$

  such that $e_n \xrightarrow{\omega}$ for some $n \geq 0$.
- p must e if for every such maximal computation, $e_n \xrightarrow{\omega}$ for some $n \geq 0$.

Then the testing preorders and the induced equivalences are defined as usual:

- $p \mathrel{\underset{\sim}{\sqsubset}}_{may} q$ if p may e implies q may e for every test e
- $p \mathrel{\underset{\sim}{\sqsubset}}_{must} q$ if p must e implies q must e for every test e
- $p \mathrel{\underset{\sim}{\sqsubset}} q$ if $p \mathrel{\underset{\sim}{\sqsubset}}_{may} q$ and $p \mathrel{\underset{\sim}{\sqsubset}}_{must} q$

- $p \simeq_{may} q$ if $p \mathrel{\underset{\sim}{\sqsubset}}_{may} q$ and $q \mathrel{\underset{\sim}{\sqsubset}}_{may} p$
- $p \simeq_{must} q$ if $p \mathrel{\underset{\sim}{\sqsubset}}_{must} q$ and $q \mathrel{\underset{\sim}{\sqsubset}}_{must} p$
- $p \simeq q$ if $p \mathrel{\underset{\sim}{\sqsubset}} q$ and $q \mathrel{\underset{\sim}{\sqsubset}} p$

Note that the definitions of the preorders are formally the same as in the synchronous case [7], but because they are applied to an asynchronous semantics we expect these relations to be weaker. It has been argued in previous work on asynchronous calculi [10, 1], that an *asynchronous observer* should not be capable of observing the inputs of a process, since this would amount to detecting when its own outputs are consumed; indeed this restriction on what can be observed seems to be the essence of asynchrony. In our testing scenario this restriction is automatically enforced; the observer loses control over the inputs of the process being tested because its own output actions are non blocking, and thus in particular they cannot block a success action ω.

Let us look at some examples. In these examples p will always denote the left-hand process and q the right-hand one.

Example 1. $\boxed{a \;\sqsubseteq\; \mathsf{0}}$

To see that $p \sqsubseteq_{must} q$, note that in any test that p must pass, an output on a can never block a success action ω. Typically the test $e = \widehat{a}.\, \omega$ cannot serve as a failing test for q as in the synchronous case. So if $p \;\mathsf{must}\; e$ this is because all the internal sequences of e itself lead to success, hence also $q \;\mathsf{must}\; e$. In a similar way one argues that $p \sqsubseteq_{may} q$.

Example 2. $\boxed{\mathsf{0} \;\not\sqsubseteq_{must}\; a}$ $\boxed{\mathsf{0} \;\sqsubseteq_{may}\; a}$

The distinguishing test in the must case is $(a.\, \omega \,\|\, \overline{a})$.

As this example shows, the tests $(a.\, \omega \,\|\, \overline{a})$ allow inputs to be observed to some extent. In the synchronous case the test normally used to observe the input a is $e = (\omega \oplus \omega) + \overline{a}$. However with our "asynchronous" rule for $+$ the test e could silently move to $\overline{a}$, and thus fail for every process. Note that the asynchronous test $(a.\, \omega \,\|\, \overline{a})$ is weaker than the corresponding test e in the synchronous case: it cannot distinguish the inability to perform an action a from the possibility of performing an a immediately followed by an $\overline{a}$. This is illustrated by the next example.

Example 3. $\boxed{\mathsf{0} \;\simeq\; a.\overline{a}}$

We leave the reader to verify the various inequalities. Let us just remark, as regards $\mathsf{0} \;\sqsubseteq_{must}\; a.\overline{a}$, that the test $(a.\, \omega \,\|\overline{a})$ cannot be used here as a discriminating test as in Example 2, since $a.\overline{a} \;\mathsf{must}\; (a.\, \omega \,\|\overline{a})$. Note that with the standard rule for $+$ we would have $\mathsf{0} \;\mathsf{must}\; (\omega \oplus \omega) + \overline{a}$ and therefore it would *not* be true that $\mathsf{0} \;\sqsubseteq_{must}\; a.\overline{a}$. On the other hand this seems to be a basic law for asynchrony.

The testing preorders $\sqsubseteq_{may}$ and $\sqsubseteq_{must}$ are *contextual*, in the sense that they are defined using a quantification over the behaviour of processes in all contexts of a certain form. We now give alternative characterisations for them, which are independent of these contexts. To motivate these characterisations it will be convenient to refer to similar results in the synchronous case. So we let $\sqsubseteq^{s}_{may}$ and $\sqsubseteq^{s}_{must}$ denote the synchronous may and must testing preorders; that is the preorders based on the standard operational semantics of *CCS*, as defined in [7].

3.1 Characterising may testing

Let us start with some terminology. We define *weak transitions* as usual:

$$p \stackrel{\varepsilon}{\Longrightarrow} p' \;\Leftrightarrow\; p \stackrel{\tau}{\longrightarrow}{}^{*} p'$$
$$p \stackrel{\alpha}{\Longrightarrow} p' \;\Leftrightarrow\; p \stackrel{\varepsilon}{\Longrightarrow} \cdot \stackrel{\alpha}{\longrightarrow} \cdot \stackrel{\varepsilon}{\Longrightarrow} p'$$

Note the difference between $p \stackrel{\varepsilon}{\Longrightarrow} p'$ and $p \stackrel{\tau}{\Longrightarrow} p'$ (the latter involving at least one τ step). The weak transitions are extended to sequences of observable actions in the standard way:

$$p \stackrel{s}{\Longrightarrow} p' \;\Leftrightarrow\; p \stackrel{\alpha_1}{\Longrightarrow} \cdots \stackrel{\alpha_n}{\Longrightarrow} p' \qquad\qquad s = \alpha_1 \cdots \alpha_n, \quad \alpha_i \neq \tau$$

Thus $p \stackrel{s}{\Longrightarrow} p'$ means that p can carry out the observable sequence of actions s to arrive at p'. In a synchronous world this is exactly the same as saying that p can react to the sequence of messages $\bar{s}$ (or to the obvious sequential process constructed from $\bar{s}$) to arrive at p'. Let $\equiv$ denote the structural congruence over terms generated by the monoidal laws for $\|$. Formally we have, using the notation $p \stackrel{\alpha}{\longrightarrow} \equiv p'$ to mean $\exists\, p''.\; p \stackrel{\alpha}{\longrightarrow} p'' \equiv p'$

$$p \stackrel{s}{\Longrightarrow} p' \text{ if and only if } p \,\|\, \bar{s} \stackrel{\varepsilon}{\Longrightarrow} \equiv p'$$

However in an asynchronous world these two statements no longer coincide. First of all, the notion of "sequence of messages" is not as clear, since output messages are non blocking for the rest of the sequence. The asynchronous operational semantics allows for a larger number of computations from $p\|\bar{s}$ (where $\bar{s}$ represents here the process obtained by replacing output actions with asynchronous prefix in the corresponding sequence). For example if $\bar{s}$ is the sequence $\bar{b}b$ then we can have $p \,\|\, \bar{s} \stackrel{\varepsilon}{\Longrightarrow} p$, which means that in the asynchronous world we should allow p to move to itself via the sequence s. To formalise this idea we introduce new transition relations $\stackrel{\alpha}{\Longrightarrow}_a$ and $\stackrel{s}{\Longrightarrow}_a$.

Definition 1. *Let the strong* asynchronous transition relation $\stackrel{\alpha}{\longrightarrow}_a$ *be the least relation over processes such that*

- $\stackrel{\alpha}{\longrightarrow} \subseteq \stackrel{\alpha}{\longrightarrow}_a$
- *for any* $b \in \mathsf{Names}$, $p \stackrel{b}{\longrightarrow}_a p \,\|\, \bar{b}$

The weak transition relations $\stackrel{\alpha}{\Longrightarrow}_a$ *and* $\stackrel{s}{\Longrightarrow}_a$ *are then defined as above.*

Note that the $\stackrel{\alpha}{\longrightarrow}_a$ determine a kind of *input-enabled* transition system, similar to that introduced in [10] by Honda and Tokoro for the asynchronous π-calculus.

Lemma 1. $p \stackrel{s}{\Longrightarrow}_a p'$ *and* $q \stackrel{\bar{s}}{\Longrightarrow} q'$ *imply* $p \,\|\, q \stackrel{\varepsilon}{\Longrightarrow} \equiv p' \,\|\, q'$

For any process p, let $\mathcal{L}(p)$ denote the language of observable sequences of p:

$$\mathcal{L}(p) = \{\, s \mid \exists p'.\; p \stackrel{s}{\Longrightarrow} p' \,\}$$

We define similarly the language of observable *asynchronous sequences* of p:

$$\mathcal{L}^a(p) = \{\, s \mid \exists p'.\; p \stackrel{s}{\Longrightarrow}_a p' \,\}$$

We call $\mathcal{L}^a(p)$ the *asynchronous language* of p. It is well-known that in the synchronous case may testing is characterised by language inclusion; that is, for synchronous processes

$$p \sqsubseteq^s_{may} q \text{ if and only if } \mathcal{L}(p) \subseteq \mathcal{L}(q)$$

As it turns out, this result can be naturally adapted to the asynchronous case:

Theorem 1. (Characterisation of may testing). *In TACCS :*

$$p \sqsubseteq_{may} q \;\; \textit{if and only if} \;\; \mathcal{L}(p) \subseteq \mathcal{L}^a(q)$$

3.2 Characterising must testing

Unlike that of $\sqsubseteq_{may}$, the characterisation of $\sqsubseteq_{must}$ is rather involved. In order to motivate it, it will be helpful to recall the corresponding characterisation for the synchronous case. To do so we need the following notation:

Input and Output sets $I(p) = \{ a \mid p \xrightarrow{a} \}$, $O(p) = \{ \bar{a} \mid p \xrightarrow{\bar{a}} \}$
Ready sets $R(p) = I(p) \cup O(p)$
Acceptance sets $\mathcal{A}(p, s) = \{ R(p') \mid p \stackrel{s}{\Longrightarrow} p' \stackrel{\tau}{\not\longrightarrow} \}$
Convergence predicate Let $\Downarrow$ be the least predicate over processes which satisfies
- $p \Downarrow \varepsilon$ if there is no infinite reduction of the form $p \xrightarrow{\tau} \cdots$
- $p \Downarrow \alpha s$ if $p \Downarrow \varepsilon$ and $p \stackrel{\alpha}{\Longrightarrow} q$ implies $q \Downarrow s$.

Definition 2. *Let $p \ll^s q$ if for every $s \in \mathsf{Act}^*$, $p \Downarrow s$ implies*

- *$q \Downarrow s$*
- *for every $A \in \mathcal{A}(q, s)$ there exists some $A' \in \mathcal{A}(p, s)$ such that $A' \subseteq A$.*

The classical result for synchronous processes [7] is

$$p \sqsubseteq^s_{must} q \text{ if and only if } p \ll^s q$$

We wish to adapt this result to the asynchronous case. Consider first the convergence predicate. An asynchronous version can be defined in the obvious way:

- $p \Downarrow^a \varepsilon$ if there is no infinite reduction of the form $p \xrightarrow{\tau} \cdots$
- $p \Downarrow^a \alpha s$ if $p \Downarrow^a \varepsilon$ and $p \stackrel{\alpha}{\Longrightarrow}_a q$ implies $q \Downarrow^a s$.

Acceptance sets will also require considerable modification. An immediate remark is that they should not include input actions. This is illustrated by a simple example, already encountered in Section 3.

Example 4. Let $p = a$ and $q = \mathsf{0}$. We have seen that $p \sqsubseteq_{must} q$. On the other hand $\mathcal{A}(q, \varepsilon) = \{\emptyset\}$ and $\mathcal{A}(p, \varepsilon) = \{\{a\}\}$, so the condition on acceptance sets of Definition 2 is not satisfied.

This example suggests that the $\mathcal{A}(p,s)$ should be restricted to output actions, i.e. replaced by *output acceptance sets* $\mathcal{O}(p,s) = \{\, O(p') \mid p \stackrel{s}{\Longrightarrow} p' \stackrel{\tau}{\not\longrightarrow} \}$, in keeping with the intuition that only outputs should be directly observable.

A consequence of the synchronous characterisation is that $p \sqsubseteq^{s}_{must} q$ implies $\mathcal{L}(q) \subseteq \mathcal{L}(p)$, since $s \in \mathcal{L}(p) \Leftrightarrow \mathcal{A}(p,s) \neq \emptyset$. This is not true in the asynchronous case; $p \sqsubseteq_{must} q$ does not imply $\mathcal{L}(q) \subseteq \mathcal{L}(p)$, as shown by the next example.

Example 5. Let $p = \mathsf{0}$ and $q = a.\overline{a}$. We have $p \sqsubseteq_{must} q$ (as seen earlier in this section) and $a \in \mathcal{L}(q)$, but $a \notin \mathcal{L}(p)$.

This indicates that p should be allowed to match the execution sequences of q in a more liberal way than in the synchronous case: if q executes a sequence s then p should be allowed to execute s "asynchronously". This is where the new transition relation $\stackrel{s}{\Longrightarrow}_a$ comes into play for must testing. Intuitively we expect $p \sqsubseteq_{must} q \Rightarrow \mathcal{L}(q) \subseteq \mathcal{L}^a(p)$.

These two remarks lead to our first attempt at a definition. We will use two kinds of acceptance sets: $\mathcal{O}(p,s) = \{\, O(p') \mid p \stackrel{s}{\Longrightarrow} p' \stackrel{\tau}{\not\longrightarrow} \}$ and $\mathcal{O}^a(p,s) = \{\, O(p') \mid p \stackrel{s}{\Longrightarrow}_a p' \stackrel{\tau}{\not\longrightarrow} \}$.

Definition 3. *Let $p \ll' q$ if for every $s \in \mathsf{Act}^*$, $p \Downarrow^a s$ implies*

- *$q \Downarrow^a s$*
- *for every $O \in \mathcal{O}(q,s)$ there exists some $O' \in \mathcal{O}^a(p,s)$ such that $O' \subseteq O$.*

It may be shown that $p \sqsubseteq_{must} q$ implies $p \ll' q$. On the other hand $\ll'$ is still too generous and it is not true that $p \ll' q$ implies $p \sqsubseteq_{must} q$.

Example 6. Let $p = a.\overline{b}$ and $q = \mathsf{0}$. Then

- $p \ll' q$, because $\mathcal{O}(q,\varepsilon) = \{\emptyset\}$ and $\mathcal{O}^a(p,\varepsilon) = \{\emptyset\}$.
- $p \not\sqsubseteq_{must} q$, because p must e while q $\not\mathsf{m}\mathsf{ust}$ e, where e is the test $\overline{a} \parallel b.\omega$.

This example suggests that to compute the acceptance set of p after s we should not only consider its s-derivatives but also look further at their input-derivatives (derivatives via a sequence of inputs). Intuitively this is because an observer with output capabilities interacting with a process may or may not discharge some of these outputs by communication with the process. In the above example the observer provokes the derivation $p \stackrel{\varepsilon}{\Longrightarrow} p' \stackrel{a}{\longrightarrow} p''$ where $p' = p$ and $p'' = \overline{b}$ and this should be reflected in some manner in the acceptance set of p after ε.

We thus want to generalise the sets $\mathcal{O}^a(p,s)$ by looking at the input-derivatives of p after s. For a reason that will soon become clear we need to consider multisets of inputs rather than sets. In the following $\uplus$ denotes multiset union. We use the notation $\{\!|\ \ |\!\}$ for multisets, writing for instance $\{\!|\, a, a \,|\!\}$.

Definition 4. *For any finite multiset of* input *actions I let $\stackrel{}{\rightsquigarrow}_I$ be the binary predicate defined by*

- *$(p \stackrel{\tau}{\not\longrightarrow}$ and $I(p) \cap I = \emptyset)$ implies $p \rightsquigarrow_I p$*
- *$(p \stackrel{a}{\Longrightarrow} p'$, and $p' \rightsquigarrow_I p''$) implies $p \rightsquigarrow_{I \uplus \{\!|\, a \,|\!\}} p''$*

Intuitively $p \rightsquigarrow_I p'$ means that by performing a subset of the input actions in I, p can arrive at the stable state p' which cannot perform any of the remaining actions in I. Note that if p is stable then $p \rightsquigarrow_\emptyset p$.

Based on this predicate, we can define the generalised acceptance sets:

$$\mathcal{O}_I^a(p,s) = \{\, O(p'') \mid p \overset{s}{\Longrightarrow}_a p', p' \rightsquigarrow_I p'' \,\}$$

Let $IM(p,s)$, the set of *input multisets* of p after s, be given by

$$IM(p,s) = \{\, \{\!|\, a_1, \dots, a_n \,|\!\} \mid a_i \in \mathsf{Names},\ p \overset{s}{\Longrightarrow}_a \overset{a_1}{\Longrightarrow} \cdots \overset{a_n}{\Longrightarrow} p_n \,\}$$

We can now finally define our alternative preorder:

Definition 5. *Let $p \ll q$ if for every $s \in \mathsf{Act}^*$, $p \Downarrow^a s$ implies*

- $q \Downarrow^a s$
- *for every $A \in \mathcal{A}(q,s)$ and every $I \in IM(p,s)$ such that $I \cap A = \emptyset$ there exists some $O \in \mathcal{O}_I^a(p,s)$ such that $O \backslash \overline{I} \subseteq A$.*

Two comments about this definition will be helpful.

- The requirement $I \cap A = \emptyset$ can be justified as follows. The multiset I represents the inputs that p could be doing (after s) in reaction to some test, that is the complement of the outputs that the test is offering. Since the process q has reached a *stable state* with ready set A, where it is confronted with the same test, the test should be unable to make q react. Thus in particular the complement I of its outputs $\overline{I}$ should be disjoint from the inputs in A. Note that without this requirement we would have e.g. $a.\overline{b} \not\ll a.\overline{b}$!
- The condition $O \backslash \overline{I} \subseteq A$ can also be explained intuitively. In an asynchronous setting, an observer providing some outputs $\overline{I}$ to the environment, will have subsequently no way of telling, when observing these outputs, whether they come from himself or from the observed process p. So all he can require is that when p reaches a stable state p'', with outputs, say, O, the *remaining* outputs of p'', $O \backslash \overline{I}$, be included in the actions A of the corresponding state of q. In fact the condition $O \backslash \overline{I} \subseteq A$ could be reformulated as $O \subseteq A \uplus \overline{I}$, which is reminiscent of the definition of *asynchronous bisimulation* in [1].

Theorem 2. (Characterisation of must testing) *In TACCS :*

$$p \sqsubseteq_{must} q \text{ if and only if } p \ll q$$

4 Equational Characterisation

In this section we restrict attention to finite terms, without recursion. Standard techniques may be used to extend equational theories to recursive terms.

In what follows $\mu, \nu \in \mathsf{Act}$ will denote visible actions. We will use the notation $\mu.t$ to represent $a.t$ when μ is the input action a and $\widehat{a}.t$ when μ is the output action $\bar{a}$. With this convention the basic testing axioms for $\sqsubseteq$ (concerning the

Asynchrony:

$$X = X + a.\,(\overline{a} \parallel X)$$
$$\widehat{a}.\,X = \overline{a} \parallel X$$

Fig. 2. Extra laws for asynchronous testing

choice operators and their interplay with prefixing and parallelism) are exactly the same as in [7] and are not reported here.

In addition we need two axioms for asynchrony, given in Figure 2. The first is a law holding also for weak asynchronous bisimulation, taken from [1]. The second is the natural law for asynchronous output prefixing. Indeed it shows that in the presence of the atoms $\overline{a}$ this form of prefix is redundant in our language. However its presence facilitates the comparison between the operational semantics of synchronous and asynchronous outputs.

We shall not present the characterisation of $\sqsubseteq_{may}$ here, since it is a simple adaptation of that given in [4] for a variant of our language. Let us just recall some interesting laws holding for asynchronous may testing. Let $a \in \mathsf{Names}$, $\mu \in \mathsf{Act}$. Then $\sqsubseteq_{may}$ satisfies:

$$a.\,\mu.\,X \leq \mu.\,a.\,X$$
$$a.\,(\overline{a} \parallel X) \leq X$$

In fact, for $\sqsubseteq_{may}$ the first law in Figure 2 can be derived from these two laws.

We give now the equational characterisation for must testing. To obtain the theory for $\sqsubseteq_{must}$ we add to the standard laws and those of Figure 2 the usual axiom for must testing:

$$X \oplus Y \;\leq\; X$$

However this is not sufficient to yield a complete theory for $\sqsubseteq_{must}$. For instance we have the inequality $\bar{a} + b.\,\bar{a} \sqsubseteq_{must} \bar{a}$, which the reader may want to check using the alternative characterisation of Theorem 2. This inequality is not derivable from the axioms considered so far. More generally we have the law

$$p + b.\,p \;\sqsubseteq_{must}\; p$$

In our theory this will be an instance of the conditional rule **R1** in Figure 3.

Two more conditional rules are required, **R2** and **R3**, which we briefly motivate. As regards **R2**, note that the may testing law $a.\,(\overline{a} \parallel X) \leq X$ is not valid

$$X \oplus Y \leq X$$

$$X + \widehat{a}.\,Y = (X + \widehat{a}.\,Y) \oplus \widehat{a}.\,Y$$

$$\mathbf{R1}\ \frac{X + Y \leq X}{X + a.\,Y \leq X}$$

$$\mathbf{R2}\ \frac{X + Y \leq X}{X + a.\,(\overline{a} \parallel Y) \leq X}$$

$$\mathbf{R3}\ \frac{X + Y \leq X}{(X + \widehat{a}.\,Z) \oplus Y \leq X}$$

Fig. 3. Extra laws for must testing

for must testing. For instance $p = a.\,(\overline{a} \parallel \overline{b}) \not\sqsubseteq_{must} \overline{b} = q$, since $\mathcal{A}(q, \bar{b}) \neq \emptyset$ while $\mathcal{O}^a_I(p, \bar{b}) = \emptyset$ for any I. For must testing we have the more restricted law

$$X \ + \ a.\,(\overline{a} \parallel X) \ \leq \ X$$

which is one half of the asynchrony law in Figure 2. However, this is not enough since it does not allow one to derive for instance: $\overline{a} \ + \ c.\,(\overline{c} \parallel b.\,\bar{a}) \ \leq \ \overline{a}$. Whence the need for the more general conditional rule **R2**.

Similarly, the use of **R3** can be exemplified by the simple inequality

$$(a \ + \ \overline{b}) \ \oplus \ \mathsf{0} \ \leq \ a$$

which again appears not to be derivable from the other laws.

Finally, we have a law expressing the asynchronous behaviour of outputs with respect to external choice

$$X + \widehat{a}.\,Y \ = \ (X + \widehat{a}.\,Y) \ \oplus \ \widehat{a}.\,Y$$

This shows that a process of the form $X + \widehat{a}.\,Y$ can spontaneously resolve its choice by releasing its output to the environment.

Let $\vdash_{must} p \leq q$ mean that $p \leq q$ is provable in the theory obtained by adding the equations in Figure 3 to the standard laws and those of Figure 2.

Theorem 3. *For finite asynchronous processes* $p \sqsubseteq_{must} q$ *iff* $\vdash_{must} p \leq q$.

5 Conclusions

We have given an asynchonous operational semantics to a version of *CCS* with internal and external choice operators, called *TACCS*. Based on this asynchonous semantics we developed semantic theories of may and must testing. In particular we gave alternative behavioural characterisations of the resulting preorders, and equational characterisations for the finite fragment of the language.

Some interesting questions remain. For example, is it possible to eliminate the use of conditional equations in the characterisation of $\sqsubseteq_{must}$? Or would there be a simpler algebraic characterisation for the sublanguage in which the external choice operator is only applied to input prefixes, essentially the language studied in [3, 1]? But perhaps the most important question is the extent to which our approach can be generalised to yield an equational characterisation of must testing over the π-calculus.

The may testing preorder has been equationally characterised for an asynchronous version of *CCS* and for the asynchronous π-calculus in [4]. But both these languages have syntactic restrictions that we do not impose; specifically external choice may only be applied to input prefixes. Strong bisimulation has also been characterised for the asynchronous π-calculus in [1], again with the same restriction on external choice. As regards weak asynchronous bisimulation, the recent work [11] shows how restrictions on the behaviour of asynchronous π-processes can bring up interesting new algebraic laws.

References

1. R. Amadio, I. Castellani, and D. Sangiorgi. On bisimulations for the asynchronous π-calculus. *Theoretical Computer Science*, 195:291–324, 1998.
2. J. C. M. Baeten and W. P. Weijland. *Process Algebra*, volume 18 of *Cambridge Tracts in Computer Science*. Cambridge University Press, 1990.
3. M. Boreale, R. DeNicola, and R. Pugliese. Asynchronous observations of processes. In *Proc. FOSSACS'98,* LNCS 1378, 1998.
4. M. Boreale, R. DeNicola, and R. Pugliese. Laws for asynchrony, 1998. Draft.
5. G. Boudol. Asynchrony and the π-calculus. Research Report 1702, INRIA, Sophia-Antipolis, 1992.
6. R. DeNicola and M. Hennessy. Testing equivalences for provesses. *Theoretical Computer Science*, 43:83–133, 1984.
7. M. Hennessy. *Algebraic Theory of Processes.* MIT Press, Cambridge, Massachusetts, 1988.
8. C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.
9. K. Honda and M. Tokoro. An object calculus for asynchronous communication. In *Proc. ECOOP'91,* LNCS 512, 1991.
10. K. Honda and M. Tokoro. On asynchronous communication semantics. In *Proc. Object-Based Concurrent Computing,* LNCS 612, 1992.
11. M. Merro and D. Sangiorgi. On asynchrony in name-passing calculi. In *Proc. ICALP'98,* LNCS 1443, 1998.
12. R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.
13. U. Nestmann and B. Pierce. Decoding choice encodings. In *Proc. CONCUR'96,* LNCS 1119, 1996.
14. Benjamin C. Pierce and David N. Turner. Pict: A programming language based on the pi-calculus. Technical Report CSCI 476, Computer Science Department, Indiana University, 1997. To appear in *Proof, Language and Interaction: Essays in Honour of Robin Milner*, Gordon Plotkin, Colin Stirling, and Mads Tofte, editors, MIT Press.
15. P. Selinger. First-order axioms for asynchrony. In *Proc. CONCUR'97,* LNCS 1243, 1997.