Concurrent Testing of Processes [*]

M. Hennessy
University of Sussex

July 1, 1992

Abstract

We develop a noninterleaving semantic theory of processes based on testing. We assume that all actions have a non-zero duration and the allowed tests take advantage of this assumption. The result is a semantic theory in which concurrency is differentiated from nondeterminism.
We show that the semantic preorder based on these tests is preserved by so-called "stable" action refinements and may be characterised as the largest such preorder contained in the standard testing preorder.

[*]This work has been supported by the ESPRIT/BRA CEDISYS project and the SERC

1 Introduction

In recent years there has been much research into semantic theories of processes which distinguish nondeterminism from concurrency. See for example [DD89, DNM90, vGV87], [BC89]. Most of these are based on some variation of bisimulation equivalence, [Mil89]. This is a well-established behavioural equivalence between processes based on their ability to perform actions. Roughly speaking two processes p, q are bisimulation equivalent if whenever either can perform an action and be transformed into the process r then the other can also perform the same action and be transformed into some r' which is bisimulation equivalent to r. This equivalence leads to a so-called "interleaving" theory of concurrency in that it reduces parallelism to nondeterminism. For example the process which can perform the actions a and b in parallel is deemed to be equivalent to the purely sequential but nondeterministic process which either can perform a followed by b or b followed by a. However "non-interleaving" theories of concurrency, i.e. theories which distinguish parallelism from concurrency, can be obtained by varying the basic ingredients of the definition of bisimulation. For example the basic actions may be replaced by partial orders of actions where the order is induced by some idea of causality as in [DNM90] or the structure of processes may be taken into account as in [BCHK91].

Another well-established "interleaving" theory of processes is based on testing, [DH84]. Here processes are said to be equivalent if they are guaranteed to pass exactly the same tests. Although the framework of testing equivalence is quite general, apart from [MP91] it has only been applied, at least as far as the author is aware, to generate "interleaving theories". The purpose of this paper is use this framework to develop a "non-interleaving theory" of processes and in particular to investigate the application of this theory to action refinement.

In the standard theory a test, which is usually itself a process, is applied to a process by running both together in parallel. A particular run is considered to be successful if the test reaches a certain designated successful state and the process guarantees the test if every run is successful. The test and the process under observation interact by communicating with each other or synchronising. In most process algebras synchronisation is modelled as the simultaneous occurrence of complementary actions although there is a variety of definitions of complementation. But regardless of the precise definition the actions which comprise the synchronisations are considered to be instantaneous and indivisible. This of course is an idealisation and abstraction from reality but it has proved to be most useful as it has lead to a range of elegant mathematical theories of processes. Here we relax this restriction. Now we will assume that the synchronisations take a non-zero but indefinite amount of time. This is still an abstraction from reality as we are not saying exactly what form the interaction takes; only that it takes time. For example we could have in mind the rendez-vous mechanism of ADA or the existence of some non-trivial communication medium connecting the tester and the process. Under these assumptions we can see how the process performing a and b in parallel can be differentiated from its sequential counterpart which performs the actions in either order. Consider the test which requests a synchronisation via the action a and then will succeed only if it can successfully initiate a second synchronisation via the action b before the first synchronisation has terminated. The first process will always pass this test whereas the second will always fail.

Let us now discuss how this intuitive idea of non-instantaneous actions can be formalised. The simplest consequence is that each action a has a distinct beginning and end, $s(a)$ and $f(a)$ respectively. If we are thinking of synchronisation over a communication network $s(a)$ corresponds to the sending of a synchronisation signal while $f(a)$ corresponds to the receipt of a confirmation that the signal has been received. The elements of Λ may be viewed as virtual synchronisation channels or ports which are supported by the underlying communication network. But only using these two sub-events we do not capture the idea that we have individual actions with duration, particularly if two distinct occurrences of an action a can be active at the same time. In terms of the communication network we have to assume that it is sufficiently intelligent to distinguish distinct instances of the same virtual synchronisation channel. In this way we are lead to view the behaviour of processes in terms of their ability to produce *labeled beginning and endings of actions.* Such an operational semantics has been given for a process algebra in [AH91b, AH91a] and a similar semantics had previously been developed in [vGV87] for Petri nets which is used to define a variation on bisimulation equivalence called *ST-bisimulation.* So we will call this type of semantics *ST-operational semantics* and using it we can easily apply the standard framework of testing to define a behavioural equivalence, or more generally a preorder, between processes which captures the intuitions of concurrent testing described above: $p \underset{\sim}{\sqsubset}_c q$ if q guarantees every concurrent test guaranteed by p and $p \eqsim_c q$ if they guarantee exactly the same set of concurrent tests. A concurrent test will be a test which can demand of a process either the beginning of a synchronisation along a specific instance of a communication channel or wait for the end of such a synchronisation, i.e. await confirmation that the synchronisation has been successfully completed.

This theory is developed for a process algebra which is a slight extension to *CCS*, essentially the language studied in [AH92]. It contains all of the operators of *CCS* together with sequential composition and recursion. The first major result of the paper is that $\underset{\sim}{\sqsubset}_c$ is preserved by all the operators in the language apart from, as usual, the choice operator $+$ from *CCS.* This is proved using an alternative characterisation in terms of so-called st-sequences and Acceptance sets which is a generalisation of the alternative characterisation of the standard testing preorder, [Hen88]. St-sequences are simply sequences of labeled beginning and ending of actions where the actual labels involved are not important; they are only a mechanism for describing sequences which may contain a number of distinct occurrences of the sub-actions $s(a)$ or $f(a)$. They have been used previously in a number of papers, such as [Ace91, Vog91a, AH91b, vGV87].

However the main result of the paper concerns *action refinement.* We add to the language a new operator which allows the refinement of an action by a process: the process $p[a \rightsquigarrow q]$ is supposed to act like the process p where the action a is replaced by the process q. We first make this notion precise and then show that $\underset{\sim}{\sqsubset}_c^c$, the closure of $\underset{\sim}{\sqsubset}_c$ with respect to all $+$ contexts, is also preserved by it, i.e. if $p \underset{\sim}{\sqsubset}_c^c p'$ and $q \underset{\sim}{\sqsubset}_c^c q'$ then, subject to some restrictions, $p[a \rightsquigarrow q] \underset{\sim}{\sqsubset}_c^c p'[a \rightsquigarrow q']$.
The final result in the paper shows that $\underset{\sim}{\sqsubset}_c^c$ may in fact be characterised using action refinement and the standard testing preorder which we call $\underset{\sim}{\sqsubset}_s$. The latter may be defined using the standard operational semantics where actions are instantaneous and it generates an "interleaving theory". We show that $\underset{\sim}{\sqsubset}_c^c$ is the largest preorder contained in $\underset{\sim}{\sqsubset}_s^c$ which is preserved by these forms of action refinement.

We now give a more detailed account of the contents of each section of the paper. In the next section we define the language used in the paper and give the st-operational semantics. The language is a very expressive process algebra; it has combinators or operators for nondeterminism, parallelism, restriction, sequential composition and there are also two forms of termination, deadlock and successful termination; in addition recursion is allowed.
The operational semantics is similar in style to that used in [AH91a], although it is formulated somewhat differently. This section also contains the formal definition of how to apply a test to a process and of the concurrent testing preorder $\underset{\sim}{\sqsubset}_c$. This involves a language for tests which can demand of a process to start an action then interrogate further the process before accepting confirmation that the started action has finished. Tests also need the ability to detect successful termination.

Section 3 contains a number of alternative characterisations of this testing preorder. The general idea is that the preorder can be determined essentially by the acceptance sets of processes after performing *st-sequences*, although some information on divergence and the ability to terminate successfully is also required. The results of this section allow us to compare our work with that of [AE91], [Vog91a]. They are also used to show that the preorder $\underset{\sim}{\sqsubset}_c^c$ is preserved by the operators in the language.

In section 4 we turn our attention to action refinements. In general an action refinement is a mapping ρ from actions to processes and the result of applying a refinement to a process is denoted by $p\rho$. (Above we used $[a \rightsquigarrow q]$ to represent a simple form of refinement which maps a to q and leaves all other actions untouched.) Formally $p\rho$ is defined to be the process which results from substituting $\rho(a)$ for each occurrence of a in p; the precise definition views restriction as an action "binder", which is natural as restricted actions are in some sense hidden or similar to local variables in a conventional programming language. (A similar approach is taken in [AH91a].) This definition of action refinement is very general. In $p\rho$ we allow arbitrary interaction between the processes $\rho(a)$ and $\rho(b)$ for any a and b and we also allow arbitrary interaction between the parent process p and the refined process $\rho(a)$.
The main result of this section, the *refinement theorem* is that the testing preorder is preserved by a large class of action refinements, i. e. if $p \underset{\sim}{\sqsubset}_c q$ and $\rho \underset{\sim}{\sqsubset}_c \rho$ (pointwise) then $p\rho \underset{\sim}{\sqsubset}_c q\sigma$. This result is subject to some restrictions on the refinements. The first states that in some sense the complementation on actions should be preserved by action refinements. Specifically $\rho(a)$ in parallel with $\rho(\overline{a})$ should be able to reach a properly terminated state by a series of communications; these communications reflect the ability of a and $\overline{a}$ to perform a communication. The second restriction states that $\rho(a)$ should not be a properly terminated process. Refinements which satisfy these constraints have been called *standard* in [AH91a]. Here we need some further restrictions which in some sense say that the refinements are initially stable: we demand that no process $\rho(a)$ can initially perform an internal move and moreover if $\rho(a)$ and $\rho(b)$ can perform an initial communication then $a = \overline{b}$. We call standard refinements which satisfy these extra constraints *stable*; we prove the refinement theorem for stable refinements.

The proof of this result involves a Whence theorem which decomposes a move from $p\rho$ into the contributions from p and ρ and a Whither theorem which shows how to combine moves from p and ρ to form a move from $p\rho$. These must then be generalised to arbitrary sequences of external actions and we must also characterise the ability of $p\rho$ to diverge in terms of properties of the behaviour of p and ρ. The latter is not particularly straightforward as $p\rho$ can diverge although p and $\rho(a)$, for every a, may be perfectly well behaved.

In the final section we prove a *characterisation theorem* for the new preorder. First we formulate a more standard testing preorder based on tests which can only demand complete actions of processes. This is similar to the notion of testing on [Hen88] and we denote it by $\underset{\sim}{\sqsubset}_s$. As one would expect it is not preserved by action refinement. We show that $\underset{\sim}{\sqsubset}_c^c$ is contained in $\underset{\sim}{\sqsubset}_s^c$ and may be characterised as the largest preorder contained in it which is preserved by stable action refinements. To prove this we exhibit a particular stable refinement σ with the property that $p\sigma \underset{\sim}{\sqsubset}_s q\sigma$ implies $p \underset{\sim}{\sqsubset}_c q$.

The development of *non-interleaving* semantic theories for concurrent processes has recently received much attention in the literature. See for example [DNM90], [TV87], [vGV87], [BC89], [Vog91a]. However most of the semantic equivalences investigated in these papers are based on modifying in some way the basic definition of *bisimulation equivalence*, [Mil89]. There are exceptions such as [TV87] and [Vog91a] but as far as the author is aware the present paper is the first attempt at applying the paradigm of testing equivalence, [DH84], to process algebras so as to obtain a non-interleaving semantic theory. However the properties of this equivalence, or preorder, have not been fully investigated and the main contribution of the present paper is the refinement theorem and the characterisation theorem.

The research literature on the subject of action refinements is also quite extensive. The novelty of our results on this topic is in the conjunction of three characteristics:

the semantic theory and the definition of action refinement applies directly to the syntax of the process algebra, rather than some intermediate intensional model such as *event structures* or *causal trees*

the semantic theory used is based on testing equivalence, rather than the much finer equivalence *bisimulation equivalence*

the class of allowed action refinements is quite general, as is the manner in which these refinements are applied to processes.

However it does leave open the question of finding a testing based equivalence between processes which is preserved by the more general class of *standard refinements.* We conjecture that this can be defined using a *step-st-operational semantics* in which processes and tests interact via multi-sets of beginnings and endings of labeled actions.

In papers such as [DD91], [DD90], [vG90] [Vog90], restricted forms of action refinements are defined on *event structures, causal trees* or *synchronisation trees* and related refinement theorems are proved with respect to *causal bisimulation* or *st-bisimulation.* The last reference also contains characterisation theorems for a variety of equivalences based on bisimulation. In [AH91a] refinements are defined directly on a process algebra and a refinement theorem and characterisation theorem is given with respect to weak bisimulation. But in the language considered there is no recursion so all processes are necessarily finite. In [AE91], failures equivalence is used and there are similar results. However the language used is very restricted; all processes are finite, there is no communication and no restriction. Nevertheless the results of the present paper may be viewed as a direct generalisation of the results of [AE91] to a full process algebra. (However we have not been able to generalise their results on fully-abstract models).

The existing work which appears closest to our results is reported in [Vog91a] and [JM92]. In [Vog91a] the author deals with *safe Petri Nets* and *failure equivalence.* A restricted form of refinement theorem is proved for a generalisation of *failure equivalence* based on *interval semi-words* and there is a characterisation with respect to the standard failures equivalence. As the author points out in a separate paper, [Vog91b], *interval semi-words* are more or less equivalent to st-sequences; it therefore follows (from the results of section three) that this equivalence is closely related to $\eqsim_c$. However the notion of action refinement used is more restrictive than what we allow. In particular when a refinement ρ is applied to a process there can be no interaction between occurrences of $\rho(a)$ and $\rho(b)$ in the refined process. In [JM92] this work is extended to a more general class of Petri Nets but the restrictions on the type of action refinements remain.

2 The Language

In this section we describe the language, taken from [AH92], its st-operational semantics and the testing preorder.

The language is parameterised on a set of actions Act which is ranged over by $a, b, \ldots$. We assume that there is a possibly partial complementation function defined over Act; we write the complement of a, if it exists, as $\overline{a}$ and we asssume that $\overline{\overline{a}}$ is a. We also assume a special action symbol, τ, different from all symbols in Act and a set of recursion variables X ranged over by x. Then the syntax of the language is given in the usual way using a finite set of process operators and a mechanism for recursive definitions:

$$\begin{array}{lll} t & ::= & \Omega \mid nil \mid \delta \mid a \mid \tau \mid x \\ & & \mid t+t \mid t \,|\, t \mid t\backslash a \\ & & \mid t;t \mid rec\,x.\,t \end{array}$$

We have two kinds of terminated processes, *nil* the successfully terminated process and δ the deadlocked process, while Ω denotes a process which can only diverge internally and $rec\,x.\,t$ stands for the process defined recursively by the equation $x = t$. We use the choice, parallel and restriction operators from *CCS*, $|, +$ and $\backslash$, while ; represents sequential composition. As usual $rec\,x.\,t$ binds occurrences of x in t and we have free and bound occurrences of variables in terms and an appropriate notion of substitution of terms for free occurrences of variables : $t[u/x]$. Let BP_{Act} denote the set of *closed* terms, i.e. those terms with no free variables, which we often refer to as *processes*; this set is ranged over by p, q, i. e. . Much of the development does not depend on the set of actions Act and so we usually abbreviate BP_{Act} to BP. But in later in the paper we will have a need for particular instantiations of BP_{Act} when we choose particular action sets Act.

The st-operational semantics of the language is given in terms of a termination relation $\surd$ and a next state relation with respect to labelled beginning and ending of actions. We also need a relation to define the effect of an internal move or communication between subprocesses, which is represented in the language by the special action τ. One way of giving rise to such an action is by the simultaneous occurrence of two complementary *complete* actions and therefore it will convenient to define in addition a next state relation with respect to complete actions also although in principle these are derivable from those for sub-actions. Let L be an infinite set of labels, ranged over by l, $LSAct$ denote the set of labelled sub-actions $\{\, s(a_l), f(a_l) \mid a \in Act, l \in L \,\}$ and $LAct$ denote the union of all the external actions, $LSAct \cup Act$. For any set S we use S_τ to denote $S \cup \{\tau\}$. So for example $LSAct_\tau$, Act_τ, denote the sets $LSAct \cup \{\tau\}$, $Act \cup \{\tau\}$ respectively. We let μ range over the set of all possible actions $LAct_\tau$, α over the set of external actions $LAct$, a over the set of complete actions Act and finally e over $LSAct$, the set of (labelled) sub-actions. The execution of the sub-actions will often lead to states of processes where actions have started and not yet terminated and therefore we have to enrich the language in order to define such states. We call the more general terms *configurations* and they are defined by

$$c \;::=\; p \;\mid\; a_l \;\mid\; c \mid c \;\mid\; c;p \;\mid\; c\backslash a$$

where we assume that in $c\backslash a$ c contains no occurrences of any $a_l, \overline{a}_l$ and more importantly that every occurrence of a labelled action a_l is unique. An occurrence of a_l is meant to denote that there is an a action active and since we use the labels to distinguish different occurrences it is important that there is no duplication of labels. So for example the configuration $a_l;p \mid b_l;q \mid a_k;r$ describes a process which has three subprocesses, two of which are performing an a action and one a b action. We let $\mathcal{C}$ denote the set of configurations, ranged over by c, and for any $c \in \mathcal{C}$ $L(c)$ denotes $\{\, a_k \mid a_k \text{ occurs in } c \,\}$.

Definition 2.1 Let $\surd$ be the least relation over configurations which satisfies

1. $nil\surd$
2. $p\surd$, $q\surd$ implies $p+q\surd$
3. $p\surd$, $c\surd$ implies $p;c\surd$
4. $c\surd$, $c'\surd$ implies $c \mid c'\surd$
5. $c\surd$ implies $c\backslash a\surd$
6. $t[rec\,x.\ t/t]\surd$ implies $rec\,x.\ t\surd$

$\square$

Since BP is contained in $\mathcal{C}$ this also gives a definition of $\surd$ for the set of processes. Because of the way in which recursion is handled we also need, in the definition of testing, a "well-definedness" predicate on configurations, $\downarrow$.

Definition 2.2 Let $\downarrow$ be the least relation over configurations which satisfies

1. $nil\downarrow$ and $\delta\downarrow$
2. $p\downarrow$, $q\downarrow$ implies $p+q\downarrow$
3. $p\downarrow$, $c\downarrow$ implies $p;c\downarrow$
4. $c\downarrow$, $c'\downarrow$ implies $c \mid c'\downarrow$

(O1) $\mu \xrightarrow{\mu} nil$

$a \xrightarrow{s(a_l)} a_l$ for every label l

$a_l \xrightarrow{f(a_l)} nil$

(O2) $p \xrightarrow{\mu} c$ implies $p + q \xrightarrow{\mu} c$

(O3) $c_1 \xrightarrow{\mu} c_1'$ implies $c_1 \mid c_2 \xrightarrow{\mu} c_1' \mid c_2$
provided $c_1' \mid c_2$ is in $\mathcal{C}$

(O4) $c \xrightarrow{\mu} c'$ implies $c \backslash a \xrightarrow{\mu} c' \backslash a$
provided a admits μ

(O5) $c \xrightarrow{\mu} c'$ $c; p \xrightarrow{\mu} c'; p$

$c\surd,\ p \xrightarrow{\mu} c'$ implies $c; p \xrightarrow{\mu} c'$

(O6) $c_1 \xrightarrow{a} c_1',\ c_2 \xrightarrow{\overline{a}} c_2'$ implies $c_1 \mid c_2 \xrightarrow{\tau} c_1' \mid c_2'$

(O8) $t[rec\,x.\ t/x] \xrightarrow{\mu} q$ implies $rec\,x.\ t \xrightarrow{\mu} q$

(O9) $\Omega \xrightarrow{\tau} \Omega$

Figure 1: Operational semantics

5. $c \downarrow$ implies $c \backslash a \downarrow$

6. $t[rec\,x.\ t/t] \downarrow$ implies $rec\,x.\ t \downarrow$

$\square$

We often use $\uparrow$ for the converse to $\downarrow$. So for example $\Omega \uparrow$ and $rec\,x.\ a + x \uparrow$. The next state relations $\xrightarrow{\mu}$, for each $\mu \in LAct_\tau$ are given in Figure 1. Many of the rules are straightforward and do not require comment. The obvious symmetric components of the rules $(O2)$ and $(O3)$ have been omitted and the predicate admits used in (O4) has the obvious definition: a admits μ unless μ has one of the forms $a, \overline{a}, s(a_l)$ or $f(a_l)$. The main nonstandard rule is (O1) which allows the basic process a to perform not only the complete action a but also simply start the action by performing the move $s(a_l)$ for any label l to arrive at the state a_l: this indicates that an instance of the action a is running. The side condition in $(O3)$ ensures that labels continue to be unique so that the actual labels used in applications of (O1) will be restricted by the labels not already occurring in the configurations which contain the process to which this rule is applied.

The last three rules are concerned with the derivation of internal moves and $(O6)$ is the most important. It says that an internal move may occur because of a communication between two subprocesses. The remaining rules are straightforward; Ω can only perform internal moves and the moves of a recursive definition are determined by its body.

One can check that if $c \xrightarrow{\mu} c'$ and $c \in \mathcal{C}$ then c' is also in $\mathcal{C}$. One can also show that the actual identity of the labels generated in the derivations are relatively unimportant. Specifically if $c \xrightarrow{s(a_l)} c'$ then for almost all labels k $c \xrightarrow{s(a_k)} c'[k/l]$; one can use any k which does not occur already in c, which is a finite set of labels. As stated previously it is unnecessary to define the operational semantics of complete actions as they can be derived. This can be seen from the second part of the following lemma.

Lemma 2.3 *for every configuration c*

1. $c \xrightarrow{s(a_l)} c' \xrightarrow{f(a_l)} d$ *implies* $c \xrightarrow{a} d$
2. $c \xrightarrow{a} d$ *implies there exists a configuration* c' *such that for some* l $c \xrightarrow{s(a_l)} c' \xrightarrow{f(a_l)} d$
3. $c \xrightarrow{s(a_i)} \xrightarrow{s(\overline{a_j})} \xrightarrow{f(a_i)} \xrightarrow{f(\overline{a_j})} d$ *implies* $c \xrightarrow{\tau} d$.

□

Of slightly more interest is the fact that many pairs of moves can be permuted.

Definition 2.4 Two elements, μ, μ' of $LSAct_\tau$ *weakly commutes* if for every pair of configurations c, c' $\exists d. c \xrightarrow{\mu} d, d \xrightarrow{\mu'} c'$ implies $\exists d. c \xrightarrow{\mu'} d, d \xrightarrow{\mu} c'$. □

The set of weakly commuting moves can be characterised as follows:

Proposition 2.5 *A pair of moves* $< \mu, \mu' >$ *is weakly commuting if and only if they are of one of the forms the forms*

1. $< s(a_i), \mu >$ *where* μ *is different than* $f(a_i)$
2. $< \mu, f(a_i) >$ *where* μ *is different than* $s(a_i)$.

Proof: For pairs not of this form it is easy to think of counterexamples. If the pair $< \mu, \mu' >$ is of this form and $c \xrightarrow{\mu} d \xrightarrow{\mu'} c'$ then one can show by induction on the derivation of $c \xrightarrow{\mu} d$ that there exists a d' such that $c \xrightarrow{\mu'} d' \xrightarrow{\mu} c'$ The detailed case analysis depends crucially on the allowed structure of configurations. □

A stronger notion of commuting may be obtained by replacing the implication in the above definition with "if and only if". Let us say that such a pair is *strongly commuting"*. There are far fewer strongly commuting pairs; the only ones are of the form $< s(a_l), s(b_k) >$ and $< f(a_l), f(b_k) >$.

The reason for developing this st-operational semantics is to formalise the concurrent testing discussed in the introduction. However in order to be able to describe the appropriate tests we need a language which is strictly more expressive than BP_{Act}. This is because in this form of testing we need to be able to test the ability of processes to initiate new synchronisations before other previously initiated synchronisations have terminated. Such tests are not possible in the basic language BP_{Act}. So we include in our set of tests processes which can perform as fully-fledged actions the subactions of the language BP_{Act}. Specifically we use as the set of actions

$$\Delta = \{\, s(a_l), \overline{s(a_l)}, f(a_l), \overline{f(a_l)} \mid a \in Act \} \ \cup \ Act.$$

There is another problem which can not be resolved by mimicing the framework of testing developed in [Hen88]. Here we have two forms of termination, in *nil* and δ and it is difficult to see how they can be differentiated by purely computation means. Accordingly we introduce into the testing language the ability to recognise proper termination; this takes the form of a special action *term* which the test can execute only when the process under observation is properly terminated. So we use as the set of tests, *Tests*, closed terms in the language BP_Φ where Φ is the set of actions

$$\Delta \cup \{term, \omega\}.$$

Here the action ω will be used to report the successful completion of an experiment. Of course there are many tests in this language which will not be used but, at least here, it is not of great importance to characterise the collection of meaningful tests.

We now define formally how tests and processes, or more generally configurations, interact. An *experimental state* takes the form $e \parallel c$ where e is a test and c a configuration. An experiment proceeds by moving from state to state and this is defined using a transition relation of the form $e \parallel c \ \mapsto \ e' \parallel c'$.

Definition 2.6 Let $\mapsto$ be the least relation between experimental states which satisfies

1. $e \xrightarrow{\alpha} e',\ c \xrightarrow{\alpha} c'$ implies $e \parallel c \ \mapsto \ e' \parallel c'$ for every α in $LAct$
2. $e \xrightarrow{\tau} e'$ implies $e \parallel c \ \mapsto \ e' \parallel c$
3. $c \xrightarrow{\tau} c'$ implies $e \parallel c \ \mapsto \ e \parallel c'$
4. $e \xrightarrow{term} e',\ c\surd$ implies $e \parallel c \ \mapsto \ e' \parallel c$. □.

We can now define in the standard way when processes guarantee tests. A *computation* from the state $e \parallel p$ is a maximal sequence of the form

$$e \parallel p \equiv e_0 \parallel c_0 \ \mapsto \ e_1 \parallel c_1 \ldots \ \mapsto \ e_n \parallel c_n \ \mapsto \ \ldots$$

that is, it is either infinite or has a maximal element $e_m \parallel c_m$ from which no further derivation can be made. A state $e_m \parallel c_m$ in such a computation is successful if e_m can perform the action ω and for every $n < m$ $p_n \downarrow$. We say p *guarantees* e or p *must* e if every computation from $e \parallel p$ is successful. Finally we write $p \underset{\sim}{\sqsubset}_c q$ if for every test e p *must* e implies q *must* e. The associated equivalence relation, the kernel of $\underset{\sim}{\sqsubset}_c$, is denoted by $\eqsim_c$.

Example 2.7 The two processes *nil* and δ are incomparable. On the one hand *nil must term*; ω while δ $\mathit{m\!\!/ust}$ *term*; ω and therefore *nil* $\not\underset{\sim}{\sqsubset}_c$ δ and on the other δ $\not\underset{\sim}{\sqsubset}_c$ *nil* because of the test $\tau; \omega + term$. □

Example 2.8 The processes $a \mid b$ and $a; b + b; a$ are also incomparable: $a \mid b$ guarantees the test $s(a_l); s(b_l); \omega$ whereas we have the unsuccessful computation

$$s(a_l); s(b_l); \omega \parallel a; b + b; a \;\mapsto\; s(b_l); \omega \parallel a_l; b.$$

In the other direction $a; b + b; a$ guarantees the test $s(a_l); (\tau; \omega + s(b_l))$ whereas $a \mid b$ generates an unsuccessful computation:

$$s(a_l); (\tau; \omega + s(b_l)) \parallel a \mid b \;\mapsto\; \tau; \omega + s(b_l) \parallel a_l \mid b \;\mapsto\; nil \parallel a_l \mid b_l.$$

$\square$

Example 2.9 The operator $+$ does not in general distribute over the parallel operator $\mid$. For example $a \mid (b + c)$ guarantees the test $s(a_l); s(b_l); \omega$ whereas

$$s(a_l); s(b_l); \omega \parallel a \mid b + a \mid c \;\mapsto\; s(b_l); \omega \parallel a_l \mid c$$

is an unsuccessful computation. However it turns out that $a \mid b + a \mid c \sqsubseteq_c a \mid (b + c)$ as we will be able to check in a straightforward manner using the results of the next section. $\square$

Example 2.10 In the theory of testing presented in [Hen88] $a + a; (b + c) \sqsubseteq a + a; b + a; c)$ but here this is no longer the case because of the ability of processes to check for termination. These processes can now be distinguished by the test $a; (term; \omega + b; \omega)$. $\square$

Example 2.11 On page 51 of [AH91b] the *owl example* of R. van Glabbeek is described in *CCS* and discussed in detail. The two terms used there, P and Q, are also in the language *BP* but we refrain from reproducing them here. However the interested reader can check that P *must* e, Q $\cancel{must}$ e where e is the test $b; s(c_l); a; s(c_k); f(c_l); s(d); \omega$. Since the processes are symmetric one can also easily find a test which establishes $Q \not\sqsubseteq_c P$. This example shows that the labelling of active actions is essential. For P and Q are "timed equivalent" - an variation on bisimulation equivalence where unlabelled beginnings and endings of actions are used - and this ensures that there are also equivalent in a version of testing equivalence which uses unlabelled beginnings and endings. $\square$

In the next section we give an alternative characterisation of $\sqsubseteq_c$ which will enable us to develop its properties more easily.

3 ST-Acceptance sets

Here we outline the properties of processes which determine their ability to to guarantee tests. They are essentially the st-sequences they can produce and their acceptance sets after these sequences. These acceptance sets are finite collections of finite subsets of *Act* but because of the ability to test for proper termination they have to be defined slightly differently than in [Hen88]. First some notation.

The single arrow relations $\xrightarrow{\mu}$ are extended in the natural way to abstract from internal moves. For each $s \in LAct_\tau^*$ we define the relation $\overset{s}{\Longrightarrow}$ between configurations as follows:

1. $c \overset{\varepsilon}{\Longrightarrow} c$
2. $c \overset{s}{\Longrightarrow} d,\ \ d \overset{\mu}{\longrightarrow} d'$ implies $c \overset{s.\mu}{\Longrightarrow} d'$ for every μ in $LAct_\tau$
3. $c \overset{s}{\Longrightarrow} d,\ \ d \overset{\tau}{\longrightarrow} d'$ implies $c \overset{s}{\Longrightarrow} d'$.

Note the subtlety in this definition; $c \overset{\tau^n}{\Longrightarrow} d$ means that c can move to d by performing at least n internal moves. For any c let $S(c) = \{\, a \in Act \mid c \overset{a}{\Longrightarrow}\ \}$; $S(c)$ only contains complete actions which are by definition unlabelled.
We say c is *stable* if $c\downarrow$ and $c \overset{\tau}{\longrightarrow} c'$ for no c' and it is *live* if $c\surd$ is not true. i.e. if it has not properly terminated. We also say it *converges*, written $c\Downarrow$, if intuitively it can not diverge, i.e. every sequence of the form

$$c = c_0 \overset{\tau}{\longrightarrow} c_1 \overset{\tau}{\longrightarrow} c_2 \ldots$$

is finite and for every configuration c_k in such a sequence $c_k\downarrow$. The set of *acceptance sets of p after s*, for $s \in LAct^*$, is defined by

$$\mathcal{A}(p,s) = \{\, S(c) \mid p \overset{s}{\Longrightarrow} c,\ c\ \textit{stable}\ \text{and}\ \ \textit{live}\,\}.$$

In this definition the requirement that c be live is crucial. Acceptance sets are compared as in [Hen88], using a slight variation on subset inclusion; We write $\mathcal{A}(p,s) \ll \mathcal{A}(q,s)$ if for every $A \in \mathcal{A}(q,s)$ there is a $B \in \mathcal{A}(p,s)$ such that $B \subseteq A$.

To obtain the alternative characterisation of $\underset{\sim}{\sqsubset}_c$ we need to parameterise both the proper termination and the convergence predicates to sequences of actions from $LAct$. We say *c (weakly) terminates with respect to s* if there exists a c' such that $c \overset{s}{\Longrightarrow} c'$ and $c\surd$. The generalisation of $\Downarrow$ is much stronger; $c \Downarrow s$ guarantees that c will never diverge when performing any subsequence of s. It is defined by induction on s:

1. $c \Downarrow \varepsilon$ if $p \Downarrow$
2. $c \Downarrow \alpha.s$ if $c \Downarrow$ and for every c' such that $c \overset{\alpha}{\Longrightarrow} c;,\ \ c' \Downarrow s$.

We will sometimes use $c \Uparrow s$ to indicate the negation of $c \Downarrow s$.

Definition 3.1 (The alternative preorder) For processes p, q we write $p \ll_{st} q$ if for every $s \in LAct^*$

$$\begin{array}{lll} p \Downarrow s & \text{implies} & i)\ \ q \Downarrow s \\ & & ii)\ \ q\surd s \text{ implies } p\surd s \\ & & iii)\ \ \mathcal{A}(p,s) \ll \mathcal{A}(q,s) \end{array}$$

□

The main result of this section is that $\underset{\sim}{\sqsubset}_c$ and $\ll_{st}$ coincide on processes.

We first need some simple lemmas about the permanency of finish moves. We say $f(a_l)$ is *active* in c if $c \overset{f(a_l)}{\longrightarrow}$ and it is active in the sequence s from $LAct^*$ if s has the form $s_1.s(a_l).s_2$ where $f(a_l)$ does not occur in s_2.

Lemma 3.2 *If $f(a_l)$ is active in c*

1. $c \xrightarrow{\mu} d$ and μ is different from $f(a_l)$ imply $f(a_l)$ is active in d

2. $d \xrightarrow{\mu} c$ and μ is different from $s(a_l)$ imply $f(a_l)$ is active in d

Proof: Each case is an easy induction on the derivation of $\xrightarrow{\mu}$. □

Lemma 3.3 *If $c \overset{s}{\Longrightarrow} d$ and $f(a_l)$ is active in s then it is also active in d.*

Proof: The proof is by induction on the derivation of $c \overset{s}{\Longrightarrow} d$. There are two cases:

1. $c \overset{\varepsilon}{\Longrightarrow} c$. Vacuous
2. $c \overset{s}{\Longrightarrow} c' \xrightarrow{\mu} d$. If μ is $s(a_l)$ then one can show by induction on the derivation of $c' \xrightarrow{s(a_l)} d$ that $f(a_l)$ is active in d. Otherwise we may apply induction to obtain that $f(a_l)$ is active in c' and then, since μ can not be $f(a_l)$, apply the previous lemma.

□

In a similar manner we can prove

Lemma 3.4 *If $c \overset{s}{\Longrightarrow} d$ and $f(a_l)$ is active in d then it is active in c or in s.* □

With these lemmas we can now derive one part of the required characterisation.

Proposition 3.5 *For all processes p, q, $p \ll_{st} q$ implies $p \sqsubseteq_c q$.*

Proof: Suppose p *must* e. To show q *must* e we consider an arbitrary computation

$$e \parallel q \equiv e_0 \parallel q_0 \;\mapsto\; e_1 \parallel q_1 \ldots \qquad (1)$$

We must show that some e_k is successful. The computation (1) can be decomposed into the individual contributions from e and q

$$\begin{array}{cccc} e_{i_o} & e_{i_1} & e_{i_2} & \cdots \\ q_{i_o} & q_{i_1} & q_{i_2} & \cdots \end{array}$$

where for each k either $e_{i_k} \xrightarrow{term} e_{i_{k+1}}$ and $q_{i_k}\surd$ or $e_{i_k} \overset{\alpha}{\Longrightarrow} e_{i_{k+1}}$ and $q_{i_k} \overset{\alpha}{\Longrightarrow} q_{i_{k+1}}$ for some $\alpha \in LAct$. For the moment let us assume that the former does not occur and let $s_k \in LAct$ denote the sequence $\alpha_1 \ldots \alpha_k$. We may assume that $q \Downarrow s_k$ for otherwise we would have $p \Uparrow s_k$ and we would be able to recombine the diverging computation from p with that above from e and since p *must* e there would be some successful e_k.
The decomposition above may either be finite or infinite. In the latter case we can follow the standard proof, Lemma 4.4.13 from [Hen88], and so we may assume that (1) above is finite with maximal element $e_n \parallel q_n$. There are two possibilities.
i) $term \in S(e_n)$. Then we know that not $q_n\surd$ and therefore $S(q_n) \in \mathcal{A}(q, s_n)$. Since $p \ll_{st} q$ it follows that $p \overset{s_n}{\Longrightarrow} p'$ for some stable and live p' such that $S(p') \subseteq S(q_n)$. So the derivation $p \overset{s_n}{\Longrightarrow} p'$ can be combined with that from e above to obtain a computation

from $e \parallel p$ to $e_n \parallel p'$ which contains only the test states from the original computation (1). The result would now follow if we can show that this is a maximal computation. The only possible way for $e_n \parallel p'$ to make a further step is for $f(a_l)$ to be active in both e_n and p'. Applying Lemma 3.4 $f(a_l)$ must be active in s_n and by Lemma 3.3 this would imply that $f(a_l)$ is active in q_n which contradicts the fact that $e_n \parallel q_n$ can not make another step.
ii) $term \notin S(e_n)$. If not $q_n\surd$ then we can proceed as in the previous case. So suppose $q_n\surd$. Then since $p \ll_{st} q$ it follows that $p\surd s_n$ and therefore $p \overset{s_n}{\Longrightarrow} p'$ for some p' such that $p'\surd$. It is easy to check that any r such that $r\surd$ is stable and can make no move. Therefore we can recombine $p \overset{s_n}{\Longrightarrow} p'$ with the derivation from e as before to obtain a computation from $e \parallel p$ with maximal element $e_n \parallel p'$.

This leaves the case when at some point in the decomposition of (1) we have $q_{i_n}\surd$ and $e_{i_n} \xrightarrow{term} e_{i_{n+1}}$ for some n. However if we consider the first such occurrence we can proceed as in case ii) above. $\square$

The converse is more straightforward in that the proof is much the same as the corresponding proof in [Hen88], Lemma 4.4.9.

Proposition 3.6 *For all processes p, q $p \sqsubseteq_c q$ implies $p \ll_{st} q$.*

Proof: The idea of the proof is to define particular tests which capture the behaviour of processes which determine the properties used in the definition of $\ll_{st}$.
For each $s \in LAct$ let $conv(s)$ be defined by

$$\begin{aligned} conv(\varepsilon) &= \tau;\omega \\ conv(\alpha.s) &= \tau;\omega + \alpha; conv(s) \end{aligned}$$

Then $p \Downarrow s$ if and only if p *must* $conv(s)$.
Similarly we can define a test for weak termination. For every $s \in Act$ let $notterm(s)$ be defined by

$$\begin{aligned} notterm(\varepsilon) &= \tau;\omega + term \\ noterm(\alpha.s) &= (\tau;\omega) + \alpha; notterm(s) \end{aligned}$$

Then if $p \Downarrow s$ one can check that p *must* $notterm(s)$ if and only if not $p\surd s$.
Finally we can check the acceptance sets of a process by the following tests: for each finite $B \subseteq Act$ let $acc(s, B)$ be defined by

$$\begin{aligned} acc(\varepsilon, B) &= term;\omega + \sum\{\, a;\omega \mid a \in B \,\} \\ acc(\alpha.s, B) &= \tau + \alpha; acc(s, B) \end{aligned}$$

Again if $p \Downarrow s$ it follows that p *must* $acc(s, B)$ if and only if for every $A \in \mathcal{A}(p, s)$ $A \cap B \neq \emptyset$.
Using these three sets of tests the result now follows easily. $\square$

Putting these two results together we obtain the main result of the section:

Theorem 3.7 *(Alternative Characterisation) For all processes p, q $p \sqsubseteq_c q$ if and only if $p \ll_{st} q$.* $\square$

This alternative characterisation makes the behavioural preorder much more amenable to investigation. For example one can now easily check the examples in the previous section. One can also prove that the preorder is preserved by all the operators of the language other than $+$:

Proposition 3.8 *For each operator in the language op except $+$, $p \sqsubseteq\!\!\!\!\sim_c q$ implies that $op(\ldots, p, \ldots) \sqsubseteq\!\!\!\!\sim_c op(\ldots, q, \ldots)$.*

Proof: It is a matter of checking each of the operators in turn but this is made much more manageable if $\ll_{st}$ is used in place of $\sqsubseteq\!\!\!\!\sim_c$. □

It is not preserved by $+$ for the usual reasons: $a \sqsubseteq\!\!\!\!\sim_c \tau; a$ but $a + b$ guarantees the test $b; \omega$ which may be failed by $\tau; a + b$. However it is very easy to adapt it so that it is preserved by $+$; Let $p \sqsubseteq\!\!\!\!\sim_c^c q$ if

1. $p \sqsubseteq\!\!\!\!\sim_c q$
2. p *stable* implies q *stable*

and let $\eqsim_c^c$ be the associated equivalence. We state without proof:

Proposition 3.9 *The relation $\sqsubseteq\!\!\!\!\sim_c^c$ is the largest relation contained in $\sqsubseteq\!\!\!\!\sim_c$ which is preserved by all the operators in the language.* □

We will also use of the alternative characterisation in the next section to show that $\sqsubseteq\!\!\!\!\sim_c^c$ is preserved by action refinement. However in the definition of $\ll_{st}$ there is a tremendous amount of redundant information. This is principally caused by the fact that many of the sequences in $LAct^*$ can not be generated by processes and the fact the the individual identity of the labels are unimportant. In the remainder of this section we outline how some of this redundancy can be removed but the uninterested reader by proceed immediately to the next section.

Definition 3.10 A sequence s in $LAct$ is called and *st-sequence* if it satisfies the following conditions:

1. each $e \in LSAct$ has at most one occurrence in s
2. every occurrence of a subaction of the form $f(a_l)$ is preceded by an occurrence of $s(a_l)$.

□

In these sequences labels are not reused and the begins and ends are in the proper order. We could also eliminate all occurrences of complete actions by replacing a with $s(a_l); f(a_l)$ for an arbitrary label l. However in the next section it will be convenient to have an alternative characterisation in which they are retained. Let $\ll_{st}'$ be defined by restricting attention in the definition of $\ll_{st}$ to st-sequences. Then

Proposition 3.11 *For all processes p, q $\;p \ll_{st} q$ if and only if $p \ll_{st}' q$.*

Proof: (Outline) The proof relies on the fact that if $s \in LAct^*$ $p \overset{s}{\Longrightarrow} p_1$ then the starts and finishes in s are in the proper order and $p \overset{s}{\Longrightarrow} p_1$ if and only if $p \overset{s'}{\Longrightarrow} p_1'$ where $s' \in LAct^*$ is obtained from s by some systematic renaming to the duplicate labels in the resulting sequence to ensure uniqueness and p_1' is obtained from p' by the same label renaming. Note that $S(p_1, s) = S(p_1', s')$. □

We can go even further by abstracting away from the actual label names in the st-sequences. An *association* h is a finite subset of $Act \times L \times L$ which satisfies

$$\begin{array}{lll} (<a,i,j>, <a,i,j'>) \in h & \text{implies} & j = j' \\ (<a,i,j>, <a,i',j>) \in h & \text{implies} & i = i' \end{array}$$

So h sets up a one-one correspondence between two finite subsets of L for each action a. Let us restrict our attention to st-sequences which contain no complete actions, which we call `st`-sequences. Then $\equiv_h$ is defined on `st`-sequences as follows:

1. $\varepsilon \equiv_{\emptyset} \varepsilon$
2. $s \equiv_h s'$ implies $s.s(a_l) \equiv_{h \cup \{<a,l,k>\}} s'.s(a_k)$
3. $s \equiv_h s'$ implies $s.f(a_l) \equiv_h s'.f(a_k)$ provided $<a,l,k> \in h$

If $s \equiv_h s'$ then intuitively s and s' are the same sequence of moves up to a renaming via h. Now let $s \equiv s'$ if $s \equiv_h s'$ for some h. One can check that $\equiv$ is an equivalence relation over `st`-sequences; it is essentially the equivalence used in [AE91]. Moreover it is sufficient to consider the behaviour of processes with respect to arbitrary representatives from the equivalence clauses rather than all `st`-sequences. One can check that if $s \equiv s'$ then

$$\begin{array}{rcl} p \Downarrow s & \Longleftrightarrow & p \Downarrow s' \\ p \surd s & \Longleftrightarrow & p \surd s' \\ \mathcal{A}(p,s) & = & \mathcal{A}(p,s'). \end{array}$$

So one can unambiguously define

1. $p \Downarrow [s]$ if for any $s' \equiv s$ $p \Downarrow s'$
2. $p \surd [s]$ if for any $s' \equiv s$ $p \surd s'$
3. $\mathcal{A}(p,[s]) = \mathcal{A}(p,s')$ where s' is an arbitrary element of $[s]$.

Now let $\ll''_{st}$ be defined by modifying the original definition of $\ll_{st}$ so that equivalence classes are used rather than sequences over $LAct$. Specifically

Definition 3.12 For processes p, q we write $p \ll''_{st} q$ if for every `st`-sequence s

$$\begin{array}{lll} p \Downarrow [s] & \text{implies} & i)\ q \Downarrow [s] \\ & & ii)\ \mathcal{A}(p,[s]) \ll \mathcal{A}(q,[s]) \\ & & iii)\ q \surd [s] \text{ implies } p \surd [s] \end{array}$$

□

It follows more or less immediately that

Proposition 3.13 *For all processes p, q $p \ll_{st} q$ if and only if $p \ll''_{st} q$.* $\Box$

One could go even further and show how these equivalence classes can be interpreted as labelled partial orders. But the connection between st-sequences and labelled partial orders has been extensively studied in [Vog91b]. In particular he has pointed out that st-sequences can be viewed as representing a restricted class of labelled partial orders where the underlying partial order is an *interval order.*

4 Action Refinement

The aim of this section is to show that the behavioural preorder $\underset{\sim_c}{\sqsubset}$ is preserved by action refinement and we use the approach and notation from [AH91a]. An *action refinement* is a mapping

$$\rho\colon Act \longmapsto BP.$$

It associates with each action a a process $\rho(a)$. The application of an action refinement ρ to the process p will be denoted by $p\rho$. This is to be considered as a process which behaves like the process p where the action a has been replaced by the process $\rho(a)$. We will not give a direct operational semantics to $p\rho$; instead we will simply define a substitution operator *sub* which gives the effect of substituting $\rho(a)$ for a and then say that $p\rho$ behaves in exactly the same way the process $sub(\rho, p) \in BP$. The definition of substitution is straightforward except that we wish to consider the restriction operator $\backslash a$ as a binding operator on actions. This is reasonable because, for example, the behaviour of $(a; p \mid c; q)\backslash a$ does not depend on the particular action a; it has the same behaviour as $(b; p \mid c; q)\backslash b$ assuming a, b does not appear in $p, c; q$. We would also expect $(a; p \mid c; q)\backslash a\ \rho$ to have the same behaviour as $(b; p \mid a; q)\backslash b\ \rho$. Now consider a ρ which maps a to $rec\,x.\ a; x$. If we simply used syntactic substitution we would require that $(a; p \mid (rec\,x.\ a; x); q)\backslash a$ and $(a; p \mid (rec\,x.\ a; x); q)\backslash b$ have the same behaviour, and they are obviously different; the latter can perform a a move which the former can not. The problem occurs because in the former the free occurrence of a in $\rho(a)$ is captured by the restriction in $(a; p \mid c; q)\backslash a$. So in order to avoid this we need to rename the actions being restricted so that no capturing occurs.

We assume that the reader is familiar with the usual treatment of binding operators, (see for example [Sto88]), and here we only sketch the details. Precise definitions and the proofs of associated properties for the treatment of restriction as a binder may be found in [AH91a]. We use $FA(t)$ to denote the set of free actions occurring in the term t (an occurrence of $\overline{a}$ is considered to be an occurernce of a also) and $=_\alpha$ to denote the associated α-conversion; this is the least equivalence relation preserved by all the operators of the language and $rec\,x.$ which satisfies

$$b \notin FA(t) \text{ and } t\mathrm{id}[a \mapsto b] =_\alpha u \text{ imply } t\backslash a =_\alpha u\backslash b.$$

Here id represents the identity refinement and $\rho[a \mapsto p]$ is the refinement which is identical to ρ except that a and $\overline{a}$ are mapped to p and $\overline{p}$ respectively, where $\overline{p}$ is obtained from p by replacing each action a with its complement $\overline{a}$.

Definition 4.1 For each action refinement ρ and term t let $sub(\rho, t)$ be the term defined by

1. $sub(\rho, a) = \rho(a)$
2. $sub(\rho, (t\backslash a)) = sub(\rho[a \mapsto a'], t)\backslash a'$ where a' is different than all action names in $FA(\rho(a))$ for each a occurring free in t.
3. $sub(\rho, op(\ldots, t, \ldots)) = \ op(\ldots, sub(\rho, t), \ldots)$
4. $sub(\rho, x) = x$
5. $sub(\rho, rec\, x.\ t) = rec\, x.\ sub(\rho, t)$.

□

For each process p and action refinement ρ $sub(\rho, p)$ is also a process and we take the behaviour of $p\rho$ to be that of $sub(\rho, p)$. Via this reduction we may view $p\rho$ as a process in BP and in what follows we will take this for granted.

We wish to investigate under circumstances the new semantic preorder is preserved by action refinement. Specifically let $\rho \ \underset{c}{\sqsubseteq}^{c} \sigma$ if $\rho(a) \ \underset{c}{\sqsubseteq}^{c} \sigma(a)$ for every a. Then we wish to know under what circumstances

$$p \ \underset{c}{\sqsubseteq}^{c} q \text{ and } \rho \ \underset{c}{\sqsubseteq}^{c} \sigma \text{ imply } p\rho \ \underset{c}{\sqsubseteq}^{c} q\sigma,$$

or speaking more strictly $p \ \underset{c}{\sqsubseteq}^{c} q$ and $\rho \ \underset{c}{\sqsubseteq}^{c} \sigma$ imply $sub(\rho, p) \ \underset{c}{\sqsubseteq}^{c} sub(\sigma, q)$.

In [AH91a] a similar problem was posed for bisimulation equivalence and it was shown to be true for *standard* refinements.

Definition 4.2 An action refinement ρ is *standard* if it satisfies

1. for each a $\ \rho(a) \mid \rho(\overline{a}) \overset{\varepsilon}{\Longrightarrow} r$ for some r such that $r\surd$.
2. for each a not $\rho(a)\surd$.

□

The first condition says that after the refinement the resulting process should be able to mimic the complete synchronisation between a and $\overline{a}$ and the second says that an action can not be refined to a process which is properly terminated. These conditions are also necessary if we wish $\ \underset{c}{\sqsubseteq}\ $ to be preserved by refinements.

Example 4.3 Let p be $(a + d) \mid \overline{d}; b$ and q be $p + \tau; b$. Then $p \eqsim_c^c q$ but if $\rho(d) = c$, $\rho(\overline{d}) = e$ then $p\rho \not\eqsim_c q\rho$; the former guarantees the test $a; \omega$. Note that this refinement does not satisfy the first requirement of *standard.* □

Example 4.4 Let p, q be $((c + a; \overline{d}) \mid d; b)\backslash d,\ c + a; b$ respectively. Then $p \eqsim_c^c q$ but if $\rho(a) = nil$, a refinement which does not satisfy the second condition, then $p\rho \not\eqsim_c q\rho$. □

However more restrictions are neccessary as can be seen from the next example.

Example 4.5 Let p be $a \mid (b;c) + a + b$, q be $a + b$ and ρ a standard refinement such that $\rho(a) = d$ and $\rho(b) = \overline{d}$. Then $p \underset{c}{\sqsubseteq} q$ but p guarantees the test $c\omega$ whereas q fails it. □

Example 4.6 An standard action refinemant is called *stable* if

1. $\rho(a) \not\xrightarrow{\tau}$for every action a
2. if $\rho(a) \mid \rho(b) \xrightarrow{\tau}$ implies $a = \overline{b}$

□

The example just discussed violates the second condition but we conjecture that the first is unnecessary.

Theorem 4.7 *(The Refinement Theorem) For every pair of stable refinements ρ, σ, $p \underset{c}{\sqsubseteq}^{c} q$ and $\rho \underset{c}{\sqsubseteq}^{c} \sigma$ imply $p\rho \underset{c}{\sqsubseteq}^{c} q\sigma$.*

Before beginning the proof of this theorem we should point out that it depends on the syntax of the language used. In particular it does not hold for the language *EPL*, [Hen88], with respect to which much of the previous work on testing has been developed. The following example has been pointed out by L. Jategoankar, [Jat92]. The main difference between *EPL* and *CCS* is that τ is replaced by a binary "internal choice operator" $\oplus$ with the operational semantics determined by the rules

$$p \oplus q \xrightarrow{\iota} p \text{ and } p \oplus q \xrightarrow{\iota} q.$$

The relation $\xrightarrow{\iota}$ is much the same as $\xrightarrow{\tau}$ except that it does not decide the choice in $p + q$. Specifically it satisfies

$$p \xrightarrow{\iota} p' \text{ implies } p + q \xrightarrow{\iota} p' + q$$

and the obvious symmetric couterpart.
Using this different operational semantics on can check that $a \mid b + a;b + b;a \underset{c}{\sqsubseteq} a \mid b$ but if we apply the action refinement ρ which is the identity except that it maps b to $\overline{a}$ then they can be distingushed by the test $a;\omega$.

The remainder of this section is devoted to proving this theorem. Most of intermediate results only depend on refinements being *standard*; the extra conditions are needed is one place, the proof of Theorem 4.22. So we only assume that all refinements are standard and we will explicitly mention stability when it is required.

We use the characterisation theorem of the previous section and therefore much of the proof is concerned with $\ll_{st}$ instead of $\underset{c}{\sqsubseteq}$. The proof depends on the ability to break down a sequence of moves from $p\rho$ into the contributions made by p and each $\rho(a)$ and conversely the ability to recombine appropriate sequences of moves from p and each $\rho(a)$ in order to obtain a sequence from $p\rho$. The first step is to analyse single moves from $p\rho$ or more generally from the configurations which can occur when such processes start making moves. So we need to generalise the definition of refinements from processes to configurations.

Definition 4.8 An *(extended) action refinement* is a mapping

$$\rho\colon Act \cup \{\, a_i \mid a \in Act \,\} \longmapsto C$$

such that $\rho(a) \in BP$ for all $a \in Act$. □

We say that a pair of such refinements are *compatible* if they agree on all the unlabelled actions, i. e. for all $a \in Act$, $\rho(a) = \rho'(a)$. A pair (c, ρ) is *compatible* if $sub(\rho, c)$, defined in the same way as above for terms, is a configuration. This simply means that the labels used in $\rho(a_i)$, $\rho(b_j)$ where a_i, b_j occur in c must be distinct. We will assume that whenever we write $c\rho$ in the pair is actually compatible. We will also say the pair is *proper* if it is compatible and $\rho(a_i)\surd$ if and only if $a_i \notin L(c)$. We will identify an action refinement ρ with its automatic extension, defined by letting $\rho(a_i) = nil$ for each a_i. For extended action refinements obtained in this way every pair (p, ρ) is proper and we tend to refer to "extended action refinements" simply as "action refinements".

Because substitution in general changes the names of restricted channels it is convenient to work modulo $=_\alpha$ or even a slightly more general relation which includes $=_\alpha$. Let $\equiv$ denote strong bisimulation defined over configurations using labelled sub-actions. Specifically it is the largest equivalence relation over C such if $c \equiv d$ then

1. $c\surd$ implies $c\surd$
2. for all $\mu \in LAct_\tau$ $c \xrightarrow{\mu} c'$ implies there exists a d' such that $d \xrightarrow{\mu} d'$ where $c' \equiv d'$.

This is quite a strong relation as even the labels in the moves must be matched. One can show that $p =_\alpha q$ implies that $p \equiv q$ and that $\equiv$ satisfies the equations

$$\begin{aligned} nil; p &= nil \\ nil + p &= p \\ p \mid nil &= p \end{aligned}$$

We will use these equations freely throughout the following proofs. As an example of the identities allowed using $\equiv$ we show

Lemma 4.9 *If $c \xrightarrow{f(a_i)} c'$ and $\rho(a_i)\surd$ then $c\rho \equiv c'\rho$.*

Proof: By induction on the derivation of $c \xrightarrow{f(a_i)} c'$. □

We first consider the proper termination of $c\rho$.

Proposition 4.10 *For every configuration c and refinement ρ*

1. *$c\surd$ implies $c\rho\surd$*
2. *$c\rho\surd$ implies either $c\surd$ (in which case c is a process) or $c \xrightarrow{f(a_i)} c'$ for some $f(a_i)$, c' such that $\rho(a_i)\surd$ and $c'\rho\surd$*
3. *if (c, ρ) is proper then $c\rho\surd$ implies $c\surd$*

Proof: The first statement may be proved by induction on why $c\surd$. The second uses structural induction on c from which the last statement follows immediately. For the second statement to be true it is essential for ρ to be standard. $\square$

We now analyse the moves from processes of the form $c\rho$. The first result, the *Whence theorem*, decomposes a move of the form $c\rho \xrightarrow{\mu} \overline{c}$ into the contributions from c and the individual processes $\rho(a)$. This theorem was originally stated in [AH91a] for finite processes although here we simplify the statement of the final possibility.

Proposition 4.11 *(The Whence Theorem) If $c\rho \xrightarrow{\mu} \overline{c}$ then one of the following holds:*

- *starting moves*

  1. $c \xrightarrow{s(a_i)} c'$, $\rho(a) \xrightarrow{\mu} x$ *and* $\overline{c} \equiv c'\rho[a_i \mapsto x]$
  2. $c \xrightarrow{s(a_i)} c'' \xrightarrow{s(b_j)} c'$, $a_i \neq b_j$, $\rho(a) \xrightarrow{a'} x$, $\rho(b) \xrightarrow{\overline{a'}} y$ *and* $\mu = \tau$, $\overline{c} \equiv c'\rho[a_i \mapsto x, b_j \mapsto y]$
  3. b_j *occurs in* c, $c \xrightarrow{s(a_i)} c'$, $\rho(a) \xrightarrow{a'} x$, $\rho(b_j) \xrightarrow{\overline{a'}} y$ *and* $\mu = \tau$, $\overline{c} \equiv c'\rho[a_i \mapsto x, b_j \mapsto y]$

- *continuing moves*

  4. a_i *occurs in* c, $\rho(a_i) \xrightarrow{\mu} x$ *and* $\overline{c} \equiv c\rho[a_i \mapsto x]$
  5. a_i, b_j *occur in* c, $a_i \neq b_j$ $\rho(a_i) \xrightarrow{a'} x$, $\rho(b_j) \xrightarrow{\overline{a'}} y$ *and* $\mu = \tau$, $\overline{c} \equiv c'\rho[a_i \mapsto x, b_j \mapsto y]$

- *only* c *moving*

  6. $c \xrightarrow{\tau} c'$ *and* $\mu = \tau$, $\overline{c} \equiv c'\rho$
  7. a_i *occurs in* c, $c \xrightarrow{f(a_i)} c'$, $\rho(a_i)\surd$ *and* $c'\rho \xrightarrow{\mu} \overline{c}$.

Proof: The proof is by induction on the length of the derivation of $c\rho \xrightarrow{\mu} \overline{c}$ and is carried out by a case analysis on the structure of c. The only case not covered by Theorem 4.2 of [AH91a] is when c has the form $rec\,x.\ t$.
Now $rec\,x.\ t\rho \equiv rec\,x.\ t\rho$ and since $rec\,x.\ t\rho \xrightarrow{\mu} \overline{c}$ it follows that (up to the α-conversion) $t\rho[rec\,x.\ t\rho/x] \xrightarrow{\mu} \overline{c}$ and moreover the proof of the latter is shorter than that of the former. Accordingly we can apply induction since $t\rho[rec\,x.\ t\rho/x] = t[rec\,x.\ t/x]\rho$ and because $t[rec\,x.\ t/x]$ is a process there are only three possibilities, (1), (2) and (6). As an example consider the second. Here $t[rec\,x.\ t/x] \xrightarrow{s(a_i)} c'' \xrightarrow{s(b_j)} c'$, $\rho(a) \xrightarrow{a'} x, \rho(a) \xrightarrow{\overline{a'}} y$, $\mu = \tau$ and $\overline{c} \equiv c'\rho[a_i \mapsto x, b_j \mapsto y]$. This means that $rec\,x.\ t \xrightarrow{s(a_i)} c'' \xrightarrow{s(b_j)} c'$ and therefore case (2) applies. $\square$

There is a converse theorem, again a generalisation of the corresponding theorem from [AH91a], for recombining appropriate moves from c and ρ to obtain a single move from $c\rho$. There are six possibilities corresponding to the first six methods for decomposing a move from $c\rho$ in the Whither theorem. The last method will not be of interest.

Proposition 4.12 *(The Whither theorem) Let c be a configuration and ρ a refinement. Then*

- *starting moves*

  1. $c \xrightarrow{s(a_i)} c'$, $\rho(a) \overset{\mu}{\Longrightarrow} x$ *implies* $c\rho \overset{\mu}{\Longrightarrow} c'\rho[a_i \mapsto x]$
  2. *if* $a_i \neq b_j$ *then* $c \xrightarrow{s(a_i)}\xrightarrow{s(b_j)} c'$, $\rho(a_i) \overset{a'}{\Longrightarrow} x$, $\rho(b_j) \overset{\overline{a'}}{\Longrightarrow} y$ *implies* $c\rho \overset{\tau}{\Longrightarrow} c'\rho[a_i \mapsto x, b_j \mapsto y]$
  3. *if* b_j *occurs in* c *then* $c \xrightarrow{s(a_i)} c'$, $\rho(a) \overset{a'}{\Longrightarrow} x$, $\rho(b_j) \overset{\overline{a'}}{\Longrightarrow} y$ *implies* $c\rho \overset{\tau}{\Longrightarrow} c'\rho[a_i \mapsto x, b_j \mapsto y]$

- *continuing moves*

  4. *if* a_j *occurs in* c *then* $\rho(a_j) \overset{\mu}{\Longrightarrow} x$ *implies* $c\rho \overset{\mu}{\Longrightarrow} c\rho[a_i \mapsto x]$
  5. *If* a_i, b_j *occur in* c *and* $a_i \neq b_j$ *then* $\rho(a_i) \overset{a'}{\Longrightarrow} x$, $\rho(b_i) \overset{\overline{a'}}{\Longrightarrow} y$ *imply* $c\rho \overset{\tau}{\Longrightarrow} c\rho[a_i \mapsto x, b_j \mapsto y]$

- *only c moving*

  6. $c \xrightarrow{\tau} c'$ *implies* $c\rho \overset{\tau}{\Longrightarrow} c'\rho$.

Proof: The first two statements are proved directly by induction on the length of the proof of $c \xrightarrow{s(a_i)} c'$; however the first is needed to prove the second. The fourth statement is proved by structural induction on c and note that in this case c can not be of the form $rec\, x.\ t$. Statements (3) and (5) are also proved by structural induction on c and (1) and (4) are needed in the proof of (3) while only (4) is used in that of (5). The final statement is proved by induction on the length of the proof of the derivation $c \xrightarrow{\tau} c'$. However in this last case an auxilary lemma is required, namely: $c \xrightarrow{a} c'$, $\rho(a) \overset{s}{\Longrightarrow} r\surd$ implies $c\rho \overset{s}{\Longrightarrow} c'\rho$ and for this to be true it is essential that ρ be standard.
In each case the details follows those in Theorem 4.3 of [AH91a]. $\square$

As an example of the use of these results we can show that stability is preserved by refinement.

Theorem 4.13 *If $p \mathrel{\underset{c}{\overset{c}{\sqsubset\!\!\!\sim}}} q$ and $\rho \mathrel{\underset{c}{\overset{c}{\sqsubset\!\!\!\sim}}} \sigma$ then $p\rho$ stable implies $q\rho$ stable.*

Proof: We show that $q\rho \xrightarrow{\tau}$ implies $p\rho \xrightarrow{\tau}$. By the Whence theorem if $q\rho \xrightarrow{\tau}$ then, since $q\rho$ is a process, there are only three possibilities, (1), (2) or (6). As an example consider the second possibility where $q \xrightarrow{s(a_i)}\xrightarrow{s(b_j)}$, $a_i \neq b_j, \rho(a) \xrightarrow{a'}$ and $\rho(b) \xrightarrow{\overline{a'}}$. Since $p \mathrel{\underset{c}{\overset{c}{\sqsubset\!\!\!\sim}}} q$ it follows that $p \overset{s(a_i)}{\Longrightarrow}\overset{s(b_j)}{\Longrightarrow}$ and since $\rho \mathrel{\underset{c}{\overset{c}{\sqsubset\!\!\!\sim}}} \sigma$ we also have that $\rho(a) \overset{a'}{\Longrightarrow}$ and $\rho(b) \overset{\overline{a'}}{\Longrightarrow}$. Applying Proposition 2.5 we may assume $p \overset{\tau}{\Longrightarrow}\xrightarrow{s(a_i)}\xrightarrow{s(b_j)}$ and if $p \xrightarrow{\tau}$ then the last case of the Whither Theorem gives $p\rho \xrightarrow{\tau}$ immediately. So we may assume $p \xrightarrow{s(a_i)}\xrightarrow{s(b_j)}$ and the second case of the Whither theorem also gives $p\rho \xrightarrow{\tau}$. $\square$

Since $\ll_{st}$ is defined in terms of sequences the Whence and Whither theorems must be generalised to sequences. This is not particularly easy but it helps somewhat if we confine our interest to st-sequences. The idea is to define a predicate which relates tuples of sequences of contributions from the action refinement ρ and the underlying process p to the sequence produced by the refined process $p\rho$. However in the production of a sequence of moves from $p\rho$ we may have used the fact that for some a $\rho(a)$ can successfully terminate after certain sequences of moves. (Recall that in order to derive moves from $p; q$ it is necessary to know if p has successfully terminated.) So in general the *merge* predicate, $\mathcal{M}_T$, is parameterised by a *termination set* T which is a set of pairs of the form (a, s). Each action refinement gives rise to a particular termination set $T(\rho)$ defined by $\{\, (a, s) \mid \rho(a)\surd s \,\}$. $\mathcal{M}_T$ will have three components:

$$< u, \mathbf{r}, s >$$

where

1. $u \in LAct^*$ records the contribution from the process being refined
2. $s \in LAct^*$ is the sequence produced by the refined process
3. $\mathbf{r}$ is a partial function from $Act \times N$ to $LAct^+_\tau$

   $\mathbf{r}(a_i)$ records the contribution of $\rho(a_i)$ to the computation of s which may be a non-empty sequence of elements from $LAct$ or simply τ, representing the some unknown internal activity.

The definition of this predicate is obtained by reading off from the Whence theorem the possible ways in which derivations from $c\rho$ can be extended. To make the definition more readable we assume that, since the relation is only defined over st-sequences, the clauses in the definition can only be applied if the components in the resulting tuple are also st-sequences. We also use the same approach to appending μ to a sequence s; when $\mu = \tau$ it acts like a right unit both when applied to elements of $LAct^*$ and $LAct^+_\tau$. We use $\mathbf{r}[a_i{+}\mu]$ to denote the function which results from modifying $\mathbf{r}$ so the new function now maps a_i to the result of appending μ to $\mathbf{r}(a_i)$; it is only defined when a_i is in the domain of $\mathbf{r}$. However we use $\mathbf{r}[a_i{\leftarrow}\mu]$, where a_i is not in the domain of $\mathbf{r}$, to denote the extension of $\mathbf{r}$, which now maps a_i to μ.

Definition 4.14 Let $\mathcal{M}_T$ be the least predicate such that $< \varepsilon, \emptyset, \varepsilon > \in \mathcal{M}_\emptyset$ and whenever $< u, \mathbf{r}, s > \in \mathcal{M}_T$ then

1. $< u.s(a_i), \mathbf{r}[a_i{\leftarrow}\mu], s.\mu > \in \mathcal{M}_T$
2. $< u.s(a_i).s(b_j), \mathbf{r}[a_i{\leftarrow}a', b_j{\leftarrow}\overline{a'}], s > \in \mathcal{M}_T$
3. if b_j active in u then $< u.s(a_i), \mathbf{r}[a_i{\leftarrow}a', b_j{+}\overline{a'}], s > \in \mathcal{M}_T$
4. if a_i active in u then $< u, \mathbf{r}[a_i{+}\alpha], s.\alpha > \in \mathcal{M}_T$
5. if a_i, b_j active in u, for $a_i \neq b_j$, then $< u, \mathbf{r}[a_i{+}a', b_j{+}\overline{a'}], s > \in \mathcal{M}_T$
6. $< u.f(a_i), \mathbf{r}, s > \in \mathcal{M}_{T\cup\{(a,\mathbf{r}(a_i))\}}$

$\square$

One can prove various properties of $\mathcal{M}_T$ by induction on its definition. For example if $< u, \mathbf{r}, s > \in \mathcal{M}_T$ then a_i is in the domain of $\mathbf{r}$ if and only if $s(a_i)$ appears in u. Also if $< u, \mathbf{r}, s > \in \mathcal{M}_T$ then $(a, s') \in T$ if and only if there is some $f(a_i)$ in u such that $\mathbf{r}(a_i) = s'$. The main property of $\mathcal{M}_T$ is that it can record the manner in which derivations are made from processes of the form $p\rho$. The next two theorems state that $p\rho$ can perform a sequence s from $LAct^*$ essentially if and only if $< u, \mathbf{r}, s > \in \mathcal{M}_T$ for some $T \subseteq T(\rho)$ where u and $\mathbf{r}$ record the contributions of p and ρ to the computation of s. The many extra conditions in both theorems are required in order to carry out the proofs using induction.

Theorem 4.15 *(The Decomposition Theorem)*
If (p, ρ) is proper and $p\rho \overset{s}{\Longrightarrow} \overline{c}$ then there exists a configuration c, a refinement ρ' and a tuple $< u, \mathbf{r}, s > \in \mathcal{M}_T$ such that

1. $\overline{c} \equiv c\rho'$
2. $p \overset{u}{\Longrightarrow} c$
3. $\rho(a) \overset{\mathbf{r}(a_i)}{\Longrightarrow} \rho'(a_i)$ *for every* $a_i \in L(c)$
4. $T \subseteq T(\rho)$
5. (c, ρ') *is proper and* ρ *and* ρ' *are compatible.*

Proof: The proof is by induction on the size of the derivation $p\rho \overset{s}{\Longrightarrow} \overline{c}$ i.e. the total number of inference rules in the proofs of all the individual moves which make up the derivation. If the number is zero then the theorem is trivially satisfied since $< \varepsilon, \emptyset, \varepsilon > \in \mathcal{M}_\emptyset$. So we can assume that for some configuration $\overline{c_1}$ $p\rho \overset{s'}{\Longrightarrow} \overline{c_1} \xrightarrow{\mu} \overline{c}$. By considering the derivation $p\rho \overset{s'}{\Longrightarrow} \overline{c_1}$ we may apply induction to obtain a configuration d_1, a refinement ρ_1 and a tuple $< u', \mathbf{r}', s' > \in \mathcal{M}_{T'}$ such that

1. $\overline{c_1} \equiv d\rho_1$
2. $p \overset{u'}{\Longrightarrow} d$
3. $\rho(a) \overset{\mathbf{r}'(a_i)}{\Longrightarrow} \rho_1(a_i)$ for every $a_i \in L(d)$
4. $T' \subseteq T(\rho)$
5. (d, ρ_1) is proper and ρ, ρ_1 are compatible.

We can assume $d\rho_1 \xrightarrow{\mu} \overline{c}$ and we examine the way in which this final move can be inferred. The Whence theorem gives us seven possibilities for this move but in fact the last one is not possible because the pair (d, ρ_1) is proper and in the sixth case the existing element of $\mathcal{M}_T$ $< u', \mathbf{r}', s' >$ will satisfy the requirements of the theorem. In each of the other cases we employ a corresponding clause in the definition of $\mathcal{M}_T$. However in certain cases there may be some tidying up to do, using the final clause in the definition

of $\mathcal{M}_T$, to ensure that the pair (c, ρ') is proper.
As an example suppose that clause (1) of the Whence theorem applies, $d \xrightarrow{s(a_i)} d''$, $\rho_1(a) \xrightarrow{\mu} x$ and $\overline{c} \equiv d''\rho_1[a_i \mapsto x]$. Here we apply clause (1) in the definition of $\mathcal{M}_T$ to obtain $< u'.s(a_i), \mathbf{r}'[a_i{\leftarrow}\mu], s'.\mu > \in \mathcal{M}_{T'}$ and one can check that the requirements are satisfied with $c = d''$, $u = u'.s(a_i)$, $s = s'.\mu$, $\rho' = \rho_1[a_i \mapsto x]$ and $\mathbf{r} = \mathbf{r}'[a_i{\leftarrow}\mu]$, except possibly that (d'', ρ') is not proper because $x\surd$. In this case let d' be such that $d'' \xrightarrow{f(a_i)} d'$. By Lemma 4.9 we have $\overline{c} \equiv d'\rho'$ and employing the last clause in the definition of $\mathcal{M}_T$ we have $< u'.s(a_i).f(a_i), \mathbf{r}, s > \in \mathcal{M}_{T' \cup \{(a, \mathbf{r}(a_i))\}}$. In this case the requirements are also satisfied since $T' \cup \{(a, \mathbf{r}(a_i))\} \subseteq T(\rho)$. □

Not only may $\mathcal{M}_T$ be used to record the derivation of sequences of moves from terms of the form $p\rho$ but it captures precisely when such sequences can be made; the decomposition theorem has a converse:

Theorem 4.16 *(The Composition Theorem)*
Suppose $< u, \mathbf{r}, s > \in \mathcal{M}_T$ *where* $T \subseteq T(\rho)$. *If*

1. $p \overset{u}{\Longrightarrow} c$ *where* $c \in \mathcal{C}$
2. *for every* $a_i \in L(c)$ $\rho(a) \overset{\mathbf{r}(a_i)}{\Longrightarrow} \rho'(a_i)$
3. (p, ρ), (c, ρ') *and* ρ, ρ' *are all compatible*

then $p\rho \overset{s}{\Longrightarrow} c\rho'$.

Proof: This time we use induction on the proof that $< u, \mathbf{r}, s > \in \mathcal{M}_T$.
First consider the base case, $< \varepsilon, \emptyset, \varepsilon > \in \mathcal{M}_\emptyset$.
Here $p \overset{\varepsilon}{\Longrightarrow} c$ and c must be a process and therefore ρ must coincide with ρ'. So $p\rho \overset{\varepsilon}{\Longrightarrow} c'\rho'$ follows by possibly more than one application of the final clause of the Whither theorem.

Otherwise there are six cases depending on how membership in $\mathcal{M}_T$ is inferred. In all but the last case we can apply induction followed by a corresponding clause in the Whither theorem; the last case is treated separately using clause (6). As examples we look at two cases.

1. if a_i, b_j active in u, $a_i \neq b_j$, then $< u, \mathbf{r}[a_i{+}a, b_j{+}\overline{a}], s > \in \mathcal{M}_T$ because $< u, \mathbf{r}, s > \in \mathcal{M}_T$.
   Let the derivations from $\rho(a), \rho(b)$ be of the form
   $$\rho(a) \overset{\mathbf{r}(a_i)}{\Longrightarrow} x \overset{a'}{\Longrightarrow} \rho'(a_i)$$
   $$\rho(b) \overset{\mathbf{r}(b_j)}{\Longrightarrow} y \overset{\overline{a'}}{\Longrightarrow} \rho'(b_j)$$
   and let ρ_1 denote $\rho'[a_i \mapsto x, b_j \mapsto y]$. By induction we have $p\rho \overset{s}{\Longrightarrow} c\rho_1$. By Lemma 3.3 we can apply clause (5) of the Whither Theorem to obtain $c\rho_1 \overset{\tau}{\Longrightarrow} c\rho'$ and therefore $p\rho \overset{s}{\Longrightarrow} c\rho'$.

2. $< u.f(a_i), \mathbf{r}, s > \in \mathcal{M}_{T \cup (a, \mathbf{r}(a_i))}$ because $< u, \mathbf{r}, s > \in \mathcal{M}_T$.
   Let c_1, c_2 be such that $p \overset{u}{\Longrightarrow} c_1 \xrightarrow{f(a_i)} c_2 \overset{\varepsilon}{\Longrightarrow} c$. Now by the statement of the theorem

$\rho(a)\surd\mathbf{r}(a_i)$. So let ρ'_1 be the refinement $\rho'[a_i \mapsto x]$ where x is any process such that $\rho(a) \xrightarrow{\mathbf{r}(a_i)} x\surd$. Then by induction $p\rho \overset{s}{\Longrightarrow} c_1\rho'_1$. by Lemma 4.9 $c_1\rho'_1 \equiv c_2\rho'_1 \equiv c_2\rho'$ and by repeated application of clause (6) of the Whither Theorem $c_2\rho' \overset{\varepsilon}{\Longrightarrow} c\rho'$. It follows that (up to $\equiv$) $p\rho \overset{s}{\Longrightarrow} c\rho'$. $\square$

These composition and decomposition results enable us to analyse and compare arbitrary sequences of external moves from processes of the form $p\rho$ but we also need to be able to analyse why such processes diverge internally. The following proposition tells us conditions under which we can expect $p\rho \Uparrow s$. These conditions seem rather special but they occur in various places during the proof of the refinement theorem.
First some notation. Let us write $T \prec T(\rho)$ if $(a,s) \in T$ implies that either $(a,s) \in T(\rho)$ or $\rho(a) \Uparrow s$. One can check that if $T \subseteq T(\sigma)$ and $\rho \underset{\sim_c}{\sqsubset} \sigma$ then $T \prec T(\rho)$.

Proposition 4.17 *Suppose* $< u, \mathbf{r}, s > \in \mathcal{M}_T$ *where* $T \prec T(\rho)$ *and suppose that for some* c

1. $p \Uparrow u$ *or* $p \overset{u}{\Longrightarrow} c$
2. *for every* $a_i \in L(c)$ *either* $\rho(a) \Uparrow \mathbf{r}(a_i)$ *or* $\rho(a) \overset{\mathbf{r}(a_i)}{\Longrightarrow}$
3. *one of the following holds:*
   (a) *for some* $a \in S(c)$ $\rho(a) \Uparrow$
   (b) *for some* $a_i \in L(c)$ $\rho(a) \Uparrow \mathbf{r}(a_i)$
   (c) $p \Uparrow u$

Then $p\rho \Uparrow s$.

Proof: By induction on why $< u, \mathbf{r}, s > \in \mathcal{M}_T$. The base case $< \varepsilon, \emptyset, \varepsilon > \in \mathcal{M}_\emptyset$ is trivial. For the induction step there are six cases, one for each way of inferring membership in $\mathcal{M}_T$. As an example we consider one case:

$$< u.s(a_i), \mathbf{r}[a_i \leftarrow \mu], s.\mu > \in \mathcal{M}_T \quad \text{because} \quad < u, \mathbf{r}, s > \in \mathcal{M}_T.$$

If $p \Uparrow u$ or $\rho(a) \Uparrow \mathbf{r}(a_j)$ for any j then we can apply induction to obtain $p\rho \Uparrow s$ and therefore $p\rho \Uparrow s.\mu$. Otherwise we have $p \overset{u}{\Longrightarrow} c' \xrightarrow{s(a_i)} c''$ for some c', c'' and $\rho(a) \overset{\mathbf{r}(a_j)}{\Longrightarrow} x_{a_j}$ for every $a_j \in L(c)$. Under these circumstances if $\rho(a) \Uparrow s'$ for some $(a, s') \in T$ then one can show by induction on why $< u, \mathbf{r}, s > \in \mathcal{M}_T$ that $p\rho \Uparrow s$. So we can assume that $T \subseteq T(\rho)$ and therefore apply the Composition theorem to obtain $p\rho \overset{s}{\Longrightarrow} c'\rho'$ where ρ' is defined so as to be compatible with ρ and such that $\rho'(a_j) = x_{a_j}$ for each a_j in the domain of $\mathbf{r}$.
We know that $c' \xrightarrow{s(a_i)} c''$ and $\rho'(a_i) \overset{\mu}{\Longrightarrow} x$ for some x. Applying clause (1) of the Whither theorem we obtain $c'\rho \overset{\mu}{\Longrightarrow} c''\rho'[a_i \mapsto x]$. Now if $x \Uparrow$ or $c'' \Uparrow$ again the Whither theorem shows that $c''\rho'[a_i \mapsto x] \Uparrow$ and therefore $p\rho \Uparrow s.\mu$. The only remaining possibility is that $\rho(a) \Uparrow$ for some $a \in S(c'')$ in which case since $\rho(a) = \rho'(a)$ it again follows that $c''\rho'[a_i \mapsto x] \Uparrow$ and so $p\rho \Uparrow s.\mu$. $\square$

Unfortunately this is not the only result we need about divergence. This is because $c\rho$ may diverge although c and $\rho(a)$, for every a, are well behaved processes which never diverge. As a simple example let c be $rec\,x.\ a;x \mid rec\,x.\ b;x$ with ρ the standard refinement defined by $\rho(a) = a + a'$, $\rho(b) = b + \overline{a'}$. Note that ρ is not stable. There are similar forms of divergence for stable refinements although the behaviour is not quite as complicated. For example let c be $a \mid b$ and ρ the stable refinement defined by $\rho(a) = c; rec\,x.\ d;x$ and $\rho(b) = c'; rec\,x.\ \overline{d};x$. It is this form of divergence from $c\rho$ which requires the most careful analysis.

Definition 4.18 The pair (c,ρ) *weakly converges*, written $(c,\rho)\Downarrow$, if for every $\overline{c}$ such that $c\rho \overset{\varepsilon}{\Longrightarrow} \overline{c}$ and pair (c',ρ') such that $\overline{c} \equiv c'\rho'$ $\quad c'\Downarrow$ and $\rho'(a_i)\Downarrow$, $\rho(a)\Downarrow$ for every $a_i \in L(c')$, $a \in S(c')$ respectively. □

We examine why $c\rho$ diverges for weakly convergent pairs (c,ρ). For the sake of generality this analysis is carried out for standard ρ although the situation is slightly less complicatied when ρ is assumed to be in addition stable. The labelled transition system which defines the operational semantics of BP is finite branching and therefore a term diverges if and only if it can produce the sequence of moves τ^n for every $n \geq 0$. So in analogy with the predicate $\mathcal{M}_T$ we define a predicate $\mathcal{N}_T$ which characterises when terms of the form $c\rho$ can perform the sequence τ^n. Specifically $c\rho \overset{\tau^n}{\Longrightarrow}$ if and only if there is a triple $< u, \mathbf{r}, n > \in \mathcal{N}_T$ for some particular T such that $c \overset{u}{\Longrightarrow}$ and $\rho(a) \overset{\mathbf{r}(a_i)}{\Longrightarrow}$ for every a_i in the domain of $\mathbf{r}$.

Definition 4.19 Let $\mathcal{N}_T$ be the least predicate such that $< \varepsilon, \emptyset, 0 > \in \mathcal{N}_\emptyset$ and whenever $< u, \mathbf{r}, n > \in \mathcal{N}_T$ then

1. $< u.s(a_i), \mathbf{r}[a_i \leftarrow \tau, n+1 > \in \mathcal{N}_T$
2. $< u.s(a_i).s(b_j), \mathbf{r}[a_i \leftarrow a', b_j \leftarrow \overline{a'}], n+1 > \in \mathcal{N}_T$
3. if b_j is active in u then $< u.s(a_i), \mathbf{r}[a_i \leftarrow a', b_j + \overline{a'}], n+1 > \in \mathcal{N}_T$
4. if a_i, b_j are active in u, for $a_i \neq b_j$, then $< u, \mathbf{r}[a_i + a', b_j + \overline{a'}], n+1 > \in \mathcal{N}_T$
5. $< u.f(a_i), \mathbf{r}, n > \in \mathcal{N}_{T \cup \{(a, \mathbf{r}(a_i))\}}$

□

The corresponding composition theorem for $\mathcal{N}_T$ is:

Theorem 4.20 *(The Composition Theorem for $\mathcal{N}_T$)*
Suppose $< u, \mathbf{r}, n > \in \mathcal{N}_T$ where $T \subseteq T(\rho)$. If

1. $c \overset{u}{\Longrightarrow} c'$
2. *for every $a_i \in L(c') - L(c)$ $\rho(a) \overset{\mathbf{r}(a_i)}{\Longrightarrow} \rho'(a_i)$*
3. *for every $a_i \in L(c') \cap L(c) \cap domain(\mathbf{r})$ $\rho(a_i) \overset{\mathbf{r}(a_i)}{\Longrightarrow} \rho'(a_i)$*
4. *(p,ρ), (c,ρ') and ρ, ρ' are all compatible*

then $c\rho \overset{\tau^n}{\Longrightarrow} c\rho'$.

Proof: Similar to that of Theorem 4.16. $\square$

The decomposition theorem for $\mathcal{N}_T$ applies to weakly convergent pairs and is slightly different in form from that for $\mathcal{M}_T$. It basically says that if $d\rho \Uparrow$ then for every n there exists some $< u, \mathbf{r}, n > \in \mathcal{N}_T$ where u records the contribution of d and $\mathbf{r}$ that of ρ to some internal computation of length at least n. The other components in the statement of the theorem are required for the inductive proof.

Theorem 4.21 *(The Decomposition Theorem for $\mathcal{N}_T$)*
Suppose (d, ρ) is proper and weakly convergent. If $d\rho \Uparrow$ then for every $n \geq 0$ there exists a configuration d', a refinement ρ' and a tuple $< u, \mathbf{r}, n > \in \mathcal{N}_T$ such that

1. $d'\rho' \Uparrow$
2. $d\rho \overset{\tau^n}{\Longrightarrow} d'\rho'$
3. $d \overset{u}{\Longrightarrow} d'$
4. $\rho(a) \overset{\mathbf{r}(a_i)}{\Longrightarrow} \rho'(a_i)$ *for every* $a_i \in L(d') - L(d)$
   and $\rho(a_i) \overset{\mathbf{r}(a_i)}{\Longrightarrow} \rho'(a_i)$ *for every* $a_i \in L(d') \cap L(d) \cap domain(\mathbf{r})$.
5. $T \subseteq T(\rho)$
6. *(d', ρ') is proper and ρ and ρ' are compatible.*

Proof: By induction on n. If $n = 0$ the required tuple is $< \varepsilon, \emptyset, 0 > \in \mathcal{N}_\emptyset$ and the pair (d', ρ') is (d, ρ) itself. So assume the result is true for k; we prove it true for $k + 1$.
By induction we know that $d\rho \overset{\tau^k}{\Longrightarrow} d_1\rho_1$ for some $d_1,\ \rho_1$ and there exists some $< u, \mathbf{r}, s > \in \mathcal{N}_T$ such that

1. $d_1\rho_1 \Uparrow$
2. $d\rho \overset{\tau^n}{\Longrightarrow} d_1\rho_1$
3. $\rho(a) \overset{\mathbf{r}(a_i)}{\Longrightarrow} \rho_1(a_i)$ for every $a_i \in L(d_1) - L(d)$
   and $\rho(a_i) \overset{\mathbf{r}(a_i)}{\Longrightarrow} \rho_1(a_i)$ for every $a_i \in L(d_1) \cap L(d) \cap domain(\mathbf{r})$
4. $T \subseteq T(\rho)$
5. (d_1, ρ_1) is proper and ρ and ρ_1 are compatible.

To prove the result it is sufficient to show that exists a (d', ρ') and a T' such that

1. $d_1\rho_1 \overset{\tau}{\Longrightarrow} d'\rho'$
2. $d'\rho' \Uparrow$
3. there exists an extension u' of u and an extension $\mathbf{r}'$ of $\mathbf{r}$ such that
   (a) $< u', \mathbf{r}', k + 1 > \in \mathcal{N}_{T'}$

(b) $d_1 \overset{u'/u}{\Longrightarrow} d'$

(c) $\rho(a) \overset{\mathbf{r}'(a_i)}{\Longrightarrow} \rho'(a_i)$ for all $a_i \in L(d') - L(d)$
and $\rho(a_i) \overset{\mathbf{r}'(a_i)}{\Longrightarrow} \rho'(a_i)$ for all $a_i \in L(d') \cap L(d) \cap domain(\mathbf{r}')$

4. $T \subseteq T' \subseteq T(\rho)$

5. (d', ρ') is proper and ρ, ρ' are compatible.

This ensures that the requirements of the theorem are satisfied for $k+1$.

We show that the required d', ρ' exist by using induction on a measure of the length of the internal derivations which d_1 and the relevant $\rho_1(\nu)$ can perform. For any convergent configuration d let $\tau(d)$ be the length of the longest τ-derivation it can perform and for ν of the form a or a_i let $occ(\nu, d)$ be the number of top-level occurrences of ν in d. We then define $size(d, \rho)$ to be

$$\tau(d) + \sum_{\nu} \{ occ(\nu, d) * \tau(\rho(name(\nu))) \}$$

where $name(a) = name(a_i) = a$. If $(d, \rho) \Downarrow$ then $size(d, \rho)$ is finite and we use induction on this measure.

Since $d_1\rho_1 \Uparrow$ we know that $d_1\rho_1 \xrightarrow{\tau} \overline{c}$ for some $\overline{c}$ such that $\overline{c} \Uparrow$. If we apply the Whence theorem to this derivation there are six possibilities; the last is ruled out because (d_1, ρ_1) is proper. For each of these six possibilities we can either use induction on *size* or else we can apply an appropriate clause in the definition of $\mathcal{N}_T$. We consider two example cases.

1. a_i occurs in c, $\rho_1(a) \overset{\tau}{\Longrightarrow} x$ and $\overline{c} \equiv d_1\rho_1'$ where ρ_1' denotes $\rho_1[a_i \mapsto x]$.
Here $size(d_1, \rho_1') < size(d_1, \rho_1)$ and therefore if (d_1, ρ_1') is proper we may apply induction to obtain a d', ρ' which satisfy the requirements for d_1, ρ_1'; however they will also satisy them for d_1, ρ. If (d_1, ρ_1') is not proper then it is because $x\surd$. Let $d_1 \xrightarrow{f(a_i)} d'$. Then by Lemma 4.9 $d_1\rho_1[a_i \mapsto x] \equiv d'\rho_1$ and $< u.f(a_i), \mathbf{r}, k > \in \mathcal{N}_{T \cup \{(a, \mathbf{r}(a_i))\}}$. Since $\rho(a) \overset{\mathbf{r}(a_i)}{\Longrightarrow} \rho_1(a_i) \xrightarrow{\tau} x$ it follows that $T \cup \{(a, \mathbf{r}(a_i))\} \subseteq T(\rho)$ and therefore the requirements are satisfied.

2. $d_1 \xrightarrow{s(a_i)} \xrightarrow{s(b_j)} d_1'$, $\rho_1(a) \xrightarrow{a'} x$, $\rho_1(b) \xrightarrow{\overline{a'}} y$, and $\overline{c} \equiv d_1'\rho_1[a_i \mapsto x, b_j \mapsto y]$.
It follows that $< u.s(a_i).s(a_j), \mathbf{r}[a_i \leftarrow a', b_j \leftarrow \overline{a'}], k+1 > \in \mathcal{N}_T$, from clause (2) of the definition of $\mathcal{N}_T$. The required u', $\mathbf{r}'$ are u, $\mathbf{r}[a_i \leftarrow a', b_j \leftarrow \overline{a'}]$ respectively and the Whither theorem gives $d_1\rho_1 \xrightarrow{i} d_1'\rho_1[a_i \mapsto x, b_j \mapsto y])$. But $(d_1', \rho_1[a_i \mapsto x, b_j \mapsto y])$ may not be proper; for example it may be that $x\surd$. In this case let d' be such that $d_1' \xrightarrow{f(a_i)} d'$. Then by Lemma 4.9 $d'\rho_1[a_i \mapsto x, b_j \mapsto y] \equiv d_1'\rho_1[a_i \mapsto x, b_j \mapsto y]$. In this way, by applying the last clause of the definition of $\mathcal{N}_T$, as in the previous case, we can be sure of obtaining a proper pair satisfying the requirements. □

With these results we now have all the required information to prove the main theorem of this section.

Theorem 4.22 *(The Refinement Theorem) For every pair of stable refinements* ρ, σ, $p \sqsubseteq_c^c q$ *and* $\rho \sqsubseteq_c^c \sigma$ *imply* $p\rho \sqsubseteq_c^c q\sigma$.

Proof: We use the alternative characterisation of $\underset{\sim}{\sqsubset}{}_c^c$, expressed in terms of st-sequences and stability. In view of Proposition 4.13 it is sufficient to show $p\rho \ll_{st} q\sigma$. This will follow from $p \ll_{st} q$ and $\rho \underset{\sim}{\sqsubset}{}_c^c \sigma$.
Now for any st-sequence s suppose $p\rho \Downarrow s$. We have to prove three statements:

1. $q\sigma \Downarrow s$.
   We prove the contrapositive: assuming $q\sigma \Uparrow s$ we prove $p\sigma \Uparrow s$. If $q\sigma \Uparrow s$ then for some prefix s' of s $q\sigma \overset{s'}{\Longrightarrow} \overline{c}$ such that $\overline{c} \Uparrow$. To avoid unnecessary complexity we just assume that s' is s.
   To any derivation $q\sigma \overset{s}{\Longrightarrow} \overline{c}$ we can apply the Decomposition theorem to obtain $< u, \mathbf{r}, s > \in \mathcal{M}_T$ for some $T \subseteq T(\sigma)$ such that

   (a) $\overline{c} \equiv c'\sigma'$

   (b) $T \subseteq T(\sigma)$

   (c) $q \overset{u}{\Longrightarrow} c'$

   (d) for every $a_i \in L(c')$ $\sigma(a) \overset{\mathbf{r}(a_i)}{\Longrightarrow} \sigma'(a_i)$.

   In order to prove $p\rho \Uparrow s$ we will try to apply Proposition 4.17 and in most cases we will be successful. Note that becasue $\rho \underset{\sim}{\sqsubset}{}_c \sigma$ we can always assume that for such a decomposition $T \prec T(\rho)$. There are three possibilities:

   - There exist some $\overline{c}$ such that $q\sigma \overset{s}{\Longrightarrow} \overline{c}$ and the above decomposition gives a c' such that $c' \Uparrow$.
     In this case since $p \underset{\sim}{\sqsubset}{}_c q$ it follows that $p \Uparrow u$. From $\rho \underset{\sim}{\sqsubset}{}_c \sigma$ it then follows that all the requirements of Proposition 4.17 hold and we can conclude $p\rho \Uparrow$.
   - There exist some $\overline{c}$ such that $q\sigma \overset{s}{\Longrightarrow} \overline{c}$ and the above decomposition gives either some $a_i \in L(c')$ such that $\sigma'(a_i) \Uparrow$ or some $a \in S(c')$ such that $\sigma(a) \Uparrow$. Again one can show that all the requirements of Proposition 4.17 are satisfied.
   - Otherwise we can assume that in the decomposition above that $(d, \sigma') \Downarrow$.
     Here we use the predicate $\mathcal{N}_T$. By its decomposition theorem we obtain for each $n \geq 0$, $< u_1, \mathbf{r}_1, n > \in \mathcal{N}_{T_1}$ for some $T_1 \subseteq T(\sigma)$ such that
     $c' \overset{u_1}{\Longrightarrow} c_n$
     $\sigma'(a_i) \overset{\mathbf{r}_1(a_i)}{\Longrightarrow}$ for each $a_i \in L(c') \cap L(c_n) \cap domain(\mathbf{r}_1)$
     $\sigma'(a) \overset{\mathbf{r}_1(a_i)}{\Longrightarrow}$ for each $a_i \in L(c_n) - L(c')$.
     Let $u', T', \mathbf{r}'$ denote $u.u_1, T \cup T_1$ and $\mathbf{r} \cdot \mathbf{r}_1$ respectively.
     Then since $p \underset{\sim}{\sqsubset}{}_c q$ and $\rho \underset{\sim}{\sqsubset}{}_c \sigma$ it follows that conditions i) and ii) of Proposition 4.17 are true. If for some n condition iii) is also true then, since one can show (using the definitions of $\mathcal{M}_T$ and $\mathcal{N}_T$) that $< u', \mathbf{r}', s > \in \mathcal{M}_T$, we can apply this proposition to obtain $p\rho \Uparrow s$.
     So we may assume $\rho(a) \Downarrow \mathbf{r}'(a_i)$ for each $a_i \in L(c_n)$ and since $(a, s') \in T'$ implies $s' = \mathbf{r}'(a_j)$ for some j it follows that $T' \subseteq T(\rho)$. So we may apply the Composition theorem for $\mathcal{M}_T$ followed by that for $\mathcal{N}_T$ to obtain $p\rho \overset{s\tau^n}{\Longrightarrow}$.
     Since this is true for each n it follows that $p\rho \Uparrow s$.

2. If $q\surd s$ then $p\surd s$.
   We leave this to the reader.

3. If $A \in \mathcal{A}(q\sigma, s)$ then $B \subseteq A$ for some $B \in \mathcal{A}(p\rho, s)$.
   If $A \in \mathcal{A}(q\sigma, s)$ then $q\sigma \overset{s}{\Longrightarrow} \overline{c}$ for some live and stable $\overline{c}$ such that $A = S(\overline{c})$. Applying the decomposition theorem to this derivation we obtain $< u, \mathbf{r}, s > \in \mathcal{M}_T$ for some $T \subseteq T(\sigma)$ and $\overline{c} \equiv d\sigma'$, where (d, σ') is proper, such that $q \overset{u}{\Longrightarrow} d$ and $\sigma(a) \overset{\mathbf{r}(a_i)}{\Longrightarrow} \sigma'(a_i)$ for every $a_i \in L(d)$. In view of Proposition 4.17 and the fact that $p\rho \Downarrow s$ we may assume that $p \Downarrow u$ and $\rho(a) \Downarrow \mathbf{r}(a_i)$, $\rho(a) \Downarrow$ for each $a \in S(d)$ and $T \subseteq T(\rho)$.
   Since $d\sigma'$ is stable it follows that d, $\sigma'(a_i)$, $\sigma'(a)$ are all stable, for each $a_i \in L(d)$ and $a \in S(d)$. These are also live because $d\sigma'$ is live and (d, σ') is proper.
   So from $p \underset{\sim}{\sqsubset}{}_c^c q$ and $\rho \underset{\sim}{\sqsubset}{}_c \sigma$ we can obtain stable and live c and x_{a_i} such that $p \overset{u}{\Longrightarrow} c$, $\rho(a) \overset{\mathbf{r}(a_i)}{\Longrightarrow} x_{a_i}$ and $S(c) \subseteq S(d)$, $S(x_{a_i}) \subseteq S(\sigma'(a_i))$. Applying the Composition theorem for $\mathcal{M}_T$ we obtain $p\rho \overset{s}{\Longrightarrow} c\rho_1$ where ρ_1 denotes $\rho[a_i \mapsto x_{a_i}]$.

   We show that $c\rho_1$ is stable and for this we need the fact that the refinements are stable. Suppose $c\rho_1 \xrightarrow{\tau}$. We use the Whence theorem to derive a contradiction. This theorem gives seven different possible decompositions for this internal move and examining each in turn we see that none of them are possible. The first is not possible since ρ is stable. In the second decomposition the stability of ρ implies that $a = \overline{b}$ and so by Lemma 2.3 $c \xrightarrow{\tau}$ which contradicts the fact that c is stable. In the third case since $S(c) \subseteq S(d)$ we can use the third clause of the Whither theorem to show that $d\sigma' \xrightarrow{\tau}$ which contradicts the stability of $d\sigma'$. A similar arguement rules out the fifth case while the fourth is impossible because each x_{a_i} is stable. The stabilty of c rules out the sixth case and finally the last case is not possible since each x_{a_i} is live.

   One can also check that $c\rho_1$ is live and by construction $S(c\rho_1) \subseteq A$. The result now follows since $p\rho \overset{s}{\Longrightarrow} c\rho_1$.

$\square$

5 Characterising ST-Testing

The more standard notion of testing, based on complete, non-divisible actions as in [Hen88] may also be applied to our language; we denote the resulting preorder by $\underset{\sim}{\sqsubset}{}_s$. Because only complete actions are used this equivalence does not distinguish between nondeterminism and concurrency. It is also not preserved by action refinement.

In this section we examine the relationship between $\underset{\sim}{\sqsubset}{}_c$ and $\underset{\sim}{\sqsubset}{}_s$. The main result is that $\underset{\sim}{\sqsubset}{}_c^c$ may be characterised by $\underset{\sim}{\sqsubset}{}_s^c$ and action refinement. Specifically $\underset{\sim}{\sqsubset}{}_c^c$ satisfies the properties

1. it is contained in $\underset{\sim}{\sqsubset}{}_s^c$
2. it is preserved by stable action refinements.

We prove that $\underset{\sim}{\sqsubset}{}_c^c$ is the largest relation with these two properties.

Let us first define the testing relation $\underset{\sim_s}{\sqsubset}$. Although it may be obtained by restricting the tests in section 2 it makes the development clearer if we reiterate the various definitions. For each $\nu \in Act_\tau$ let $\xrightarrow{\nu}_s$ be the least relation over BP which satisfies all of the requirements $(O1)$ to $(O7)$ in Figure 1 with the single change that in $(O1)$ the second and third conditions are dropped. So in $\xrightarrow{\nu}_s$ only complete actions are ever used. Sequential testing is now defined using this operational semantics. We use as sequential tests the set of processes BP_Γ where

$$\Gamma = Act \cup \{term, \omega\}.$$

So the tests are also only able to perform complete actions although they can still detect the successful termination of a process.

Definition 5.1 Let $\mapsto_s$ be the least relation between experimental states of the form $e \parallel p$ where e is a sequential test and p a process which satisfies

1. $e \xrightarrow{a}_s e'$, $p \xrightarrow{a}_s p'$ implies $e \parallel p \mapsto_s e' \parallel p'$ for every a in Act
2. $e \xrightarrow{\tau}_s e'$ implies $e \parallel p \mapsto_s e' \parallel p$
3. $p \xrightarrow{\tau}_s p'$ implies $e \parallel p \mapsto_s e \parallel p'$
4. $e \xrightarrow{term}_s e'$, $p\surd$ implies $e \parallel p \mapsto_s e' \parallel p$

□

A sequential computation is then a maximal sequence of the form

$$e \parallel p \equiv e_0 \parallel p_0 \mapsto_s e_1 \parallel p_1 \mapsto_s \dots$$

and it is successful if, as usual, it contains a successful state. Then $p\ must_s\ e$ if every computation from $e \parallel p$ is successful. Finally $p \underset{\sim_s}{\sqsubset} q$ if for all sequential tests e $p\ must_s\ e$ implies $q\ must_s\ e$.

We will not develop to any great extent the theory of this behavioural preorder; it is similar to that in [DH84] except that here it is applied to a slightly more general language which has two kinds of termination and sequential composition. However it can be characterised in terms of acceptance sets along the lines of the alternative characterisation of $\underset{\sim_c}{\sqsubset}$ given in section 3; it is sufficient to restrict attention to sequences from Act:

Definition 5.2 For processes p, q we write $p \ll q$ if for every $s \in Act^*$

$$p \downarrow s \quad \text{implies} \quad \begin{array}{l} i)\ q \downarrow s \\ iii)\ q\surd s \text{ implies } p\surd s \\ ii)\ \mathcal{A}(p,s) \ll \mathcal{A}(q,s). \end{array}$$

□

We state without proof:

Theorem 5.3 *(Alternative Characterisation for sequential testing)*
For all processes p, q $p \underset{\sim_s}{\sqsubset} q$ if and only if $p \ll q$. □

As an immediate corollary we have

Corollary 5.4 *For all processes p, q $p \underset{\sim_c}{\sqsubset} q$ implies $p \underset{\sim_s}{\sqsubset} q$.* $\square$

Let $\underset{\sim_s}{\sqsubset}^c$ be defined in the obvious way from $\underset{\sim_s}{\sqsubset}$: $p \underset{\sim_s}{\sqsubset}^c q$ if $p \underset{\sim_s}{\sqsubset} q$ and p *stable* implies q *stable*. It follows trivially that $\underset{\sim_c}{\sqsubset}^c \subseteq \underset{\sim_s}{\sqsubset}^c$. We also know that $\underset{\sim_c}{\sqsubset}^c$ is preserved by action refinement and therefore the main result of this section will follow if we can construct a particular stable refinement σ with the property that for all processes p, q, $p\sigma \ll q\sigma$ implies $p \ll_{st} q$. We define σ as a mapping

$$\sigma\colon Act \longmapsto BP_\Delta$$

where Δ is the set of actions defined in section 3; it contains all the elements of Act and all labelled begin and end actions together with their complements. Then σ is defined by

$$\sigma(a) = a + \sum\{\, s(a_k); f(a_k) \mid k \in K \,\} \text{ for } K \subseteq L \text{ a sufficiently large finite set.}$$

Note that σ is a stable refinement. The use of the alphabet Δ leads to some blurring of the distinction between processes and configurations as a configuration for the process language BP_{Act} is a process for the language BP_Δ. However this is the basic reason why the concurrent behaviour of processes in BP_{Act} can be simulated by the sequential behaviour of proceses in BP_Δ.

We proceed by proving a sequence of lemmas which relate the sequential operational semantics of $p\sigma$ to the st-operational semantics of p.

For any `st`-sequence u let $\mathbf{r}_u$, a partial function from $Act \times N$ to $LAct^+$ be defined by

$$\mathbf{r}_u(a_i) = u \mid \{s(a_i), f(a_i)\} \text{ if } s(a_i) \text{ or } f(a_i) \text{ appear in } u.$$

Also let $T_s = \{\, (a, s(a_i); f(a_i)) \mid i \in N, a \in Act\}$.

Lemma 5.5 *For every `st`-sequence u there exists a finite subset T of T_s such that $< u, \mathbf{r}_u, u > \in \mathcal{M}_T$.*

Proof: By induction on u. $\square$

For any configuration c we say that the action refinement σ' is an *extension of σ compatible with c* if it is compatible with σ, i. e. coincides with σ on all the elements of Act, and otherwise is characterised by

$$\sigma'(a) = \begin{cases} f(a_j) \text{ for some } j & a_i \in L(c) \\ nil & a_i \notin L(c) \end{cases}$$

A particular extension of σ compatible with c, σ_c, may be defined by letting $\sigma_c(a_i) = f(a_i)$ for each $a_i \in L(c)$. Then as an immediate corollary we have

Corollary 5.6 *For any `st`-sequence u if $p \overset{u}{\Longrightarrow} c$ then $p\sigma \overset{u}{\Longrightarrow}_s c\sigma_c$.*

Proof: We use the Composition theorem for $\mathcal{M}_T$ and we have to assume that the K used in the definition of σ contains any index used in u.
By the previous lemma there exists a $T \subseteq T_s$ such that $< u, \mathbf{r}_u, u > \in \mathcal{M}_T$ and by definition $T \subseteq T(\sigma)$. So all the requirements of the composition theorem hold and we may conclude $p \overset{u}{\Longrightarrow}_s c\sigma_c$. □

We now examine sequential derivations from processes in BP_Δ of the form $p\sigma$.

Lemma 5.7 *For any* `st`*-sequence s if $p\sigma \overset{s}{\Longrightarrow}_s \overline{c}$ then there exists an* `st`*-sequence u such the $s \equiv u$, $p \overset{u}{\Longrightarrow} c$ and $\overline{c} \equiv c\sigma'$ where σ' is an extension of σ compatible with c and (c, σ') is proper.*

Proof: By induction on the length of the derivation $p\sigma \overset{s}{\Longrightarrow}_s \overline{c}$. If the derivation is empty, i. e. $\overline{c}$ is $p\sigma$, then the requirements are trivially satisified. Otherwise it has the form

$$p\sigma \overset{s'}{\Longrightarrow}_s \overline{c_1} \xrightarrow{\nu}_s \overline{c}.$$

By induction there exists u', c_1, σ_1 and an association h such that $s' \equiv_h u'$, $p \overset{u'}{\Longrightarrow} c_1$ and $\overline{c_1} \equiv c_1\sigma_1$. We now apply the Whence theorem to the move $c_1\sigma_1 \xrightarrow{\nu}_s \overline{c}$ and there are only three possible results; third and fifth are ruled out because of the definition of σ while the last is not possible because (c_1, σ_1) is proper. We examine two representative cases.

1. $c_1 \xrightarrow{s(a_i)}\xrightarrow{s(b_j)} c$, $a_i \neq b_j$, $\sigma_1(a) \xrightarrow{a'}_s x$, $\sigma_1(b) \xrightarrow{\overline{a'}}_s y$ and $\overline{c} \equiv c\sigma_1[a_i \mapsto x, b_j \mapsto y]$.
   Now by the definition of σ a must be a' and b must be $\overline{a'}$ and therefore we have $p \overset{u'}{\Longrightarrow}\xrightarrow{s(a'_i)}\xrightarrow{s(\overline{a'}_j)} c$. Let c' be such that
   $$p \overset{u'}{\Longrightarrow}\xrightarrow{s(a'_i)}\xrightarrow{s(\overline{a'}_j)}\xrightarrow{f(a'_i)}\xrightarrow{f(\overline{a'}_j)} c'.$$
   Then by Lemma 2.3 $p \overset{u'}{\Longrightarrow} c$. Also by Lemma 4.9, since both x and y must be *nil*, $c\sigma_1[a_i \mapsto x, b_j \mapsto y] \equiv c'\sigma_1$.

2. $c_1 \xrightarrow{s(a_i)} c$, $\sigma(a) \xrightarrow{\nu}_s x$ and $\overline{c} = \sigma_1[a_i \mapsto x]$.
   Here ν must be $s(a_j)$ for some j and so x must be $f(a_j)$. Then let σ' be $\sigma_1[a_i \mapsto f(a_j)]$ which is an extension of σ compatible with c and by definition $s.s(a_i) \equiv_{h\cup\{<a,i,j>\}} u'.s(a_j)$. So the required association is $h \cup \{< a, i, j >\}$.

□

Lemma 5.8 *If σ' is an extension of σ compatible with c and $c\sigma' \Uparrow$ then $c \Uparrow$.*

Proof: It is sufficient to show that if c, σ' satisfy the conditions of the lemma then either $c \uparrow$ or $c \xrightarrow{\tau} c'$ such that $c'\sigma'' \Uparrow$ for some extension of σ, σ'' compatible with c'.
If $c\sigma' \Uparrow$ then either $c\sigma' \uparrow$ or $c\sigma' \xrightarrow{\tau}_s \overline{c}$ for some $\overline{c}$ such that $\overline{c} \Uparrow$. In the former case we can conclude $c \uparrow$ because of the form of σ' while in the latter we can apply the Whence theorem to the move. Because of the definition of σ only two of the possibilities can

actually occur, clauses (2) and (6). In each case it is straightforward to find the required c' and σ''. $\square$

As an immediate corollary we have

Corollary 5.9 *For every* `st`*-sequence* s, $p \Uparrow s$ *if and only if* $p\sigma \Uparrow s$.

Proof: First suppose $p \Uparrow s$. Without loss of generality we can assume $p \overset{s}{\Longrightarrow} c$ such that $c \Uparrow$. By Corollary 5.6 $p\sigma \overset{s}{\Longrightarrow}_s c\sigma_c$ and by the Whither theorem $c \Uparrow$ implies $c\sigma_c \Uparrow$.

Conversely suppose $p\sigma \Uparrow s$, say $p\sigma \overset{s}{\Longrightarrow}_s \overline{c}$ such that $\overline{c} \Uparrow$. Applying Lemma 5.7 $\overline{c} \equiv c\sigma'$ for some extension σ' of σ compatible with c and $p \overset{s'}{\Longrightarrow} c$ for some $s' \equiv s$. By the previous lemma it follows that $c \Uparrow$, i. e. $p \Uparrow s'$ and so $p \Uparrow s$. $\square$

We also have the following properties of the refinement σ.

Lemma 5.10 *For every extension* σ' *of* σ *compatible with* c

1. $c\surd$ *if and only if* $c\sigma'\surd$
2. c *is stable if and only if* $c\sigma'$ *is stable*
3. *if* c *is stable* $S(c) = S(c\sigma')$.

Proof: Straighforward using Proposition 4.10 and the Whither and Whence theorems. $\square$

Finally we can prove the crucial property of the refinement σ.

Theorem 5.11 $p\sigma \sqsubseteq_s q\sigma$ *implies* $p \sqsubseteq_c q$.

Proof: Suppose $q \Uparrow s$ where s is an `st`-sequence. By the previous corollary $q\sigma \Uparrow s$. This in turn implies $p\sigma \Uparrow s$ from which $p \Uparrow s$ follows, again by the corollary.

We leave the reader to check $q\surd s$ implies $p\surd s$ and we check the condition on acceptance sets. Suppose $A \in \mathcal{A}(q,s)$, i. e. $q \overset{s}{\Longrightarrow} c$ such that c is live and stable and $A = S(c)$. By Corollary 5.6 $q\sigma \overset{s}{\Longrightarrow}_s c\sigma_c$ and by the previous lemma $c\sigma_c$ is also live and stable and $A = S(c\sigma_c)$. So $p\sigma \overset{s}{\Longrightarrow}_s \overline{c}$ for some live and stable $\overline{c}$ such that $S(\overline{c}) \subseteq A$. Applying Lemma 5.7 $\overline{c} \equiv c\sigma'$ for some extension σ' compatible with c and $p \overset{s'}{\Longrightarrow} c$ for some s' such that $s \equiv s'$. Again by the previous lemma c is live and stable and $S(\overline{c}) = S(c\sigma') = S(c)$. So $S(\overline{c}) \in \mathcal{A}(p,s') = \mathcal{A}(p,s)$. $\square$

As a corollary we have the main result of the section:

Corollary 5.12 *The relation* $\sqsubseteq_c^c$ *is the largest preorder contained in* $\sqsubseteq_s^c$ *which is preserved by stable action refinements.* $\square$

References

[Ace91] L. Aceto. Full abstraction for series-parallel-pomsets. In *Proceedings of CAAP*, volume 493 of *Lecture Notes in Computer Science*, pages 1–25. Springer–Verlag, 1991.

[AE91] L. Aceto and U. Engberg. Failure semantics for a simple process language with refinement. Technical report, INRIA, Sophia-Antipolis, 1991.

[AH91a] L. Aceto and M. Hennessy. Adding action refinement to a finite process algebra. In *Proceedings of* 18^{th} *ICALP*, Lecture Notes in Computer Science. Springer–Verlag, 1991.

[AH91b] L. Aceto and M. Hennessy. Towards action refinement in process algebras. *Information and Computation*, 1991. to appear.

[AH92] L. Aceto and M. Hennessy. Termination, deadlock and divergence in process algebras. *Journal of the ACM*, 39(1):147–187, 1992.

[BC89] G. Boudol and I. Castellani. Permutation of transitions: an event structure semantics for CCS and SCCS. In *Proceedings of Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, number 354 in Lecture Notes in Computer Science, pages 411–427, 1989.

[BCHK91] G. Boudol, I. Castellani, M. Hennessy, and A. Kiehn. Observing localities. In *Proceedings of MFCS '91.* Lecture Notes in Computer Science, Springer–Verlag, 1991. to appear.

[DD89a] Ph. Darondeau and P. Degano. About semantic action refinement. Technical Report 11/89, Dipartimento di Informatica, Università di Pisa, 1989. To appear in *Fundamenta Informaticae.*

[DD89b] P. Degano and P. Darondeau. Causal trees. In *Proceedings of ICALP 89*, number 372 in Lecture Notes in Computer Science, pages 234–248. Springer–Verlag, 1989.

[DD90] P. Darondeau and P. Degano. Event structures, causal trees and refinements, 1990. Submitted to *Theoretical Computer Science.*

[DH84] R. DeNicola and M. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 24:83–113, 1984.

[DNM90] P. Degano, R. De Nicola, and U. Montanari. A partial ordering semantics for CCS. *Theoretical Computer Science*, 75:223–262, 1990.

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

[Jat92] L. Jategaonkar. Personal communication. 1992.

[JM92] L. Jategoankar and A. Meyer. Testing equivalence for petri nets with action refinement. In *Proceedings of CONCUR92*, 1992.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice-Hall, 1989.

[MP91] D. Murphy and D. Pitt. Testing, betting and true concurrency. In *Proceedings of Concur 91*, number 527 in Lecture Notes in Computer Science, 1991.

[Sto88] A. Stoughton. *Fully Abstract Models of Programming Languages.* Research Notes in Theoretical Computer Science, Pitman/Wiley, 1988.

[TV87] D. Taubner and W. Vogler. The step failures semantics. In F.J. Brandenburg et. al., editor, *Proceedings of STACS 87*, number 247 in Lecture Notes in Computer Science, pages 348–359. Springer–Verlag, 1987.

[vG90] R.J. van Glabbeek. The refinement theorem for ST-bisimulation. In *Prooceedings IFIP Working Group, Sea of Galilee*, Lecture Notes in Computer Science. Springer–Verlag, 1990.

[vGV87] R.J. van Glabbeek and F.W. Vaandrager. Petri net models for algebraic theories of concurrency. In J.W. de Bakker, A.J. Nijman, and P.C. Treleaven, editors, *Prooceedings PARLE conference*, number 259 in Lecture Notes in Computer Science, pages 224–242. Springer–Verlag, 1987.

[Vog90] W. Vogler. Bisimulation and action refinement. Technical report, Technische Universität München, 1990.

[Vog91a] W. Vogler. Failure semantics based on interval semiwords is a congruence for refinement. *Distributed Computing*, 4:139–162, 1991.

[Vog91b] W. Vogler. Is partial order semantics necessary for action refinement ? Technical report, Technische Universität München, 1991.