Equivalences on Observable Processes

Irek Ulidowski
Department of Computing
Imperial College of Science, Technology and Medicine
180 Queens Gate
London SW7 2BZ.
E-mail: iu@doc.ic.ac.uk

Abstract

The aim of this paper is to find the finest 'observable' and 'implementable' equivalence on concurrent processes. This is a part of a larger programme to develop a theory of observable processes where semantics of processes are based on locally and finitely *observable* process behaviour, and all process constructs are allowed, provided their operational meaning is defined by realistically *implementable* transition rules.

Process behaviour which can be established by so-called local testing but not global testing is called locally and finitely observable. We define *copy+refusal* testing equivalence as indistinguishability by copy+refusal tests consisting of *traces, refusals* and *copying*, all of which are local. It is argued that copy+refusal tests are sufficient for local testing—adding any other local tests to copy+refusal tests does not increase their testing power. Hence, copy+refusal equivalence is the finest observable equivalence.

By examining the structure of transition rules we propose several conditions which all realistically implementable rules should satisfy. Using these conditions, we define the ISOS format of rules. We show that the ISOS contexts capture exactly the observable behaviour of processes—ISOS trace congruence is proved to coincide with copy+refusal equivalence. Hence, copy+refusal equivalence is also the finest implementable equivalence.

1 Introduction

An essential element of any theory of concurrent processes is a notion of *process equivalence* whose purpose is to identify processes with essentially the same behaviour. But what behaviour do we mean? Consider two of the fundamental equivalences on concurrent processes, namely *observation* equivalence [11] and *trace* equivalence [5, 13]. We contend that observation equivalence, considered as the finest extensional equivalence, distinguishes between too many processes — it can examine some process behaviour which can not be finitely and locally observed. On the other hand, trace equivalence, based on sequences of visible actions called traces, is not discriminating enough. It does not take into account some observable aspects of process behaviour like action *refusals*. Thus we can ask,

> what is the finest process equivalence which identifies processes with exactly the same locally and finitely observable behaviour?

We take the view that the only reasonable method to establish the behaviour of processes is by experimenting on them and recording and interpreting their responses [12]. The different experiments proposed in the concurrency literature [12, 1, 16, 10, 4] can be divided into two groups. The first consists of the so-called *local* experiments. Among these is the *attempt to communicate* on channel a ('pressing the a button') [12, 16]. Such an attempt may produce, in finite time, two kinds of responses: acceptance on a and refusal on a. We consider them to be the basic *observable* process behaviour. Also, in the first group, is the *empty* experiment (meaning stop with either success or failure) and the *delay* experiment (pause for some time before performing the next experiment). The last experiment in this group is *copying* [1, 10, 4] (make a finite number of copies of the process, experiment separately on each of them and put the results together only at the end). Copying clearly has a local and finite character. In contrast, the experiment of the second group has a distinctive global character—it involves combining at any time the results of all possible runs of some other experiment on the process. That is why

it is called the *global* experiment. This experiment is what Milner calls 'controlling the weather conditions' [12]. It is proved in [1] that combined local and global experiments are powerful enough to test if processes are observation equivalent. We agree with [10, 4] that the global experiments are too powerful—they make some unobservable behaviour observable, and they are unimplementable. On the other hand, the local experiments depict locally and finitely observable fragments of process behaviour. As we reject the global experiments we may ask,

> what is the finest process equivalence testable by the local experiments?

As each experiment together with the interpretation of the process's responses produces a number of tests, the obvious candidate for the required equivalence is what is called *copy+refusal* testing equivalence [1, 15]. This represents indistinguishability by the set of copy+refusal tests, CR. In the spirit of the Operator Completeness result of Abramsky [1] we prove that adding any monotone and linear testing operators to CR will not give more testing power. Hence, we argue that, if we identify the computability and locality of experiments with the monotonicity and linearity of their induced tests, then copy+refusal equivalence *is* the finest equivalence testable by the local experiments.

In order to gain more intuition about copy+refusal equivalence we also look at it from two different angles. We compare copy+refusal equivalence with bisimulation-like equivalences and trace congruences induced by a number of formats of structured transition rules.

We start by defining the *observational* transition system. This has the form of a derived transition system with divergence, where the transition relation $\Rightarrow$ is labelled by acceptances and refusals of communications (or actions)— the basic observable process behaviour.

It is known that observation equivalence and n-nested simulation equivalence [8] are strictly finer than copy+refusal equivalence. This means that, in terms of testing, we need both local and some form of global tests to characterise them. We define, on the observational transition system, a new simulation-like process relation called *refusal* simulation. An equivalence associated with it is strictly coarser than observation and n-nested simulation equivalences. We prove that, for sort-finite processes, refusal simulation equivalence corresponds exactly to copy+refusal equivalence.

Now we turn our attention to trace congruences. We define two processes to be refusal simulation equivalent iff their responses to copy+refusal testing are the same. But the responses can be viewed as a form of generalised traces (by which divergence can be expressed). Hence, two processes are copy+refusal equivalent iff they are trace congruent with respect to a class of process contexts powerful enough to mimic copy+refusal testing. Since the contexts are built up from process constructs (operators) we can represent the class of contexts by a class of process constructs. However, we can group the constructs according to a format of transition rules defining their operational semantics. This suggests an alternative way of looking at process equivalences: we should be able to divide them into classes each of which is characterised be a trace congruence induced by a particular format of transition rules.

It has been shown that (strong) bisimulation can be characterised by a trace congruence generated by the pure *ntyft/ntyxt* format [7], and 2-nested simulation by a trace congruence of the simpler pure *tyft/tyxt* format [8]. However, we feel that many rules in the above formats, and thus the process constructs whose operational meaning they define, are not realistically implementable. Hence we introduce, at some level of abstraction from the physical details, a notion of implementability which says how processes are constructed. Using this notion, we analyse the structure of transition rules in order to set a number of conditions which all realistically implementable rules should satisfy. These conditions restrict a general format of rules with negative premises (for example the *ntyft/ntyxt* format) to a new format called the *Implementable* Structured Operational Semantics, ISOS for short. Here we can ask,

> what is the process equivalence which coincides with a trace congruence induced by the ISOS format of rules?

We show that the ISOS trace congruence corresponds to copy+refusal testing equivalence, proving that copy+refusal equivalence is the finest observable and implementable equivalence.

Remark. Since the processes we are discussing may diverge we shall consider, in general, the preorders on processes rather than the equivalences we are interested in. The equivalences can be obtained from the preorders in a standard way.

The paper is organised as follows. In the next section we review the basic concepts of testing: local and global tests, the semantics of tests and the testing preorders. In particular, we define copy+refusal tests CR and copy+refusal testing preorder. In section 3, we introduce several other local tests and

prove that adding them to CR does not increase the power of copy+refusal testing. In the next section we give a characterisation of copy+refusal preorder by a simulation-like preorder called refusal simulation. The second part of the paper begins, in section 5, with the introduction of our notion of implementability. Then, using this notion, we identify the ISOS format of transition rules. Next, in section 6, we define the ISOS trace precongruence and show that it coincides with copy+refusal preorder. The last section contains conclusions and suggestions for future research.

2 Copy+refusal Testing

In this section we present a notion of testing as a reliable tool to examine the behaviour of processes. We argue that all finite and local testing can be done by copy+refusal testing.

We start by defining a new kind of derived transition system with divergence. It no only abstracts from silent behaviour, but it also internalises refusals of actions. Hence it represents both acceptances and refusals of actions, i.e. the basic observable aspects of process behaviour, treating them as equally important. For this reason we call it the *observational* transition system.

Throughout the paper the usual abbreviations will be used: $p \xrightarrow{a}$ for $\exists q \in Proc.\ p \xrightarrow{a} q$, $p \not\xrightarrow{a}$ for $\neg\exists q \in Proc.\ p \xrightarrow{a} q$ and $p \xrightarrow{ab}$ for $\exists q, r \in Proc.\ p \xrightarrow{a} q \xrightarrow{b} r$. Similarly for $\Rightarrow$.

Definition 2.1 Let $(Proc, A \cup \{\tau\}, \rightarrow, \uparrow)$ be a transition system with divergence. The *observational* transition system with divergence is a structure $(Proc, \mathsf{RAct}, \Rightarrow, \Uparrow)$ where

- $\mathsf{RAct} \equiv A \cup \tilde{A} \cup \{\varepsilon\}$ where $\tilde{A} \equiv \{\tilde{a} \mid a \in A\}$
- $\Rightarrow \subseteq Proc \times \mathsf{RAct} \times Proc$ is a *derived* transition relation defined as follows:

  $p \overset{a}{\Rightarrow} q \equiv p \xrightarrow{\tau^*} \xrightarrow{a} q$

  $p \overset{\tilde{a}}{\Rightarrow} q \equiv p \xrightarrow{\tau^*} q \not\xrightarrow{\tau}\ \&\ q{\downarrow}\ \&\ q \not\xrightarrow{a}$

  $p \overset{\varepsilon}{\Rightarrow} q \equiv p \xrightarrow{\tau^*} q$
- $p{\Uparrow} \equiv (\exists q.\ p \overset{\varepsilon}{\Rightarrow} q\ \&\ q{\uparrow}) \vee p \xrightarrow{\tau^\omega}$.

Notice that we can have both $p \overset{a}{\Rightarrow} q$ and $p \overset{\tilde{a}}{\Rightarrow} r$. The last is clearly different from $p \overset{a}{\not\Rightarrow}$.

In the introduction we briefly discussed five types of experiments: the empty experiment, the attempt to communicate on a, the delay, copying and the global experiment. These experiments together with the interpretation of the process's responses produce five groups of tests (e_i are any experiments and t_i are tests induced by them, for $i \in I \subseteq \mathbb{N}$):

- s and f—we stop experimenting with either a success or with a failure;
- the attempt to communicate on a. In finite time two things can happen: either the communication is accepted—then we continue with the experiment e_1; or it is refused in a stable state (no silent actions τ are possible and thus no divergence: the green light is off [12])—then we continue with e_2. Hence the general form of the test is $at_1 + \tilde{a}t_2$. However, it is showed in [16] that we can work with four simpler tests without losing expressiveness. These tests and their abbreviations are

  $$\begin{array}{lll} at + \tilde{a}\mathsf{f} & \text{abbreviated by} & at, \\ at + \tilde{a}\mathsf{s} & \text{abbreviated by} & \lfloor a \rfloor t, \\ \tilde{a}t + a\mathsf{f} & \text{abbreviated by} & \tilde{a}t, \\ \tilde{a}t + a\mathsf{s} & \text{abbreviated by} & \lfloor \tilde{a} \rfloor t. \end{array}$$

  We can think of the above tests as HML formulae on RAct with the satisfaction relation defined over the observational transition system: if $(\ )^*$ is a translation function, then $(at)^* = \langle a \rangle t^*$, $(\lfloor a \rfloor t)^* = [a]t^*$ as well as $(\tilde{a}t)^* = \langle \tilde{a} \rangle t^*$, $(\lfloor \tilde{a} \rfloor t)^* = [\tilde{a}]t^*$. Abramsky in [1] uses only two tests: at and $\tilde{a}t$ with the last being the same as our $\lfloor a \rfloor t$.
- delay—wait a while, then continue experimenting with e_1. This gives us the test εt_1 ;
- copying—make two copies of the process, perform e_1 on the first copy and e_2 on the second copy, then 'combine' the responses together. Informally, if we require both experiments to succeed then the induced test is $t_1 \wedge t_2$, whereas if we only need a success of one of them then we obtain $t_1 \vee t_2$.
- global experiment—enumerate all possible runs of the experiment e_1 on the process. We have two tests $\forall t_1$, $\exists t_1$ and as the notation suggests we interpret them as *all* respectively *some* runs of the experiment e_1 succeed [1].

We call $\mathsf{s}, \mathsf{f}, at, \lfloor a \rfloor t$ tests the *traces* and $\tilde{a}t, \lfloor \tilde{a} \rfloor t$ the *refusals*. Following [1, 16] the language T of tests t is defined thus:

$$t ::= \mathsf{s} \mid \mathsf{f} \mid at \mid \tilde{a}t \mid \lfloor a \rfloor t \mid \lfloor \tilde{a} \rfloor t \mid \varepsilon t \mid t \wedge t \mid t \vee t \mid \exists t \mid \forall t$$

$$
\begin{array}{lcl}
O(\mathsf{s},p) & = & \{\top\} \\
O(\mathsf{f},p) & = & \{\bot\} \\
O(at,p) & = & \bigcup\{O(t,p') \mid p \stackrel{a}{\Rightarrow} p'\} \\
 & & \cup\, \{\bot \mid p \stackrel{\tilde{a}}{\Rightarrow}\} \;\cup\; \{\bot \mid p\Uparrow\} \\
O(\tilde{a}t,p) & = & \bigcup\{O(t,p') \mid p \stackrel{\tilde{a}}{\Rightarrow} p'\} \\
 & & \cup\, \{\bot \mid p \stackrel{a}{\Rightarrow}\} \;\cup\; \{\bot \mid p\Uparrow\} \\
O(\lfloor a \rfloor t,p) & = & \bigcup\{O(t,p') \mid p \stackrel{a}{\Rightarrow} p'\} \\
 & & \cup\, \{\top \mid p \stackrel{\tilde{a}}{\Rightarrow}\} \;\cup\; \{\bot \mid p\Uparrow\} \\
O(\lfloor \tilde{a} \rfloor t,p) & = & \bigcup\{O(t,p') \mid p \stackrel{\tilde{a}}{\Rightarrow} p'\} \\
 & & \cup\, \{\top \mid p \stackrel{a}{\Rightarrow}\} \;\cup\; \{\bot \mid p\Uparrow\} \\
O(\varepsilon t,p) & = & \bigcup\{O(t,p') \mid p \stackrel{\varepsilon}{\Rightarrow} p'\} \;\cup\; \{\bot \mid p\Uparrow\} \\
O(t \wedge t',p) & = & O(t,p) \wedge O(t',p) \\
O(t \vee t',p) & = & O(t,p) \vee O(t',p) \\
O(\exists t,p) & = & \exists O(t,p) \\
O(\forall t,p) & = & \forall O(t,p)
\end{array}
$$

Figure 1.

In Figure 1 we define the 'meaning' of tests by the outcome function $O : \mathrm{T} \times \mathit{Proc} \to \mathcal{P}[\mathbb{O}]$, after [1, 15]. The set $\mathbb{O} = \{\top, \bot\}$ is a domain of outcomes. $\top$ represents a success and $\bot$ a failure (or divergence) of a particular run of a test on a process. Since processes are nondeterministic we need to use subsets of $\mathbb{O}$ to denote the outcomes of all possible runs of the test.

Remark. In Figure 1, by abuse of notation, we write $\wedge$ ($\vee$) for both the testing operator and the binary operator on $\mathcal{P}[\mathbb{O}]$ which is just a *pointwise extension* of $\wedge$ ($\vee$) on $\mathbb{O}$. Similarly we use $\forall$ ($\exists$) to denote both the testing operator and the unary operator on $\mathcal{P}[\mathbb{O}]$ defined in Figure 2.

We require some orderings to compare subsets of $\mathbb{O}$ and thus, to compare processes being tested. The most commonly used ordering is the *convex* (Egli-Milner or Plotkin) ordering, $\sqsubseteq_C$. We will also use the *lower* (Hoare) ordering, $\sqsubseteq_L$, and the *upper* (Smyth) ordering $\sqsubseteq_U$. In fact $\sqsubseteq_C$ is a partial order and $\sqsubseteq_L$, $\sqsubseteq_U$ are preorders on $\mathcal{P}[\mathbb{O}]$. Also $\sqsubseteq_C = \sqsubseteq_L \cap \sqsubseteq_U$.

Now we define a testing relation on processes, $\sqsubseteq$, parametrised by a subset of T in the superscript and in the subscript by a type of ordering used.

Definition 2.2 Let $Y \subseteq \mathrm{T}$ and $X \in \{C, L, U\}$. A binary relation $\sqsubseteq_X^Y$ on *Proc* is defined as follows, for $p,\ q \in \mathit{Proc}$

$$p \sqsubseteq_X^Y q \;\equiv\; \forall t \in Y.\ O(t,p) \sqsubseteq_X O(t,q).$$

$$
\exists X = \begin{cases} \{\bot\} & \text{if } X = \{\bot\} \\ \{\top\} & \text{otherwise} \end{cases}
$$

$$
\forall X = \begin{cases} \{\top\} & \text{if } X = \{\top\} \\ \{\bot\} & \text{otherwise} \end{cases}
$$

Figure 2.

Abramsky showed that, by taking a set S of tests generated by all testing constructs except $\tilde{a}$ and $\lfloor \tilde{a} \rfloor$, preorder $\sqsubseteq_C^S$ coincides with observation preorder $\sqsubseteq$, for sort-finite transition systems. Clearly, also $\sqsubseteq_C^T$ coincides with $\sqsubseteq$.

Phillips investigated in [15] the properties of the language of *copy+refusal* tests, CR, generated by:

$$t ::= \mathsf{s} \mid \mathsf{f} \mid at \mid \tilde{a}t \mid \lfloor a \rfloor t \mid \lfloor \tilde{a} \rfloor t \mid \varepsilon t \mid t \wedge t \mid t \vee t.$$

Clearly $\sqsubseteq_C^{CR}$ is a preorder and $\sqsubseteq_C^T \subset \sqsubseteq_C^{CR}$.

Example. Consider processes p, q, r and s represented by their derivation trees in Figure 3. A label Ω at the node of q denotes divergence i.e. an infinite unary tree of τ. Let $t_1 = a\mathsf{s}$, then $O(t_1,p) = O(t_1,q) = \{\top, \bot\}$. Hence, $O(\forall t_1, p) = \{\bot\}$ but $O(\exists t_1, p) = \{\top\}$. Next consider $t_2 = abc\mathsf{s} \wedge abd\mathsf{s}$ and $t_3 = a(bc\mathsf{s} \wedge bd\mathsf{s})$. Clearly we have $O(t_2,r) = O(t_2,s) = \{\top, \bot\}$, where $\bot$ is due to the nondeterministic nature of r and s. However, $O(t_3,s) = \{\top, \bot\}$ but $O(t_3,r) = \{\bot\}$: s *may* do a and then both bc and bd whereas r may not. Hence r and s are not copy+refusal equivalent. They are refusal (not copying) equivalent [16]. Lastly consider processes p and q. Clearly they are lower copy+refusal equivalent, but not upper copy+refusal equivalent. This is due to the possibility of divergence of q after a. A test $t_4 = [a]\tilde{a}\mathsf{s}$ distinguishes them: $O(t_4,p) = \{\top\}$ but $O(t_4,q) = \{\top, \bot\}$. □

Since $\sqsubseteq_C^{CR} = \sqsubseteq_L^{CR} \cap \sqsubseteq_U^{CR}$, we find it more intuitive and easier to work with the lower (*may*) and upper (*must*) parts of copy+refusal preorder, $\sqsubseteq_L^{CR}$, $\sqsubseteq_U^{CR}$, than with $\sqsubseteq_C^{CR}$. Moreover, it is easy to prove that only LCR, a subset of CR, formed by

$$t ::= \mathsf{s} \mid at \mid \tilde{a}t \mid t \wedge t$$

is required for lower copy+refusal testing. Also, for upper testing we only need to consider UCR—a subset of CR generated by

$$t ::= \mathsf{f} \mid \lfloor a \rfloor t \mid \lfloor \tilde{a} \rfloor t \mid t \vee t.$$

Thus $\sqsubseteq_L^{CR} = \sqsubseteq_L^{LCR}$ and $\sqsubseteq_U^{CR} = \sqsubseteq_U^{UCR}$.

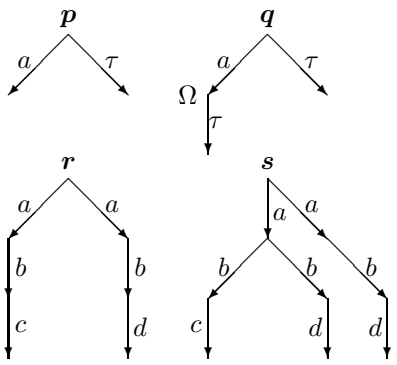

Figure 3.

3 Operator Completeness of CR

In the last section we presented CR as the set of tests induced by the local experiments. One may ask

- are there other interesting experiments which are local and finite in character?
- does adding tests induced by those experiments to CR increase its distinguishing power?

We argue (if somewhat informally) that, although other finite and local experiments can be devised, we do not gain more distinguishing power by adding the induced tests to CR. The line of inquiry in this section is motivated by the Operator Completeness result for tests S due to Abramsky [1].

Consider the following experiment. We attempt to communicate simultaneously on finitely many channels a_i (or in the 'black box' terminology: simultaneously press a finite number of buttons). If we succeed on a_i then we continue with an experiment e_i. When the communication on all a_i is refused we continue with an experiment e. Clearly this experiment is finite and local. It is proved in [16] that it can be expressed as the combination of the attempt to communicate on single channel and copying.

The next experiment we discuss involves copying. We make n copies of the process, then, for all $1 \leq i \leq n$, we run an experiment e_i on the ith copy obtaining a set of responses $R_i \in \mathcal{P}[\mathbb{O}]$. Finally we combine all R_i by means of some operator $\mathsf{op} : \mathcal{P}[\mathbb{O}]^n \to \mathcal{P}[\mathbb{O}]$. The experiment can be denoted by $\mathsf{OP}(e_1, \ldots, e_n)$ and the induced test as $\mathsf{OP}(t_1, \ldots, t_n)$. Notice that the testing operators $\wedge, \vee$ and $\forall, \exists$ are binary respectively unary versions of OP. Whether or not adding OP to CR increases its testing power clearly depends on properties of the operator op. These properties will also determine the character the experiment $\mathsf{OP}(e_1, \ldots, e_n)$.

Consider LCR, UCR the *may* and *must* parts of copy+refusal tests. To formalise the idea of adding a class of testing operators OP to a given set of tests we use a notion of *operator completion* (see [1]).

Definition 3.1 LCR′ (UCR′) is the operator completion of LCR (UCR) obtained by extending the syntax of tests LCR (UCR) by a construct $\mathsf{OP}(t_1, \ldots, t_n)$ $(n \geq 1)$ for each monotone w.r.t. the lower (upper) ordering function $\mathsf{op} : \mathcal{P}[\mathbb{O}]^n \to \mathcal{P}[\mathbb{O}]$. The outcome function O is augmented with

$$O(\mathsf{OP}(t_1, \ldots, t_n), p) \;=\; \mathsf{op}(O(t_1, p), \ldots, O(t_n, p)).$$

Lemma 3.2 Adding any monotone w.r.t. the lower (upper) ordering testing operators to LCR (UCR) does not increase the power of testing, since for all $p, q \in \mathit{Proc}$

$$p \sqsubseteq_L^{LCR} q \;\text{ iff }\; p \sqsubseteq_L^{LCR'} q \qquad (p \sqsubseteq_U^{UCR} q \;\text{ iff }\; p \sqsubseteq_U^{UCR'} q).$$

Proof. See the next section.
This lemma shows that LCR is a sufficient and necessary set of tests for the *may* copy+refusal testing and similarly UCR for the *must* copy+refusal testing. Since $\exists$ is lower monotone then, by lemma 3.2, adding it to LCR does not increase the distinguishing power. However, if $\exists$ is added to UCR it will increase the power of testing since $\exists$ is not upper monotone.

Now we return to CR. The question remains: what class of testing operators can be added to CR without increasing its testing power? Lemma 3.2 suggests the answer. Let CR′ be the operator completion of CR by constructs $\mathsf{OP}(t_1, \ldots, t_n)$ for each lower and upper monotone function $\mathsf{op} : \mathcal{P}[\mathbb{O}]^n \to \mathcal{P}[\mathbb{O}]$. We deduce that CR and CR′ have the same distinguishing power:

Theorem 3.3 For $p, q \in \mathit{Proc}$ $\quad p \sqsubseteq_C^{CR} q \;\text{ iff }\; p \sqsubseteq_C^{CR'} q.$

To shed more light on the type of operators OP, which can be added to CR without increasing the testing power, we propose an alternative characterisation of the operator completion of CR. It would be wrong to characterise CR″ as the completion of CR by OP for each convex monotone function op—both $\forall$, $\exists$ are convex monotone but adding them to CR would give us Abramsky's tests S. However, imposing another condition on op defined as follows

$$\mathsf{op}(\ldots, X_i \cup Y_i, \ldots) = \mathsf{op}(\ldots, X_i, \ldots) \cup \mathsf{op}(\ldots, Y_i, \ldots),$$

for all $1 \leq i \leq n$, seems sufficient. This condition is called *linearity* and it means that op preserves union in

each argument separately. It tells us how the the outcome of the experiment $\mathsf{op}(R_1,\ldots,R_n)$ is computed. By linearity, $\mathsf{op}(R_1,\ldots,R_n)$ equals to

$$\bigcup\{\mathsf{op}(\{r_1\},\ldots,\{r_n\}) \mid r_i \in R_i, 1 \le i \le n\}.$$

This expression says that we run repeatedly the experiments $e_1,\ldots,e_n$ on the copies of the process to obtain partial results $\mathsf{op}(\{r_1\},\ldots,\{r_n\})$. Then, at the end, we put these independent results together to obtain $\mathsf{op}(R_1,\ldots,R_n)$. The next theorem summarises the main result of the section:

Theorem 3.4 The testing power of CR$''$, the operator completion of CR by constructs $\mathsf{OP}(t_1,\ldots,t_n)$ for each convex monotone and linear function $\mathsf{op} : \mathcal{P}[\mathbb{O}]^n \to \mathcal{P}[\mathbb{O}]$, is the same as that of CR since, for $p, q \in Proc$, we have

$$p \sqsubseteq^{CR}_{\equiv C} q \quad \text{iff} \quad p \sqsubseteq^{CR''}_{\equiv C} q.$$

To complete the section we notice (by inspecting the outcome function O) that the local testing can be done without the delay operator ε (see [15]). This suggests a criterion by which a 'local' equivalence can be identified:

- local equivalences are those whose testing characterisations do not depend on the delay test.

Leaving the delay out of the language S decreases its testing power. Thus observation equivalence is not a local equivalence also by this criterion.

4 Refusal Simulation

So far we have shown that copy+refusal equivalence is the finest locally and finitely observable testing equivalence. In this section we argue that it also is a *natural* and interesting equivalence on its own, away from the testing background. We show that it can be characterised by a simulation-like equivalence, called *refusal simulation*, defined on the observational transition system.

We define refusal simulation in terms of two auxiliary relations: *L-simulation* and *U-simulation*. These auxiliary relations represent the lower (*may* or liveness) and the upper (*must* or safety) parts of refusal simulation. The intention is that, for sort-finite processes, L-simulation coincides with $\sqsubseteq^{CR}_{\equiv L}$ and U-simulation coincides with $\sqsubseteq^{CR}_{\equiv U}$. Notice that refusal simulation is a generalisation of *ready simulation* [3] and of $\frac{2}{3}$-bisimulation [10] and is similar to firework simulations of Bloom [2].

We define sort-finiteness [1] for the observational transition system $(Proc, \mathsf{RAct}, \Rightarrow, \Uparrow)$ as for all $p \in Proc$

$$\{\mu \in \mathsf{RAct} \mid \exists s \in \mathsf{RAct}^*.\ p \stackrel{s}{\Rightarrow}\stackrel{\mu}{\Rightarrow}\} \text{ is finite.}$$

Definition 4.1 For the observational transition system D we define two families of binary relations $\sqsubseteq^n_{\approx L}$, $\sqsubseteq^n_{\approx U}$ and binary relations $\sqsubseteq_{\approx L}, \sqsubseteq_{\approx U}$ over $Proc$ as follows:

$$p \sqsubseteq^0_{\approx L} q,\ p \sqsubseteq^0_{\approx U} q \quad \text{always} \quad (\text{i.e. } \sqsubseteq^0_{\approx L}, \sqsubseteq^0_{\approx U} = Proc \times Proc)$$

$$\begin{aligned} p \sqsubseteq^{n+1}_{\approx L} q \quad &\equiv \quad \forall \mu \in \mathsf{RAct}. \\ &\quad \forall p'.\ p \stackrel{\mu}{\Rightarrow} p' \text{ implies } \exists q'.\ q \stackrel{\mu}{\Rightarrow} q' \ \&\ p' \sqsubseteq^n_{\approx L} q' \end{aligned}$$

$$\begin{aligned} p \sqsubseteq^{n+1}_{\approx U} q \quad &\equiv \quad \forall \mu \in \mathsf{RAct}.\ p\Downarrow \text{ implies} \\ &\quad q\Downarrow\ \& \\ &\quad \forall q'.\ q \stackrel{\mu}{\Rightarrow} q' \text{ implies } \exists p'.\ p \stackrel{\mu}{\Rightarrow} p' \ \&\ p' \sqsubseteq^n_{\approx U} q \end{aligned}$$

$$p \sqsubseteq_{\approx L} q \quad \equiv \quad \forall n.\ p \sqsubseteq^n_{\approx L} q$$

$$p \sqsubseteq_{\approx U} q \quad \equiv \quad \forall n.\ p \sqsubseteq^n_{\approx U} q$$

We call $\sqsubseteq_{\approx L}$ a L-simulation and $\sqsubseteq_{\approx U}$ a U-simulation relations. They are preorders on D.

Example. Consider processes p, q, r and s as in Figure 3. It is easy to see that $p \sqsubseteq_{\approx L} q$ and $q \sqsubseteq_{\approx L} p$. On the other hand $q \sqsubseteq_{\approx U} p$ but $p \not\sqsubseteq_{\approx U} q$: since $p\Downarrow$ and $q\Downarrow$ we have $q \stackrel{a}{\Rightarrow} \Omega$, $p \stackrel{a}{\Rightarrow} \mathbf{0}$. However, $\mathbf{0}\Downarrow$ but $\Omega\Uparrow$. It is clear that r and s are both L- and U-simulation equivalent. □

It easy to show by induction on tests that:

Theorem 4.2 For sort-finite observational transition systems $\sqsubseteq_{\approx L} = \sqsubseteq^{LCR}_{\equiv L}$ and $\sqsubseteq_{\approx U} = \sqsubseteq^{UCR}_{\equiv U}$.

Now we define refusal simulation:

Definition 4.3 *Refusal simulation*, $\sqsubseteq_{\approx RS}$, is a binary relation over $Proc$ defined as follows

$$p \sqsubseteq_{\approx RS} q \ \equiv\ p \sqsubseteq_{\approx L} q \ \ \&\ \ p \sqsubseteq_{\approx U} q.$$

As a simple consequence of theorem 4.2 we have:

Corollary 4.4 For sort-finite observational transition systems $\sqsubseteq_{\approx RS} = \sqsubseteq^{CR}_{\equiv C}$.

Now we return to the proof of lemma 3.2. We shall use the following lemma which states that whatever monotone operators are added to LCR (UCR), we shall not go beyond the L-simulation (U-simulation) preorders.

Lemma 4.5 For $p, q \in Proc$, $p \sqsubseteq_{\approx L} q$ implies $p \sqsubseteq^{LCR'}_{\equiv L} q$ and $p \sqsubseteq_{\approx U} q$ implies $p \sqsubseteq^{UCR'}_{\equiv U} q$.

Proof. By induction on the depth of tests LCR$'$ respectively UCR$'$.

Proof of 3.2. By lemma 4.5 and theorem 4.2.

5 The ISOS Format

In the first part of the paper the discussion on concurrent processes was conducted in a 'syntax-free' manner—processes were the states of the observational transition system. We established which aspects of process behaviour were observable, and we found copy+refusal preorder to be the finest preorder relating processes with the same observational behaviour. In the second part we change our point of view and consider how concurrent processes are constructed. We propose a general notion of implementability which says how complex processes are built from simpler processes by means of process constructs. In order to decide which constructs are realistically implementable we analyse the structure of transition rules defining their meaning. Guided by our notion of implementability we define the ISOS format of rules. Thus, the implementable process constructs are those whose operational semantics are in terms of the ISOS rules.

We start by informally defining our notion of implementability. Concurrent processes are thought as 'black boxes' with labelled channels on them, and with a 'green light' to indicate the presence or absence of internal behaviour [12]. Thus, the internal structure of processes is not known. A new process can be constructed by 'connecting in some way' the channels of several simpler processes. The connection is such that a behaviour of the new process solely depends on the *observable* behaviour of the subprocesses and not on their internal structure or working. We remain intentionally vague as to how the connection of channels is made, but we concentrate on what behaviour of the subprocesses the connection uses in order to produce the behaviour of the new process. Once a process is built it gains the status of a black box, so it can be used to construct more complex processes.

Example. For many process constructs the required connection of channels is quite simple. Consider a parallel interleaving construct $|$. Given black boxes p, q, we construct $p \mid q$ by putting p, q 'side by side'. The channels of $p \mid q$ are those of p and q. Now consider the CSP parallel composition operator $\parallel$ of [5]. By definition, $p \parallel q$ accepts a communication on channel a iff both p, q can communicate on a. Thus, to construct $p \parallel q$ we join channels with the same label by the 'and' gate, one from p and one from q, into pairs and disabling all other channels. □

We analyse the structure of transition rules in the framework of *Transition System Specifications* [8, 7], TSS for short. A TSS is a tuple $(\Sigma, A \cup \{\tau\}, R)$ where Σ denotes a signature of process constructs (terms over Σ are called process terms) and $A \cup \{\tau\}$ is a set of channel names (actions). R is a set of structured transition rules.

Our notion of implementability suggests that the premises of implementable rules should describe the behaviour of subprocesses $X_1, \ldots, X_n$, and the conclusion the behaviour of the implemented process $f(X_1, \ldots, X_n)$, for some construct $f \in \Sigma$. This means that physical complexity of the implemented process is strictly larger than that of its subprocesses. Hence, many rules used in [8, 7] are not implementable, for example, the following two rules expressing certain 'global closure properties':

$$\frac{X \xrightarrow{a} X' \quad Y \xrightarrow{a} Y' \quad s(X', Y') \xrightarrow{ok} Z}{s(X, Y) \xrightarrow{ok} Z'}$$

$$\frac{X \xrightarrow{\tau} X' \quad X' \xrightarrow{a} X''}{X \xrightarrow{a} X''}$$

One of the consequences of the black box character of subprocesses is that we are forced to treat them as distinct processes. Although in constructing a new process we may use several copies of a given subprocess, due to their internal nondeterminacy, we have no method to tell if the copies represent the same subprocess at the time of use. Thus, we require all process variables $X_1, \ldots, X_n$ in the premises of implementable rules to be distinct. The next example shows what goes wrong when rules not satisfying this condition are carelessly used.

Example. Consider an unary operator *a-and-b* which is intended to show whether or not its argument can communicate on a and on b. The operational meaning of *a-and-b*(X) could be given by the following rule:

$$\frac{X \xrightarrow{a} X' \quad X \xrightarrow{b} X''}{a\text{-}and\text{-}b(X) \xrightarrow{ok} \mathbf{0}}$$

Now we notice that *a-and-b* is too 'powerful'—it can distinguish between observationally equivalent processes p and q shown in Figure 4. This happens because the premises of the above rule say more about the behaviour of X than is observationally justifiable: X can do a and b in the *same* state rather than just X can do a and b ($X \stackrel{a}{\Rightarrow}$ and $X \stackrel{b}{\Rightarrow}$). □

The next example shows that it is not safe to use any matching of subprocess names in the premises of implementable rules.

Example. Consider an unary construct *a-then-b* which tells us whether or not its argument can do a

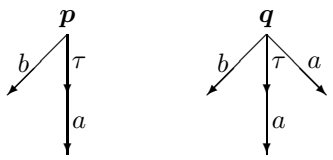

Figure 4.

and then b. Notice that we are quite vague what the precise intended meaning of *a-then-b* is. One possible rule defining it is

$$\frac{X \xrightarrow{a} X' \quad X' \xrightarrow{b} X''}{\textit{a-then-b}(X) \xrightarrow{ok} \mathbf{0}.}$$

As in the previous example the operator *a-then-b* does more than intended—it says that the argument communicates on a and then immediately on b. The communication on b is observable but whether it happens immediately (in the state X') or after several silent communications is not observable. As a result of this extra distinguishing power *a-then-b* distinguishes between observationally equivalent (CCS-like) processes $a\tau b\mathbf{0}$ and $ab\mathbf{0}$. □

The consequence of these examples is that

- all process variables in the premises of implementable rules must be distinct.

Now we are ready to present a first approximation of the format of implementable rules:

$$\frac{\{\ X_i \xrightarrow{a_i} Y_i\}_{i\in I_1} \quad \{\ X_i \xrightarrow{b_i}\}_{i\in I_2}}{f(X_1,\ldots,X_r) \xrightarrow{a} t}$$

where $a_i, b_i, a \in A\cup\{\tau\}$; f is a function name of arity r; $X_1,\ldots,X_r,Y_i$ are distinct variables in the set of variables V; $I_1, I_2 \subseteq \{1,\ldots,r\}$; and $t \in \mathit{Terms}(\Sigma, V)$ is yet unspecified term.

This format of rules is similar to the so-called GSOS (Structured Operational Semantics with Guarded Recursion) format introduced by Bloom, Istrail and Meyer in [3]. Inspecting the premises of the above format we notice that they represent more than the observable behaviour of subprocesses. It is because we have not decided yet how to represent the behaviour of processes, embodied in the $\Rightarrow$ relation, in terms of the $\rightarrow$ relation.

Example. Generality of the above format allows one to define other unwanted and perverse operators. Consider two unary operators *see-*τ and *no-initial-a* defined by the following rules:

$$\frac{X \xrightarrow{\tau} X'}{\textit{see-}\tau(X) \xrightarrow{a} \textit{see-}\tau(X')} \qquad \frac{X \xrightarrow{a}}{\textit{no-initial-}a(X) \xrightarrow{ok} \mathbf{0}}$$

We see that they distinguish between observationally equivalent processes: for *see-*τ take observationally equivalent $\tau\mathbf{0}$ and $\tau\tau\mathbf{0}$; for *no-initial-a* take $\tau a\mathbf{0}$ and $a\mathbf{0}$. This happens because we defined observable behaviour of *see-*$\tau(X)$ and *no-initial-*$a(X)$ in terms of unobservable behaviour of X: $X \xrightarrow{\tau}$ and $X \xrightarrow{a}$. □

We conclude that the implementable rules must satisfy the following:

- the behaviour of $f(X_1,\ldots,X_r)$ represented in the conclusion should only depend on the observable behaviour of $X_1,\ldots,X_r$ represented in the premises and
- the unobservable behaviour (i.e. sequences of τ) of $X_1,\ldots,X_r$ should always be inherited by $f(X_1,\ldots,X_r)$.

In earlier sections we decided that the basic notions of the observable process behaviour are: (1) performing a visible action a (a communicating on channel a) and (2) refusing, in a stable state, to perform a visible action b (communicate on b), for any $a, b \in A$. Hence, the premises of implementable rule will look like

$$\{X_i \xrightarrow{a_i} Y_i\}_{i\in I_1} \quad \{\ X_i \xrightarrow{\tau},\ X_i \xrightarrow{b_i}\}_{i\in I_2}$$

where $a_i, b_i \in A$; $X_1,\ldots,X_r,Y_i$ are distinct process variables in V, $I_1, I_2 \subseteq \{1,\ldots,r\}$ and r is an arity of f. Clearly neither of the premises of the rules defining *see-*τ and *no-initial-a* has this form.

The last condition above refers to the behaviour of $f(\boldsymbol{X})$ ($\boldsymbol{X}$ stands for $X_1,\ldots,X_r$) in the situation where some of the subprocesses X_i evolve spontaneously by performing τ actions. Since they are not externally observable, by our notion of implementability, they will not cause any observable behaviour or structural change of $f(\boldsymbol{X})$. But, they will be inherited by $f(\boldsymbol{X})$. This idea is embodied by a τ-rule defined below (introduced in [14] and [2]) where $\boldsymbol{X}$ is the vector $X_1,\ldots,X_r$

$$\frac{X_i \xrightarrow{\tau} X_i'}{f(\boldsymbol{X}) \xrightarrow{\tau} f(\boldsymbol{X})[X_i'/X_i]} \quad \tau\text{-rule}$$

Hence, for all operators f in Σ and all arguments X_i such that the behaviour of $f(\boldsymbol{X})$ depends on X_i, we include an appropriate τ-rule in a set R. These τ-rules are called the τ-rules associated with R. Thus,

we have solved the problem of representing the process behaviour by using ordinary ($\rightarrow$) rules together with their associated τ-rules.

Now we establish a structure of a term t to which $f(X_1, \ldots, X_r)$ evolves. Referring to our notion of implementability, we see that the process term t is built from what has become of processes $X_1, \ldots, X_r$:

- if the behaviour of $f(X_1, \ldots, X_r)$ depends on X_i performing a visible action a then X_i can not be used in t. For, by performing a, X_i ceases to exist—it evolves to X_i'. Hence, only X_i' can be used in t;
- if the behaviour of $f(X_1, \ldots, X_r)$ depends on X_j refusing b then it can be used in t. The reason is that X_j represents a stable state. Thus observing the refusal of b does not change X_j, so it is there for a reuse;
- if the behaviour of $f(X_1, \ldots, X_r)$ does not depend on X_i then it can be used in t

Hence, the conclusion of implementable rules would be $f(\boldsymbol{X}) \stackrel{c}{\rightarrow} t$, where $t \in \mathit{Terms}(\Sigma, \{Y_1, \ldots, Y_r\})$ and Y_i is defined as follows:

$$Y_i = \begin{cases} X_i' & \text{if } i \in I_1 \\ X_i & \text{otherwise} \end{cases}$$

De Simone's format of architectural rules of [6] has a process term t defined in a similar way.

The reader may ask if we allow any form of copying in our rules. Since in many operating systems it can be simulated by a simple dumping routine [17] or, where the system components are geographically scattered, by a combination of a dump routine and protocols [9] we choose to have some form of copying. Moreover, to conform with our requirement of realistic implementability, we allow only the *explicit* form of copying. That means if the identical copies of a process are required they have to be created *before* being used. So we will have some copying operators and separate rules defining them.

Hence, the last condition which we put on the structure of ISOS rules says that only explicit form of copying is allowed:

- a multiple use of identical process variables in the structure of ISOS rule is only permitted in the term t of the conclusion.

This contrasts with other forms of copying where copies are used without being made beforehand, which we call *implicit* copying. Among these are the 'branching' in the premises [3, 8] represented by the rule for *a-and-b* above and the use in the term t of variables other than Y_i's [8]. The second form of implicit copying and the consequences of its use are shown in the example below.

Example. Consider an unary process construct *a-then-b′* which has the same intended meaning as *a-then-b* described above. We could represent this meaning by the following rules and their τ-rules, which are not shown:

$$\frac{X \stackrel{a}{\rightarrow} X'}{\textit{a-then-b}'(X) \stackrel{\tau}{\rightarrow} \textit{then-b}(X)} \qquad \frac{X \stackrel{b}{\rightarrow} X'}{\textit{then-b}(X) \stackrel{ok}{\rightarrow} \mathbf{0}}$$

The rule for *a-then-b′* uses implicit copying. The right hand side of the conclusion uses X although it evolved to X' in order to produce the behaviour of *a-then-b′*(X), hence X is not available. This hidden copying capacity allows *a-then-b′* to distinguish between observationally equivalent processes p and q in Figure 4. □

Finally, we are ready to present the main subform of the ISOS format and the ISOS format itself. We call it the *CR*-rule because it expresses both information about refusals and copying:

$$\frac{\{\, X_i \stackrel{a_i}{\rightarrow} X_i'\}_{i \in I_1} \quad \{\, X_i \stackrel{\tau}{\not\rightarrow},\ X_i \stackrel{b_i}{\not\rightarrow}\}_{i \in I_2}}{f(\boldsymbol{X}) \stackrel{a}{\rightarrow} t} \qquad \textit{CR}\text{-rule}$$

where $a_i, b_i \in A$, $a \in A \cup \{\tau\}$; X_i of $\boldsymbol{X}$, X_i' are distinct process variables; $I_1, I_2 \subseteq \{1, \ldots, r\}$; r is an arity of f and $t \in \mathit{Terms}(\Sigma, \{Y_1, \ldots, Y_r\})$ for Y_i defined as before.

Definition 5.1 A TSS (Σ, A, R) is in the ISOS format iff R consists of *CR*-rules for each operator in Σ and the associated τ-rules (and no other rules).

We may extend this definition to talk about a set of rules R in the ISOS format.

It is clear that the operational semantics of the CCS choice operator + are not definable by rules in the ISOS format. The following are not ISOS rules:

$$\frac{X \stackrel{\tau}{\rightarrow} X'}{X + Y \stackrel{\tau}{\rightarrow} X'} \qquad \frac{Y \stackrel{\tau}{\rightarrow} Y'}{X + Y \stackrel{\tau}{\rightarrow} Y'}$$

We notice that the unobservable behaviour of subprocess X (or Y) causes a structural change of $X + Y$. If we wished to construct a process $p + q$ from p and q, we would have to built the 'connection' of channels of p, q in such way that it would be required to react to the unobservable behaviour of p or q to make the choice. This is not allowed according to our

notion of implementability—only the observable behaviour of subprocesses can cause the observable behaviour or the structural change of the constructed process. On the other hand the CSP-like internal and external choice constructs are are ISOS definable as well as other CCS-, CSP-like operators.

To conclude this section we present two unary process constructs *a-and-b′* and *a-then-b″* plus several auxiliary constructs and the ISOS rules defining their operational semantics. The intended meaning of *a-and-b′* and *a-then-b″* is the same as that of *a-and-b* respectively *a-then-b* discussed above. Due to lack of space only the *CR*-rules are shown below but not the associated τ-rules:

$$a\text{-}and\text{-}b'(X) \xrightarrow{\tau} a\text{-}And\text{-}b(X,X)$$

$$\frac{X \xrightarrow{a} X' \quad Y \xrightarrow{b} Y'}{a\text{-}And\text{-}b(X,Y) \xrightarrow{ok} \mathbf{0}}$$

$$a\text{-}then\text{-}b''(X) \xrightarrow{\tau} a\text{-}Then\text{-}b(X,X)$$

$$\frac{X \xrightarrow{a} X'}{a\text{-}Then\text{-}b(X,Y) \xrightarrow{\tau} then\text{-}b(X')}$$

$$\frac{X \xrightarrow{a} X'}{a\text{-}Then\text{-}b(X,Y) \xrightarrow{\tau} then\text{-}b(Y)}$$

The rule for *then-b* is same as before. We notice that explicit copying rules have been used. *a-then-b″* is given as general a meaning as possible due to the inclusion of the last two rules. The first says that we check for b physically after a, the second says that we check for b only if a happens but not necessarily after a.

6 Mimicking Copy+refusal Testing by the ISOS Contexts

The aim of this section is to prove theorem 6.7, which says that a trace precongruence induced by the ISOS contexts exactly corresponds to copy+refusal preorder, $\sqsubseteq_C^{CR}$.

We begin by defining a transition system with divergence B for a given TSS $P = (\Sigma, A \cup \{\tau\}, R)$ in the ISOS format. The states (processes) of B are members of $Terms(\Sigma)$, the labels are in $A \cup \{\tau\}$ and the divergence predicate is set to empty. The transition relation of B is the unique transition relation $\rightarrow_P$ generated by P. Its existence and uniqueness are guaranteed by the result for more general *ntyft/ntyxt* format in [7]. Hence $B = (Terms(\Sigma), A \cup \{\tau\}, \rightarrow_P, \emptyset)$. Using B, we define the observational transition system $D = (Terms(\Sigma), \mathsf{RAct}, \Rightarrow_P, \Uparrow)$, where $p\Uparrow$ iff $p \xrightarrow{\tau^\omega}$ and $\mathsf{RAct} \equiv A \cup \tilde{A} \cup \{\varepsilon\}$.

Theorem 6.7 is proved as follows. We show, in lemma 6.1, that $\sqsubseteq_{RS}$ is a precongruence for the ISOS contexts (proof in [18]). We introduce a notion of generalised traces *GTraces* allowing us to represent possible divergence. Next, we define three (the lower, upper and convex) trace preorders. Following the definitions on conservative extensions from [7], we introduce three corresponding trace precongruences. Since $\sqsubseteq_{RS}$ is ISOS precongruence and since it clearly refines the convex trace preorder we deduce, in lemma 6.5, that $\sqsubseteq_{RS}$ refines ISOS (convex) trace preorder. This result together with corollary 4.4 shows that $\sqsubseteq_C^{CR}$ refines ISOS trace preorder. To obtain the other half of theorem 6.7 we mimic copy+refusal testing by the ISOS contexts. This is done in lemma 6.6.

Lemma 6.1 Let $P = (\Sigma, A \cup \{\tau\}, R)$ be a given TSS in the ISOS format. $\sqsubseteq_{RS}$ is a precongruence on the derived transition system D: for all contexts $C[\,]$over Σ, all $p, q \in Terms(\Sigma)$

$$p \sqsubseteq_{RS} q \text{ implies } C[p] \sqsubseteq_{RS} C[q].$$

Definition 6.2

$$\begin{aligned} Traces(p) &\equiv \{t \mid t \in \mathsf{RAct}^* \ \&\ t \neq \varepsilon \ \&\ \exists q.\ p \stackrel{t}{\Rightarrow} q\} \\ PTraces(p) &\equiv \{t\cdot \mid t \in \mathsf{RAct}^* \ \&\ \exists q.\ p \stackrel{t}{\Rightarrow} q \ \&\ q\Uparrow\} \\ GTraces(p) &\equiv Traces(p) \cup PTraces(p). \end{aligned}$$

To define trace preorders we require an ordering $\leq$ on generalised traces: $s \leq t$, $s\cdot \leq t$ and $s\cdot \leq t\cdot$ hold if s, t are traces (possibly ε) and s is a prefix of t.

Definition 6.3

$$\begin{aligned} p \preccurlyeq_L q &\equiv GTraces(p) \sqsubseteq_L GTraces(q) \\ p \preccurlyeq_U q &\equiv GTraces(p) \sqsubseteq_U GTraces(q) \\ p \preccurlyeq q &\equiv p \preccurlyeq_L q \ \&\ p \preccurlyeq_U q. \end{aligned}$$

Definition 6.4 Let $\mathcal{F}$ be some format of TSS rules and $P = (\Sigma, B, \mathcal{R})$ a TSS in $\mathcal{F}$ format. Relations $\preccurlyeq_L^{\mathcal{F}}, \preccurlyeq_U^{\mathcal{F}}, \preccurlyeq^{\mathcal{F}}$, the lower, upper respectively convex $\mathcal{F}$ trace precongruences, are defined as follows: for any $p, q \in Terms(\Sigma)$ and any TSS $P' = (\Sigma', B', \mathcal{R}')$ in $\mathcal{F}$ format which can be added conservatively to P

$$\begin{aligned} p \preccurlyeq_L^{\mathcal{F}} q &\equiv \forall C[\,] \text{ over } \Sigma \oplus \Sigma'.\ C[p] \preccurlyeq_L C[q] \\ p \preccurlyeq_U^{\mathcal{F}} q &\equiv \forall C[\,] \text{ over } \Sigma \oplus \Sigma'.\ C[p] \preccurlyeq_U C[q] \\ p \preccurlyeq^{\mathcal{F}} q &\equiv p \preccurlyeq_L^{\mathcal{F}} q \ \&\ p \preccurlyeq_U^{\mathcal{F}} q. \end{aligned}$$

Lemma 6.5 $\sqsubseteq_{RS} \subseteq \preccurlyeq^{ISOS}$.

Lemma 6.6 Let $P = (\Sigma, A \cup \{\tau\}, \mathcal{R})$ be a TSS in the ISOS format. There is a TSS $P' = (\Sigma', A', \mathcal{R}')$ in the ISOS format which can be added conservatively to P in such a way that, in $P \oplus P'$, we have

$$\preccurlyeq^{ISOS} \subseteq \sqsubseteq_C^{CR}.$$

Proof. Let $P' = (\Sigma', A', \mathcal{R}')$ be a TSS defined as follows. The signature Σ' consists of a constant $\mathbf{0}$, unary prefix operators α for each $\alpha \in A'$, binary, infix operators $+, |, |_l, |_r$ and $\wedge$, $\vee$; the set of action labels A' is $A \cup \tilde{A} \cup \lfloor A \rfloor \cup \lfloor \tilde{A} \rfloor \cup \{\tau, \mathsf{s}, \mathsf{f}, \&, \#, l, r, \omega\}$. We use & (#) to encode the 'and' ('or') testing operators and l, r to denote the left, right operands. + is the CSP-like external choice operator of [5], ω denotes a success hence $\tilde{\omega}$ denotes a finite failure. We translate copy+refusal tests into processes by means of a function $()^*$:

$$\begin{array}{lcllcl}
\mathsf{s}^* & = & \mathsf{s}\mathbf{0} & \mathsf{f}^* & = & \mathsf{f}\mathbf{0} \\
(at)^* & = & at^* & (\lfloor a \rfloor t)^* & = & \lfloor a \rfloor t^* \\
(\tilde{a}t)^* & = & \tilde{a}t^* & (\lfloor \tilde{a} \rfloor t)^* & = & \lfloor \tilde{a} \rfloor t^* \\
(t \wedge t')^* & = & \&(lt^* + rt'^*) & & & \\
(t \vee t')^* & = & \#(lt^* + rt'^*) & & &
\end{array}$$

The set of rules $\mathcal{R}'$ contains the usual rules for + and prefixing and the rules given below (with appropriate τ-rules).

success and **failure** rules

$$\frac{t \xrightarrow{\mathsf{s}} t'}{p|t \xrightarrow{\omega} \mathbf{0}} \qquad \frac{t \xrightarrow{\mathsf{f}} t'}{p|t \xrightarrow{\tau} \mathbf{0}}$$

refusal rules

$$\frac{t \xrightarrow{a} t' \quad p \xrightarrow{a} p'}{p|t \xrightarrow{\tau} p'|t'} \qquad \frac{t \xrightarrow{a} t' \quad p \xrightarrow{\tau}\!\!\!\!\!\!\rightarrow \xrightarrow{a}\!\!\!\!\!\!\rightarrow}{p|t \xrightarrow{\tau} \mathbf{0}}$$

$$\frac{t \xrightarrow{\tilde{a}} t' \quad p \xrightarrow{a} p'}{p|t \xrightarrow{\tau} \mathbf{0}} \qquad \frac{t \xrightarrow{\tilde{a}} t' \quad p \xrightarrow{\tau}\!\!\!\!\!\!\rightarrow \xrightarrow{a}\!\!\!\!\!\!\rightarrow}{p|t \xrightarrow{\tau} p|t'}$$

$$\frac{t \xrightarrow{\lfloor a \rfloor} t' \quad p \xrightarrow{a} p'}{p|t \xrightarrow{\tau} p'|t'} \qquad \frac{t \xrightarrow{\lfloor a \rfloor} t' \quad p \xrightarrow{\tau}\!\!\!\!\!\!\rightarrow \xrightarrow{a}\!\!\!\!\!\!\rightarrow}{p|t \xrightarrow{\omega} \mathbf{0}}$$

$$\frac{t \xrightarrow{\lfloor \tilde{a} \rfloor} t' \quad p \xrightarrow{a} p'}{p|t \xrightarrow{\omega} \mathbf{0}} \qquad \frac{t \xrightarrow{\lfloor \tilde{a} \rfloor} t' \quad p \xrightarrow{\tau}\!\!\!\!\!\!\rightarrow \xrightarrow{a}\!\!\!\!\!\!\rightarrow}{p|t \xrightarrow{\tau} p|t'}$$

copying rules

$$\frac{t \xrightarrow{\&} t'}{p|t \xrightarrow{\tau} p|_l t' \wedge p|_r t'} \qquad \frac{t \xrightarrow{\#} t'}{p|t \xrightarrow{\tau} p|_l t' \vee p|_r t'}$$

$$\frac{t \xrightarrow{l} t'}{p|_l t \xrightarrow{\tau} p|t'} \qquad \frac{t \xrightarrow{r} t'}{p|_r t \xrightarrow{\tau} p|t'}$$

$$\frac{E \xrightarrow{\omega} E'}{E \wedge G \xrightarrow{\tau} G} \qquad \frac{E \xrightarrow{\tau}\!\!\!\!\!\!\rightarrow \xrightarrow{\omega}\!\!\!\!\!\!\rightarrow}{E \wedge G \xrightarrow{\tau} \mathbf{0}} \qquad \frac{G \xrightarrow{\omega} G'}{E \wedge G \xrightarrow{\tau} E} \qquad \frac{G \xrightarrow{\tau}\!\!\!\!\!\!\rightarrow \xrightarrow{\omega}\!\!\!\!\!\!\rightarrow}{E \wedge G \xrightarrow{\tau} \mathbf{0}}$$

$$\frac{E \xrightarrow{\omega} E'}{E \vee G \xrightarrow{\omega} \mathbf{0}} \qquad \frac{E \xrightarrow{\tau}\!\!\!\!\!\!\rightarrow \xrightarrow{\omega}\!\!\!\!\!\!\rightarrow}{E \vee G \xrightarrow{\tau} G} \qquad \frac{G \xrightarrow{\omega} G'}{E \vee G \xrightarrow{\omega} \mathbf{0}} \qquad \frac{G \xrightarrow{\tau}\!\!\!\!\!\!\rightarrow \xrightarrow{\omega}\!\!\!\!\!\!\rightarrow}{E \vee G \xrightarrow{\tau} E}$$

It is clear that $P \oplus P'$ is a conservative extension of P. We prove lemma 6.6 by showing that, for $p, q \in Proc$, $t \in \mathrm{LCR}$ and $t' \in \mathrm{UCR}$,

$$p|t^* \preccurlyeq_L q|t^* \text{ implies } O(p,t) \sqsubseteq_L O(q,t)$$

$$p|t'^* \preccurlyeq_U q|t'^* \text{ implies } O(p,t') \sqsubseteq_U O(q,t').$$

Remark. The above rules are similar to the rules in [1] defining the operational semantics of tests S. A few changes are made to the copying rules to make them fit with the ISOS format. However the rules defining global testing operators $\forall$, $\exists$ can not be massaged into the ISOS format.

The main result of the section is

Theorem 6.7 For sort-finite observational transition systems we have $\sqsubseteq_C^{CR} = \preccurlyeq^{ISOS}$.

7 Conclusion

We have argued that locally and finitely observable process behaviour consists of observable sequential behaviour (sequences of action acceptances and refusals) and the local part of branching behaviour. This information is also present in the structure of implementable rules: (1) behaviour of terms in the conclusion depends on the observable behaviour of subterms in the premises and (2) only explicit copying allowed.

We have defined copy+refusal equivalence and proved it to be the finest locally and finitely observable process equivalence. In section 4 we introduced a natural, simulation-like equivalence called refusal simulation equivalence. It characterises copy+refusal equivalence for sort-finite processes. To support our claims

about copy+refusal equivalence we have established the ISOS format of implementable rules such that ISOS trace congruence coincides with copy+refusal equivalence. Thus our results are:

> two processes are copy+refusal equivalent iff they are refusal simulation equivalent iff they are ISOS trace congruent.

It would be interesting to find an ISOS process language adequate for ISOS trace congruence. We think it will contain some form of 'refusal check' operator, prefixing or sequential composition, a form of choice and parallel operators as well as explicit copying operator. Finding a complete proof system for that language would be the next task.

Acknowledgements

I would like to thank Iain Phillips for many stimulating discussions on this work. Mark Ryan, Mark Dawson and Sarah Liebert helped in preparing the text. Paul Taylor's prooftree macros were used to produce our transition rules. I thank the referees for their comments and suggestions.

References

[1] S. Abramsky. Observation equivalence as a testing equivalence. *Theoretical Computer Science*, 53, 1987.

[2] B. Bloom. Strong process equivalence in the presence of hidden moves. Preliminary report, 1990.

[3] B. Bloom, S. Istrail, and A.R. Meyer. Bisimulation can't be traced: preliminary report. In *Conference Record of the 15th ACM Symposium on Principles of Programming Languages*, San Diego, California, 1988.

[4] B. Bloom and A. R. Meyer. Experimenting with process equivalence. In M. Z. Kwiatkowska, M. W. Shilds, and R. M. Thomas, editors, *Semantics for Concurrency, Leicester 1990*, pages 81–95, Berlin, 1990. Springer-Verlag.

[5] S.D. Brookes, C.A.R. Hoare, and W. Roscoe. A theory of communicating sequential processes. *Journal of the Association for Computer Machinery*, 31, 1984.

[6] R. de Simone. Higher-level synchronising devices in MEIJE-SCCS. *Theoretical Computer Science*, 37, 1985.

[7] J.F. Groote. Transition system specifications with negative premises. Technical Report CS-8950, CWI, 1989. An extended abstract appeared in J.C.M. Baeten and J.W. Klop, editors, *Proceedings of Concur90*, Amsterdam, LNCS 458, pages 332-341. Springer-Verlag, 1990.

[8] J.F. Groote and F. Vaandrager. Structured operational semantics and bisimulation as a congruence. Technical Report CS-8845, CWI, 1988. An extended abstract appeared in G. Ausiello and M. Dezani-Ciancaglini, and S. Ronchi Della Rocca, editors, *Proceedings ICALP, 89*, Stresa, LNCS 372, pages 423-438. Springer-Verlag, 1989.

[9] L. Lamport. Time, clocks and the ordering of events in a distributed system. *CACM*, 21, 1978.

[10] K.G. Larsen and A. Skou. Bisimulation through probabilistic testing. Technical Report R 88–29, Aalborg Universitetscenter, 1988. In *16th Symp. Principles of Programming Languages*, pages 344-352.ACM.

[11] R. Milner. *A Calculus for Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1980.

[12] R. Milner. A modal characterisation of observable machine behaviours. In G. Astesiano and C. Böhm, editors, *CAAP 81*, pages 25–34, Berlin, 1981. Springer-Verlag. Lecture Notes in Computer Science Vol. 112.

[13] R. De Nicola and M. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34, 1984.

[14] R. De Nicola and M. Hennessy. CCS without τ's. In H. Ehrig, R. Kowalski, G. Levi, and U. Montanari, editors, *TAPSOFT '87*, Berlin, 1987. Springer-Verlag. LNCS 250.

[15] I. Phillips. Copy testing. Unpublished manuscript, 1985.

[16] I. Phillips. Refusal testing. *Theoretical Computer Science*, 50, 1987.

[17] A. Skou. *Validation of concurrent processes*. PhD thesis, The University of Aalborg, 1989.

[18] I. Ulidowski. *A theory of observable processes*. PhD thesis, The University of London, Imperial College, to appear in 1992.