Reward Testing Equivalences for Processes

Rob van Glabbeek
Data61, CSIRO, Sydney, Australia
School of Computer Science and Engineering, University of New South Wales, Sydney, Australia
rvg@cs.stanford.edu

May and must testing were introduced by De Nicola and Hennessy to define semantic equivalences on processes. May-testing equivalence exactly captures safety properties, and must-testing equivalence liveness properties. This paper proposes *reward testing* and shows that the resulting semantic equivalence also captures conditional liveness properties. It is strictly finer than both the may- and must-testing equivalence.

This paper is dedicated to Rocco De Nicola, on the occasion of his 60th birthday. Rocco's work has been a source of inspiration to my own.

Introduction

The idea behind semantic equivalences $\equiv$ and refinement preorders $\sqsubseteq$ on processes is that $P \equiv Q$ says, essentially, that for practical purposes processes P and Q are equally suitable, i.e. one can be replaced for by the other without untoward side effects. Likewise, $P \sqsubseteq Q$ says that for all practical purposes under consideration, Q is at least as suitable as P, i.e. it will never harm to replace P by Q. To this end, Q must have all relevant good properties that P enjoys. Among the properties that ought to be so *preserved*, are *safety properties*, saying that nothing bad will even happen, and *liveness properties*, saying that something good will happen eventually.

In the setting of the process algebra CCS, refinement preorders $\sqsubseteq_{\text{may}}$ and $\sqsubseteq_{\text{must}}$, and associated semantic equivalences $\equiv_{\text{may}}$ and $\equiv_{\text{must}}$, were proposed by De Nicola & Hennessy in [6]. In [12] I argue that $\equiv_{\text{may}}$ and $\equiv_{\text{must}}$ are the coarsest equivalences that enjoy some basic compositionality requirements[1] and preserve safety and liveness properties, respectively. Yet neither preserves so-called *conditional liveness properties*. This is illustrated in Figure 1, showing two processes that are identified under both

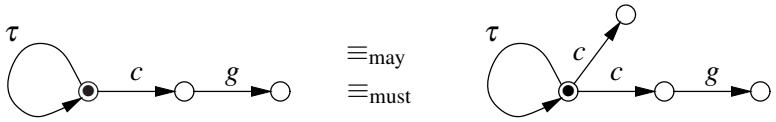

Figure 1: *Processes identified by may and must testing, but with different conditional liveness properties*

may and must testing. From a practical point of view, the difference between these two processes may be enormous. It could be that the action c comes with a huge cost, that is only worth making when the good action g happens afterwards. Only the right-hand side process is able to incur the cost without any benefits, and for this reason it lacks an important property that the left-hand process has. I call such properties *conditional liveness properties*. A conditional liveness property says that

under certain conditions something good will eventually happen.

This paper introduces a stronger form of testing that preserves conditional liveness properties.

[1] Namely being congruences for injective renaming and partially synchronous interleaving operators, or equivalently all operators of CSP, or equivalently the CCS operators parallel composition, restriction and relabelling.

1 General setting

It is natural to view the semantics of processes as being determined by their ability to pass tests [6, 17]; processes P_1 and P_2 are deemed to be semantically equivalent unless there is a test which can distinguish them. The actual tests used typically represent the ways in which users, or indeed other processes, can interact with P_i. This idea can be formulated in the following general testing scenario [9], of which the testing scenarios of [6, 17] are instances. It assumes

- a set of processes $\mathbb{P}$,
- a set of tests $\mathbb{T}$, which can be applied to processes,
- a set of outcomes $\mathbb{O}$, the possible results from applying a test to a process, and
- a function $\mathcal{A}pply : \mathbb{T} \times \mathbb{P} \to \mathscr{P}^+(\mathbb{O})$, representing the possible results of applying a specific test to a specific process.

Here $\mathscr{P}^+(\mathbb{O})$ denotes the collection of non-empty subsets of $\mathbb{O}$; so the result of applying a test T to a process P, $\mathcal{A}pply(T,P)$, is in general a *set* of outcomes, representing the fact that the behaviour of processes, and indeed tests, may be nondeterministic.

Moreover, some outcomes are considered better then others; for example the application of a test may simply succeed, or it may fail, with success being better than failure. So one can assume that $\mathbb{O}$ is endowed with a partial order, in which $o_1 \leq o_2$ means that o_2 is a better outcome than o_1.

When comparing the result of applying tests to processes one needs to compare subsets of $\mathbb{O}$. There are two standard approaches to make this comparison, based on viewing these sets as elements of either the Hoare or Smyth powerdomain [16, 1] of $\mathbb{O}$. For $O_1, O_2 \in \mathscr{P}^+(\mathbb{O})$ let

(i) $O_1 \sqsubseteq_{\text{Ho}} O_2$ if for every $o_1 \in O_1$ there exists some $o_2 \in O_2$ such that $o_1 \leq o_2$

(ii) $O_1 \sqsubseteq_{\text{Sm}} O_2$ if for every $o_2 \in O_2$ there exists some $o_1 \in O_1$ such that $o_1 \leq o_2$.

Using these two comparison methods one obtains two different semantic preorders for processes:

(i) For $P, Q \in \mathbb{P}$ let $P \sqsubseteq_{\text{may}} Q$ if $\mathcal{A}pply(T,P) \sqsubseteq_{\text{Ho}} \mathcal{A}pply(T,Q)$ for every test T

(ii) Similarly, let $P \sqsubseteq_{\text{must}} Q$ if $\mathcal{A}pply(T,P) \sqsubseteq_{\text{Sm}} \mathcal{A}pply(T,Q)$ for every test T.

Note that $\sqsubseteq_{\text{may}}$ and $\sqsubseteq_{\text{must}}$ are reflexive and transitive, and hence preorders. I use $P \equiv_{\text{may}} Q$ and $P \equiv_{\text{must}} Q$ to denote the associated equivalences.

The terminology *may* and *must* refers to the following reformulation of the same idea. Let $\mathit{Pass} \subseteq \mathbb{O}$ be an upwards-closed subset of $\mathbb{O}$, i.e. satisfying $o' \geq o \in \mathit{Pass} \Rightarrow o' \in \mathit{Pass}$, thought of as the set of outcomes that can be regarded as *passing* a test. Then one says that a process P *may* pass a test T with an outcome in *Pass*, notation "P **may** *Pass* T", if there is an outcome $o \in \mathcal{A}pply(P,T)$ with $o \in \mathit{Pass}$, and likewise P *must* pass a test T with an outcome in *Pass*, notation "P **must** *Pass* T", if for all $o \in \mathcal{A}pply(P,T)$ one has $o \in \mathit{Pass}$. Now

$$P \sqsubseteq_{\text{may}} Q \;\text{ iff }\; \forall T \in \mathbb{T}\, \forall \mathit{Pass} \in \mathrm{P}^{\uparrow}(\mathbb{O})\, (P \textbf{ may } \mathit{Pass}\ T \;\Rightarrow\; Q \textbf{ may } \mathit{Pass}\ T)$$
$$P \sqsubseteq_{\text{must}} Q \;\text{ iff }\; \forall T \in \mathbb{T}\, \forall \mathit{Pass} \in \mathrm{P}^{\uparrow}(\mathbb{O})\, (P \textbf{ must } \mathit{Pass}\ T \;\Rightarrow\; Q \textbf{ must } \mathit{Pass}\ T)$$

where $\mathrm{P}^{\uparrow}(\mathbb{O})$ is the set of upwards-closed subsets of $\mathbb{O}$.

The original theory of testing [6, 17] is obtained by using as the set of outcomes $\mathbb{O}$ the two-point lattice

with $\top$ representing the success of a test application, and $\bot$ failure.

Table 1: Structural operational semantics of CCS

$$\alpha.E \xrightarrow{\alpha} E \ (\textsc{Act}) \qquad \frac{E_j \xrightarrow{\alpha} E'_j}{\sum_{i\in I} E_i \xrightarrow{\alpha} E'_j}\ (j\in I)\ (\textsc{Sum})$$

$$\frac{E \xrightarrow{\alpha} E'}{E|F \xrightarrow{\alpha} E'|F}\ (\textsc{Par-l}) \qquad \frac{E \xrightarrow{a} E',\ F \xrightarrow{\bar{a}} F'}{E|F \xrightarrow{\tau} E'|F'}\ (\textsc{Comm}) \qquad \frac{F \xrightarrow{\alpha} F'}{E|F \xrightarrow{\alpha} E|F'}\ (\textsc{Par-r})$$

$$\frac{E \xrightarrow{\alpha} E'}{E\backslash L \xrightarrow{\alpha} E'\backslash L}\ (\alpha,\bar{\alpha}\notin L)\ (\textsc{Res}) \qquad \frac{E \xrightarrow{\alpha} E'}{E[f] \xrightarrow{f(\alpha)} E'[f]}\ (\textsc{Rel}) \qquad \frac{\mathbf{fix}(\!|S_X{:}S|\!) \xrightarrow{\alpha} E}{\mathbf{fix}(\!|X{:}S|\!) \xrightarrow{\alpha} E}\ (\textsc{Rec})$$

2 CCS: The Calculus of Communicating Systems

CCS [24] is parametrised with a set $\mathscr{C}$ of *names*; $Act := \mathscr{C} \dot\cup \bar{\mathscr{C}} \dot\cup \{\tau\}$ is the set of *actions*, where τ is a special *internal action* and $\bar{\mathscr{C}} := \{\bar{c} \mid c \in \mathscr{C}\}$ is the set of *co-names*. Complementation is extended to $\bar{\mathscr{C}}$ by setting $\bar{\bar{c}} = c$. Below, a ranges over $\mathscr{A} := \mathscr{C} \cup \bar{\mathscr{C}}$ and α over Act. A *relabelling* is a function $f : \mathscr{C} \to \mathscr{C}$; it extends to Act by $f(\bar{c}) = \overline{f(c)}$ and $f(\tau) := \tau$. Let $\mathscr{X}$ be a set $X, Y, \ldots$ of *process variables*. The set $\mathbb{E}_{\rm CCS}$ of CCS expressions is the smallest set including:

$\alpha.E$	for $\alpha \in Act$ and $E \in \mathbb{E}_{\rm CCS}$	*action prefixing*
$\sum_{i\in I} E_i$	for I an index set and $E_i \in \mathbb{E}_{\rm CCS}$	*choice*
$E\|F$	for $E, F \in \mathbb{E}_{\rm CCS}$	*parallel composition*
$E\backslash L$	for $L \subseteq \mathscr{C}$ and $E \in \mathbb{E}_{\rm CCS}$	*restriction*
$E[f]$	for f a relabelling and $E \in \mathbb{E}_{\rm CCS}$	*relabelling*
X	for $X \in \mathscr{X}$	*process variable*
$\mathbf{fix}(\!\|X{:}S\|\!)$	for $S : \mathscr{X} \rightharpoonup \mathbb{E}_{\rm CCS}$ and $X \in dom(S)$	*recursion.*

The expression $\sum_{i\in\{1,2\}} \alpha_i.E_i$ is often written as $\alpha_1.E_1 + \alpha_2.E_2$, $\sum_{i\in\{1\}} \alpha_i.E_i$ as $\alpha_1.E_1$, and $\sum_{i\in\emptyset} \alpha_i.E_i$ as $\mathbf{0}$. Moreover, one abbreviates $\alpha.\mathbf{0}$ by α, and $P\backslash\{c\}$ by $P\backslash c$. A partial function $S : \mathscr{X} \rightharpoonup \mathbb{E}_{\rm CCS}$ is called a *recursive specification*, and traditionally written as $\{Y \stackrel{def}{=} S(Y) \mid Y \in dom(S)\}$. A CCS expression E is *closed* if each occurrence of a process variable Y in E lays within a subexpression $\mathbf{fix}(\!|X{:}S|\!)$ of E with $Y \in dom(S)$; $\mathbb{P}_{\rm CCS}$, ranged over by $P, Q, \ldots$, denotes the set of closed CCS expressions, or *processes*.

The semantics of CCS is given by the labelled transition relation $\to \subseteq \mathbb{P}_{\rm CCS} \times Act \times \mathbb{P}_{\rm CCS}$, where transitions $P \xrightarrow{\alpha} Q$ are derived from the rules of Table 1. Here $\mathbf{fix}(\!|S_X{:}S|\!)$ denotes the expression $S(X)$ (written S_X) with $\mathbf{fix}(\!|Y{:}S|\!)$ substituted for each free occurrence of Y, for all $Y \in dom(S)$, while renaming bound variables in S_X as necessary to avoid name-clashes.

The process $\alpha.P$ performs the action α first and subsequently acts as P. The choice operator $\sum_{i\in I} P_i$ may act as any of its arguments P_i, depending on which of these processes is able to act at all. The parallel composition $P|Q$ executes an action from P, an action from Q, or in the case where P and Q can perform complementary actions a and $\bar{a}$, the process can perform a synchronisation, resulting in an internal action τ. The restriction operator $P\backslash L$ inhibits execution of the actions from L and their complements. The relabelling $P[f]$ acts like process P with all labels α replaced by $f(\alpha)$. Finally, the rule for recursion says that a recursively defined process $\mathbf{fix}(\!|X{:}S|\!)$ behaves exactly as the body S_X of the recursive equation $X \stackrel{def}{=} S_X$, but with recursive calls $\mathbf{fix}(\!|Y{:}S|\!)$ substituted for the variables $Y \in dom(S)$.

3 Classical may and must testing for CCS

Let $Act^\omega := Act \cup \{\omega\}$, where $\omega \notin Act$ is a special action reporting success. A CCS *test* $T \in \mathbb{T}_{\rm CCS}$ is defined just like a CCS process, but with α ranging over Act^ω. So a CCS process is a special kind of CCS test, namely one that never performs the action ω. To apply the test T to the process P one runs them in parallel; that is, one runs the combined process $T|P$—which is itself a CCS test.

Definition 1 A *computation* π is a finite or infinite sequence $T_0, T_1, T_2, \ldots$ of tests, such that (i) if T_n is the final element in the sequence, then $T_n \stackrel{\tau}{\longrightarrow} T$ for no T, and (ii) otherwise $T_n \stackrel{\tau}{\longrightarrow} T_{n+1}$.

A computation π is *successful* if it contains a state T with $T \stackrel{\omega}{\longrightarrow} T'$ for some T'.

For $T \in \mathbb{T}_{\rm CCS}$, $P \in \mathbb{P}_{\rm CCS}$, let $Comp(T,P)$ be the set of computations whose initial element is $T|P$.

Let $\mathcal{A}pply(T,P) := \{\top \mid \exists \text{ successful } \pi \in Comp(T,P)\} \cup \{\bot \mid \exists \text{ unsuccessful } \pi \in Comp(T,P)\}$.

Using this definition of $\mathcal{A}pply$ it follows that $P \sqsubseteq_{\rm may} Q$ holds unless there is a test T such that $T|P$ has (that is, is the initial state of) a successful computation but Q has not. Likewise $P \sqsubseteq_{\rm must} Q$ holds unless there is a test T such that $T|P$ has only successful computations but Q has not.

4 Dual may and must testing

A *liveness property* [20] is a property that says that *something good will eventually happen*. In the context of CCS, any test T can be regarded to specify a liveness property; a process P is defined to have this property iff all computations of $T|P$ are successful. Now $P \sqsubseteq_{\rm must} Q$ holds iff all liveness properties $T \in \mathbb{T}_{\rm CCS}$ that are enjoyed by P also hold for Q.

A *safety property* [20] is a property that says that *something bad will never happen*. When thinking of the special action ω as reporting that something bad has occurred, rather than something good, any test T can also be regarded to specify a safety property; a process P is defined to have this property iff none of the computations of $T|P$ are catastrophic; here *catastrophic* is simply another word for "successful", when reversing the connotation of ω. Now $Q \sqsubseteq_{\rm may} P$ holds iff all safety properties $T \in \mathbb{T}_{\rm CCS}$ that are enjoyed by P also hold for Q.

A *labelled transition system* (LTS) over a set Act is a pair $(\mathbb{P}, \rightarrow)$ where $\mathbb{P}$ is a set of *processes* or *states* and $\rightarrow \subseteq \mathbb{P} \times Act \times \mathbb{P}$ a set of *transitions*. In [12] preorders $\sqsubseteq_{liveness}$ and $\sqsubseteq_{safety}$ are defined on LTSs. Specialised to the LTS $(\mathbb{P}_{\rm CCS}, \rightarrow)$ induced by CCS, $\sqsubseteq_{liveness}$ coincides with $\sqsubseteq_{\rm must}$, and $\sqsubseteq_{safety}$ is exactly the reverse of $\sqsubseteq_{\rm may}$, in accordance with the reasoning above.

To explain the reversal of $\sqsubseteq_{\rm may}$ when dealing with safety properties, I propose a variant of CCS testing where in Definition 1 the word "catastrophic" is used for "successful" and $\mathcal{A}pply$ is redefined by

$$\mathcal{A}pply(T,P) := \{\bot \mid \exists \text{ catastrophic } \pi \in Comp(T,P)\} \cup \{\top \mid \exists \text{ uncatastrophic } \pi \in Comp(T,P)\}.$$

An equivalent alternative to redefining $\mathcal{A}pply$ is to simply invert the order between $\bot$ and $\top$. Let $\sqsubseteq^{\rm dual}_{\rm may}$ and $\sqsubseteq^{\rm dual}_{\rm must}$ be the versions of the may- and must-testing preorders obtained from this alternative definition. It follows immediately from the definitions that $P \sqsubseteq^{\rm dual}_{\rm may} Q$ iff $Q \sqsubseteq_{\rm must} P$ and that $P \sqsubseteq^{\rm dual}_{\rm must} Q$ iff $Q \sqsubseteq_{\rm may} P$. Based on this, it may be more accurate to say that $\sqsubseteq_{safety}$ coincides with $\sqsubseteq^{\rm dual}_{\rm must}$.

A *possibility property* [21] is a property that says that *something good might eventually happen*. A test T can be regarded to specify a possibility property; a process P is defined to have this property iff some computation of $T|P$ is successful. Now $P \sqsubseteq_{\rm may} Q$ holds iff all possibility properties $T \in \mathbb{T}_{\rm CCS}$ that are enjoyed by P also hold for Q. Lamport argues that "verifying possibility properties tells you nothing interesting about a system" [21]. As an example, consider the following models of coffee machines:

$$C_1 := \tau \qquad C_2 := \tau.c + \tau \qquad C_3 := \tau.c$$

where c is the act of dispensing coffee. The machine C_1 surely will not make coffee, C_2 makes a non-deterministic choice between making coffee or not, and C_3 surely makes coffee. Under may testing, systems $C2$ and C_3 are equivalent—both have the possibility of making coffee—and each of them is better than C_1: $C_1 \sqsubset_{\rm may} C_2 \equiv_{\rm may} C_3$. The relevance of this indeed is questionable. It takes must testing to formalise that C_3 is better than C_2: only C_3 guarantees that coffee will eventually be dispensed.

When employing dual testing, the same example applies, but with c denoting a catastrophe. Now C_1 is safe, whereas C_2 and C_3 are not: $C_1 \sqsupset^{\rm dual}_{\rm must} C_2 \equiv^{\rm dual}_{\rm must} C_3$. Dual may testing would argue that C_2 is better than C_3 because a catastrophe might be avoided. This however, can be deemed a weak argument.

In view of these considerations, I will focus on the preorders $\sqsubseteq_{\rm must}$ and $\sqsubseteq^{\rm dual}_{\rm must}$ (or $\sqsubseteq_{\mathit{safety}}$). The (dual) may preorders simply arise as their inverses, and hence do not require explicit treatment.

5 Reward testing for CCS

A CCS *reward test* is defined just like a CCS process, but with α ranging over $Act \times \mathbb{R}$, the *valued actions*. A valued action is an action tagged with a real number, the *reward* for executing this action. A negative reward can be seen as a penalty. Let $\mathbb{T}^R_{\rm CCS}$ be the set of CCS reward tests. The structural operational semantics for CCS reward tests has the following modified rules:

$$\frac{P \xrightarrow{a,r} P',\ Q \xrightarrow{\bar{a},r'} Q'}{P|Q \xrightarrow{\tau,r+r'} P'|Q'}\ (\textsc{Comm}') \qquad \frac{P \xrightarrow{\alpha,r} P'}{P\backslash L \xrightarrow{\alpha,r} P'\backslash L}\ (\alpha,\bar{\alpha} \notin L)\ (\textsc{Res}') \qquad \frac{P \xrightarrow{\alpha,r} P'}{P[f] \xrightarrow{f(\alpha),r} P'[f]}\ (\textsc{Rel}')$$

Thus, in synchronising two actions one reaps the rewards of both. In all other rules of Table 1, α is simply replaced by α,r, with $r \in \mathbb{R}$. A valued action $\alpha,0$ is simply denoted α, so that a CCS process can be seen as a special CCS reward test, namely one in which all rewards are 0. To apply a reward test T to a process P one again runs them in parallel.

Definition 2 A *reward computation* π is a finite or infinite sequence $T_0, r_1, T_1, r_2, T_2 \ldots$ of reward tests, such that (i) if T_n is the final element in π, then $T_n \xrightarrow{\tau,r} T$ for no r and T, and (ii) otherwise $T_n \xrightarrow{\tau,r_{n+1}} T_{n+1}$.

The *reward* of a finite computation π ending in T_n is $\sum_{i=1}^{n} r_i$. The *reward* of an infinite computation $T_0, r_1, T_1, r_2, T_2 \ldots$ is

$$\inf_{n\to\infty} \sum_{i=1}^{n} r_i \qquad \in \mathbb{R} \cup \{-\infty, \infty\}.$$

For $T \in \mathbb{T}^R_{\rm CCS}$, $P \in \mathbb{P}_{\rm CCS}$, let $\mathit{Comp}^R(T,P)$ be the set of reward computations with initial element $T|P$. Let $\mathcal{A}\mathit{pply}(T,P) := \{\mathit{reward}(\pi) \mid \pi \in \mathit{Comp}^R(T,P)\}$.

This defines reward preorders $\sqsubseteq^{\rm may}_{\rm reward}$ and $\sqsubseteq^{\rm must}_{\rm reward}$ on $\mathbb{P}_{\rm CCS}$. It will turn out that $P \sqsubseteq^{\rm may}_{\rm reward} Q$ iff $Q \sqsubseteq^{\rm must}_{\rm reward} P$. As a consequence I will focus on $\sqsubseteq^{\rm must}_{\rm reward}$, and simply call it $\sqsubseteq_{\rm reward}$.

6 Characterising reward testing

Assuming a fixed LTS $(\mathbb{P}, \rightarrow)$, labelled over a set $Act = \mathscr{A} \dot\cup \{\tau\}$, the ternary relation $\stackrel{-}{\Longrightarrow} \subseteq \mathbb{P} \times \mathscr{A}^* \times \mathbb{P}$ is the least relation satisfying

$$P \stackrel{\varepsilon}{\Longrightarrow} P\ , \qquad \frac{P \xrightarrow{\tau} Q}{P \stackrel{\varepsilon}{\Longrightarrow} Q}\ , \qquad \frac{P \xrightarrow{a} Q,\ a \neq \tau}{P \stackrel{a}{\Longrightarrow} Q} \qquad \text{and} \qquad \frac{P \stackrel{\sigma}{\Longrightarrow} Q \stackrel{\rho}{\Longrightarrow} r}{P \stackrel{\sigma\rho}{\Longrightarrow} r}\ .$$

For $\sigma \in \mathscr{A}^*$ and $\nu \in \mathscr{A}^* \cup \mathscr{A}^\infty$ write $\sigma \leq \nu$ for "σ is a prefix of ρ", i.e. "$\exists \rho\ .\sigma\rho = \nu$".

Definition 3 Let $P \in \mathbb{P}$.

- $a_1a_2a_3\cdots \in \mathscr{A}^\infty$ is an *infinite trace* of P if there are $P_1, P_2, \ldots$ such that $P \xrightarrow{a_1} P_1 \xrightarrow{a_2} P_2 \xrightarrow{a_3} \cdots$.
- $inf(P)$ denotes the set of infinite traces of P.
- P *diverges*, notation $P\Uparrow$, if there are $P_i \in \mathbb{P}$ for all $i > 0$ such that $P \xrightarrow{\tau} P_1 \xrightarrow{\tau} P_2 \xrightarrow{\tau} \cdots$.
- $divergences(P) := \{\sigma \in \mathscr{A}^* \mid \exists Q.\ P \stackrel{\sigma}{\Longrightarrow} Q\Uparrow\}$ is the set of *divergence traces* of P.
- $initials(P) := \{\alpha \in \mathscr{A} \mid \exists Q.\ P \xrightarrow{\alpha} Q\}$.
- $deadlocks(P) := \{\sigma \in \mathscr{A}^* \mid \exists Q.\ P \stackrel{\sigma}{\Longrightarrow} Q \wedge initials(Q) = \emptyset\}$ is the set of *deadlock traces* of P.
- $CT(P) := inf(P) \cup divergences(P) \cup deadlocks(P)$ is the set of *complete traces* of P.
- $ptraces(P) := \{\sigma \in \mathscr{A}^* \mid \exists Q.\ P \stackrel{\sigma}{\Longrightarrow} Q\}$ is the set of *partial traces* of P.
- $failures(P) := \{\langle \sigma, X\rangle \in \mathscr{A}^* \times \mathscr{P}(\mathscr{A}) \mid \exists Q.\ P \stackrel{\sigma}{\Longrightarrow} Q \wedge initials(Q) \cap (X \cup \{\tau\}) = \emptyset\}$.
- $failures_d(P) := failures(P) \cup \{\langle \sigma, X\rangle \mid \sigma \in divergences(P) \wedge X \subseteq \mathscr{A}\}$.
- $inf_d(P) := inf(P) \cup \{\nu \in \mathscr{A}^\infty \mid \forall \sigma < \nu\ \exists \rho \in divergences(P).\ \sigma \leq \rho < \nu\}$.
- $divergences_\perp(P) := \{\sigma\rho \mid \sigma \in divergences(P) \wedge \rho \in \mathscr{A}^*\}$.
- $inf_\perp(P) := inf(P) \cup \{\sigma\nu \mid \sigma \in divergences(P) \wedge \nu \in \mathscr{A}^\infty\}$.
- $failures_\perp(P) := failures(P) \cup \{\langle \sigma\rho, X\rangle \mid \sigma \in divergences(P) \wedge \rho \in \mathscr{A}^* \wedge X \subseteq \mathscr{A}\}$.

Note that $ptraces(R) = \{\sigma \mid \langle \sigma, \emptyset\rangle \in failures_d(R)\}$ for any $R \in \mathbb{P}$. (*)

A *path* of a process $P \in \mathbb{P}$ is an alternating sequence $P_0\,\alpha_1\,P_1\,\alpha_2\,P_2\cdots$ of processes/states and actions, starting with a state and either being infinite or ending with a state, such that $P_i \xrightarrow{\alpha_{i+1}} P_{i+1}$ for all relevant i. Let $l(\pi) := \alpha_1\alpha_2\cdots$ be the sequence of actions in π, and $\ell(\pi)$ the same sequence after all τs are removed. Now $\sigma \in inf(P) \cup divergences(P)$ iff P has an infinite path π with $\ell(\pi) = \sigma$. Likewise, $\sigma \in ptraces(P)$ iff P has a finite path π with $\ell(\pi) = \sigma$. Finally, $\sigma \in inf(P) \cup ptraces(P)$ iff P has an path π with $\ell(\pi) = \sigma$.

Any transition $P|Q \xrightarrow{\alpha} R$ derives, through the rules of Table 1, from

- a transition $P \xrightarrow{\alpha} P'$ and a state Q, where $R = P'|Q$,
- two transitions $P \xrightarrow{a_1} P'$ and $Q \xrightarrow{\bar{a}_2} Q'$, where $R = P'|Q'$,
- or from a state P and a transition $Q \xrightarrow{\alpha} Q'$, where $R = P|Q'$.

This transition/state, transition/transition or state/transition pair is called a *decomposition* of $P|Q \xrightarrow{\alpha} R$; it need not be unique. Now a *decomposition* of a path π of $P|Q$ into paths π_1 and π_2 of P and Q, respectively, is obtained by decomposing each transition in the path, and concatenating all left-projections into a path of P and all right-projections into a path of Q—notation $\pi \in \pi_1|\pi_2$ [15]. Here it could be that π is infinite, yet either π_1 or π_2 (but not both) are finite. Again, decomposition of paths need not be unique.

Theorem 1 Let $P, Q \in \mathbb{P}_{\rm CCS}$. Then $P \sqsubseteq_{\rm reward} Q \;\Leftrightarrow\; divergences(P) \supseteq divergences(Q) \wedge\; inf(P) \supseteq inf(Q) \wedge\; failures_d(P) \supseteq failures_d(Q)$.

Proof: Let $\sqsubseteq_{NDFD}$ be the preorder defined by: $P \sqsubseteq_{NDFD} Q$ iff the right-hand side of Theorem 1 holds.

For $\sigma = a_1a_2\cdots a_n \in \mathscr{A}^*$, let $\bar{\sigma}.T$ with $T \in \mathbb{T}^R_{\rm CCS}$ be the CCS reward test $\bar{a}_1.\bar{a}_2.\cdots\bar{a}_1.T$. It starts with performing the complements of the actions in σ, where each of these actions is given a reward 0.

Write α^r for $(\alpha, r) \in Act \times \mathbb{R}$. For $\nu = a_1a_2a_3\cdots \in \mathscr{A}^\infty$, let $\bar{\nu}^r$ be the CCS reward test $\mathbf{fix}(\!|X_0{:}S|\!)$ where $S = \{X_i \stackrel{def}{=} \bar{a}^r_{i+1}.X_{i+1} \mid i \geq 0\}$. This test simply performs the infinite sequence of complements of the actions in ν, where each of these actions is given a reward r.

"$\Rightarrow$": Suppose $P \not\sqsubseteq_{NDFD} Q$.

Case 1: Let $\sigma \in divergences(Q) \setminus divergences(P)$. Take $T := \bar{\sigma}.\tau^{-1}.\tau^1 \in \mathbb{T}^R_{\rm CCS}$. Then $T|Q$ has a computation π with $reward(\pi) < 0$, whereas $T|P$ has no such computation. Hence $P \not\sqsubseteq_{\rm reward} Q$.

Case 2: Let $\nu \in inf(Q) \setminus inf(P)$. Take $T := \bar{\nu}^{-1} \in \mathbb{T}^R_{\rm CCS}$. Then $T|Q$ has a computation π with $reward(\pi) = -\infty$, whereas $T|P$ has no such computation. Hence $P \not\sqsubseteq_{\rm reward} Q$.

Case 3: Let $\langle\sigma,X\rangle \in \mathit{failures}_d(Q) \setminus \mathit{failures}_d(P)$. Take $T := \bar{\sigma}.\tau^{-1}.\sum_{a\in X} a^1 \in \mathbb{T}^R_{\rm CCS}$. Then $T|Q$ has a computation π with $\mathit{reward}(\pi) < 0$, whereas $T|P$ has no such computation. Hence $P \not\sqsubseteq_{\rm reward} Q$.

"$\Leftarrow$": Suppose $P \sqsubseteq_{NDFD} Q$. Let $T \in \mathbb{T}^R_{\rm CCS}$ and $r \in \mathbb{R}$ be such that $\exists \pi \in \mathit{Comp}(T|Q)$ with $\mathit{reward}(\pi) = r$. It suffices to find a $\pi' \in \mathit{Comp}(T|P)$ with $\mathit{reward}(\pi') \leq r$. The computation π can be seen as a path of $T|Q$ in which all actions are τ. Decompose this path into paths π_1 of T and π_2 of Q. Note that $\mathit{reward}(\pi) = \mathit{reward}(\pi_1)$.

Case 1: Let π_2 be infinite. Then $\ell(\pi_2) \in \mathit{inf}(Q) \cup \mathit{divergences}(Q) \subseteq \mathit{inf}(P) \cup \mathit{divergences}(P)$. Thus P has an infinite path π'_2 with $\ell(\pi'_2) = \ell(\pi_2)$. Consequently, $T|P$ has an infinite path $\pi' \in \pi_1|\pi'_2$ that is a computation with $\mathit{reward}(\pi') = r$.

Case 2: Let π_2 be finite and π_1 be infinite. Then $\ell(\pi_1) \in \mathit{divergences}(T)$ and $\ell(\pi_2) \in \mathit{ptraces}(Q) \subseteq \mathit{ptraces}(P)$. The latter inclusion follows by (*). Thus P has a finite path π'_2 with $\ell(\pi'_2) = \ell(\pi_2)$. Consequently, $T|P$ has an infinite path $\pi' \in \pi_1|\pi'_2$ that is a computation with $\mathit{reward}(\pi') = r$.

Case 3: Let π_1 and π_2 be finite. Let T' and Q' be the last states of π_1 and π_2, respectively. Let $X := \{a \in \mathit{Act} \mid a^r \in \mathit{initials}(T')\}$. Then $\tau \notin X$, $\tau \notin \mathit{initials}(Q')$ and $\mathit{initials}(Q') \cap X = \emptyset$. So $\langle \ell(\pi_2), X\rangle \in \mathit{failures}(Q) \subseteq \mathit{failures}_d(Q) \subseteq \mathit{failures}_d(P)$. Thus P has either an infinite path π'_2 with $\ell(\pi'_2) = \ell(\pi_2)$ or a finite path π'_2 with $\ell(\pi'_2) = \ell(\pi_2)$ and whose last state P' satisfies $\mathit{initials}(P') \cap (X \cup \{\tau\}) = \emptyset$. Consequently, $T|P$ has a finite or infinite path $\pi' \in \pi_1|\pi'_2$ that is a computation with $\mathit{reward}(\pi') = r$. □

7 Weaker notions of reward testing

Finite-penalty reward testing doesn't allow computations that incur infinitely many penalties. A test $T \in \mathbb{T}^R_{\rm CCS}$ has *finite penalties* if each infinite path $T\alpha_1^{r_1}T_1\alpha_2^{r_2}T_2\cdots$ has only finitely many transitions i with $r_i < 0$. Let $P \sqsubseteq_{\text{fp-reward}} Q$ iff $\mathcal{A}\mathit{pply}(T,P) \sqsubseteq_{\rm Sm} \mathcal{A}\mathit{pply}(T,Q)$ for every finite-penalty reward test T.

Theorem 2 Let $P,Q \in \mathbb{P}_{\rm CCS}$. Then $P \sqsubseteq_{\text{fp-reward}} Q \;\Leftrightarrow\; \begin{array}{rl} \mathit{divergences}(P) & \supseteq \mathit{divergences}(Q) \wedge \\ \mathit{inf}_d(P) & \supseteq \mathit{inf}_d(Q) \wedge \\ \mathit{failures}_d(P) & \supseteq \mathit{failures}_d(Q). \end{array}$

Proof: Let $\sqsubseteq^d_{FDI}$ be the preorder defined by: $P \sqsubseteq^d_{FDI} Q$ iff the right-hand side of Theorem 2 holds.

"$\Rightarrow$": Suppose $P \not\sqsubseteq^d_{FDI} Q$. Case 1 and 3 proceed exactly as in the proof of Theorem 1, but the proof of Case 2 needs to be revised, as its proof uses a test with infinitely many penalties. So assume

$$\mathit{divergences}(P) \supseteq \mathit{divergences}(Q) \quad \wedge \quad \mathit{failures}_d(P) \supseteq \mathit{failures}_d(Q)$$

and let $\nu \in \mathit{inf}_d(Q) \setminus \mathit{inf}_d(P)$. I can rule out the case $\forall \sigma{<}\nu\ \exists \rho \in \mathit{divergences}(q).\ \sigma \leq \rho < \nu$ because then $\nu \in \mathit{inf}_d(P)$, using that $\mathit{divergences}(Q) \subseteq \mathit{divergences}(P)$. So $\nu \in \mathit{inf}(Q)$. Let $\nu := \nu_1\nu_2$, where each $\rho \in \mathit{divergences}(Q)$ with $\rho < \nu$ satisfies $\rho < \nu_1$. Let $\nu_2 = b_1b_2\cdots \in \mathscr{A}^\infty$. Take $T := \bar{\nu}_1.\tau^{-1}.\mathbf{fix}(\!|Y_0{:}S|\!)$, where $S = \{Y_i \stackrel{def}{=} \tau^1 + \bar{b}_{i+1}.Y_{i+1} \mid i \geq 0\}$. Then $T|Q$ has a computation π with $\mathit{reward}(\pi) < 0$, whereas $T|P$ has no such computation. Hence $P \not\sqsubseteq_{\text{fp-reward}} Q$.

"$\Leftarrow$": Suppose $P \sqsubseteq^d_{FDI} Q$. The proof proceeds just as the one of Theorem 1, except for Case 1.

Case 1: Let π_2 be infinite. Then $\ell(\pi_2) \in \mathit{inf}(Q) \cup \mathit{divergences}(Q) \subseteq \mathit{inf}_d(P) \cup \mathit{divergences}(P)$. In case $\ell(\pi_2) \in \mathit{inf}(P) \cup \mathit{divergences}(P)$ the proof concludes as for Theorem 1. So assume that $\ell(\pi_2) \in \mathscr{A}^\infty$ and $\forall \sigma{<}\ell(\pi_2)\ \exists \rho \in \mathit{divergences}(P).\ \sigma \leq \rho < \ell(\pi_2)$. Then there are prefixes $\pi^\dagger$, $\pi_1^\dagger$ and $\pi_2^\dagger$ of π, π_1 and π_2 such that (i) $\pi^\dagger \in \pi_1^\dagger|\pi_2^\dagger$, (ii) there are no negative rewards allocated in the suffix of π_1 past $\pi_1^\dagger$, and (iii) $\ell(\pi_2^\dagger) \in \mathit{divergences}(P)$. Let π'_2 be an infinite path of P with $\ell(\pi'_2) = \ell(\pi_2^\dagger)$. Then there is a computation $\pi' \in \pi_1^\dagger|\pi'_2$ of $T|P$ with $\mathit{reward}(\pi') = \mathit{reward}(\pi_1^\dagger) \leq \mathit{reward}(\pi_1) = r$. □

Single penalty reward testing doesn't allow computations that incur multiple penalties. A test $T \in \mathbb{T}^R_{\mathrm{CCS}}$ has the *single penalty* property if each path $T\alpha_1^{r_1}T_1\alpha_2^{r_2}T_2\cdots$ has at most one transition i with $r_i < 0$. Let $P \sqsubseteq_{\text{sp-reward}} Q$ iff $\mathcal{A}pply(T,P) \sqsubseteq_{\mathrm{Sm}} \mathcal{A}pply(T,Q)$ for every single penalty reward test T. Obviously, $\sqsubseteq_{\text{sp-reward}}$ coincides with $\sqsubseteq_{\text{fp-reward}}$. This follows because all test used in the proof of Theorem 2 have the single penalty property.

Analogously one might weaken reward testing and/or single penalty reward testing by requiring that in each computation only finitely many, or at most one, positive reward can be reaped. This does not constitute a real weakening, as the tests used in Theorems 1 and 2 already allot at most a single positive reward per computation only.

Nonnegative reward testing requires all rewards to be nonnegative. Let $P \sqsubseteq_{+\text{reward}} Q$ iff $\mathcal{A}pply(T,P) \sqsubseteq_{\mathrm{Sm}} \mathcal{A}pply(T,Q)$ for every nonnegative reward test T. Likewise $\sqsubseteq_{-\text{reward}}$ requires all rewards to be 0 or negative.

Theorem 3 Let $P,Q \in \mathbb{P}_{\mathrm{CCS}}$. Then $P \sqsubseteq_{+\text{reward}} Q \;\Leftrightarrow\; \begin{array}{rl} \mathit{divergences}_\perp(P) & \supseteq\ \mathit{divergences}_\perp(Q) \wedge \\ \mathit{inf}_\perp(P) & \supseteq\ \mathit{inf}_\perp(Q) \wedge \\ \mathit{failures}_\perp(P) & \supseteq\ \mathit{failures}_\perp(Q). \end{array}$

Proof: Let $\sqsubseteq^\perp_{FDI}$ be the preorder defined by: $P \sqsubseteq^\perp_{FDI} Q$ iff the right-hand side of Theorem 3 holds.

"$\Rightarrow$": Suppose $P \not\sqsubseteq^\perp_{FDI} Q$.

Case 1: Let $\sigma = a_1a_2\cdots a_n \in \mathit{divergences}_\perp(Q) \setminus \mathit{divergences}_\perp(P)$. Take $T := \mathbf{fix}(\!|X_0{:}S|\!)$ in which

$$S = \{X_i \stackrel{def}{=} \tau^1 + a_{i+1}.X_{i+1} \mid 0 \le i < n\} \cup \{X_n \stackrel{def}{=} \tau^1\}.$$

Then $T|Q$ has a computation π with $\mathit{reward}(\pi) < 1$, which $T|P$ has not. Hence $P \not\sqsubseteq_{+\text{reward}} Q$.

Case 2: Let $\nu = a_1a_2\cdots \in \mathit{inf}_\perp(Q) \setminus \mathit{inf}_\perp(P)$. Let $T := \mathbf{fix}(\!|X_0{:}S|\!)$ with $S = \{X_{i-1} \stackrel{def}{=} \tau^1 + a_i.X_i \mid i \ge 1\}$. Then $T|Q$ has a computation π with $\mathit{reward}(\pi) < 1$, which $T|P$ has not. Hence $P \not\sqsubseteq_{+\text{reward}} Q$.

Case 3: Let $\langle a_1a_2\cdots a_n, X\rangle \in \mathit{failures}_\perp(Q) \setminus \mathit{failures}_\perp(P)$. Take $T := \mathbf{fix}(\!|X_0{:}S|\!)$ in which

$$S = \{X_i \stackrel{def}{=} \tau^1 + a_{i+1}.X_{i+1} \mid 0 \le i < n\} \cup \{X_n \stackrel{def}{=} \sum_{a\in X} a^1\}.$$

Then $T|Q$ has a computation π with $\mathit{reward}(\pi) < 1$, which $T|P$ has not. Hence $P \not\sqsubseteq_{+\text{reward}} Q$.

"$\Leftarrow$": Suppose $P \sqsubseteq^\perp_{FDI} Q$. Let $T \in \mathbb{T}^R_{\mathrm{CCS}}$ be a nonnegative rewards test and $r \in \mathbb{R}$ be such that there is a $\pi \in \mathit{Comp}(T|Q)$ with $\mathit{reward}(\pi) = r$. It suffices to find a $\pi' \in \mathit{Comp}(T|P)$ with $\mathit{reward}(\pi') \le r$. The computation π can be seen as a path of $T|Q$ in which all actions are τ. Decompose this path into paths π_1 of T and π_2 of Q. Note that $\mathit{reward}(\pi) = \mathit{reward}(\pi_1)$.

Case 1: Let π_2 be infinite. Then $\ell(\pi_2) \in \mathit{inf}(Q) \cup \mathit{divergences}(Q) \subseteq \mathit{inf}_\perp(P) \cup \mathit{divergences}_\perp(P)$. If $\ell(\pi_2) \in \mathit{inf}(P) \cup \mathit{divergences}(P)$ then P has an infinite path π_2' with $\ell(\pi_2') = \ell(\pi_2)$. Consequently, $T|P$ has an infinite path $\pi' \in \pi_1|\pi_2'$ that is a computation with $\mathit{reward}(\pi') = r$. The alternative is that $\ell(\pi_2)$ has a prefix in $\mathit{divergences}(P)$. In that case there are prefixes $\pi^\dagger$, $\pi_1^\dagger$ and $\pi_2^\dagger$ of π, π_1 and π_2 such that $\pi^\dagger \in \pi_1^\dagger|\pi_2^\dagger$ and $\ell(\pi_2^\dagger) \in \mathit{divergences}(P)$. Let π_2' be an infinite path of P with $\ell(\pi_2') = \ell(\pi_2^\dagger)$. Then there is a computation $\pi' \in \pi_1^\dagger|\pi_2'$ of $T|P$ with $\mathit{reward}(\pi') = \mathit{reward}(\pi_1^\dagger) \le \mathit{reward}(\pi_1) = r$.

Case 2: Let π_2 be finite and π_1 be infinite. Then $\ell(\pi_1) \in \mathit{divergences}(T)$ and $\ell(\pi_2) \in \mathit{ptraces}(Q) \subseteq \mathit{ptraces}(P) \cup \mathit{divergences}_\perp(P)$. The latter inclusion follows since

$$\mathit{ptraces}(R) \cup \mathit{divergences}_\perp(R) = \{\sigma \mid \langle\sigma,\emptyset\rangle \in \mathit{failures}_\perp(R)\}$$

for any $R \in \mathbb{P}$. If $\ell(\pi_2) \in \mathit{ptraces}(P)$ then P has a finite path π_2' with $\ell(\pi_2') = \ell(\pi_2)$. Consequently, $T|P$ has an infinite path $\pi' \in \pi_1|\pi_2'$ that is a computation with $\mathit{reward}(\pi') = r$. The alternative is handled just as for Case 1 above.

Case 3: Let π_1 and π_2 be finite. Let T' and Q' be the last states of π_1 and π_2, respectively. Let $X := \{a \in Act \mid a^r \in initials(T')\}$. Then $\tau \notin X$, $\tau \notin initials(Q')$ and $initials(Q') \cap X = \emptyset$. So $\langle \ell(\pi_2), X\rangle \in failures(Q) \subseteq failures_\perp(Q) \subseteq failures_\perp(P)$. If $\langle \ell(\pi_2) \in failures(P)$ then P has a finite path π_2' with $\ell(\pi_2') = \ell(\pi_2)$ and whose last state P' satisfies $initials(P') \cap (X \cup \{\tau\}) = \emptyset$. Consequently, $T|P$ has a finite or infinite path $\pi' \in \pi_1|\pi_2'$ that is a computation with $reward(\pi') = r$. The alternative is handled just as for Case 1 above. □

One might weaken nonnegative reward testing by requiring that in each computation only finitely many, or at most one, reward can be reaped. This does not constitute a real weakening, as the tests used in Theorem 3 already allot at most a single reward per computation only.

Theorem 4 Let $P, Q \in \mathbb{P}_{\rm CCS}$. Then $P \sqsubseteq_{-\rm reward} Q \;\Leftrightarrow\; ptraces(P) \supseteq ptraces(Q) \wedge inf(P) \supseteq inf(Q)$

Proof: Let $\sqsubseteq_T^\infty$ be the preorder defined by: $P \sqsubseteq_T^\infty Q$ iff the right-hand side of Theorem 4 holds.

"⇒": Suppose $P \not\sqsubseteq_T^\infty Q$.

Case 1: Let $\sigma \in ptraces(Q) \setminus ptraces(P)$. Take $T := \bar\sigma.\tau^{-1}$. Then $T|Q$ has a computation π with $reward(\pi) < 1$, which $T|P$ has not. Hence $P \not\sqsubseteq_{-\rm reward} Q$.

Case 2 proceeds exactly as in the proof of Theorem 1.

"⇐": Suppose $P \sqsubseteq_T^\infty Q$. Let $T \in \mathbb{T}^R_{\rm CCS}$ be a nonpositive rewards test and $r \in \mathbb{R}$ be such that there is a $\pi \in Comp(T|Q)$ with $reward(\pi) = r$. It suffices to find a $\pi' \in Comp(T|P)$ with $reward(\pi') \leq r$. The computation π can be seen as a path of $T|Q$ in which all actions are τ. Decompose this path into paths π_1 of T and π_2 of Q. Note that $reward(\pi) = reward(\pi_1)$.

Moreover, $\ell(\pi_2) \in inf(Q) \cup ptraces(Q) \subseteq inf(P) \cup ptraces(P)$. So P has a path π_2' with $\ell(\pi_2') = \ell(\pi_2)$. Consequently, $T|P$ has an path $\pi' \in \pi_1|\pi_2'$ that is either a computation, or a prefix of a computation, with $reward(\pi') = r$. In case it is a prefix of a computation π'' then $reward(\pi'') \leq reward(\pi') = r$. □

Finite-penalty nonpositive reward testing only allows computations that incur no positive rewards and merely finitely many penalties. Let $P \sqsubseteq_{\text{fp-}-\text{reward}} Q$ iff $\mathcal{A}pply(T,P) \sqsubseteq_{\rm Sm} \mathcal{A}pply(T,Q)$ for every finite-penalty nonpositive reward test T.

Theorem 5 Let $P, Q \in \mathbb{P}_{\rm CCS}$. Then $P \sqsubseteq_{\text{fp-}-\text{reward}} Q \;\Leftrightarrow\; ptraces(P) \supseteq ptraces(Q)$

Proof: Let $\sqsubseteq_T$ be the preorder defined by: $P \sqsubseteq_T Q$ iff the right-hand side of Theorem 5 holds.

"⇒": Suppose $P \not\sqsubseteq_T Q$. Let $\sigma \in ptraces(Q) \setminus ptraces(P)$. Take $T := \bar\sigma.\tau^{-1}$. Then $T|Q$ has a computation π with $reward(\pi) < 1$, which $T|P$ has not. Hence $P \not\sqsubseteq_{-\rm reward} Q$.

"⇐": Suppose $P \sqsubseteq_T Q$. Let $T \in \mathbb{T}^R_{\rm CCS}$ be a finite-penalty nonpositive rewards test and $r \in \mathbb{R}$ be such that there is a $\pi \in Comp(T|Q)$ with $reward(\pi) = r$. Then π has a finite prefix $\pi^\dagger$ (not necessarily a computation) with $reward(\pi) = r$. It suffices to find a prefix π' of a computation of $T|P$ with $reward(\pi') = r$. The finite prefix $\pi^\dagger$ can be seen as a path of $T|Q$ in which all actions are τ. Decompose this path into finite paths π_1 of T and π_2 of Q. Now $\ell(\pi_2) \in ptraces(Q) \subseteq ptraces(P)$. So P has a path π_2' with $\ell(\pi_2') = \ell(\pi_2)$. Consequently, $T|P$ has a path $\pi' \in \pi_1|\pi_2'$ that is a prefix of a computation, with $reward(\pi') = r$. □

Single penalty nonpositive reward testing only allows computations that incur no positive rewards and at most one penalty. Let $P \sqsubseteq_{\text{sp-}-\text{reward}} Q$ iff $\mathcal{A}pply(T,P) \sqsubseteq_{\rm Sm} \mathcal{A}pply(T,Q)$ for every single penalty nonpositive reward test T. Obviously, $\sqsubseteq_{\text{sp-}-\text{reward}}$ coincides with $\sqsubseteq_{\text{fp-}-\text{reward}}$. This follows because all test used in the proof of Theorem 5 have the single penalty property.

8 Reward may testing

Call a test $T \in \mathbb{T}^R_{\mathrm{CCS}}$ *well-behaved* if for each infinite path $T\alpha_1^{r_1}T_1\alpha_2^{r_2}T_2\cdots$ the limit $\lim_{n\to\infty}\sum_{i=1}^{n} r_i \in \mathbb{R}\cup\{-\infty,\infty\}$ exists. If the sequence $(r_i)_{i=1}^{\infty}$ alternates between 1 and -1 for instance, the test is not well-behaved. Since all tests used in the proof of Theorem 1 are well-behaved, the reward testing preorder $\sqsubseteq_{\mathrm{reward}}$ would not change if one restricts the collection of available test to the well-behaved ones only. When restricting to well-behaved tests, the infimum $\inf_{n\to\infty}$ in Definition 2 may be read as $\lim_{n\to\infty}$.

Theorem 6 $P \sqsubseteq^{\mathrm{may}}_{\mathrm{reward}} Q$ iff $Q \sqsubseteq^{\mathrm{must}}_{\mathrm{reward}} P$.

Proof: For any well-behaved test T, let $-T$ be obtained by changing all occurrences of actions (α, r) into $(\alpha,-r)$. Now $\mathcal{A}pply(-T,P) = \{-r \mid r \in \mathcal{A}pply(T,P)\}$. This immediately yields the claimed result.□

All weaker notions of testing contemplated in Section 7 employ well-behaved tests only. The same reasoning as above yields (besides $\sqsubseteq^{\mathrm{may}}_{\mathrm{reward}} = \sqsubseteq^{-1}_{\mathrm{reward}}$)

$$\sqsubseteq^{\mathrm{may}}_{\text{fp-reward}} = \sqsubseteq^{-1}_{\mathrm{reward}} \quad , \quad \sqsubseteq^{\mathrm{may}}_{+\mathrm{reward}} = \sqsubseteq^{-1}_{-\mathrm{reward}} \quad , \quad \sqsubseteq^{\mathrm{may}}_{-\mathrm{reward}} = \sqsubseteq^{-1}_{+\mathrm{reward}} \quad \text{and} \quad \sqsubseteq^{\mathrm{may}}_{\text{fp-}-\text{reward}} = \sqsubseteq^{-1}_{-\mathrm{reward}} \quad .$$

9 A hierarchy of testing preorders

Theorem 7 $P \sqsubseteq_{\mathrm{must}} Q$ iff $P \sqsubseteq_{+\mathrm{reward}} Q$. Likewise, $P \sqsubseteq^{\mathrm{dual}}_{\mathrm{must}} Q$ iff $P \sqsubseteq_{\text{fp-}-\text{reward}} Q$.

Proof: "If": Without affecting $\sqsubseteq_{\mathrm{must}}$ one may restrict attention to tests $T \in \mathbb{T}_{\mathrm{CCS}}$ with the property that each path of T contains at most one success state—one with an outgoing transition labelled ω. Namely, any outgoing transition of a success state may safely be omitted. Now each such test T can be converted into a nonnegative reward test T', namely by assigning a reward 1 to any action leading into a success state, keeping the rewards of all other actions 0. The success action itself may then be renamed into τ, or omitted. Now trivially, a computation of $T|P$ is successful iff the matching computation of T' yields a reward 1; a computation of $T|P$ is unsuccessful iff the matching computation of T' yields a reward 0. It follows that must-testing can be emulated by nonnegative reward testing.

"Only if": As remarked in Section 7, nonnegative reward testing looses no power when allowing only one reward per computation. For the same reasons it looses no power if each positive reward is 1. Now any reward test $T' \in \mathbb{T}^R_{\mathrm{CCS}}$ with these restrictions can be converted to a test $T \in \mathbb{T}_{\mathrm{CCS}}$ by making any target state of a reward-1 transition into a success state. It follows that nonnegative reward testing can be emulated by must-testing.

The second statement follows in the same way, but using a reward -1. □

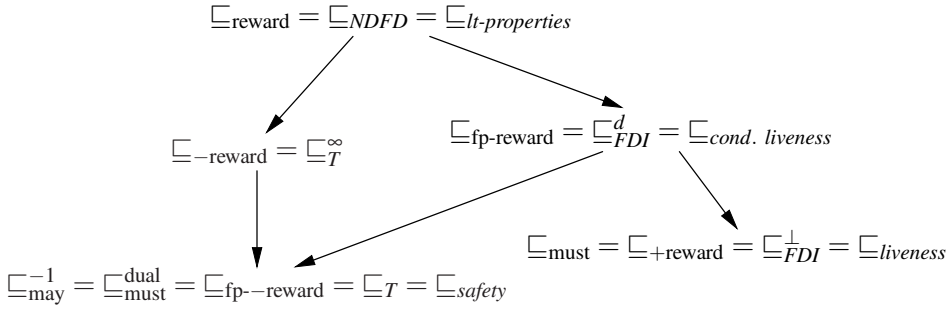

Figure 2: A spectrum of testing preorders

A preorder $\sqsubseteq_X$ is said to be *finer* than or equal to a preorder $\sqsubseteq_Y$ iff $P \sqsubseteq_X Q \Rightarrow P \sqsubseteq_Y Q$ for all P and Q; in that case $\sqsubseteq_Y$ is *coarser* than or equal to $\sqsubseteq_X$.

Theorem 8 The preorders occurring in this paper are related as indicated in Figure 2, where the arrows point in the coarser direction.

Proof: The relations between $\sqsubseteq_{\text{reward}}$, $\sqsubseteq_{\text{fp-reward}}$, $\sqsubseteq_{+\text{reward}}$, $\sqsubseteq_{-\text{reward}}$ and $\sqsubseteq_{\text{fp-}-\text{reward}}$ follow immediately from the definitions, as the coarser variant uses only a subset of the tests available to the finer variant. The strictness of all these relations is obtained by the examples below.

The connections with $\sqsubseteq_{\text{must}}$, $\sqsubseteq_{\text{must}}^{\text{dual}}$ and the inverse of $\sqsubseteq_{\text{may}}$ are provided by Theorem 7 and Section 4. The characterisations in terms of $\sqsubseteq_{NDFD}$, $\sqsubseteq_{FDI}^{d}$, $\sqsubseteq_{FDI}^{\perp}$, $\sqsubseteq_{F}^{\infty}$ and $\sqsubseteq_{T}$ are provided by Theorems 1–5. The connections with $\sqsubseteq_{lt\text{-}properties}$, $\sqsubseteq_{cond.\ liveness}$, $\sqsubseteq_{liveness}$ and $\sqsubseteq_{safety}$ will be established in Section 10. □

Let $a^n.P$ be defined by $a^0.P := P$ and $a^{i+1}.P = a.a^i.P$. Furthermore, let $a^\infty := \mathbf{fix}(\!|X{:}X \stackrel{def}{=} a.X|\!)$ be a process that performs infinitely many as. Let Δ be the unary operator given by $\Delta P := \mathbf{fix}(\!|X{:}X \stackrel{def}{=} \tau.X + P\,|\!)$. It first performs 0 or more τ-actions, and if this number is finite subsequently behaves as its argument P. So $\Delta\mathbf{0} = \tau^\infty$ just performs an infinite sequence of τ-moves.

Example 1 $\sum_{n\geq 1} a^n.\Delta\mathbf{0} \equiv_{\text{fp-reward}} a^\infty + \sum_{n\geq 1} a^n.\Delta\mathbf{0}$, but $\sum_{n\geq 1} a^n.\Delta\mathbf{0} \not\sqsubseteq_{-\text{reward}} a^\infty + \sum_{n\geq 1} a^n.\Delta\mathbf{0}$ (and thus $\not\sqsubseteq_{\text{reward}}$).

Example 2 $\Delta(c.g) \equiv_{\text{must}} \Delta(c + c.g)$ and $\Delta(c.g) \equiv_{-\text{reward}} \Delta(c + c.g)$, yet $\Delta(c.g) \not\sqsubseteq_{\text{fp-reward}} \Delta(c + c.g)$. These are the processes displayed in Figure 1. A test showing the latter is $c^{-1}.g^1$.

Example 3 $c.g \equiv_{\text{fp-}-\text{reward}} c + c.g$, yet $c.g \not\sqsubseteq_{+\text{reward}} c + c.g$. A test showing the latter is $c.g^1$.

Example 4 $\Delta a \equiv_{+\text{reward}} \Delta\mathbf{0}$, yet $\Delta a \not\sqsubseteq_{\text{fp-}-\text{reward}} \Delta\mathbf{0}$. A test showing the latter is a^{-1}.

A process P is *divergence-free* if $\mathit{divergences}(P) = \emptyset$. It is *regular*, or *finite-state*, if there only finitely many processes Q such that $\exists \sigma \in \mathscr{A}^*.\ P \stackrel{\sigma}{\Longrightarrow} Q$. It is $\Longrightarrow$*-image-finite* if for each $\sigma \in A^*$ there are only finitely many Q such that $P \stackrel{\sigma}{\Longrightarrow} Q$. Note that the class of $\Longrightarrow$-image-finite processes is not closed under parallel composition, or under renaming transition labels $a \in \mathscr{A}$ into τ. Regular processes are $\Longrightarrow$-image-finite. Any $P \in \mathbb{P}_{\text{CCS}}$ without parallel composition, relabelling or restriction is regular. Any $P \in \mathbb{P}_{\text{CCS}}$ without recursion is both divergence-free and regular.

Proposition 1 If $P \in \mathbb{P}_{\text{CCS}}$ is divergence-free, then $P \sqsubseteq_{+\text{reward}} Q$ iff $P \sqsubseteq_{\text{reward}} Q$.

Proof: This follows immediately from Theorems 1 and 3, using that $\mathit{divergences}(P) = \emptyset$, $\mathit{inf}_\perp(P) = \mathit{inf}_d(P) = \mathit{inf}(P)$ and $\mathit{failures}_\perp(P) = \mathit{failures}_d(P) = \mathit{failures}(P)$. (In case Q is not divergence-free one has neither $P \sqsubseteq_{+\text{reward}} Q$ nor $P \sqsubseteq_{\text{reward}} Q$.) □

Proposition 2 If P is $\Longrightarrow$-image-finite then (a) $P \sqsubseteq_{\text{fp-}-\text{reward}} Q$ iff $P \sqsubseteq_{-\text{reward}} Q$
and (b) $P \sqsubseteq_{\text{fp-reward}} Q$ iff $P \sqsubseteq_{\text{reward}} Q$.

Proof: By Königs lemma $\nu \in \mathscr{A}^\infty$ is an infinite trace of P iff only if each finite prefix of ν is a partial trace of P. Now (a) follows immediately from Theorems 4 and 5: Suppose $P \sqsubseteq_{-\text{reward}} Q$ and $\nu \in \mathit{inf}(Q)$. Then each finite prefix of ν is in $\mathit{ptraces}(Q)$ and thus in $\mathit{ptraces}(P)$. Thus $\nu \in \mathit{inf}(P)$.

(b) follows in the same way from Theorems 1 and 2, using (*). □

10 Conditional liveness properties

To obtain a general liveness property for labelled transition systems, assume that some notion of *good* is defined. Now, to judge whether a process P satisfies this liveness property, one should judge whether P can reach a state in which one would say that something good had happened. But all observable behaviour of P that is recorded in a labelled transition system until one comes to such a verdict, is the sequence of visible actions performed until that point. Thus the liveness property is completely determined by the set sequences of visible actions that, when performed by P, lead to such a judgement. Therefore one can just as well define a liveness property in terms of such a set.

Definition 4 A *liveness property* of processes in an LTS is given by a set $G \subseteq \mathscr{A}^*$. A process P *satisfies* this liveness property, notation $P \models \mathit{liveness}(G)$, when each complete trace of P has a prefix in G.

This formalisation of liveness properties stems from [12] and is essentially different from the one in [2] and most subsequent work on liveness properties; this point is discussed in [12, Section 6].

A preorder $\sqsubseteq$ *preserves* liveness properties if $P \sqsubseteq Q$ implies that Q enjoys any liveness property that P has. It is a *precongruence* for an n-ary operator op if $P_i \sqsubseteq Q_i$ for $i = 1,\ldots,n$ implies $op(P_1,\ldots,P_n) \sqsubseteq op(Q_1,\ldots,Q_n)$. Now let $\sqsubseteq_{\mathit{liveness}}$ be the coarsest preorder that is a precongruence for the operators of CSP and preserves liveness properties. In [12] it is shown that this preorder exists, and equals $\sqsubseteq^{\perp}_{FDI}$, as defined in the proof of Theorem 3. The proof of this result does not require that $\sqsubseteq_{\mathit{liveness}}$ be a preorder for all operators of CSP; it goes through already when merely requiring it to be precongruence for injective renaming and partially synchronous interleaving operators. Looking at this proof, the same can also be obtained requiring $\sqsubseteq_{\mathit{liveness}}$ to be a precongruence for the CCS operators $|$, $\backslash L$ and injective relabelling.

It follows that $\sqsubseteq_{\mathit{liveness}}$ coincides with $\sqsubseteq_{+\text{reward}}$ (cf. Theorem 8). This connection can be illustrated by a translation from liveness properties $G \subseteq \mathscr{A}^*$ (w.l.o.g. assumed to have the property that if $\sigma \in G$ then $\sigma\rho \notin G$ for any $\rho \neq \varepsilon$) to nonnegative reward tests T_G. Here T_G can be rendered as a deterministic tree in which all transitions completing a trace from $\bar{G}$ yield a reward 1, so that all computations of $T|P$ earn a positive reward iff $P \models \mathit{liveness}(G)$.

One obtains a general concept of safety property by means of the same argument as for liveness properties above, but using "bad" instead of "good".

Definition 5 A *safety property* of processes in an LTS is given by a set $B \subseteq \mathscr{A}^*$. A process P *satisfies* this safety property, notation $P \models \mathit{safety}(B)$, when $\mathit{ptraces}(p) \cap B = \emptyset$.

This formalisation of safety properties stems from [12] and is in line with the one in [2]. Now let $\sqsubseteq_{\mathit{safety}}$ be the coarsest precongruence (for the same choice of operators as above) that preserves safety properties. In [12] it is shown that this preorder exists, and equals $\sqsubseteq_T$, as defined in the proof of Theorem 5.

It follows that $\sqsubseteq_{\mathit{safety}}$ coincides with $\sqsubseteq_{\text{fp-}-\text{reward}}$ (cf. Theorem 8). This connection can be illustrated by a translation from safety properties $B \subseteq \mathscr{A}^*$ (w.l.o.g. assumed to have the property that if $\sigma \in B$ then $\sigma\rho \notin B$ for any $\rho \neq \varepsilon$) to nonnegative reward tests T_B. Here T_B can be rendered as a deterministic tree in which all transitions completing a trace from $\bar{B}$ yield a reward -1, so that all computations of $T|P$ earn a nonnegative reward iff $P \models \mathit{safety}(B)$.

A conditional liveness property says that *under certain conditions something good will eventually happen*. To obtain a general conditional liveness property for LTSs, assume that some condition, and some notion of *good* is defined. Now, to judge whether a process P satisfies this conditional liveness property, one should judge first of all in which states the condition is fulfilled. All observable behaviour of P that is recorded in an LTS until one comes to such a verdict, is the sequence of visible actions performed until that point. Thus the condition is completely determined by the set of sequences of visible actions that, when performed by P, lead to such a judgement. Next one should judge whether P can reach a state in which one would say that something good had happened. Again, this judgement can be expressed in terms of the sequences of visible actions that lead to such a state.

Definition 6 ([12]) A *conditional liveness property* of processes in an LTS is given by two sets $C, G \subseteq \mathscr{A}^*$. A process P *satisfies* this conditional liveness property, notation $P \models \mathit{liveness}_C(G)$, when each complete trace of P that has a prefix in C, also has a prefix in G.

Now let $\sqsubseteq_{\mathit{cond.\ liveness}}$ be the coarsest precongruence (for the same choice of operators as above) that preserves conditional liveness properties. In [12] it is shown that this preorder exists, and equals $\sqsubseteq^{d}_{FDI}$, as defined in the proof of Theorem 2. It follows that $\sqsubseteq_{\mathit{safety}}$ coincides with $\sqsubseteq_{\text{fp-reward}}$ (cf. Theorem 8).

Similar to the above cases, this connection can be illustrated by a translation from conditional liveness properties $C, G \subseteq \mathscr{A}^*$ to reward tests in which each computation has at most one negative and one positive reward, which are always -1 and $+1$.

Definition 7 A *linear time property* of processes in an LTS is given by a set $\Phi \subseteq \mathscr{A}^* \cup \mathscr{A}^\infty$ of finite and infinite sequences of actions. A process P *satisfies* this property, notation $P \models \Phi$, when $CT(P) \subseteq \Phi$.

A liveness property is a special kind of linear time property:

$$\mathit{liveness}(G) = \{\sigma \in \mathscr{A}^* \cup \mathscr{A}^\infty \mid \exists \rho \in G.\ \rho \leq \sigma\}.$$

Likewise, $\mathit{safety}(B) = \{\sigma \in \mathscr{A}^* \cup \mathscr{A}^\infty \mid \neg\exists \rho \in B.\ \rho \leq \sigma\}$, and

$$\mathit{liveness}_C(G) = \{\sigma \in \mathscr{A}^* \cup \mathscr{A}^\infty \mid (\exists \rho \in C.\ \rho \leq \sigma) \Rightarrow (\exists \nu \in G.\ \nu \leq \sigma)\}.$$

Now let $\sqsubseteq_{\mathit{lt.\ properties}}$ be the coarsest precongruence (for the same choice of operators as above) that preserves linear time properties. In [19, 12] it is shown that this preorder exists, and equals $\sqsubseteq_{NDFD}$, as defined in the proof of Theorem 1. It follows that $\sqsubseteq_{\mathit{lt.\ properties}}$ coincides with $\sqsubseteq_{\text{reward}}$ (cf. Theorem 8).

11 Congruence properties

Theorem 9 The preorders of this paper are precongruences for the CCS operators $|$, $\backslash L$ and $[f]$.

Proof: Note that $\mathcal{A}pply(T, R|P) = \mathcal{A}pply(T|R, P)$, using the associativity (up to strong bisimilarity) of $|$. Therefore $P \sqsubseteq_{\text{reward}} Q$ implies $R|P \sqsubseteq_{\text{reward}} R|Q$, showing that $\sqsubseteq_{\text{reward}}$ is a precongruence for parallel composition. The same holds for $\sqsubseteq_{\text{fp-reward}}$, $\sqsubseteq_{+\text{reward}}$, $\sqsubseteq_{-\text{reward}}$ and $\sqsubseteq_{\text{fp-}-\text{reward}}$.

Likewise $\mathcal{A}pply(T, P\backslash L) = \mathcal{A}pply(T\backslash L, P)$. This yields precongruence results for restriction.

Finally, $\mathcal{A}pply(T, P[f]) = \mathcal{A}pply(T[f^{-1}], P)$, yielding precongruence results for relabelling.

Here $[f^{-1}]$ is an operator with rule $\dfrac{E \xrightarrow{\alpha,r} E'}{E[f^{-1}] \xrightarrow{\beta,r} E'[f^{-1}]}$ $(f(\beta) = \alpha)$. Although this is not a CCS operator, for any test T the test $T[f^{-1}]$ is expressible in CCS, on grounds that each process in an LTS is expressible in CCS. □

Theorem 10 The preorders of this paper are precongruences for action prefixing.

Proof: This follows in a straightforward way from the characterisations of the preorders in Sections 6 and 7. For instance, $\mathit{failures}_d(a.P) = \{\langle a\sigma, X\rangle \mid \langle \sigma, X\rangle \in \mathit{failures}_d(a.P)\}$. □

In the same way it follows that $\sqsubseteq_{\text{fp-}-\text{reward}}$ and $\sqsubseteq_{-\text{reward}}$ are precongruences for the CCS operator $+$. However, the preorders $\sqsubseteq_{\text{reward}}$, $\sqsubseteq_{\text{fp-reward}}$ and $\sqsubseteq_{+\text{reward}}$ fail to be congruences for choice:

Example 5 $\mathbf{0} \equiv_{\text{reward}} \tau$, yet $\mathbf{0} + a \not\sqsubseteq_{+\text{reward}} \tau + a$, using that $\langle \varepsilon, \mathscr{A}\rangle \in \mathit{failures}_\perp(\tau + a) \setminus \mathit{failures}_\perp(\mathbf{0} + a)$.

This issue occurs for almost all semantic equivalences and preorders that abstract from internal actions. The standard solution is to replace each such preorder $\sqsubseteq_X$ by the coarsest precongruence for the operators of CCS that is finer than $\sqsubseteq_X$. Let *stable* be the predicate that holds for a process P iff there is no P' with $P \xrightarrow{\tau} P'$. Write $P \sqsubseteq_X^\tau Q$ iff $P \sqsubseteq_X Q \wedge (\mathit{stable}(P) \Rightarrow \mathit{stable}(Q))$.

Theorem 11 Let $X \in \{\text{reward, fp-reward, } +\text{reward}\}$. Then $\sqsubseteq_X^\tau$ is the coarsest precongruence for the operators of CCS that is contained in $\sqsubseteq_X$.

Proof: That $\sqsubseteq^{\tau}_{+\text{reward}}$ is a precongruence for $+$ follows with Theorem 3 since

$$\begin{aligned}
\mathit{stable}(P+Q) &\Leftrightarrow \mathit{stable}(P) \wedge \mathit{stable}(Q)\\
\mathit{failures}_{\perp}(P+Q) &= \{\langle \sigma, X\rangle \in \mathit{failures}_{\perp}(P) \mid \sigma \neq \varepsilon \vee \neg \mathit{stable}(P)\} \cup\\
&\quad \{\langle \sigma, X\rangle \in \mathit{failures}_{\perp}(Q) \mid \sigma \neq \varepsilon \vee \neg \mathit{stable}(Q)\} \cup\\
&\quad \{\langle \varepsilon, X\rangle \mid \langle \varepsilon, X\rangle \in \mathit{failures}_{\perp}(P) \cap \mathit{failures}_{\perp}(Q)\},\\
\mathit{inf}_{\perp}(P+Q) &= \mathit{inf}_{\perp}(P) \cup \mathit{inf}_{\perp}(Q)\\
\mathit{divergences}_{\perp}(P+Q) &= \mathit{divergences}_{\perp}(P) \cup \mathit{divergences}_{\perp}(Q)\ .
\end{aligned}$$

That it is a congruence for action prefixing, $|$, $\backslash L$ and $[f]$ follows since

$$\begin{array}{ll}
\mathit{stable}(\alpha.P) & \text{iff } \alpha \neq \tau\\
\mathit{stable}(P|Q) & \text{iff } \mathit{stable}(P) \wedge \mathit{stable}(Q) \wedge \neg \exists a \in \mathscr{A}.\ (\langle a, \emptyset\rangle \in \mathit{failures}_{\perp}(P) \wedge \langle \bar{a}, \emptyset\rangle \in \mathit{failures}_{\perp}(P))\\
\mathit{stable}(P\backslash L) & \text{iff } \mathit{stable}(P)\\
\mathit{stable}(P[f]) & \text{iff } \mathit{stable}(P).
\end{array}$$

By definition, $\sqsubseteq^{\tau}_{+\text{reward}}$ is contained in $\sqsubseteq_{+\text{reward}}$. To see that it is the coarsest precongruence contained in $\sqsubseteq_{+\text{reward}}$, suppose $P \not\sqsubseteq^{\tau}_{+\text{reward}} Q$. It suffices to build a context $C[\_]$ from CCS operators such that $C[P] \not\sqsubseteq_{+\text{reward}} C[Q]$. The case $P \not\sqsubseteq_{+\text{reward}} Q$ is immediate—take the trivial context with $C[P] := P$. So assume $P \sqsubseteq_{+\text{reward}} Q$. Then $\mathit{stable}(P)$ and $\neg \mathit{stable}(Q)$. Hence $\varepsilon \notin \mathit{divergences}_{\perp}(P) \supseteq \mathit{divergences}_{\perp}(Q)$. Choose $a \notin \mathit{ptraces}(Q)$—in case no such a exists, one first applies an injective relabelling to P and Q such that $a \notin \text{range}(f)$. Now $\langle \varepsilon, \{a\}\rangle \in \mathit{failures}(Q) \subseteq \mathit{failures}_{\perp}(Q) \subseteq \mathit{failures}_{\perp}(P)$. However, whereas $\langle \varepsilon, \{a\}\rangle \in \mathit{failures}_{\perp}(Q+a)$ one has $\langle \varepsilon, \{a\}\rangle \notin \mathit{failures}_{\perp}(P+a)$. It follows that $P+a \not\sqsubseteq_{+\text{reward}} Q+a$.

The arguments for $X \in \{\text{reward}, \text{ fp-reward}\}$ are very similar. □

12 Axiomatisations

The following axioms are easily seen to be sound for $\sqsubseteq^{\tau}_{\text{reward}}$. Here an equality $P \equiv Q$ can be seen as a shorthand for the two axioms $P \sqsubseteq Q$ and $Q \sqsubseteq P$. Action prefixing and Δ bind stronger than $+$.

$$\left\{\begin{array}{lrcl}
\text{(R1)} & \tau.X + Y & \equiv & \tau.X + \tau.(X+Y)\\
\text{(R2)} & \alpha.X + \tau.(\alpha.Y + Z) & \equiv & \tau(\alpha.X + \alpha.Y + Z)\\
\text{(R3)} & \alpha.(\tau.X + \tau.Y) & \equiv & \alpha.X + \alpha.Y\\
\text{(RP1)} & \tau.X + Y & \sqsubseteq & \tau.(X+Y)\\
\text{(RP2)} & \tau.X + Y & \sqsubseteq & X\\
\text{(R4)} & \tau.\Delta X + Y & \equiv & \Delta(X+Y)
\end{array}\right\}$$

For recursion-free processes, and dropping the infinite choice operator in favour of $+$ and $\mathbf{0}$, $\sqsubseteq^{\tau}_{\text{must}}$ coincides with $\sqsubseteq^{\tau}_{\text{reward}}$ and $\sqsubseteq^{\tau}_{\text{fp-reward}}$. Together with the standard axioms for strong bisimilarity [24], the three axioms (R1)–(R3) constitute a sound and complete axiomatisation of $\equiv^{\tau}_{\text{must}}$ [5, Theorem 4.2], and thus for $\equiv^{\tau}_{\text{reward}}$. Likewise, the three axioms (RP1),(RP2) and (R3) constitute a sound and complete axiomatisation of $\sqsubseteq^{\tau}_{\text{must}}$ [5, Theorem 4.1], and thus for $\sqsubseteq^{\tau}_{\text{reward}}$; the axioms (R1) and (R2) are derivable from them. The first sound and complete axiomatisation of $\sqsubseteq^{\tau}_{\text{must}}$ appears in [6]; their axioms are derivable from the ones above (and vise versa).

A sound and complete axiomatisation of $\equiv_{\text{may}}$ (and hence of $\equiv_{-\text{reward}}$) is obtained by adding the axioms $\tau.X \equiv X$ and $\alpha(X+Y) \equiv \alpha.X + \alpha.Y$ to the standard axioms for strong bisimilarity [5, Theorem 4.5]. The axioms (R1)–(R3) are derivable from them. Adding the axiom $X + Y \sqsubseteq X$ yields a sound and

complete axiomatisation of $\sqsubseteq_{\rm may}^{-1}$ (and hence of $\sqsubseteq_{-\rm reward}$) [5, Theorem 4.6]. The axioms (RP1) and (RP2) are then also derivable. The first sound and complete axiomatisation of $\sqsubseteq_{\rm may}$ appears in [6]; their axioms are derivable from the ones above (and vise versa).

To illustrate the difference between $\equiv^{\tau}_{\rm must}$ and $\equiv^{\tau}_{\rm reward}$, without having to deal with recursion, I consider recursion-free CCS with finite choice (as done above), but upgraded with the *delay operator* Δ introduced in [3] and in Section 9. Clearly all preorders of this paper are precongruences for Δ. With (R4), sound for $\equiv^{\tau}_{\rm reward}$, one can derive $\tau.\Delta X \equiv \Delta X$ and $\Delta X + Y \equiv \Delta(X+Y)$. Writing Ω for $\Delta\mathbf{0}$, the latter implies $\Delta Y \equiv \Omega + Y$ so one can equally well take Ω as Δ as primitive. It also follows that $\Delta\Delta X \equiv \Delta X$.

The above sound and complete axiomatisations of $\equiv_{\rm may}$ and $\sqsubseteq_{\rm may}^{-1}$ (and hence of $\equiv_{-\rm reward}$ and $\sqsubseteq_{-\rm reward}$) are extended with Δ by adding the trivial axiom $\Delta X = X$; (R4) is then derivable. This illustrates that these preorders abstract from divergence. The axiom

$$\text{(R5)} \qquad \Delta X \equiv \Delta Y$$

is sound for $\equiv^{\tau}_{\rm must}$. It expresses that must testing does not record any information past a divergence. Axioms (RP2), (R4) and (R5) imply $\Omega \sqsubseteq X$, an axiom featured in [6]. Neither $\Delta X = X$ nor (R5) is sound for $\equiv^{\tau}_{\rm reward}$.

13 Failure of congruence property for recursion

Each preorder $\sqsubseteq$ on CCS processes (= closed CCS expressions) can be extended to one on all CCS expressions by defining $E \sqsubseteq F$ iff all closed substitution instances of this inequality hold.

Definition 8 A preorder $\sqsubseteq$ on $\mathbb{E}_{\rm CCS}$ is a (full) precongruence for recursion if $S_Y \sqsubseteq T_Y$ for each $Y \in \mathit{dom}(S) = \mathit{dom}(T)$ implies $\mathbf{fix}(\!\langle X{:}S\rangle\!) \sqsubseteq \mathbf{fix}(\!\langle X{:}T\rangle\!)$.

The following counterexample shows that the must-testing preorder $\sqsubseteq^{\tau}_{\rm must}$ fails to be a precongruence for recursion, implying that the must-testing equivalence $\equiv^{\tau}_{\rm must}$ fails to be a congruence for recursion.

Example 6 Let $P \in \mathbb{T}_{\rm CCS}$ be such that $\varepsilon \notin \mathit{divergences}(P)$—for instance $P = \mathbf{0}$. Then by (R1) one has $\tau.P + X \equiv^{\tau}_{\rm must} \tau.P + \tau.(X+P)$. Yet $\mathbf{fix}(\!\langle X{:}X \stackrel{def}{=} \tau.P + X\rangle\!) \not\sqsubseteq^{\tau}_{\rm must} \mathbf{fix}(\!\langle X{:}X \stackrel{def}{=} \tau.P + \tau.(X+P)\rangle\!)$, because only the latter process has a divergence ε.

The same example shows that also $\sqsubseteq^{\tau}_{\rm reward}$, $\sqsubseteq^{\tau}_{\text{fp-reward}}$, $\sqsubseteq_{\rm reward}$, $\sqsubseteq_{\text{fp-reward}}$ and $\sqsubseteq_{\rm must}$ fail to be precongruences for recursion. However, I conjecture that all these preorders are *lean* precongruences for recursion as defined in [14].

14 Unguarded recursion

The must-testing preorder $\sqsubseteq_{\rm must}$ on CCS presented in this paper is not quite the same as the original one $\sqsubseteq^{\rm org}_{\rm must}$ from [6]. The following example shows the difference.

Example 7 $\quad \mathbf{0} \quad \begin{matrix}\equiv_{\rm must}\\ \not\sqsubseteq^{\rm org}_{\rm must}\end{matrix} \quad \mathbf{fix}(\!\langle X{:}X \stackrel{def}{=} X\rangle\!) \quad \begin{matrix}\not\sqsubseteq_{\rm must}\\ \equiv^{\rm org}_{\rm must}\end{matrix} \quad \mathbf{fix}(\!\langle X{:}X \stackrel{def}{=} \tau.X\rangle\!).$

The $\equiv_{\rm must}$-statement follows since neither process has a single outgoing transition; the processes are even *strongly bisimilar* [24]. The $\not\sqsubseteq_{\rm must}$-statement follows since $\varepsilon \in \mathit{divergences}(\mathbf{fix}(\!\langle X{:}X \stackrel{def}{=} \tau.X\rangle\!))$, yet $\varepsilon \notin \mathit{divergences}(\mathbf{fix}(\!\langle X{:}X \stackrel{def}{=} X\rangle\!))$. A test showing the difference is $\tau.\omega$.

The reason that in the original must-testing approach $\mathbf{fix}(\!|X{:}X \stackrel{def}{=} X|\!)$ sides with $\mathbf{fix}(\!|X{:}X \stackrel{def}{=} \tau.X|\!)$ rather than with $\mathbf{0}$, is that [6] treats a process featuring unguarded recursion (cf. [24]), such as $\mathbf{fix}(\!|X{:}X \stackrel{def}{=} X|\!)$, as if it diverges, regardless whether it can do any internal actions τ. This leads to a must-testing equivalence that is incomparable with strong bisimilarity.

In my view, the decision whether $\mathbf{fix}(\!|X{:}X \stackrel{def}{=} X|\!)$ diverges or not is part of the definition of the process algebra CCS, and entirely orthogonal to the development of testing equivalences. Below I define a process algebra $\mathrm{CCS}_\perp$ that resembles CCS in all aspects, expect that any process with unguarded recursion is declared to diverge. I see the work of [6] not so much as defining a must-testing equivalence on CCS that is incomparable with strong bisimilarity, but rather as defining a must-testing equivalence on $\mathrm{CCS}_\perp$, a languages that is almost, but not quite, the same as CCS.[2] This is a matter of opinion, as there is no technical difference between these approaches.

I now proceed to define $\mathrm{CCS}_\perp$, and apply the reward testing preorders of this paper to that language.

Definition 9 Let $\downarrow$ be the least predicate on $\mathbb{P}_{\mathrm{CCS}}$ which satisfies

- $\alpha.P\downarrow$ for any $\alpha \in Act$,
- if $P_i\downarrow$ for all $i \in I$ then $\sum_{i\in I} P_i\downarrow$,
- if $P\downarrow$ and $Q\downarrow$ then $P|Q\downarrow$, $P\backslash L\downarrow$ and $P[f]\downarrow$,
- if $\mathbf{fix}(\!|S_X{:}S|\!)\downarrow$ then $\mathbf{fix}(\!|X{:}S|\!)\downarrow$.

Let $P\uparrow$ if not $P\downarrow$. If $P\uparrow$ then P features *strongly unguarded recursion*.[3]

Note that $\mathbf{0}\downarrow$, $\mathbf{fix}(\!|X{:}X \stackrel{def}{=} X|\!)\uparrow$ and $\mathbf{fix}(\!|X{:}X \stackrel{def}{=} \tau.X|\!)\downarrow$, the latter because in Definition 9 τ is allowed as a *guard*. The definitions of this paper are adapted to $\mathrm{CCS}_\perp$ by redefining P *diverges*, notation $P\Uparrow$, if either there is a P' with $P \Longrightarrow P'\uparrow$ or there are $P_i \in \mathbb{P}$ for all $i > 0$ such that $P \xrightarrow{\tau} P_1 \xrightarrow{\tau} P_2 \xrightarrow{\tau} \cdots$. In Definition 2, and similarly for Definition 1, clause (i) is replaced by (i′) "if T_n is the final element in π, then either $T_n\uparrow$ or $T_n \xrightarrow{\tau,r} T$ for no r and T". Now all results for CCS from Sections 3–12 remain valid for $\mathrm{CCS}_\perp$ as well. The only change in the proofs of Theorems 1–3, direction "$\Leftarrow$", is that finite paths ending in $\downarrow$ are treated like infinite paths.

My definition of $\sqsubseteq_{\mathrm{must}}$ on $\mathrm{CCS}_\perp$ differs on two points from the definition of $\sqsubseteq_{\mathrm{must}}$ on $\mathrm{CCS}_\perp$ from [6]. But both differences are inessential, and the resulting notion of $\sqsubseteq_{\mathrm{must}}$ is the same. The first difference is that in [6] the notion of computation is exactly as in Definition 1, rather than the amended form above. However, in [6] a computation $\pi = T_0, T_1, T_2, \ldots \in \mathit{Comp}(T|P)$ counts as successful only if (a) it contains a state T with $T \xrightarrow{\omega} T'$ for some T', and (b) if $T_k\uparrow$ then $T_{k'} \xrightarrow{\omega} T'$ for some T' and some $k' \leq k$. It is straightforward to check that $\mathcal{A}\mathit{pply}(T|P)$ remains the same upon dropping (b) and changing (i) into (i′). The other difference is that in [6] τ does not count as a guard—their version of Definition 9 requires $\alpha \in \mathscr{A}$. So in [6] one has $\mathbf{fix}(\!|X{:}X \stackrel{def}{=} \tau.X|\!)\uparrow$. The notion of $\downarrow$ from [6] is therefore closer to unguarded recursion rather than strongly unguarded recursion. However, in the treatment of [6] one would have $\mathbf{fix}(\!|X{:}X \stackrel{def}{=} a.X|\bar{a}|\!)\downarrow$, showing that the resulting notion of guardedness is not very robust. Since the essential difference between CCS and $\mathrm{CCS}_\perp$ is that in $\mathrm{CCS}_\perp$ a strongly unguarded recursion is treated as a divergence, it does not matter whether $\downarrow$ also includes all or some not-strongly unguarded recursions, such as $\mathbf{fix}(\!|X{:}X \stackrel{def}{=} \tau.X|\!)$. For any such not-strongly unguarded recursion is already divergent, and hence it does not make difference whether it is declared syntactically divergent as well.

An alternative to moving from CCS to $\mathrm{CCS}_\perp$ is to restrict either language to processes P satisfying $P\downarrow$. This restriction rules out the process $\mathbf{fix}(\!|X{:}X \stackrel{def}{=} X|\!)$, but includes $\mathbf{fix}(\!|X{:}X \stackrel{def}{=} \tau.X|\!)$. On this restricted set of processes their is no difference between CCS and $\mathrm{CCS}_\perp$.

[2] All processes of Example 7 are *weakly bisimilar* [24]. In my view this does not mean that weak bisimulation semantics uses a variant of CCS in which none of these processes diverges. Instead it tells that weak bisimilarity abstracts from divergence.

[3] Un(strongly unguarded) recursion should not be called "strongly guarded" recursion; it is weaker than guarded recursion.

Another approach to making unguarded recursions divergent is to change the rule (REC) from Table 1 into $\mathbf{fix}(\!|X{:}S|\!) \xrightarrow{\tau} \mathbf{fix}(\!|S_X{:}S|\!)$; this is done in the setting of CSP [26]. This would not have the same result, however, as here and in [6] one has $a+\mathbf{fix}(\!|X{:}X \stackrel{def}{=} b|\!) \equiv_{\text{must}} a+b$.

The great advantage of moving from CCS to $\text{CCS}_\perp$ is that Counterexample 6, against testing preorders being congruences for recursion, disappears.

Question 1 Are $\sqsubseteq^\tau_{\text{reward}}$, $\sqsubseteq^\tau_{\text{fp-reward}}$ and $\sqsubseteq^\tau_{\text{must}}$ precongruences for recursion on $\text{CCS}_\perp$?

In [6] it is shown that, in the absence of infinite choice, $\sqsubseteq^\tau_{\text{must}}$ is a precongruences for recursion. Central in the proof is that on $\text{CCS}_\perp$ with finite choice, the clause on infinite traces ($\mathit{inf}_\perp(P) \supseteq \mathit{inf}_\perp(Q)$) may be dropped from Theorem 3, since the infinite traces $\mathit{inf}_\perp(P)$ of a $\text{CCS}_\perp$ process P with finite choice are completely determined by $\mathit{divergences}_\perp(P)$ and $\mathit{failures}_\perp(P)$. This proof does not generalise to $\sqsubseteq^\tau_{\text{reward}}$ or $\sqsubseteq^\tau_{\text{fp-reward}}$, since here, on $\text{CCS}_\perp$ with finite choice, the infinite traces are not redundant. The proof also does not generalise to $\sqsubseteq^\tau_{\text{must}}$ on CCS with infinite choice.

In [28] it is shown that $\sqsubseteq^\perp_{FDI}$ (cf. Theorem 3), which coincides with $\sqsubseteq_{\text{must}}$, is a congruence for recursion on the language CSP. I expect that similar reasoning can show that $\sqsubseteq^\tau_{\text{reward}}$ is a congruence for recursion on $\text{CCS}_\perp$. In [29] it is shown that $\sqsubseteq^d_{FDI}$ (cf. Theorem 2), which coincides with $\sqsubseteq_{\text{fp-reward}}$, is a congruence for recursion on CSP. I expect that similar reasoning can show that $\sqsubseteq^\tau_{\text{fp-reward}}$ is a congruence for recursion on $\text{CCS}_\perp$. Roscoe [29] also presents an example, independently discovered by Levy [23], showing that $\equiv_{NDFD}$ (cf. Theorem 1), which coincides with $\sqsubseteq_{\text{reward}}$, fails to be a congruence for recursion:[4] Let FA be a process that has *all* conceivable failures, divergences and infinite traces, except for the infinite trace a^∞. Then $FA+\tau.X \equiv_{NDFD} FA+a.X$, for both sides have all conceivable failures, divergences and infinite traces, with the possible exception of a^∞, and both side have the infinite trace a^∞ iff X has it. However,

$$\mathbf{fix}(\!|X{:}FA+\tau.X|\!) \not\equiv_{NDFD} \mathbf{fix}(\!|X{:}FA+a.X|\!)$$

since only the latter process has the infinite trace a^∞.

It could be argued that this example shows that the definition of being a congruence for recursion ought to be sharpened, for instance by requiring that $E \sqsubseteq F$ holds only if all closed substitutions of $E \sqsubseteq F$ employing an extended alphabet of actions hold. This would invalidate $FA+\tau.X \equiv_{NDFD} FA+a.X$, namely by substituting b for X, with b a fresh action, not alluded to in FA. With such a sharpening, the question whether $\sqsubseteq^\tau_{\text{reward}}$ is a congruence for recursion on $\text{CCS}_\perp$ is open.

15 Related work

The concept of reward testing stems from [18], in the setting of nondeterministic probabilistic processes. In the terminology of Section 7, they employ single reward nonnegative reward testing. In [10] it was shown, again in a probabilistic setting, that nonnegative reward testing is no more powerful then classical testing. This result is a probabilistic analogue of Theorem 7. Negative rewards were first proposed in [11], a predecessor of the present paper. In [8], reward testing with also negative rewards, called *real-reward* testing, was applied to nondeterministic probabilistic processes. Although technically no rewards can be gathered after a first reward has been encountered, thanks to probabilistic branching rewards can be distributed over multiple actions in a computation. This makes the approach a probabilistic generalisation of the reward testing proposed here. The main result of [8] is that for finitary (= finite-state and finitely many transitions) nondeterministic probabilistic processes without divergence, real-reward testing coincides with nonnegative reward testing. This is a generalisation (to probabilistic processes) of

[4] The example was formulated for another equivalence, but actually applies to a range of equivalences, including $\equiv_{NDFD}$.

a specialisation (to finitary processes) of Proposition 1. An explicit characterisation (as in Theorem 1) of real-reward testing for processes with divergence was not attempted in [8].

The *nondivergent failures divergences* equivalence, $\equiv_{NDFD}$, defined in the proof of Theorem 1, stems from [19]. There it was shown to be the coarsest congruence (for a collection of operators equivalent to the ones used in Section 10) that preserves those linear-time properties (cf. Definition 7) that can be expressed in linear-time temporal logic without the nexttime operator. If follows directly from their proof that it is also the coarsest congruence that preserves *all* linear-time properties as defined in Definition 7; so $\equiv_{NDFD}$ coincides with $\equiv_{lt.\ properties}$, as remarked at the end of Section 10. It is this result that inspired Theorem 1 in the current paper.

The paper [22] argues that $\equiv_{NDFD}$ can be seen as a testing equivalence, but does not offer a testing scenario in quite the same style as [6] or the current paper.

The semantic equivalence $\equiv^{d}_{FDI}$, whose associated preorder occurs in the proof of Theorem 2, stems from [27]. There it was shown to be the coarsest congruence (for the same operators) that preserves $\mathit{deadlocks}(P) \cup \mathit{divergences}(P)$, the combined deadlock and divergence traces of a process (cf. Definition 3). It is this result that directly led (via [12, Theorem 9]) to Theorem 2 in the current paper.

In [6] the action ω is used merely to mark certain states as success states, namely the states were an ω-transition is enabled; a computation is successful iff it passes through such a success state. In [30], on the other hand, it is the *actual execution* of ω that constitutes success. In [10, 7], this is called *action-based testing*; [7, Proposition 5.1 and Example 5.3] shows that action-based must testing is strictly less discriminating than state-based must-besting:

$$\tau.a.\Omega \equiv^{\text{action-based}}_{\text{must}} \tau.a.\Omega + \tau.\mathbf{0}, \qquad \text{whereas} \qquad \tau.a.\Omega \not\sqsubseteq_{\text{must}} \tau.a.\Omega + \tau.\mathbf{0}.$$

The preorders in the current paper are generalisations of state-based testing; an action-based form of reward testing could be obtained by only allowing τ-actions to carry non-0 rewards. The same counterexample as above would show the difference between state- and action-based reward testing.

The reward testing contributed here constitutes a strengthening of the testing machinery of De Nicola & Hennessy. As such it differs from testing-based approaches that lead to incomparable preorders, such as the *efficiency testing* of [31], or the *fair testing* independently proposed in [4] and [25].

In [13] I advocate an overhaul of concurrency theory to ensure liveness properties when making the reasonable assumption of *justness*. The current work is prior to any such overhaul. It is consistent with the principles of [13] when pretending that the parallel composition $|$ of CCS is in fact not a parallel composition of independent processes, but an interleaving operator, scheduling two parallel treads by means of arbitrary interleaving.

16 Conclusion

In this paper I contributed a concept of reward testing, strengthening the may and must testing of De Nicola & Hennessy. Inspired by [19, 27], I provided an explicit characterisation of the reward-testing preorder, as well as of a slight weakening, called finite-penalty reward testing. Must testing can be recovered by only considering positive rewards, and may testing by only considering negative rewards. While the must-testing preorder preserves liveness properties, and the inverse of the may-testing preorder (which can also be seen as a must-testing preorder dealing with catastrophes rather than successes) preserves safety properties, the (finite-penalty) reward testing preorder, which is finer than both, additionally preserves conditional liveness properties. I illustrated the difference between may testing, must testing and (finite-penalty) reward testing in terms of their equational axiomatisations. When applied to CCS as intended by Milner, must-testing equivalence fails to be a congruence for recursion, and the same problem exists for reward testing. The counterexample is eliminated by applying it to a small variant of

CCS that, following [6], treats a process with unguarded recursion as if it is diverging, even if it cannot make any internal moves. In this setting, by analogy with Roscoe's work on CSP [28, 29], I expect must-testing and finite-penalty reward testing to be congruences for recursion; for reward testing this question remains open.

References

[1] S. Abramsky & A. Jung (1994): *Domain Theory*. In: *Handbook of Logic and Computer Science*, 3, Clarendon Press, pp. 1–168.

[2] B. Alpern & F.B. Schneider (1985): *Defining liveness*. *Information Processing Letters* 21(4), pp. 181–185, doi:10.1016/0020-0190(85)90056-0.

[3] J.A. Bergstra, J.W. Klop & E.-R. Olderog (1987): *Failures without chaos: a new process semantics for fair abstraction*. In M. Wirsing, editor: *Formal Description of Programming Concepts – III, Proceedings of the 3th IFIP WG 2.2 working conference*, Ebberup 1986, North-Holland, Amsterdam, pp. 77–103.

[4] E. Brinksma, A. Rensink & W. Vogler (1995): *Fair Testing*. In I. Lee & S. Smolka, editors: Proceedings 6th International Conference on *Concurrency Theory*, (CONCUR'95), Philadelphia, PA, USA, August 1995, LNCS 962, Springer, pp. 313–327, doi:10.1007/3-540-60218-6_23.

[5] T. Chen, W.J. Fokkink & R.J. van Glabbeek (2015): *On the Axiomatizability of Impossible Futures*. *Logical Methods in Computer Science* 11(3):17, doi:10.2168/LMCS-11(3:17)2015.

[6] R. De Nicola & M. Hennessy (1984): *Testing equivalences for processes*. *Theoretical Computer Science* 34, pp. 83–133, doi:10.1016/0304-3975(84)90113-0.

[7] Y. Deng, R.J. van Glabbeek, M. Hennessy & C.C. Morgan (2008): *Characterising Testing Preorders for Finite Probabilistic Processes*. *Logical Methods in Computer Science* 4(4):4, doi:10.2168/LMCS-4(4:4)2008.

[8] Y. Deng, R.J. van Glabbeek, M. Hennessy & C.C. Morgan (2014): *Real-Reward Testing for Probabilistic Processes*. *Theoretical Computer Science* 538, pp. 16–36, doi:10.1016/j.tcs.2013.07.016.

[9] Y. Deng, R.J. van Glabbeek, M. Hennessy, C.C. Morgan & C. Zhang (2007): *Remarks on Testing Probabilistic Processes*. In L. Cardelli, M. Fiore & G. Winskel, editors: *Computation, Meaning, and Logic: Articles dedicated to Gordon Plotkin*, *Electronic Notes in Theoretical Computer Science* 172, Elsevier, pp. 359–397, doi:10.1016/j.entcs.2007.02.013.

[10] Y. Deng, R.J. van Glabbeek, C.C. Morgan & C. Zhang (2007): *Scalar Outcomes Suffice for Finitary Probabilistic Testing*. In R. De Nicola, editor: Proceedings 16th *European Symposium on Programming*, ESOP 2007, Braga, Portugal, LNCS 4421, Springer, pp. 363–378, doi:10.1007/978-3-540-71316-6_25.

[11] R.J. van Glabbeek (2009): *The Linear Time Branching Time Spectrum after 20 years,* or *Full abstraction for safety and liveness properties*. Copies of slides. Invited talk for IFIP WG 1.8 at CONCUR 2009 in Bologna. Available at http://theory.stanford.edu/~rvg/abstracts.html#20years.

[12] R.J. van Glabbeek (2010): *The Coarsest Precongruences Respecting Safety and Liveness Properties*. In C.S. Calude & V. Sassone, editors: Proceedings 6th IFIP TC 1/WG 2.2 International Conference on *Theoretical Computer Science* (TCS 2010); held as part of the *World Computer Congress* 2010, Brisbane, Australia, *IFIP* 323, Springer, pp. 32–52, doi:10.1007/978-3-642-15240-5_3.

[13] R.J. van Glabbeek (2016): *Ensuring Liveness Properties of Distributed Systems (A Research Agenda)*. Position paper. Available at https://arxiv.org/abs/1711.04240.

[14] R.J. van Glabbeek (2017): *Lean and Full Congruence Formats for Recursion*. In: Proceedings 32nd Annual ACM/IEEE Symposium on *Logic in Computer Science*, LICS 2017, Reykjavik, Iceland, June 2017, IEEE Computer Society Press, doi:10.1109/LICS.2017.8005142.

[15] R.J. van Glabbeek & P. Höfner (2015): *Progress, Fairness and Justness in Process Algebra*. Technical Report 8501, NICTA, Sydney, Australia. Available at http://arxiv.org/abs/1501.03268.

[16] M. Hennessy (1982): *Powerdomains and nondeterministic recursive definitions*. In: Proceedings 5th Intern. Symposium on Programming, LNCS 137, Springer, pp. 178–193, doi:10.1007/3-540-11494-7_13.

[17] M. Hennessy (1988): *An Algebraic Theory of Processes*. MIT Press.

[18] B. Jonsson, C. Ho-Stuart & W. Yi (1994): *Testing and Refinement for Nondeterministic and Probabilistic Processes*. In: Proceedings of the 3rd International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems, LNCS 863, Springer, pp. 418–430, doi:10.1007/3-540-58468-4_176.

[19] R. Kaivola & A. Valmari (1992): *The Weakest Compositional Semantic Equivalence Preserving Nexttime-less Linear Temporal Logic*. In R. Cleaveland, editor: *CONCUR'92*, LNCS 630, Springer, pp. 207–221, doi:10.1007/BFb0084793.

[20] L. Lamport (1977): *Proving the correctness of multiprocess programs*. *IEEE Transactions on Software Engineering* 3(2), pp. 125–143, doi:10.1109/TSE.1977.229904.

[21] L. Lamport (1998): *Proving Possibility Properties*. *Theoretical Computer Science* 206(1-2), pp. 341–352, doi:10.1016/S0304-3975(98)00129-7. See especially http://research.microsoft.com/en-us/um/people/lamport/pubs/pubs.html#lamport-possibility.

[22] G. Leduc (1994): *Failure-based congruences, unfair divergences and new testing theory*. In S.T. Vuong & S.T. Chanson, editors: Proceedings Fourteenth IFIP WG6.1 International Symposium on Protocol Specification, Testing and Verification, Vancouver, BC, Canada, 1994, *IFIP Conference Proceedings* 1, Chapman & Hall, pp. 252–267.

[23] P.B. Levy (2008): *Infinite trace equivalence*. *Annals of Pure and Applied Logic* 151(2-3), pp. 170–198, doi:10.1016/j.apal.2007.10.007.

[24] R. Milner (1990): *Operational and algebraic semantics of concurrent processes*. In J. van Leeuwen, editor: *Handbook of Theoretical Computer Science*, chapter 19, Elsevier Science Publishers B.V. (North-Holland), pp. 1201–1242. Alternatively see *Communication and Concurrency*, Prentice-Hall, Englewood Cliffs, 1989.

[25] V. Natarajan & R. Cleaveland (1995): *Divergence and Fair Testing*. In Z. Fülöp & F. Gécseg, editors: Proceedings 22nd International Colloquium on *Automata, Languages and Programming* (ICALP'95), Szeged, Hungary, July 1995, LNCS 944, Springer, pp. 648–659, doi:10.1007/3-540-60084-1_112.

[26] E.-R. Olderog & C.A.R. Hoare (1986): *Specification-oriented semantics for communicating processes*. *Acta Informatica* 23, pp. 9–66, doi:10.1007/BF00268075.

[27] A. Puhakka (2001): *Weakest Congruence Results Concerning "Any-Lock"*. In N. Kobayashi & B. Pierce, editors: Proceedings 4th International Symposium on *Theoretical Aspects of Computer Software*, TACS 2001, Sendai, Japan, 2001, LNCS 2215, Springer, pp. 400–419, doi:10.1007/3-540-45500-0_20.

[28] A.W. Roscoe (1997): *The Theory and Practice of Concurrency*. Prentice-Hall. Available at http://www.comlab.ox.ac.uk/bill.roscoe/publications/68b.pdf.

[29] A.W. Roscoe (2005): *Seeing Beyond Divergence*. In A.E. Abdallah, C.B. Jones & J.W. Sanders, editors: *Communicating Sequential Processes: The First 25 Years, Symposium on the Occasion of 25 Years of CSP*, London, UK, 2004, Revised Invited Papers, LNCS 3525, Springer, pp. 15–35, doi:10.1007/11423348_2.

[30] R. Segala (1996): *Testing Probabilistic Automata*. In: Proceedings of the 7th International Conference on *Concurrency Theory*, LNCS 1119, Springer, pp. 299–314, doi:10.1007/3-540-61604-7_62.

[31] W. Vogler (2002): *Efficiency of asynchronous systems, read arcs, and the MUTEX-problem*. *Theoretical Computer Science* 275(1-2), pp. 589–631, doi:10.1016/S0304-3975(01)00300-0.