Theoretical Computer Science 411 (2010) 1311–1337

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

# CSP is a retract of CCS

He Jifeng[a], Tony Hoare[b,*]

[a] SEI, East China Normal University, Shanghai, China
[b] Microsoft Research Ltd., Cambridge, United Kingdom

ARTICLE INFO

*Article history:*
Received 9 November 2006
Received in revised form 23 November 2009
Accepted 18 December 2009
Communicated by R. van Glabbeek

*Keywords:*
Unifying theories of programming
Process algebra
CCS
CSP
Retraction

ABSTRACT

Automata theory provides two ways of defining an automaton: either by its transition system, defining its states and events, or by its language, the set of sequences (traces) of events in which it can engage. For many classes of automaton, these forms of definition have been proved equivalent. For example, there is a well-known isomorphism between regular languages and finite deterministic automata. This paper suggests that for (demonically) the non-deterministic automata (as treated in process algebra), the appropriate link between transition systems and languages may be a retraction rather than an isomorphism.

A pair of automata, defined in the tradition of CCS by their transition systems, may be compared by a pre-ordering based on some kind of simulation or bisimulation, for example, weak, strong, or barbed. Automata defined in the tradition of CSP are naturally ordered by set inclusion of their languages (often called refinement); variations in ordering arise from different choices of basic event, including for example, refusals and divergences. In both cases, we characterise a theory by its underlying transition system and its choice of ordering. Our treatment is therefore wholly semantic, independent of the syntax and definition of operators of the calculus.

We put forward a series of retractions relating the above-mentioned versions of CSP to their corresponding CCS transition models. A retraction is an injection that is (with respect to the chosen ordering) monotonic, increasing and idempotent (up to equivalence). It maps the nodes of a transition system of its source theory to those of a system that has been saturated by additional transitions. Each retraction will be defined by a transition rule, in the style of operational semantics; the proofs use the familiar technique of co-induction, often abbreviated by encoding in the relational calculus.

The aim of this paper is to contribute to unification of theories of reactive system programming. More practical benefits may follow. For example, we justify a method to improve the efficiency of model checking based on simulation. Furthermore, we show how model checking of a transition network fits consistently with theorem-proving tools, which reason directly about specifications and designs that are expressed in terms of sets of sequences of observable events.

## 1. Introduction

The purpose of the study of process algebra is to provide a conceptual framework and theory for the description or specification and design of physical processes (natural or artificial) which act and interact continuously with their

---

[*] Corresponding address: Microsoft Research Ltd., 7 JJ Thomson Avenue, CB3 0FB Cambridge, United Kingdom. Tel.: +44 1223 479 800.
*E-mail address:* thoare@microsoft.com (T. Hoare).

0304-3975/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2009.12.012

environment. Environments are often similarly modelled as concurrent processes. The nature of each observed interaction is indicated by a symbol from some explicit or implicit alphabet.

A process may be defined by a transition system, defined as a directed arc-labelled graph whose nodes represent states of the process, and whose arcs are labelled by a symbol denoting an observation that can be made when the process makes a transition from the state at the tail of the arc to its head. This is the approach taken in setting up the framework of the process calculus CCS [15,16]. Alternatively, a process can be defined by its language, consisting of a set of sequences of symbols (traces); these denote possible sequences of observations of the events that constitute the interactive behaviour of the process. Traces are the foundation of the definition of CSP [4], which places certain healthiness (closure) conditions on which sets of traces can constitute a process. The two approaches have the following complementary advantages.

A transition system describes in detail how a process moves from one state to another. It therefore gives an operational semantics for a process description, showing how it can be implemented as a program or collection of programs on a computer system. If the system then does not behave properly, the operational semantics is directly applicable to identify the place and the cause of the error. Even better, the absence of errors can be checked before testing the program by the method of high-speed model checking, which is based on some appropriate notion of simulation or bisimulation.

The trace model of a process may be used at an earlier stage in the design of a system. The intended behaviour of the system is first specified by a formal statement of the desired properties of its traces. Such requirements may be built up to a complete specification by simple conjunction. Disjunction and even negation can be used freely, together with all other helpful concepts of mathematics. The design of a fully specified system proceeds similarly by specifying its components, and by combining them with operators mathematically defined on trace sets. Mathematical proof (ideally checked by computer) can then guarantee that the design logically implies (i.e., refines) its specification. This top-down design procedure (including stepwise and componentwise refinement) aims at correctness by construction: in principle errors are eliminated at source.

This paper aims to provide a sound theoretical basis for combination of the merits of both approaches to the definition of a process. This may be useful when proof tools are used in combination to ensure correctness. For example, model checking often requires or benefits from abstraction over excessively large or infinite sets of states; and refinement often supplies a natural notion of abstraction. More generally, the unification of diverse theories is accepted as an independent goal of scientific research, rightly considered worthy of pursuit in its own right [11]. A unified theory inherits and enhances the credibility of all the theories that it unifies. We claim that the technique of retraction is a common and useful form of unification, because it preserves and explains the distinctive merits of each theory. In particular, this paper aims to complete the unfinished business of the original ESPRIT Basic Research Action CONCUR [BRA 3009, 1989–92], which aimed to assimilate the theories and notations of CSP, ACP and CCS.

Our development starts with a universal transition system — universal in the sense that it includes a subsystem isomorphic to every automaton definable in CCS or any similar calculus. We select as our initial pre-order the concept of strong simulation (denoted $\leq$), which is the asymmetric version of the more familiar CCS concept of strong bisimulation. (The familiar deficiencies of simulation as a notion of correctness will be resolved later by introduction of a well-chosen notion of tests or barbs.) Equivalence of processes (denoted by $\equiv$) is defined in this paper as mutual simulation. An elementary introduction to the ideas of simulation and trace refinement is given in Section 2.

Section 3 introduces the notion of a link $L$ between transition systems. This is an injective endomorphism defined over the nodes of the universal transition system. When its domain is restricted to transition systems for its source theory, its range determines the transition systems for its target theory. A new pre-order in the source theory is defined, as in [7], by applying $L$ to the two processes of the source theory; then the pre-order of the source theory is used to compare them in the target theory. More formally, $Lp$ and $Lq$ are ordered in the target theory if and only if $p$ and $q$ are ordered in the source theory. Since the source theory uses simulation as its pre-order, the pre-orders of all the linked theories can be efficiently model-checked by the same standard algorithm, after application of $L$ to both operands; such an option is now provided by the Concurrency Workbench [6].

In Section 4, we explore some of the desirable properties of the link $L$. An important property is that it should be idempotent, in the sense that applying it twice is equivalent to applying it just once. This means that a healthiness condition defining exactly the nodes of the target theory can be expressed as a fixed-point equivalence of the form $p \equiv Lp$. Secondly $L$ should be monotonic, in the sense that it respects the source ordering. This ensures that all ordering theorems of the source theory are preserved in the target theory. Thirdly, $L$ should be decreasing, in the sense that its result is always lower in the source ordering than its argument. This means that model checking the ordering of the target theory can be optimised, by applying $L$ only to the left operand of the ordering ($Lp \leq q$), rather than both operands. This fact has been industrially exploited in the model-checking algorithm of FDR [21,22]. A function that has all these three properties is called a retraction. A closure operator of topology has exactly the same definition, with respect to the ordering of set containment. Retractions also arise as a result of composing the two functions of a Galois connection.

The next two sections define a series of functions linking various versions of CCS to corresponding versions of CSP.

(1) The trace model of CSP (with $\tau$ recorded) is a retract of CCS modulo strong simulation, by a retraction $T$ (Section 5.1).

(2) CCS modulo weak simulation is a retract of CCS modulo strong simulation, by a retraction $W$ (Section 5.2).

(3) The trace model of CSP (with $\tau$ omitted) is a retract of CCS modulo weak simulation, by the retraction $(W; T)$ (Section 5.3).

(4) CCS modulo refusal-barbed simulation is a retract of CCS modulo two-thirds simulation, by a retraction $R$ (Section 6.1).

(5) The trace/refusal model of CSP is a retract of CCS modulo weak refusal-barbed simulation, by the retraction $R; W; T$ (Section 6.2).

(6) The FDR (failures–divergence–refinement) model of CSP is a retract of CCS modulo weak refusal- and divergence-barbed simulation, by the retraction $(D; R; W; T)$ (Section 6.3).

Section 7 gives a brief survey of related work.

## 2. Background: Simulation and traces

We define both simulation and trace ordering in terms of a labelled transition system. This consists of:

| | |
|---|---|
| a denumerable set $P$ of process states: | $nil, p, q, Lp, \ldots$ |
| a denumerable set $A$ of observations: | $a, b, \ldots$ |
| including communications: | $x, y, \ldots$ |
| and hidden symbols: | $\tau, \sigma, \ldots$ |
| and barbs, which have special meanings: | $\mathrm{ref}(X), \delta, \ldots$ |
| a transition relation $T \subseteq P \times A \times P$ | |

As usual, we identify a process with its initial state. The fact that $(p, a, q) \in T$ means that a process in initial state $p$ can make a transition to state $q$, and simultaneously admit or emit the observation $a$.

A typical small transition system contains only just enough nodes to represent the full behaviour of a single process expressed syntactically in a calculus like CCS or CSP. A universal transition system is constructed as the disjoint union of all such small transition systems. The technical details are beyond the scope of this paper.

We will exploit infix relational notation, and define (using $\cong$ for definition)

$$\xrightarrow{\langle a\rangle} \cong \{(p, q) \mid (p, a, q) \in T\}.$$

We use the identity relation (Id), forward relational composition ($;$), the universal relation ($U = P \times P$), the empty relation ($\{\ \}$), relational union ($\cup$), relational converse (superscript $^{\cup}$) and inclusion ($\subseteq$); we appeal without mention to the familiar algebraic properties of the relational calculus: unit laws, associativity, monotonicity, distribution through union.

The concept of an observation extends to sequences of none or more observations, denoted by $s, t, \ldots \in A^*$ (the set of all sequences of symbols drawn from $A$). The following definitions are standard:

$$\begin{aligned}
p \xrightarrow{\varepsilon} q \quad &\cong p = q \\
p \xrightarrow{\langle a\rangle s} r \quad &\cong p(\xrightarrow{\langle a\rangle}; \xrightarrow{s})r \\
p \xrightarrow{s}\_ \quad &\cong \exists q.p \xrightarrow{s} q \\
traces(p) \ &\cong \{s \mid p \xrightarrow{s}\_\}.
\end{aligned}$$

Simulation is defined co-inductively as the weakest (largest in set-inclusion ordering) solution of a set of inclusions. Since relational composition distributes through arbitrary unions, such a weakest solution always exists, definable as the union of all solutions.

$\leq$ is the weakest relation $R \subseteq P \times P$ satisfying the inclusion

$$\forall a : A, R; \xrightarrow{\langle a\rangle} \subseteq \xrightarrow{\langle a\rangle}; R.$$

(For bisimulation, $R$ must be symmetric — but we shall not be using bisimulation in this paper). An important practical advantage in the choice of simulation as the standard pre-order for a theory of concurrency is that its definition describes abstractly but exactly the algorithm for an efficient model checker; model checking can automatically find counterexamples for false conjectures in the theory, and often prove true ones.

**Theorem 2.1.** *$\leq$ is a pre-order.*

**Proof** (*For Reflexivity*). Id (the identity relation on processes) satisfies the inclusions that define simulation. It is therefore included in the weakest such relation, or more formally, Id $\subseteq \leq$. That is just the relational expression of reflexivity.

(for transitivity): similarly, $(\leq; \leq)$ also satisfies the inclusions. □

**Lemma 2.1.**

$$\forall s : A^*, \leq; \xrightarrow{s} \subseteq \xrightarrow{s}; \leq .$$

**Proof.** By induction on the length of $s$. □

CSP refinement (denoted $\sqsubseteq$) can be defined in the same style as simulation:

$\sqsubseteq$ is the weakest relation $R \subseteq P \times P$ such that

$$\forall s : A^*, R; \xrightarrow{s}; U \subseteq \xrightarrow{s}; U.$$

As before, such a weakest relation exists, and satisfies its defining inclusion, actually as an equation:

**Lemma 2.2.**

$$\sqsubseteq; \xrightarrow{s}; U = \xrightarrow{s}; U.$$

**Theorem 2.2.** *$\sqsubseteq$ is a pre-order.*

**Proof.** Similar to the above. □

**Theorem 2.3.**

$$(\leq) \subseteq (\sqsubseteq).$$

**Proof.** By Lemma 2.1, $\leq$ satisfies the defining inclusion for $\sqsubseteq$. □

**Corollaries.**

$$\begin{array}{lll} \leq; \sqsubseteq & = \sqsubseteq & \{\leq \mathit{refl}, \sqsubseteq \mathit{trans}\} \\ R; \leq \subseteq S; \leq & \Rightarrow R; \sqsubseteq \subseteq S; \sqsubseteq. & \end{array}$$

*Most importantly, $\sqsubseteq$ coincides with the more usual definition of trace refinement:*

**Theorem 2.4.**

$$p \sqsubseteq q \Leftrightarrow traces(q) \subseteq traces(p).$$

**Proof.**

$$\begin{array}{lcl} & RHS & \\ \Leftrightarrow & q \xrightarrow{s} \_ \Rightarrow p \xrightarrow{s} \_, \forall s \in A^* & \{\text{by def traces}\} \\ \Leftrightarrow & \{(p,q)\}; \xrightarrow{s}; U \subseteq \xrightarrow{s}; U & \{\text{rewriting}\} \\ \Rightarrow & \{(p,q)\} \subseteq \sqsubseteq & \{\text{by def} \sqsubseteq\} \\ \Leftrightarrow & LHS & \{\text{rewriting}\} \\ \Rightarrow & q \xrightarrow{s} \_ \Rightarrow p \xrightarrow{s} \_, \forall s \in A^* & \{\text{by Lemma 2.2}\} \\ \Leftrightarrow & RHS & \{\text{by def traces}\}. \end{array}$$ □

**Corollary.**

$$R; \xrightarrow{s}; U \Rightarrow S; \xrightarrow{s}; U, \quad \textit{for all } s$$
$$\Leftrightarrow R; \sqsubseteq \subseteq S; \sqsubseteq.$$

## 3. Functions defined by transition rules

A new function symbol $L$ is commonly introduced into a process algebra by a transition rule of the form

$$\frac{p(fa)q}{Lp \xrightarrow{\langle a \rangle} Lq}$$

where $fa$ is a relation defined in terms of the observation $a$. By suitable definition of the relation $fa$, a multiplicity of transition rules defining the same symbol $L$ can often be reduced to a single rule of the form shown above.

**Example 3.1** (*Restriction*). Let $a$ name an observation of a process $p$. We may decide that we no longer wish this event to occur. We therefore define a process $Vp$ which behaves in all ways like $p$, except that it never gives rise to the observation $a$. The function $V$ (veto) is defined by the single transition rule

$$\frac{p \xrightarrow{\langle c \rangle} q}{Vp \xrightarrow{\langle c \rangle} Vq} \quad \text{if } c \neq a.$$

The side-condition can be eliminated by defining a relation to serve as antecedent:

$$\begin{array}{ll} fc \cong \{\,\} & \text{if } c = a, \\ \quad \cong \xrightarrow{\langle c \rangle} & \text{otherwise.} \end{array}$$

**Example 3.2** (*Symbol Change*). Let $a$ and $b$ be distinct names for observations of a process $p$. Suppose we wish to replace every occurrence of a $b$ with an $a$. We therefore define a process $Cp$ which behaves like $p$, except that it gives rise to an observation $b$ whenever $p$ would have given rise to observation $a$. The function $C$ (change) is defined by the pair of transition rules

$$\begin{array}{c} p \xrightarrow{\langle a\rangle} q \\ \text{-----------} \\ Cp \xrightarrow{\langle b\rangle} Cq \end{array} \qquad \begin{array}{c} p \xrightarrow{\langle c\rangle} q \\ \text{-----------} \\ Cp \xrightarrow{\langle c\rangle} Cq \end{array} \qquad \text{if } c \neq a.$$

These can be reduced to a single rule by defining

$$\begin{aligned} &fa \cong \{\ \} \\ &fb \cong \xrightarrow{\langle a\rangle} \cup \xrightarrow{\langle b\rangle} \\ &fc \cong \xrightarrow{\langle c\rangle} \qquad\qquad \text{otherwise.} \end{aligned}$$

In a structured operational semantics, a function $L$ defined by transition rules is intended to be a syntactic operator on expressions denoting processes. It is therefore an injection, i.e., a total invertible function. The image of $L$ is a region of the source transition system in which every node is labelled by a term beginning with $L$; it will contain an edge-labelled $a$ if and only if the existence of that edge can be deduced from the single transition rule defining the function. Since we want to ignore syntax, we will just assume the existence of such an injection $L$ defined on our basic universal transition system.

Relational notation can be used to express and prove the properties of functions, by defining $pLq$ and $qL^{\cup}p$ both to mean $q = Lp$. The fact that $L$ is an injection can be expressed relationally:

$$\begin{array}{lll} \text{because } L \text{ is single-valued:} & L^{\cup}; L \subseteq \mathrm{Id} & \\ \text{because } L \text{ is a total invertible function:} & L; L^{\cup} = \mathrm{Id} & \ldots\ldots(L \text{ inj}). \end{array}$$

A name appearing in brackets after ...... is used later in braces to reference the preceding fact(s).

Since a transition rule for $L$ defines exactly the set of all transitions involving $L$ that can be deduced from it, the single transition rule can be read as an equivalence:

$$p\,(fa)\,q \Leftrightarrow Lp \xrightarrow{\langle a\rangle} Lq\,, \quad \text{for all } p \text{ and } q.$$

Because all transitions deducible by the single rule lead from the image of $L$ to itself, two further equivalences may be derived from the same rule:

$$\begin{aligned} &Lp \xrightarrow{\langle a\rangle} r \Leftrightarrow \exists q.r = Lq \,\&\, p(fa)q \\ &r \xrightarrow{\langle a\rangle} Lq \Leftrightarrow \exists p.r = Lp \,\&\, p(fa)q. \end{aligned}$$

All three equivalences can be neatly coded in the relational calculus:

$$\begin{aligned} &fa && = L; \xrightarrow{a}; L^{\cup} \\ &L; \xrightarrow{\langle a\rangle} && = fa; L \\ &\xrightarrow{\langle a\rangle}; L^{\cup} = L^{\cup}; fa && \qquad \ldots\ldots.(L \text{ commut}). \end{aligned}$$

An immediate consequence is a useful commuting law for $(L; \leq)$:

**Lemma 3.1.**

$$(L; \leq); \xrightarrow{a} \subseteq fa; (L; \leq) \quad \ldots\ldots.((L; \leq) \text{ commut}).$$

This property can be used as a co-inductive characterisation of $L; \leq$.

**Theorem 3.1.** *$L; \leq$ is the weakest relation $R \subseteq P \times P$ such that*

$$\forall a : A, R; \xrightarrow{\langle a\rangle} \subseteq fa; R.$$

**Proof.** Let $R$ satisfy the commuting property quoted in the theorem.

$$\begin{array}{lll} L^{\cup}; R; \xrightarrow{\langle a\rangle} \subseteq L^{\cup}; fa; R & = \xrightarrow{\langle a\rangle}; L^{\cup}; R & \{R, L \text{ commut}\} \\ \Rightarrow \quad L^{\cup}; R \subseteq \leq & & \{\text{def} \leq\} \\ \Rightarrow \quad L; L^{\cup}; R \subseteq L; \leq & & \\ \Rightarrow \quad L; \leq \text{ is weaker than } R. & & \{L \text{ inj}\}. \end{array}$$

By Lemma 3.1, $(L; \leq)$ has the relevant commuting property. Being weaker than an arbitrary commuting $R$, $(L; \leq)$ is the weakest such commuting relation.

Note that $L; \leq$ is not in general a pre-order, and so it is not a candidate for defining the ordering relation for the target theory. For this, we will use an ordering defined by applying the injection $L$ to both operands of the source ordering:

$$p \leq_L q \cong Lp \leq Lq, \quad \text{all } p, q.$$

An algebraic presentation of the same definition, and a similar one for refinement, is

$$\begin{array}{l} \leq_L = L;\ \leq\ ;\ L^{\cup} \\ \sqsubseteq_L \cong L;\ \sqsubseteq\ ;\ L^{\cup}. \quad \square \end{array}$$

**Lemma 3.2.** *$\leq_L$ is a pre-order, and so is $\sqsubseteq_L$.*

**Proof.**

$$\begin{array}{lll} (\leq_L \text{ reflexive})\ \mathrm{Id} \subseteq L; L^{\cup} & \subseteq L;\ \leq;\ L^{\cup} & \\ (\leq_L \text{ transitive})\ (L;\ \leq;\ L^{\cup});\ (L;\ \leq;\ L^{\cup}) & \subseteq L;\ \leq;\ \leq;\ L^{\cup} & \{L \text{ inj}\} \\ & \subseteq L;\ \leq;\ L^{\cup}. \quad \square & \end{array}$$

The following theorem gives a co-inductive characterisation of this pre-order.

**Theorem 3.2.**

*$\leq_L$ is the weakest relation $R \subseteq P \times P$ such that*

$$\forall a : A. \quad R; fa \subseteq fa; R.$$

**Proof.** As in Theorem 3.1, we establish $L^{\cup}\ ;\ R;\ L \subseteq \leq$, and use ($L$ inj). $\square$

## 4. Retractions

In this section we discuss the desirable properties of the ordering $\leq_L$, and relate them to desirable properties of the function $L$.

**Monotonicity.** First, we would like $\leq_L$ to be uniformly weaker than standard simulation, so that all the simulations provable in CCS will still be valid in the new theory as well; consequently, the new calculus will be a member of the CCS family.

This requirement is just that $L$ should be monotonic in the source ordering, which can be expressed relationally in three equivalent ways:

$$\begin{array}{ll} \leq & \subseteq \leq_L \\ \leq; L & \subseteq L; \leq \\ \leq; L; \leq\ = L; \leq & \quad \ldots\ldots.(L \text{ mon}). \end{array}$$

**Example 4.1.** Symbol change (Example 3.2) and restriction (Example 3.1) are monotonic.

Monotonicity can often be proved from the definition of $L$, using the defining relation $fa$, as described in Section 3. The following theorem is in the spirit of the congruence rules given in [10]; however, the format of the conclusion of our rules is far more rigid. Conversely, we place no syntactic restriction at all on the relation $fa$ in the antecedent of the rule.

**Theorem 4.1.**

*$L$ is monotonic if $\leq; fa \subseteq fa; \leq$,*

*for all $a$ s.t. $L; \xrightarrow{\langle a\rangle} \neq \{\}$.*

**Proof.**

$$\begin{array}{lllll} & L^{\cup};\ \leq;\ L;\ \xrightarrow{\langle a\rangle} & = L^{\cup};\ \leq;\ fa;\ L & & \{L \text{ commut}\} \\ & & \subseteq L^{\cup};\ fa;\ \leq;\ L & & \{\text{trivially if } L;\ \xrightarrow{\langle a\rangle} = \{\}, \\ & & & & \ \text{otherwise} \leq;\ fa \subseteq fa;\ \leq\} \\ & & = \xrightarrow{\langle a\rangle};\ L^{\cup};\ \leq;\ L & & \{L \text{ commut}\} \\ \Rightarrow & L^{\cup};\ \leq;\ L \subseteq & \leq & \{\text{def} \leq\} & \\ \Rightarrow & \leq;\ L \subseteq L;\ \leq & & \{\text{by } L \text{ inj}\}. \quad \square & \end{array}$$

**Decrease.** The simulation ordering $p \leq q$ can reasonably be interpreted as a statement that $p$ is a more general or more abstract description of the behaviour of $q$. For example, $p$ may be a specification, expressing abstractly the desirable general properties of a system, and $q$ may be a more detailed description of the behaviour of a particular system implementation. The relationship $p \leq q$ then states that the implementation $q$ meets its specification $p$.

Suppose we want to use the target theory primarily for specifications. It would be useful if $Lp$ were always a valid specification for the source process $p$.

$$Lp \leq p.$$

This property has two equivalent algebraic formulations

$$\begin{aligned} &L^{\cup} \subseteq \leq \\ &\leq \ \subseteq L; \leq \quad \ldots\ldots (L\ \mathrm{dec}). \end{aligned}$$

**Counterexample 4.2.** Symbol change and restriction are not decreasing.

**Theorem 4.2.**

*L is decreasing if* $\xrightarrow{\langle a\rangle}; L \quad \subseteq L; \xrightarrow{\langle a\rangle}$, *for all a*

*(or equivalently, if* $L^{\cup}; \quad \xrightarrow{\langle a\rangle} \subseteq \xrightarrow{\langle a\rangle}; L^{\cup}$*).*

**Proof.**

$$\begin{array}{llll} & L^{\cup}; \xrightarrow{\langle a\rangle}; L; L^{\cup} & \subseteq L^{\cup}; L; \xrightarrow{\langle a\rangle}; L^{\cup} & \{\text{proviso}\} \\ \Rightarrow & L^{\cup}; \xrightarrow{\langle a\rangle} & \subseteq \langle a\rangle >; L^{\cup} & \{L\ \text{inj}\} \\ \Rightarrow & L^{\cup} & \subseteq \leq & \{\text{def of simulation}\}. \end{array}$$

If $L$ is decreasing, $Lp$ actually has more traces than $p$

$$\xrightarrow{\langle a\rangle}; U \subseteq L; \xrightarrow{\langle a\rangle}; U. \quad \square$$

**Idempotence.** The processes of our new calculus will be defined as just the image of the function $L$. We would like to ensure that over the image of $L$, $\leq_L$ has the same meaning as $\leq$. This is obviously desirable, simply as a unification of the theories in question. But it also has practical value. One or both of the operands of $\leq$ are often expressed in the notations of the target calculus, and these can be proved in advance to remain in the image of $L$. For such operands there is no need to apply the function $L$ before model checking. This desirable property is defined formally:

$$\begin{array}{ll} & p(L; \leq; L^{\cup})q \quad \Leftrightarrow p \leq q, \quad \text{for all } p, q \text{ in the image of } L. \\ \text{i.e.,} & L; L; \leq; L^{\cup}; L^{\cup} = L; \leq; L^{\cup} \end{array}$$

It is simpler to state a slightly stronger requirement, that $L$ is an idempotent function, in that applying it twice is equivalent to applying it just once.

$$\begin{aligned} &L(Lp) \quad \equiv Lp \\ &\text{i.e., } L; L; \leq \ = L; \leq \quad (L\ \text{idem}). \end{aligned}$$

**Example 4.3.** Symbol change $C$ and restriction $V$ are both idempotent.

If $L$ is already known to be decreasing, idempotence can be proved simply by

$$Lp \leq L(Lp).$$

Another desirable consequence of idempotence is that the image of $L$ is just the same as its fixed points; as a result, the processes of the new theory are just those processes of the source theory that satisfy the healthiness condition that

$$Lp \equiv p.$$

For a monotonic function $L$, idempotence has yet another desirable consequence: it maps each $p$ of the source theory to the strongest process in the target theory that approximates $p$.

$$\forall q \in \text{image } L.\ q \leq p \Rightarrow q \leq Lp.$$

Note that idempotence as defined here is not true idempotence, but only idempotence up to equivalence. Idempotence is therefore always relative to the chosen ordering relation. Most commonly in this paper the ordering will be simulation.

**Theorem 4.3A.**

$$L;\, L;\, \le\ \subseteq L;\, \le \quad \textit{if } L;\, fa \subseteq fa;\, L,$$

$$\textit{(or equivalently, if } \xrightarrow{\langle a\rangle};\, L^{\cup} \subseteq L^{\cup};\, \xrightarrow{\langle a\rangle}\textit{)}$$

$$\textit{for all } a \textit{ s.t. } L;\, \xrightarrow{\langle a\rangle} \neq \{\ \}.$$

**Proof.**

$$\begin{array}{lll} L^{\cup};\, L;\, L;\, \xrightarrow{\langle a\rangle} & \subseteq L^{\cup};\, L;\, fa;\, L & \{L \text{ commut}\} \\ & \subseteq L^{\cup};\, fa;\, L;\, L & \{\text{proviso of the theorem}\} \\ & \subseteq \xrightarrow{\langle a\rangle};\, L^{\cup};\, L;\, L & \{L \text{ commut}\}. \end{array}$$

So $L^{\cup};\, L;\, L \subseteq\ \le$ , whence $L;\, L \subseteq L;\, \le$ and $L;\, L;\, \le\ \subseteq L;\, \le$. □

**Theorem 4.3B.**

$$L;\, \le\ \subseteq L;\, L;\, \le \textit{ if } L^{\cup};\, fa \subseteq fa;\, L^{\cup}$$

$$\textit{for all } a \textit{ s.t. } L;\, \xrightarrow{\langle a\rangle} \neq \{\ \}.$$

**Proof.** Similar to previous theorem. □

**Retraction.** A retraction is defined as a function that is monotonic, decreasing and idempotent. In summary, a retraction $L$ satisfies the three elegant inequations

$$\begin{array}{lll} \le & \subseteq L;\, \le & \ldots\ldots\ (L \text{ dec}) \\ \le;\, L & \subseteq L;\, \le & \ldots\ldots.\ (L \text{ mon}) \\ L;\, L;\, \le & \subseteq L;\, \le & \ldots\ldots.\ (L \text{ idem}). \end{array}$$

**Efficiency.** To test the relation $\le_L$ by model checking it is generally necessary to pre-process both operands in advance, by application of the function $L$. It would obviously be more efficient to apply $L$ to only one of the operands, provided we could still be sure of getting the same result of the test. Preferably, it is the specification (on the left of $\le$) that should be processed, because specifications are in general simpler than their implementations. Application of $L$ to the right-hand side of $\le$ can often be avoided anyway, when implementations can be proved in advance to satisfy the healthiness condition $q \equiv Lq$. This is commonly done by showing that all the operators of the calculus preserve the health of their operands.

In order to validate this optimisation, it is obviously essential that the meaning of the optimised ordering should be the same as the original:

$$L;\, \le\ =\ \le_L\ .$$

Surprisingly, this equation holds if and only if $L$ is a retraction, as shown by the following theorems.

**Theorem 4.4.** *$L$ is a retraction $\Leftrightarrow (L;\, \le)$ is a pre-order.*

**Proof.** Assume that $(L;\, \le)$ is a pre-order. The three properties of a retraction are proved separately.

$$\begin{array}{lll} (L \text{ dec})\ \le & \subseteq L;\, \le;\, \le & \{L;\, \le\ \text{ reflexive}\} \\ & \subseteq L;\, \le & \{\le \text{ is transitive}\} \\ (L \text{ mon})\ \le;L & \subseteq L;\, \le;\, L & \{\text{just proved}\} \\ & \subseteq L;\, \le;\, L;\, \le & \{\le \text{ reflexive}\} \\ & \subseteq L;\, \le & \{L;\, \le\ \text{ transitive}\} \\ (L \text{ idem})\ L;\, L;\, \le & \subseteq L;\, \le;\, L;\, \le & \{\le \text{ reflexive}\} \\ & \subseteq L;\, \le & \{L;\, \le\ \text{ transitive}\}. \end{array}$$

Now assume that $L$ is a retraction.

$$\begin{array}{lll} (L;\, \le \text{refl})\ \text{Id} & \subseteq\ \le & \{\le \text{ reflexive}\} \\ & \subseteq L;\, \le & \{L \text{ dec}\} \end{array}$$

$$\begin{array}{lll} (L;\, \le \textit{trans})\ L;\, \le;\, L;\, \le & \subseteq L;\, L;\, \le;\, \le & \{L \text{ mon}\} \\ & \subseteq L;\, \le;\, \le & \{L \text{ idem}\} \\ & \subseteq L;\, \le & \{\le \text{ trans}\}. \quad \square \end{array}$$

**Theorem 4.5.** *$L$ is a retraction $\Leftrightarrow L;\, \le\ =\ L;\, \le;\, L^{\cup}$.*

**Proof.** ($\Leftarrow$): by the preceding theorem, because $L; \leq; L^{\cup}$ is a pre-order.

$$\begin{aligned}(\Rightarrow): L; \leq; L^{\cup} &\subseteq L; \leq; \leq && \{L \text{ dec}\}\\ &= L; \leq && \{\leq \text{ trans}\}\\ &\subseteq L; \leq; L; L^{\cup} && \{L \text{ inj}\}\\ &\subseteq L; L; \leq; L^{\cup} && \{L \text{ mon}\}\\ &\subseteq L; \leq; L^{\cup} && \{L \text{ idem}\}.\end{aligned}$$

Although retractions have many desirable properties, there is one important property that they lack. The composition of two retractions is not necessarily a retraction. It is certainly monotonic, and it is certainly decreasing; but in general it is not idempotent. In the remainder of this section, we address the issue of idempotence. Our treatment will apply to functions $L$ and $M$ that are idempotent and monotonic, but not necessarily decreasing. Firstly, we define a useful relationship between functions, that two applications of a function $M$ are equivalent to a single application, even if the two applications are separated by application of a function $L$. □

**Definition 4.1.**

$$L \text{ respects } M \cong M; L; M; \leq\, = M; L; \leq.$$

An alternative statement of this definition is

$$M(L(Mp)) \equiv L(Mp), \quad \text{for all } p.$$

This means that every object *L(Mp)* of the theory defined by $(M; L)$ is also an object of the theory defined by $M$; in particular, it satisfies the standard healthiness condition that it is a fixed point of $M$.

This property of respect is strong enough to prove idempotence of composition by a monotonic idempotent function $L$.

**Theorem 4.6.** *If $L$ respects $K$, then $(K; L)$ is idempotent. (Note that $L$ must be monotonic and idempotent, but $K$ does not have to be either).*

**Proof.**

$$\begin{aligned}K; L; K; L; \leq &= K; L; K; \leq; L; \leq && \{L \text{ monotonic}\}\\ &= K; L; \leq; L; \leq && \{L \text{ respects } K\}\\ &= K; L; L; \leq && \{L \text{ monotonic}\}\\ &= K; L; \leq && \{L \text{ idempotent}\}.\end{aligned}$$

In contrast to idempotence, the notion of respect does compose nicely. □

**Theorem 4.7.** *If both $K$ and $L$ respect $M$, and $L$ and $M$ are both monotonic, then $(K; L)$ respects $M$. (Note that $K$ does not have to be monotonic or idempotent).*

**Proof.**

$$\begin{aligned}M; K; L; M; \leq &= M; K; \leq; L; M; \leq && \{L; M \text{ monotonic}\}\\ &= M; K; M; \leq; L; M; \leq && \{K \text{ respects } M\}\\ &= M; K; M; L; M; \leq && \{L; M \text{ monotonic}\}\\ &= M; K; M; L; \leq && \{L \text{ respects } M\}\\ &= M; K; M; \leq; L; \leq && \{L \text{ monotonic}\}\\ &= M; K; \leq; L; \leq && \{K \text{ respects } M\}\\ &= M; K; L; \leq && \{L \text{ monotonic}\}. \quad \square\end{aligned}$$

Finally, the following lemma will be useful in proving respect.

**Lemma 4.1.** *If $L$ is decreasing*

$$\begin{aligned}&\textit{and} \quad K; L; K; L; \leq\, \subseteq K; L; \leq\, \subseteq K; L; K; \leq\\ &\textit{then} \quad K; L; K; L; \leq\, = K; L; \leq\, = K; L; K; \leq .\end{aligned}$$

**Proof.** $\leq\, \subseteq L; \leq$ completes the cycle of inclusions. □

## 5. Simulation and refinement

In this section, we will prove the basic retraction property of the traces model of CSP. In the first subsection we will deal with strong simulation, by assuming that the hidden symbol $\tau$ is either absent, or treated in the same way as all other symbols. In the second subsection we will define weak simulation, and show that it gives a retract of strong simulation. In the third subsection, we compose the two retractions to get a retraction to the standard trace model of CSP, in which $\tau$ is effectively removed from all the traces. The effect of removal is actually achieved by saturating the transition system with $\tau$-transitions.

### 5.1. Trace refinement and strong simulation

In this subsection we define the retraction $T$ which will be used repeatedly to link transaction systems to their trace models. Let $s$ be a trace of the process $p$. Suppose $p$ has already engaged in all the actions recorded in $s$. Then the possible future behaviour of $p$ is denoted by $\text{Д}_s p$, the $s$-derivative of $p$; it is pronounced '$p$ after $s$'. The process $\text{Д}_s p$ can perform an action $a$ just if the process $p$ can perform the sequence of actions $s\langle a\rangle$. After this action $a$ has happened, subsequent behaviour is of course described by $\text{Д}_{s\langle a\rangle} p$.

$$p \xrightarrow{s\langle a\rangle}\_ \;\Leftrightarrow \text{Д}_s p \xrightarrow{\langle a\rangle} \text{Д}_{s\langle a\rangle} p.$$

This equivalence is a consequence of a family of transition rules, parameterised by $s$

$$\frac{p \xrightarrow{s\langle a\rangle}\_}{\text{Д}_s p \xrightarrow{\langle a\rangle} \text{Д}_{s\langle a\rangle} p} \quad (\text{def } \text{Д}_s).$$

As usual, $\text{Д}_s p$ is taken to be the solution of these equivalences that has the minimal number of transitions. It has no transitions at all in the case that $s$ is not a trace of $p$. This case is of no interest to us, and we shall take care to avoid it.

Because the rule given above is the only way of deriving an $\xrightarrow{\langle a\rangle}$ transition for $\text{Д}_s p$

$$\text{Д}_s p \xrightarrow{\langle a\rangle} r \Leftrightarrow p \xrightarrow{s\langle a\rangle}\_ \;\&\, r = \text{Д}_{s\langle a\rangle} p \quad (\text{Д}_s \text{ det}).$$

By existentially quantifying $r$ on both sides, we get

$$\text{Д}_s p \xrightarrow{\langle a\rangle}\_ \;\Leftrightarrow p \xrightarrow{s\langle a\rangle}\_ \;.$$

In the next theorem, we generalise this property to longer traces than just $\langle a\rangle$. The corollary expresses the intuitive trace definition of the derivative that is given by CSP.

**Lemma 5.1.1.**

$$p \xrightarrow{st}\_ \;\Leftrightarrow p \xrightarrow{s}\_ \;\&\, \text{Д}_s p \xrightarrow{t}\_ \;.$$

**Proof.** By induction on $t$:

$$\begin{aligned} p \xrightarrow{s\varepsilon}\_ &\Leftrightarrow p \xrightarrow{s}\_ \\ &\Leftrightarrow p \xrightarrow{s}\_ \;\&\, \text{Д}_s p \xrightarrow{\varepsilon}\_ \quad \{\text{added clause is true}\}. \end{aligned}$$

The induction hypothesis is assumed true for all s , so it can be specialised by substituting $s\langle a\rangle$ for $s$:

$$\begin{aligned} & p \xrightarrow{s\langle a\rangle t}\_ \\ \Leftrightarrow\;& p \xrightarrow{s\langle a\rangle}\_ \;\&\, \text{Д}_{s\langle a\rangle} p \xrightarrow{t}\_ && \{\text{ind hyp}\} \\ \Leftrightarrow\;& p \xrightarrow{s\langle a\rangle}\_ \;\&\, \text{Д}_s p \xrightarrow{\langle a\rangle} \text{Д}_{s\langle a\rangle} p \,\&\, \text{Д}_{s\langle a\rangle} p \xrightarrow{t}\_ && \{\text{def } \text{Д}_s\} \\ \Leftrightarrow\;& p \xrightarrow{s}\_ \;\&\, \text{Д}_s p \xrightarrow{\langle a\rangle t}\_ && \{\text{def };\, \text{Д}_s \text{ det}\}. \quad \square \end{aligned}$$

**Corollary.**

$$p \xrightarrow{s}\_ \;\Rightarrow traces(\text{Д}_s p) = \{t \mid st \in traces(p)\}.$$

The function $\text{Д}_s$, like all operationally defined functions, is total. It delivers a result even if its argument is incapable of performing the actions of the trace $s$. We would prefer a partial function $H_s$, which is undefined in this meaningless case.

$$pH_s q \cong p \xrightarrow{\langle s\rangle}\_ \,\&\, q = \text{Д}_s p \quad (\text{def } H_s).$$

This definition simplifies the statement of Lemma 5.1.1.

**Theorem 5.1.1.**

$$\xrightarrow{st};\, U = H_s;\, \xrightarrow{t};\, U \quad (H_s\ intro).$$

**Theorem 5.1.2.**

$$\sqsubseteq;\, \xrightarrow{s} \subseteq H_s;\, \sqsubseteq \quad (H_s\ comm).$$

**Proof.**

$$\begin{aligned}
\sqsubseteq;\xrightarrow{s};\xrightarrow{t};U &= \xrightarrow{s};\xrightarrow{t};U && \{\text{Lemma 2.2}\}\\
&= H_s;\xrightarrow{t};U && \{\text{Theorem 5.1.1}\}\\
\sqsubseteq;\xrightarrow{s};\sqsubseteq &= H_s;\sqsubseteq && \{\text{Theorem 2.4 cor}\}.
\end{aligned}$$

The result follows from reflexivity of $\sqsubseteq$.

In future we will use only the single-step versions $H_a$ and $Д_a$ in place of the generality of $H_s$ and $Д_s$. When required, we will iterate the single-step version, using the definitions

$$\begin{aligned}
H^{\varepsilon} &= \mathrm{Id} \quad H^{\langle a\rangle s} = H_a; H^s\\
Д^{\varepsilon} &= \mathrm{Id} \quad Д^{\langle a\rangle s} = Д_a; Д^s.
\end{aligned}$$

Inductive proofs can now be given to demonstrate properties of the iterated forms that they share with the general forms. □

**Theorem 5.1.3.**

$$\begin{aligned}
H_s;\xrightarrow{t};U &= H^s;\xrightarrow{t};U\\
H^s;U &= \xrightarrow{s};U\\
pH^sq &\Rightarrow q = Д^sp\\
p\xrightarrow{st}\_ &\Rightarrow (Д^sp)\xrightarrow{t}\_\ .
\end{aligned}$$

Although $Д_a(p)$ describes the behaviour of $p$ after it has done $a$, there is no guarantee that $p$ itself can actually make an $a$-transition to $Д_a(p)$. In the general transition system of CCS, each $a$-transition of $p$ may simultaneously make an internal commitment preventing it from behaving henceforth with the full generality of $Д_a(p)$, as shown in the left diagram of Fig. 1. However, in all models of CSP, a guarantee of the existence of such an $a$-transition is given. This is illustrated in the right diagram of Fig. 1, which shows the corresponding CSP process in the trace model. To map CCS to CSP, we define a retraction $T$, which supplies the missing $a$-transitions when necessary. Henceforth we will use $H_a$ in place of $H_{\langle a\rangle}$

$$\frac{pH_aq}{Tp\xrightarrow{\langle a\rangle}Tq}$$

From these we get the standard commuting laws

$$\begin{aligned}
T;\xrightarrow{\langle a\rangle} &= H_a;T\\
\xrightarrow{\langle a\rangle};T^{\cup} &= T^{\cup};H_a \quad (T\ \text{commut}).
\end{aligned}$$

These laws generalise by induction to longer traces.

$$\begin{aligned}
T;\xrightarrow{s} &= H^s;T\\
\xrightarrow{s};T^{\cup} &= T^{\cup};H^s \quad (T\ \text{commut}).
\end{aligned}$$

The following theorem states that the traces of $p$ and $Tp$ are the same.

**Theorem 5.1.4.**

$$T;\xrightarrow{s};U = \xrightarrow{s};U.$$

**Proof.**

$$\begin{aligned}
T;\xrightarrow{s};U &= H^s;T;U && \{T\ \mathit{commut}\}\\
&= H^s;U && \{T\ \text{total}\}\\
&= \xrightarrow{s};U && \{\text{Theorem 5.1.3}\}. \quad \square
\end{aligned}$$

**Corollaries.**

$$\begin{array}{ccclc}
 & \mathit{traces}(p) = \mathit{traces}(Tp) & & \{\text{def } \mathit{traces}\}\\
T & \subseteq & \sqsubseteq & \{\text{def } \sqsubseteq\}\\
T;\sqsubseteq & = & \sqsubseteq & \{\mathit{Theorem}\ 2.4.\text{cor}\}\\
T;\leq & \subseteq & \sqsubseteq & \{\mathit{because}\ \leq\subseteq\sqsubseteq\}
\end{array}$$

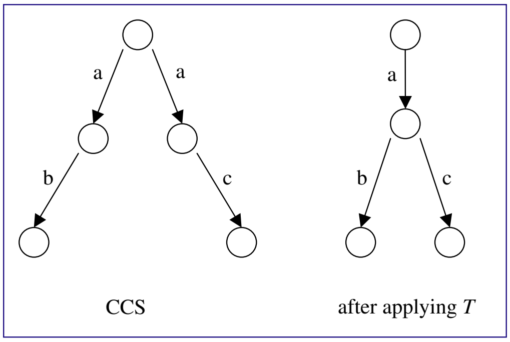

**Fig. 1.**

**Theorem 5.1.5.**

$$T^{\cup} \subseteq \sqsubseteq .$$

**Proof.**

$$\begin{aligned} T^{\cup}; \xrightarrow{s}; U &= T^{\cup}; T; \xrightarrow{s}; U && \{\text{Theorem 5.1.4}\} \\ &\subseteq \xrightarrow{s}; U && \{T \text{ inj}\}. \quad \square \end{aligned}$$

**Corollary.**

$$\sqsubseteq; T^{\cup} \subseteq \sqsubseteq \quad \{\sqsubseteq \text{ trans}\}.$$

**Theorem 5.1.6.**

$$T^{\cup}; \sqsubseteq \subseteq \leq .$$

**Proof.**

$$\begin{aligned} T^{\cup}; \sqsubseteq; \xrightarrow{\langle a\rangle} &\subseteq T^{\cup}; H_a; \sqsubseteq && \{H_a \text{ comm}\} \\ &= \xrightarrow{\langle a\rangle}; T^{\cup}; \sqsubseteq && \{T \text{ commut}\}. \quad \square \end{aligned}$$

**Corollary.**

$$\sqsubseteq \subseteq T; \leq .$$

**Theorem 5.1.7.**

$$T; \leq = \sqsubseteq \quad \{\textit{corollaries to } 5.1.4 \textit{ and } 5.1.6\}.$$

**Corollary.** *$T$ is a retraction {Theorem 4.4, since $\sqsubseteq$ is a pre-order}.*

Note that this proof of the retraction property has not used Theorems 4.1, 4.2, 4.3A and 4.3B, which are not applicable to the transition rule that defines $T$.

It follows that the trace model of CSP is a retract of CCS modulo strong simulation. This is hardly a surprising result: the function $T$ in effect determinises a process in the same way as the standard powerset construction. Determinisation is obviously idempotent, and nearly obviously it is decreasing in simulation ordering. The only novelty of our construction is that it uses simple standard transition rules to define the determinising function. This makes the proofs simpler in the relational calculus.

### *5.2. Weak simulation*

The special symbol $\tau$ is intended to stand for an event that is internal to a process; its occurrence or non-occurrence is not observable from outside. So any number of occurrences of $\tau$ are indistinguishable from any other number — even none. These intended properties of the hidden event are encoded in the following definition of a weak transition in CCS (slightly different from the familiar definition):

$$\begin{aligned} &=\tau\Rightarrow \;\widehat{=}\; (\xrightarrow{\langle\tau\rangle})* \\ &=a\Rightarrow \;\widehat{=}\; (\xrightarrow{\langle\tau\rangle})*; \xrightarrow{\langle a\rangle}; (\xrightarrow{\langle\tau\rangle})*, \quad \text{if } a \neq \tau . \end{aligned}$$

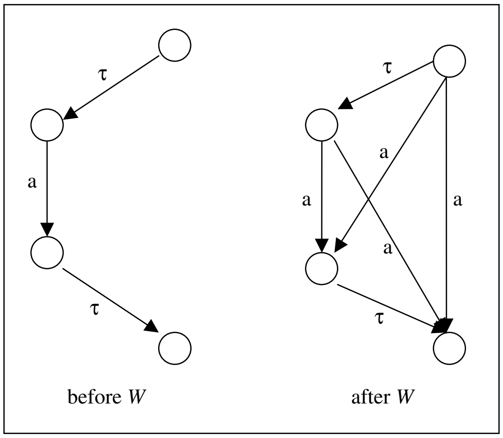

**Fig. 2.**

The definition extends readily to strings. We define a retraction $W$ which turns weak transitions of the source theory into ordinary strong transitions of the target theory. It thereby allows an implementation to optimise responsiveness by proceeding more directly to the next visible action, omitting any or all of the hidden actions which precede it or follow it.

$$\begin{array}{c} p =a\Rightarrow q \\ \hline Wp \xrightarrow{\langle a\rangle} Wq \end{array}$$

In Fig. 2, the right diagram shows a fragment of the target theory on the right; it is derived from the fragment of the source theory on the left. The right diagram is also supposed to contain a $\tau$-loop on every node.

**Theorem 5.2.1.** *$W$ is a retraction.*

**Proof.** By Theorems 4.1, 4.2, 4.3A and 4.3B, it suffices to prove the following lemmas

$$\begin{array}{llll}
\text{(mon)} \le; =a\Rightarrow & \subseteq & =a\Rightarrow; \le & \{\text{induction on def simulation}\} \\
\text{(dec)} \xrightarrow{\langle a\rangle}; W & \subseteq & =a\Rightarrow; W & \{\textit{because} \xrightarrow{\langle a\rangle} \subseteq =a\Rightarrow\} \\
 & = & W; \xrightarrow{\langle a\rangle} & \{W \text{ commut}\} \\
\text{(idem)} W; =a\Rightarrow & \subseteq & (\xrightarrow{\langle\tau\rangle})*; =a\Rightarrow; (\xrightarrow{\langle\tau\rangle})*; W & \\
 & & & \{\text{induction on W commut}\} \\
 & = & =a\Rightarrow; W & \\
\end{array}$$
$$\{\text{because } =a\Rightarrow = (\xrightarrow{\langle\tau\rangle})*; =a\Rightarrow; (\xrightarrow{\langle\tau\rangle})*\}.$$

By Theorem 3.1, $(W; \le)$ is the weakest relation satisfying the inequation

$$(W; \le); \xrightarrow{\langle a\rangle} \subseteq = a \Rightarrow; (W; \le) \quad (W; \le \text{ commut}).$$

This inequation is essentially the standard co-inductive definition of weak simulation (expressed in a single clause, rather than separating the case that $a = \tau$). We therefore claim that weak simulation is the same as $\le_W$, which is of course the same as $(W; \le)$. We conclude that CCS modulo weak simulation is a retract (by $W$) of CCS modulo strong simulation. □

### 5.3. *Weak trace refinement*

Our next claim will be that $\sqsubseteq_W$ (weak trace refinement) expresses exactly the CSP notion of trace inclusion, where all occurrences of the hidden symbol $\tau$ are removed from the traces on both sides of the inequality. We define by induction a minus operator which effects this removal, and lift the definition to sets in two ways

$$\begin{array}{ll}
(\langle\rangle)- & = \langle\rangle \\
(\langle\tau\rangle t)- & = t- \\
(\langle a\rangle t)- & = \langle a\rangle(t-) \\
S- & \cong \{s - \mid s \in S\} \\
S+ & \cong \{s \mid s - \in S-\}.
\end{array}$$

Note that $+$ and $-$ are monotonic with respect to set inclusion, $+$ is increasing, and

$$\begin{aligned}&S{+}{+} = (S-)+ = S+\\&S{-}{-} = (S+)- = S-.\end{aligned}$$

The function $traces(p)-$ gives the normal CSP traces of a process $p$, with all occurrences of $\tau$ removed. The function $traces(p)+$ inserts $\tau$ arbitrarily often into these reduced traces. It does not matter which of these functions is used to model CSP refinement, because of the simple property that

**Lemma 5.3.1.**

$$S- \subseteq T- \Leftrightarrow S+ \subseteq T+.$$

**Proof.**

$$\begin{array}{lll} LHS & \Rightarrow S{-}{+} \subseteq T{-}{+} & \{+mon\}\\ & \Leftrightarrow RHS & \{S{-}{+} = S+\}\\ & \Rightarrow S{+}{-} \subseteq T+ & -\{-mon\}\\ & \Leftrightarrow LHS. \quad \square & \end{array}$$

The following lemma shows that the traces of $Wp$ are closed with respect to addition or removal of occurrences of $\tau$.

**Lemma 5.3.2.**

$$W; \xrightarrow{s}; U = W; \overset{s}{\Longrightarrow}; U.$$

**Proof.** First we show that

$$W; \xrightarrow{s} = \overset{s}{\Longrightarrow}; W. \qquad (*)$$

Let $\lambda$ stand for a symbol or $\tau$

$$\begin{array}{ll} \quad W; \xrightarrow{\langle\lambda\rangle s} & \\ = \overset{\langle\lambda\rangle}{\Longrightarrow}; W; \xrightarrow{s} & \{W \text{ commut}\}\\ = \overset{\langle\lambda\rangle s}{\Longrightarrow}; W & \{\text{induction}\}\\ \quad W; \overset{s}{\Longrightarrow}; U & \\ = W; \overset{s}{\Longrightarrow}; W; U & \{W \text{ is total}\}\\ = W^2; \xrightarrow{s}; U & \{(*)\}\\ = W; \xrightarrow{s}; U & \{W \text{ idemp}\}. \quad \square \end{array}$$

**Corollary.**

$$traces(Wq) = traces(Wq)+.$$

**Theorem 5.3.1.**

$$traces(Wq)+ = traces(q)+.$$

**Proof.**

$$\begin{array}{ll} \quad s \in traces(Wq)+ & \\ \Leftrightarrow Wq \xrightarrow{s}\_ & \{\text{Corollary of Lemma 5.3.2}\}\\ \Leftrightarrow q \overset{s}{\Longrightarrow}\_ & \{W \text{ commut}\}\\ \Leftrightarrow q \overset{s-}{\Longrightarrow}\_ & \{\overset{\langle\tau\rangle}{\Longrightarrow}; \overset{\langle\lambda\rangle}{\Longrightarrow}; \overset{\langle\tau\rangle}{\Longrightarrow} = \overset{\langle\lambda\rangle}{\Longrightarrow}\}\\ \Leftrightarrow s- \in traces(q)- & \{\text{def of } -\}\\ \Leftrightarrow s \in traces(q)+ & \{\text{def of } S+\}. \quad \square \end{array}$$

**Theorem 5.3.2.**

$$p \sqsubseteq_w q \Leftrightarrow traces(q)- \subseteq traces(p)-.$$

**Proof.** Lemma 5.3.1 Corollary and Theorem 5.3.1. $\square$

This confirms that $\sqsubseteq_W$ corresponds to the familiar CSP concept of weak ($\tau$-hidden) trace refinement.

We explore next the relationship between $W$ and $T$.

**Theorem 5.3.3.** *$T$ respects $W$.*

**Proof.** Since $T$ and $W$ are decreasing, let us prove condition (2) of Lemma 4.1.

$$\begin{array}{lll}
W;\, T;\, W;\, \xrightarrow{s};\, U & \subseteq W;\, T;\, = s \Rightarrow;\, U & \{W \text{ commut}\} \\
 & \subseteq W;\, = s \Rightarrow;\, U & \{\text{Theorem 5.1.4}\} \\
 & = W;\, \xrightarrow{s};\, U & \{\text{Lemma 5.2.2}\} \\
W;\, T;\, W; \sqsubseteq & \subseteq W;\, \sqsubseteq & \{\text{Theorem 2.4 cor}\} \\
 & = W;\, T;\, \leq & \{\text{Theorem 5.1.7}\} \\
W;\, T;\, W;\, \leq & \subseteq W;\, T;\, \leq & \{\leq\, \subseteq\, \sqsubseteq\}. \quad \square
\end{array}$$

**Corollary.** *$W; T$ is a retraction {Theorem 4.6}.*

**Theorem 5.3.4.**

$$\sqsubseteq_W = W;\, T;\, \leq\; = W;\, T;\, \leq_W.$$

This theorem shows that CSP modulo weak trace refinement is a retract (by $W; T$) of CCS modulo weak simulation as well as CCS modulo strong simulation.

## 6. Barbs

A barb is an event which represents an observation of the current state of a process. It is effectively inserted into a transition system by a function $B$ at all places where the state has the property that is intended to be observed. It records the result of the kind of test defined in [1,7]. Barbed simulation and barbed refinement are then defined in the usual way as $B;\, \leq;\, B^{\cup}$ and $B;\, \sqsubseteq;\, B^{\cup}$ respectively. In general, $B$ is not a retraction; this is because the purpose of the barb is usually to define a form of simulation which is stronger than strong simulation, so $B$ cannot be monotonic. However, it will always be idempotent.

In a theory used for program proofs, the most important properties (to prove the absence of) are those that indicate that the program has gone wrong. And two of the most prevalent risks of concurrent programming are deadlock and livelock. Deadlock occurs when two processes are waiting for each other, but refuse to participate in communications offered by the other. Livelock occurs when a process can engage in an infinite series of hidden events, and never has to wait for a communication to be offered by its environment. These two barbs have proved to be widely useful in practice. A barb recently discovered to be useful is the *revival* of [21,22].

Simulation cannot deal properly with deadlock. Indeed, a process that never engages in any event whatsoever displays immediate deadlock; but simulation actually proves that it meets every specification whatsoever. Weak simulation cannot deal properly with livelock, because all hidden events are effectively concealed, including infinite sequences of them. The FDR model of CSP deals with both these problems, by making their observation explicit in the form of refusal barbs and divergence barbs, which are effectively added at the end of each trace that leads to them. These barbs will be the topic of this section.

### 6.1. Refusals

Many of the events modelled in a process algebra are communication events. Their successful completion depends on participation not only by the process itself, but also by some other process in its environment. If either participant refuses to engage in the events offered by the other, nothing further can happen. This is the notorious phenomenon of deadlock. This is the primary cause of unexpected deadlock.

Let $X$ be a set of communication events. Then ‘ref($X$)’ is an observation (barb) meaning that the process is deadlocked, even though its environment is prepared to synchronise with any the communications in $X$, and even allows the selection to be made by the process itself. This concept of a refusal is introduced by a function $R$, defined by the transition rules

$$\frac{p\ \mathrm{ref}_X q}{Rp \xrightarrow{\mathrm{ref}(X)} Rq} \qquad\qquad \frac{p \xrightarrow{a} q}{Rp \xrightarrow{a} Rq}$$

where $p\ \mathrm{ref}_X q \cong \forall x \in X \cup \{\tau\}. \neg p \xrightarrow{x} \_$.

Note that this definition does not depend on $q$. As a result,

$$\mathrm{ref}_X = \mathrm{ref}_X;\, U.$$

Furthermore, the second clause in the definition of $R$ guarantees by the axiom that $R;\xrightarrow{s}$ is not empty, so $U;\ R;\xrightarrow{s};\ U=U$.

One of the standard healthiness conditions of CSP is that every event that cannot happen can be added to the refusal set. This follows from the definition of $\mathrm{ref}_X$, because

$$\mathrm{ref}_X \subseteq \xrightarrow{x};\ U \cup \mathrm{ref}_{X\cup\{x\}}.$$

Furthermore, every subset of a refusal set can also be refused. These two healthiness conditions can be coded algebraically

$$\begin{aligned} R;\ \xrightarrow{\mathrm{ref}(X\cup Y)} &\subseteq R;\ \xrightarrow{\mathrm{ref}(X)} \\ R;\ \xrightarrow{\mathrm{ref}(X)} &\subseteq R;\ (\xrightarrow{x};\ U \cup \xrightarrow{\mathrm{ref}(X\cup\{x\})}). \end{aligned}$$

Of course, the validity of these conditions depends on the assumption that they are satisfied by any *a priori* refusals in the transition system, namely ones that possibly existed before application of $R$. To avoid such non-standard refusals, a process algebra usually forbids the explicit occurrence of $\mathrm{ref}(X)$ (and other barbs) in its process descriptions. Since we want to apply $R$ twice, we will not do that. Instead, we will insist on the healthiness condition that after observation of a barb, the behaviour of the process is arbitrary (other conventions are possible).

$$\xrightarrow{\mathrm{ref}(X)} = \xrightarrow{\mathrm{ref}(X)};\ U \quad (\mathrm{refbarb}).$$

We define refusal-barbed simulation in the usual way

$$\leq_R \,\cong (R;\ \leq;\ R^{\cup}).$$

By Theorem 3.2, $\leq_r$ is the weakest relation $S$ satisfying

$$\begin{aligned} S;\ \xrightarrow{a}\ &\subseteq\ \xrightarrow{a};\ S \quad \text{if } a \text{ is not a refusal} \\ S;\ \mathrm{ref}_X\ &\subseteq\ \mathrm{ref}_X. \end{aligned}$$

The second clause can be expanded

$$pSq \Rightarrow \forall X.[(\forall x \in X \cup \{\tau\}.\neg q \xrightarrow{x}) \Rightarrow (\forall x \in X \cup \{\tau\}.\neg p \xrightarrow{x})].$$

By simple contraposition of implication and set-theoretic simplification, this is equivalent to

$$pSq \Rightarrow (\forall x.p \xrightarrow{x} \ \Rightarrow q \xrightarrow{x} \vee q \xrightarrow{\tau}).$$

This is quite close to satisfying the defining property of two-thirds simulation [14]. The only difference is the involvement of $\tau$, which is needed to ensure that (as in CSP) the deadlock indicated by a refusal test can occur only in stable states. We will therefore give as $\leq_R$ the name of two-thirds simulation. The same ideas appear under the name 'ready simulation' in [2].

**Lemma 6.1.1.**

$$\begin{aligned} R;\ \xrightarrow{a}\ &=\ \xrightarrow{a};\ R \quad \textit{if } a \textit{ is not a refusal} \\ R;\ \xrightarrow{\mathrm{ref}(X)}\ &=\ (\xrightarrow{\mathrm{ref}(X)} \cup\ \mathrm{ref}_X);\ R \\ R;\ \mathrm{ref}_X\ &=\ \mathrm{ref}_X;\ R. \end{aligned}$$

Although $R$ is not a retraction, it is an idempotent function, and a decreasing one.

**Theorem 6.1.1.**

$$\begin{aligned} \leq\ &\subseteq R;\ \leq \\ R;\ R;\ \leq\ &= R;\ \leq\ . \end{aligned}$$

**Proof.** From the preceding lemma by Theorems 4.2, 4.3A and 4.3B. □

**Corollary.**

*$R$ is a retraction with respect to $\leq_r$.* {*because* $R;\ \leq_r = \leq_r$}

$R;\ R;\ \sqsubseteq\ = R;\ \sqsubseteq$.

We turn now to refusal-barbed refinement, which is also defined in the standard way

$$\sqsubseteq_R \,\cong R; \sqsubseteq; R^{\cup}.$$

This relation can be fairly efficiently computed using model checking, using the formula $R; T; \leq; R^{\cup}$. Unfortunately, the final $R^{\cup}$ cannot be omitted, because $R$ is not monotonic and cannot be a retraction.

Having defined $\sqsubseteq_r$ in a standard way, we are obliged to show that it is essentially the same as the standard CSP notion of failures refinement. In CSP, refusal barbs are restricted to appear only at the very end of a trace, whereas in a trace of *Rp* they can be followed by any trace whatsoever. Let us define a minus operator that removes all of a trace after the first refusal; and then extend it to sets in two ways

$$\begin{array}{lll} (\langle\rangle)- & = \langle\rangle & \\ (\langle \mathrm{ref}(X)\rangle t)- & = \langle \mathrm{ref}(X)\rangle & \\ (\langle a\rangle t)- & = \langle a\rangle (t-) & \text{if } a \text{ is not a refusal} \\ S- & \cong \{s- \mid s \in S\} & \\ S+ & \cong \{s \mid s- \in S-\}. & \end{array}$$

Note that $+$ and $-$ are monotonic with respect to set inclusion, $+$ is increasing, and

$$S{+}{+} = (S-)+ = S+$$
$$S{-}{-} = (S+)- = S-.$$

It should now be fairly obvious that we can define

$$\mathit{failures}(p) \cong \mathit{traces}(Rp)-.$$

Note that this definition of failures differs from the earlier CSP definitions, in that it includes also traces which do not end in a refusal.

**Lemma 6.1.2.**

$$\mathit{traces}(Rp) - + = \mathit{traces}(Rp).$$

**Proof.** We consider two cases as follows:
Case 1: $\mathrm{ref}(X) \notin s$

$$\begin{array}{ll} & s \in LHS \\ \equiv & s = s- \in \mathit{traces}(Rp)- \quad \{\mathrm{ref}(X) \notin s \text{ and def of } s-\} \\ \equiv & s \in RHS \quad \{\text{def of } S-\}. \end{array}$$

Case 2: $s = s_1\langle \mathrm{ref}(X)\rangle s_2$ and $\mathrm{ref}(X) \notin s_1$

$$\begin{array}{lll} & s \in LHS & \\ \equiv & s_1\langle \mathrm{ref}(X)\rangle \in \mathit{traces}(Rp)- & \{\text{def of } -\} \\ \equiv & \exists t \bullet s_1\langle \mathrm{ref}(X)\rangle t \in \mathit{traces}(Rp) & \{\text{def of } -\} \\ \equiv & s \in RHS & \{\text{Property (ref barb)}\}. \quad \square \end{array}$$

**Lemma 6.1.3.**

$$\mathit{traces}(Rq) \subseteq \mathit{traces}(Rp) \Rightarrow \mathit{failures}(q) \subseteq \mathit{failures}(p).$$

**Theorem 6.1.2.**

$$p \sqsubseteq_R q \Leftrightarrow \mathit{failures}(q) \subseteq \mathit{failures}(p).$$

**Proof.**

$$\begin{array}{lll} & LHS & \\ \equiv & \mathit{traces}(Rq) \subseteq \mathit{traces}(Rp) & \{\text{def of } \sqsubseteq_r\} \\ \Rightarrow & RHS & \{\text{Lemma 6.1.3}\} \\ \equiv & \mathit{traces}(Rq)- \subseteq \mathit{traces}(Rp)- & \{\text{def of } \mathit{failures}\} \\ \Rightarrow & LHS & \{\text{Lemma 6.1.2}\}. \quad \square \end{array}$$

This concludes our demonstration that we have captured the failures refinement concept of CSP. We still have to make good the claim that the simple failures model of CSP is a retract of CCS modulo two-thirds simulation. The relevant retraction will be $(R; T)$. That will be the task of the rest of this subsection.

**Lemma 6.1.4.**

$$T;\xrightarrow{s} = ((\xrightarrow{s};U)\cap\Delta_s);T.$$

**Proof.** Omitted. □

**Lemma 6.1.5.**

$$T;\mathrm{ref}_X\subseteq\mathrm{ref}_X.$$

**Proof.**

$$\begin{aligned}
& LHS \\
={} & LHS;U && \{\mathrm{ref}_X=\mathrm{ref}_X;U\} \\
={} & \neg\cup_{a\in X\cup\{\tau\}}T;\xrightarrow{a};U && \{\text{def of }\mathrm{ref}_X\} \\
\subseteq{} & \neg\cup_{a\in X\cup\{\tau\}}\xrightarrow{a};U && \{\xrightarrow{a};U=T;\xrightarrow{a};U\} \\
={} & RHS && \{\text{def of }\mathrm{ref}_X\text{ and }\mathrm{ref}_X=\mathrm{ref}_X;U\}. \quad \square
\end{aligned}$$

**Lemma 6.1.6.**

$$T;\xrightarrow{s};\mathrm{ref}_X\subseteq\xrightarrow{s};\mathrm{ref}_X.$$

**Proof.**

$$\begin{aligned}
& LHS \\
={} & LHS;U && \{\mathrm{ref}_X;U=\mathrm{ref}_X\} \\
={} & T;\xrightarrow{s};U\cap\Delta_s;T;\mathrm{ref}_X && \{\text{Lemma 6.1.4}\} \\
\subseteq{} & \xrightarrow{s};U\cap\neg\cup_{a\in X\cup\{\tau\}}\Delta_s;\xrightarrow{a};U && \{\text{Lemma 6.1.5}\} \\
\subseteq{} & \xrightarrow{s};U\cap\neg\cup_{a\in X\cup\{\tau\}}\xrightarrow{s\langle a\rangle};U && \{\Delta_s\text{ det}\} \\
\subseteq{} & \xrightarrow{s};\neg\cup_{a\in X\cup\{\tau\}}\xrightarrow{a};U && \{\xrightarrow{s\langle a\rangle}=\xrightarrow{s};\xrightarrow{\langle a\rangle}\} \\
\subseteq{} & RHS && \{\text{def of }\mathrm{ref}_X\}. \quad \square
\end{aligned}$$

**Lemma 6.1.7.**

$$T;R;\xrightarrow{s};U\subseteq R;\xrightarrow{s};U.$$

**Proof.** Assume that $\mathrm{ref}(X)\notin s$

$$\begin{aligned}
& T;R;\xrightarrow{s};U \\
={} & T;\xrightarrow{s};R;U && \{\text{Lemma 6.1.1}\} \\
={} & \xrightarrow{s};U && \{\text{Theorem 5.1.4}\} \\
={} & RHS && \{\text{Lemma 6.1.1}\} \\
& T;R;\xrightarrow{s\langle\mathrm{ref}(X)\rangle};U \\
={} & T;\xrightarrow{s};(\mathrm{ref}_X\cup\xrightarrow{\mathrm{ref}(X)});U && \{\text{Lemma 6.1.1}\} \\
\subseteq{} & (\xrightarrow{s};\mathrm{ref}_X;U)\cup T;\xrightarrow{s};\xrightarrow{\mathrm{ref}(X)};U && \{\text{Lemma 6.1.6}\} \\
={} & \xrightarrow{s};(\mathrm{ref}_X\cup\xrightarrow{\mathrm{ref}(X)});U && \{\text{Theorem 5.1.4}\} \\
={} & RHS && \{\text{Lemma 6.1.1}\}. \quad \square
\end{aligned}$$

For an arbitrary trace $s$, the conclusion follows from the fact $R;\xrightarrow{s};U=R;\xrightarrow{s-};U$.

**Theorem 6.1.3.** (1) $T;R;\sqsubseteq\,\subseteq R;\sqsubseteq$
(2) $R;T;R;\sqsubseteq\,\subseteq R;\sqsubseteq$
(3) *$T$ respects $R$*
(4) *$(R;T)$ is idempotent.*

**Proof.** The conclusion (1) follows from Lemma 6.1.7. (2) comes from (1) and Theorem 6.1.1.
(3)

$$\begin{aligned}
& R;T;R;\le \\
\subseteq{} & R;T;R;\sqsubseteq && \{\le\,\subseteq\,\sqsubseteq\} \\
\subseteq{} & R;\sqsubseteq && \{\text{Theorem 6.1.3(2)}\} \\
={} & R;T;\le && \{\text{Theorem 5.1.7}\}
\end{aligned}$$

(4) follows from (3) and Theorem 4.6. □

Finally, we can put these results together to prove that barbed refinement with refusal barbs is a retract of two-thirds simulation by the retraction $(R; T)$.

**Theorem 6.1.4.**

$$\sqsubseteq_R \; = \; R; T; \leq_R .$$

**Proof.**

$$\begin{array}{ll}
\quad LHS & \\
\subseteq RHS & \{\text{Theorem 5.1.7 and } R \text{ is decreasing}\} \\
\subseteq R; T; R; T; \leq; R^{\cup} & \{T \text{ is decreasing}\} \\
= R; T; \leq; R^{\cup} & \{\text{Theorem 6.1.3(4)}\} \\
= LHS & \{\text{Theorem 5.1.7}\}. \quad \square
\end{array}$$

### 6.2. Failures with hiding

In CSP, the standard model combines refusal barbs with the concealment of hidden events. The relevant refinement ordering is defined by composing $W$ with $R$:

$$\sqsubseteq_{RW} \; = R; W; \sqsubseteq; W^{\cup}; R^{\cup}.$$

Two-thirds simulation also has a weak form, which is similarly defined

$$\leq_{RW} \; = R; W; \leq; W^{\cup}; R^{\cup} = R; W; \leq; R^{\cup}.$$

We want to prove that the refinement is a retract of the simulation, as defined above.

One of the effects of $W$ is to insert a $\tau$ transition from every process to itself. Any subsequent application of $R$ will discover that all states are unstable; so it will never insert a refusal. As a result,

**Lemma 6.2.1.**

$$W; \mathrm{ref}_X = \{\,\}.$$

This lemma is the crux of the proof that applying $R$ after $W$ has no effect.

**Lemma 6.2.2.**

$$W; R; \leq \; = W; \leq .$$

**Proof.** If $a$ is not a refusal

$$\begin{array}{rll}
W^{\cup};\; W; R; \xrightarrow{a} & = W^{\cup};\; W; \xrightarrow{a}; R & \{\text{Lemma 6.1.1}\} \\
& = W^{\cup};\; =a\Rightarrow; W; R & \{W \text{ commut}\} \\
& = \xrightarrow{a}; W^{\cup};\; W; R & \{W \text{ commut}\} \\
W^{\cup};\; W; R; \xrightarrow{\mathrm{ref}(X)} & = W^{\cup};\; W; (\xrightarrow{\mathrm{ref}(X)} \cup \mathrm{ref}_X); R & \{R \text{ commut}\} \\
& = W^{\cup};\; = \mathrm{ref}(X) \Rightarrow; W; R & \{W; \mathrm{ref}_X = \{\,\}\} \\
& = \xrightarrow{\mathrm{ref}(X)}; W^{\cup};\; W; R & \{W \text{ commut}\}. \quad \square
\end{array}$$

The result follows from the definition of weakest simulation and $W$ inj.

**Theorem 6.2.1.** *$W$ respects $R$.*

**Corollaries.**

*$R$; $W$ is idempotent.* {*Theorem* 4.6}
*$R$; $W$ is a retraction with respect to* $\leq_{RW}$ .

**Lemma 6.2.3.** *$(W; T)$ respects $R$.*

**Proof.** From Theorem 6.1.3, 6.2.1 and Theorem 4.7. □

**Corollary.** *$(R; W; T)$ is idempotent .*

**Proof.** From Theorem 4.6 and Lemma 6.2.3. □

In conclusion, the following corollary states that the stable failures model of CSP is a retract of CCS modulo a natural definition of weak two-thirds simulation. The relevant retraction is $(R; W; T)$.

**Theorem 6.2.2.**

$$\sqsubseteq_{RW} = R; W; T; \leq_{RW} \ .$$

**Proof.**

$$\begin{aligned}
& LHS \\
= \ & R; W; T; \leq; (R; W)^{\cup} && \{\text{Theorem 5.1.7}\} \\
\subseteq \ & RHS && \{R \text{ and } W \text{ are decreasing}\} \\
\subseteq \ & R; W; T; R; W; T; \leq; (R; W)^{\cup} && \{T \text{ is decreasing}\} \\
= \ & R; W; T; \leq; (R; W)^{\cup} && \{\text{Corollary of Lemma 6.2.3}\} \\
= \ & LHS && \{\text{Theorem 5.1.7}\}. \quad \square
\end{aligned}$$

*6.3. Divergence*

Livelock, also known as divergence, is a phenomenon that occurs when a process is engaged in an unbounded series of internal events, each of which consumes some resource. It typically arises from an unguarded recursion in a program, for example an iteration with a permanently false exit condition. It can also arise from an unbounded consecutive series of hidden internal communications (infinite chatter). To distinguish the danger of unbounded resource consumption from merely ignoring the totally ignorable event $\tau$, we use a new special symbol $\sigma$ to denote a hidden event that consumes resources.

Because livelock cannot be observed from any finite trace of the actions of a process, it is called a 'liveness' property, rather than a safety property. Nevertheless, to simplify proof of the absence of livelock, we will turn it into a safety property. This is done by introducing a special barb symbol $\delta$ that stands for an infinite sequence of occurrences of $\sigma$. The barb is introduced into a transition system by a function $S$, defined as follows

$$\frac{p\,\sigma^{\infty}\,q}{Sp \xrightarrow{\langle\delta\rangle} Sq} \qquad\qquad \frac{p \xrightarrow{\langle a\rangle} q}{Sp \xrightarrow{\langle a\rangle} Sq}$$

where $\sigma^{\infty}$ is defined as the greatest binary relation $r$ satisfying the equation

$$r \subseteq \xrightarrow{\langle\sigma\rangle};\ r.$$

**Theorem 6.3.1.** *$S$ is a retraction.*

**Proof.** We are going to prove that $S$ is idempotent by showing

$$\begin{aligned}
& (Id \cup S^{\cup}; S^2) \subseteq \leq \\
& (Id \cup S^{\cup}; S^2); \xrightarrow{\delta} \\
\subseteq \ & \xrightarrow{\delta} \cup S^{\cup}; \sigma^{\infty}; S \cup \\
& S^{\cup}; (\xrightarrow{\delta} \cup \sigma^{\infty}); S^2 && \{S \text{ commut and } S; \sigma^{\infty} \subseteq \sigma^{\infty}\} \\
\subseteq \ & \xrightarrow{\delta} \cup S^{\cup}; (\xrightarrow{\delta} \cup \sigma^{\infty}); S \cup \\
& S^{\cup}; (\xrightarrow{\delta} \cup \sigma^{\infty}); S^2 && \{\text{set theory}\} \\
\subseteq \ & \xrightarrow{\delta}; (Id \cup S^{\cup}; S \cup S^{\cup}; S^2) && \{S \text{ commut}\} \\
\subseteq \ & \xrightarrow{\delta}; (Id \cup S^{\cup}; S^2) && \{S^{\cup}; S \subseteq Id\}. \quad \square
\end{aligned}$$

Divergence is a programming error like a subscript overflow: it is never a desirable behaviour for an implementation, and any process that can start with divergence at the very beginning cannot reliably satisfy any reasonable specification. That is why CSP places a divergent process at the bottom of the refinement ordering. From the standpoint of correctness reasoning, it is totally irrelevant what happens after divergence.

The divergence policy of CSP is enforced by a healthiness condition stating that any process that can diverge at a given point can also do anything else whatsoever at that same point. Of course, this condition makes CSP entirely unsuitable to help in debugging a program that suffers from a divergence error. For debugging purposes, a purely operational language definition is more suitable. That is exactly why we want to establish a link between an operationally defined theory like CCS and a denotationally presented theory like CSP, which enables correctness to be proved by normal mathematical techniques.

To formalise the divergence policy of CSP, we define a new retraction $A$, which inserts arbitrary behaviour wherever a $\delta$ can occur

$$\frac{p \xrightarrow{\langle\delta\rangle} q}{Ap \xrightarrow{\langle a\rangle} Aq} \qquad\qquad \frac{p \xrightarrow{\langle a\rangle} q}{Ap \xrightarrow{\langle a\rangle} Aq}.$$

**Theorem 6.3.2.** *$A$ is a retraction.*

**Proof.** See Appendix. □

The symbol $\sigma$ has been introduced to represent an event which is intended to be hidden — except in the case of an infinite sequence of occurrences. Once we have introduced $\delta$ to make the infinite case visible, all occurrences of $\sigma$ can now be ignored. A simple way of doing this is simply to change $\sigma$ to the standard hidden event $\tau$, so that a subsequent application of $W$ will hide it completely.

$$\frac{p \xrightarrow{\langle\sigma\rangle} q}{Cp \xrightarrow{\langle\tau\rangle} Cq} \qquad\qquad \frac{p \xrightarrow{\langle a\rangle} q}{Cp \xrightarrow{\langle a\rangle} Cq} \quad \text{if } a \neq \sigma.$$

**Theorem 6.3.3.** *$C$ is monotonic and idempotent (but not decreasing).*

**Proof.** By Theorems 4.1, 4.3A and 4.3B we are only required to prove the following lemmas:

$$\begin{array}{lll}
(mon) & \leq; f\tau & \\
 & = \leq; (\xrightarrow{\tau} \cup \xrightarrow{\sigma}) & \{\text{def of } f\tau \text{ of } C\} \\
 & = f\tau; \leq & \{\leq \text{ commut}\} \\
(idem) & C; f\tau & \\
 & = C; (\xrightarrow{\tau} \cup \xrightarrow{\sigma}) & \{\text{def of } f\tau\} \\
 & = f\tau; C & \{C; \xrightarrow{\sigma} = \{\,\} \text{ and } C \text{ commut}\}.
\end{array}$$

Furthermore we have

$$\begin{array}{ll}
C; \leq; \xrightarrow{\sigma} & \\
= \{\,\} & \{\leq \text{ commut and } C; \xrightarrow{\sigma} = \{\,\}\}. \quad \Box
\end{array}$$

Which implies the negation of $(C; \leq \subseteq Id)$, i.e., $C$ is not decreasing.

The following lemma states that when these functions are applied in the right order, they do not interfere. The first function of a pair remains idempotent, even when the second function is interposed between its two applications.

**Lemma 6.3.1.**

(1) *$A$ respects $S$* **Proof** *see Appendix*
(2) *$C$ respects $S$ and $A$* **Proof** *see Appendix*
(3) *$S$ respects $C$* **Proof** *see Appendix*.

**Corollary.** *$A$; $C$ respects $S$.*

**Proof.** From Lemma 6.3.1 by Theorem 4.7. □

The missing combinations are invalid. For example $A; S; A; \leq \neq A; S; \leq$, because the function $S$ may introduce a $\delta$, which will cause the second application of $A$ on the left to introduce additional transitions not present on the right.

The lemma tells the right order for composition of these functions.

**Theorem 6.3.4.** *$S$; $A$; $C$ is idempotent.*

**Proof.** The result follows from the preceding corollary and Theorem 4.6. □

We have defined $S$, $A$ and $C$ separately, in order to provide modular options that can be combined or omitted at will. To obtain standard models of CSP we will need the joint effect of all three functions, so henceforth we will use the single letter $D$ to stand for their composition $S; A; C$. It is simple to define $D$ independently by a more complicated transition rule:

$$\frac{p\ fa\ q}{Dp \xrightarrow{\langle a\rangle} Dq}$$

where

$$\begin{array}{lll} f\sigma & \cong \{\,\} & \\ fa & \cong \xrightarrow{\langle a\rangle} \cup \sigma^{\infty} \cup \xrightarrow{\langle \delta\rangle} \cup\, ga & \text{if } a \neq \sigma \\ g\tau & \cong \xrightarrow{\langle \sigma\rangle} & \\ ga & \cong \{\,\} & \text{if } a \neq \tau. \end{array}$$

The following proofs provide patterns for the proofs omitted earlier in this section.

**Theorem 6.3.5.** *D is idempotent.*

**Proof.** We will show that (1) $D^{\cup}; D; D; \leq$ and (2) $D^{\cup}; D^{\cup}; D; \leq$ are both simulations. So they are both contained in $\leq$. It follows that $(D; D; \leq)$ both contains and is contained in $(D; \leq)$.

$$\begin{array}{ll} (1)\ D^{\cup}; D; D; \leq; \xrightarrow{\langle a\rangle} & \\ \subseteq\ D^{\cup}; D; D; \xrightarrow{\langle a\rangle}; \leq & \{\text{def} \leq\} \\ \subseteq\ D^{\cup}; D; fa; D; \leq & \{D \text{ commut}\} \\ \subseteq\ D^{\cup}; fa; D; D; \leq & \\ & \left\{\begin{array}{l} D; \{\,\} = D; \xrightarrow{\langle \sigma\rangle} = D; \sigma^{\infty} = \{\,\}, \\ D; (\xrightarrow{\langle a\rangle} \cup \xrightarrow{\langle \delta\rangle}) = fa; D \end{array}\right\} \\ \subseteq\ \xrightarrow{\langle a\rangle}; D^{\cup}; D; D; \leq & \{D \text{ commut}\} \\ (2)\ D^{\cup}; D^{\cup}; D; \leq; \xrightarrow{\langle a\rangle} = \{\,\} & \text{if } a = \sigma \\ \qquad\qquad \subseteq D^{\cup}; \xrightarrow{\langle a\rangle}; D^{\cup}; D; \leq & \{\text{trivially}\}. \end{array}$$

In the case that $a \neq \sigma$

$$\begin{array}{lll} D^{\cup}; D^{\cup}; D; \leq; \xrightarrow{\langle a\rangle} & \subseteq D^{\cup}; \xrightarrow{\langle a\rangle}; D^{\cup}; D; \leq & \{\text{as in } (1)\} \\ & \subseteq D^{\cup}; fa; D^{\cup}; D; \leq & \{\text{since } a \neq \sigma\} \\ & \subseteq \xrightarrow{\langle a\rangle}; D^{\cup}; D^{\cup}; D; \leq & \{D \text{ commut}\}. \quad \square \end{array}$$

**Theorem 6.3.6.** *D is monotonic.*

**Proof.** By Theorem 4.1, we need only prove $\leq; fa \subseteq fa; \leq$.

The cases where $fa$ is a single transition follow from the definition of $\leq$.

$$\begin{array}{llll} & \leq; \sigma^{\infty} & = \leq; \xrightarrow{\langle \sigma\rangle}; \sigma^{\infty} & \{\text{def } \sigma^{\infty}\} \\ & & \subseteq \xrightarrow{\langle \sigma\rangle}; \leq; \sigma^{\infty} & \{\text{def} \leq\} \\ \text{So} & \leq; \sigma^{\infty} & \subseteq \sigma^{\infty} & \{\text{def } \sigma^{\infty}\} \\ & & \subseteq \sigma^{\infty}; \leq & \{\leq \text{refl}\}. \end{array}$$

We define divergence-barbed simulation and the trace-divergence model of CSP in the usual way

$$\begin{array}{l} \leq_d\ \cong D; \leq; D^{\cup} \\ \sqsubseteq_d\ \cong D; \sqsubseteq; D^{\cup}. \end{array}$$

Other versions of CCS can be obtained by adding refusal barbs, and using weak simulation in place of strong. For example weak simulation with refusal and divergence barbs can be defined

$$D; R; W; \leq; W^{\cup}; R^{\cup}; D^{\cup}. \quad \square$$

The following lemma shows the interactions between $D$ and the links defined in previous sections.

**Lemma 6.3.2.** **(1)** *T respects D*

**(2)** *R respects D*

**(3)** *W respects D.*

**Proof.** **(1)** First we show that (a) $(T;\, D;\, \sqsubseteq) \subseteq (D;\, \sqsubseteq)$

$$\begin{array}{lll}
& T;\, D;\, \xrightarrow{\langle a\rangle s};\, U & \\
= & T;\, fa;\, D;\, \xrightarrow{s};\, U & \{D \text{ commut}\} \\
\subseteq & \xrightarrow{a}\,;\, T;\, D;\, \xrightarrow{s};\, U \cup & \\
& \sigma^{\infty};\, D;\, \xrightarrow{s};\, U \cup \xrightarrow{\delta}\,;\, T;\, D;\, \xrightarrow{s};\, U & \{T \text{ commut and } T;\, \sigma^{\infty} \subseteq \sigma^{\infty}\} \\
\subseteq & \xrightarrow{a};\, D;\, \xrightarrow{s};\, U \cup & \\
& \sigma^{\infty};\, D;\, \xrightarrow{s};\, U \cup \xrightarrow{\delta};\, D;\, \xrightarrow{s};\, U & \{\text{induction hypothesis}\} \\
= & D;\, \xrightarrow{a};\, \xrightarrow{s};\, U & \{D \text{ commut}\} \\
= & D;\, \xrightarrow{\langle a\rangle s};\, U. &
\end{array}$$

Then we show that (b) $(D;\, T;\, D;\, \sqsubseteq) \subseteq (D;\, \sqsubseteq)$

$$\begin{array}{lll}
& traces(DTDp) & \\
\subseteq & traces(D^2p) & \{\text{Conclusion (a)}\} \\
= & traces(Dp) & \{\text{Theorem 6.3.5}\}.
\end{array}$$

Thirdly we show that (c) $(D;\, T;\, D;\, \leq) \subseteq (D;\, T;\, \leq)$

$$\begin{array}{lll}
& D;\, T;\, D;\, \leq & \\
\subseteq & D;\, T;\, D;\, \sqsubseteq & \{\leq\, \subseteq\, \sqsubseteq\} \\
\subseteq & D;\, \sqsubseteq & \{\text{Conclusion (b)}\} \\
= & D;\, T;\, \leq & \{\text{Theorem 5.1.4 Corollary}\}.
\end{array}$$

Finally we can show that $DTDp \leq TDp$ because

(i) $D;\, T;\, \xrightarrow{\sigma} = \{\,\}$, and

(ii) $\xrightarrow{a} \subseteq fa$ for $a \neq \sigma$. □

**Proof.** **(2)** See Appendix. □

**Proof.** **(3)** See Appendix. □

**Theorem 6.3.7.** *$R;\, W;\, T$ respects $D$.*

**Proof.** From Lemma 6.3.2. by Theorem 4.7. □

If $R$ were monotonic, the remainder of the paper would be a simple application of Theorem 4.6. In fact, we need one more lemma, stating that $R$ has no effect after $W$, even though its application is separated by both $T$ and $D$.

**Lemma 6.3.3.**

$$W;\, T;\, D;\, R;\, \leq\, =\, W;\, T;\, D;\, \leq.$$

**Proof.** Similar to Lemma 6.2.2. □

**Theorem 6.3.8.**

$$D;\, R;\, W;\, T;\, D;\, R;\, W;\, \leq\, =\, D;\, R;\, W;\, T;\, \leq.$$

**Proof.**

$$\begin{array}{lll}
LHS & = D;\, R;\, W;\, T;\, D;\, R;\, \leq;\, W;\, \leq & \{W \ mon\} \\
& = D;\, R;\, W;\, T;\, D;\, \leq;\, W;\, \leq & \{\text{Lemma 6.3.3}\} \\
& = D;\, R;\, W;\, T;\, \leq;\, W;\, \leq & \{\text{Theorem 6.3.8}\} \\
& = D;\, R;\, W;\, T;\, W;\, \leq & \{W \ mon\} \\
& = D;\, R;\, W;\, T;\, \leq & \{\text{Theorem 5.3.3}\}.
\end{array}$$

Define

$$\begin{array}{ll}
\sqsubseteq_{drw} & =_{df}\ D;\, R;\, W;\, \sqsubseteq;\, W^{\cup};\, R^{\cup};\, D^{\cup} \\
\leq_{drw} & =_{df}\ D;\, R;\, W;\, \leq;\, W^{\cup};\, R^{\cup};\, D^{\cup}.
\end{array}$$

We will show that $\sqsubseteq_{drw}$ is essentially the same as the CSP notation of failures–divergence–refinement. Let us define a minus operation that removes $\tau$ and all of a trace after first $\delta$ or refusal

$$\begin{array}{ll} (\langle\rangle)- & =_{df} \langle\rangle \\ (\langle a\rangle t)- & =_{df} \begin{cases} \langle\delta\rangle & a=\delta \\ \langle ref(X)\rangle & a = ref(X) \\ t- & a=\tau \\ \langle a\rangle(t-) & \text{otherwise} \end{cases} \\ S- & =_{df} \{s- \mid s\in S\} \\ S+ & =_{df} \{s \mid s- \in S-\}. \quad \Box \end{array}$$

**Lemma 6.3.4.**

$$traces(WRDp) = traces(WRDp)-+.$$

**Proof.** Similar to Lemma 6.1.2. □

Define

$$\begin{array}{ll} failures(p) & =_{df} \{t \mid final(t) = \langle \mathrm{ref}(X)\rangle \wedge\ t \in traces(WRDp)-\} \\ divergences(p) & =_{df} \{t \mid final(t) = \langle\delta\rangle \wedge\ t \in traces(WRDp)-\}. \end{array}$$

**Lemma 6.3.5.**

*$p \sqsubseteq_{drw} q$ if $failures(p) \supseteq failures(q)$ and $divergences(p) \supseteq divergences(q)$.*

**Proof.** From the fact that

$$\begin{array}{l} traces(WRDp) = \\ \{s\langle\delta\rangle t \mid s \in (A\cup\{\tau\})^* \wedge (s\backslash\{\tau\})\langle\delta\rangle \in traces(WRDp)-\} \cup \\ \{s\langle \mathrm{ref}(X)\rangle t \mid s \in (A\cup\{\tau\})^* \wedge (s\backslash\{\tau\})\langle \mathrm{ref}(X)\rangle \in traces(WRDp)-\} \cup \\ \{s \mid s \in (A\cup\{\tau\})^* \wedge (s\langle\delta\rangle \in traces(WRDp) \vee\ s\langle \mathrm{ref}(X)\rangle \in traces(WRDp))\}. \quad \Box \end{array}$$

**Lemma 6.3.6.**

$$\sqsubseteq_{DRW} = (D;\ R;\ W;\ T;\ \le_{DRW}).$$

**Proof.**

$$\begin{array}{ll} \phantom{=}\ LHS & \\ = D;\ R;\ W;\ T;\ \le;\ W^{\cup};\ R^{\cup};\ D^{\cup} & \{\text{Theorem 5.1.7}\} \\ = RHS & \{\text{Theorem 6.3.8}\}. \quad \Box \end{array}$$

**Theorem 6.3.9.** *The standard failure/divergence model of CSP (FDR) is a retract (by $D;\ R;\ W;\ T$) of CCS modulo weak divergence-barbed and refusal-barbed simulation.*

**Proof.** From Lemmas 6.3.5 and 6.3.6. □

**Theorem 6.3.10.** *The trace-divergence model of CSP is a retract (by $D;\ W;\ T$) of CCS modulo weak divergence-barbed simulation.*

**Proof.** Similar to Theorem 6.3.9. □

## 7. Conclusion

The ideas of this paper have been derived from many sources. The concept of a retraction was introduced from topology to Computer Science by Scott [23]. Relationships between families of process algebras and their orderings have been comprehensively explored in [25,26]. The algebraic expressive power and other properties of CCS and CSP have been analysed and compared in [3,9,24] and elsewhere. Algebra provides the primary mode of presentation of the semantics of ACP [5]. The use of transition rules to define functions is due to [19]. Barbed simulation was introduced in [17], and refusal barbs have been comprehensively treated in [18]. A series of testing equivalences (including traces and refusals) have been defined in [1]; they culminate in a testing equivalence identical to observation equivalence. Saturation of a transition system has been used in [7] as a means to relate testing pre-orders with observation equivalence. Refinement has been related to a higher-order version of simulation by Gardiner [8]. The efficiency of retractions has been exploited in the model-checking tool FDR [21,22]. The use of links to unify theories of programming is widespread in [11]. This paper can only claim to put these familiar ideas together in a possibly illuminating way.

The goals of this paper are most similar to those of Abramsky [1] and Cleavel and Hennessy [7]. Cleavel and Hennessy define a parameterised form of bisimulation, $\sqsubseteq_{L,R}$ where the parameters indicate at each step whether the matching of $a$-successors of a process has to be by simulation, by reverse simulation or both. They define tests based on may-ordering and must-ordering; these deal particularly with deadlock and divergence. A powerset algorithm is introduced to determinise a process. Finally they show that various testing pre-orders are in fact bisimulations with appropriate choice of parameters $L$ and $R$.

Abramsky proceeds in the reverse direction. He uses a standard observational equivalence, a bisimulation with just a single parameter $L$. He then defines a series of languages of varying expressive power for the specification of tests. The first allows only tests of traces, the second allows tests of single-event refusals. The most powerful languages permit conjunctions and disjunctions of tests; these tests may require taking a copy of the process under test at its current stage of progress. For example a refusal set with more than one member can be tested by testing a single-event refusal on multiple copies of the current state. The concept of passing or failing a test is defined (equivalently) both by a function and by a deductive system. The main theorem shows that the most powerful language of tests gives the same answer as observational equivalence.

This paper differs clearly in its emphasis from earlier works. We have concentrated on simulations and retractions, which have been applied specifically to illuminate the relationship between various familiar versions of CCS and of CSP. Only the simplest standard definitions of simulation and trace refinement have been used. All the interesting variation is provided by the definition of the links between the theories. The links are simply defined by transition rules. A link simultaneously specifies the healthiness conditions and the ordering of the target theory. In the case of a retraction, a link translates each process of the source theory onto its closest approximation in the target theory. The various links can be applied separately or combined in appropriate ways. The proofs are trivial, and have been presented in an unusual algebraic style. It is hoped that these or similar techniques may be found useful in the study of the varieties of more modern process algebras like the $\pi$-calculus [15,16] and bigraphical systems [13].

An unexpected result of this research is that a specification-oriented denotational semantics like that of CSP can be systematically derived by means of operationally defined links from a purely operational semantics like that of CCS. Our first attempt [11] at a derivation in the reverse direction could only prove partial correctness of an operational implementation of CSP. A unifying theory that reconciles these two styles of semantic presentation may be influential when model-checking tools based on simulation of an operational semantics are used in combination with theorem-proving tools based on the more abstract mathematical concepts in specification-oriented denotational semantics. Perhaps one day such tools will be regularly applied together for reliable system design and validation [12].

## Acknowledgements

The inspiration for this paper was a Workshop held at Microsoft Research Ltd., in Cambridge on 22–23 July 2002. Those who contributed were Ernie Cohen, Cedric Fournet, Paul Gardiner, Andy Gordon, Tony Hoare, Robin Milner, Sriram Rajamani, Jacob Rehof and Bill Roscoe. Some of the ideas from the Workshop have been incorporated into the model-checking tool, Zing [20]. The paper has benefited from comments on previous versions provided by Robin Milner, Samson Abramsky, Rob van Glabbeek and Carroll Morgan.

The first author was partially supported by National Basic Research Program of China (No. 2002CB312001 and No. 2005CB321904).

## Appendix

**Theorem 6.3.2.** *A is a retraction.*

**Proof.** By Theorems 4.1, 4.2, 4.3A and 4.3B, we are only required to establish the following results:

$$\begin{array}{lll}
(idem) & A;\, fa & \\
 & = A;\, (\xrightarrow{a} \cup \xrightarrow{\delta}) & \{\text{def of } fa \text{ of } A\} \\
 & = (\xrightarrow{a} \cup \xrightarrow{\delta});\, A \cup \xrightarrow{\delta};\, A & \{A \text{ commut}\} \\
 & = fa;\, A & \{\text{set theory}\} \\
(mon) & \le\, ;\, fa & \\
 & = \le;\, (\xrightarrow{a} \cup \xrightarrow{\delta}) & \{\text{def of } fa\} \\
 & = fa;\ \le & \{\le\ \text{commut}\} \\
(dec) & A^{\cup};\, \xrightarrow{a} & \\
 & \subseteq A^{\cup};\, fa & \{\text{set theory}\} \\
 & \subseteq \xrightarrow{a};\, A & \{A \text{ commut}\}.
\end{array}$$

**Lemma 6.3.1(1).** *A respects S .*

**Proof.** Because $S$ is decreasing one has

$$(S; A; \le) \subseteq (S; A; S; \le).$$

We will show that $(A^{\cup}; S^{\cup}; S; A; S;\ \le) \subseteq \le$

$$\begin{array}{ll}
\quad A^{\cup}; S^{\cup}; S; A; S;\ \le; \xrightarrow{\delta} & \\
= A^{\cup}; S^{\cup}; S; A; (\xrightarrow{\delta} \cup \sigma^{\infty}); S; \le & \{\le \text{ commut and } S \text{ commut}\} \\
\subseteq A^{\cup}; S^{\cup}; S; \xrightarrow{\delta}; A; S; \le \cup & \\
\quad A^{\cup}; S^{\cup}; S; \sigma^{\infty}; S; \le & \{A \text{ commut and } A; \sigma^{\infty} \subseteq \sigma^{\infty}\} \\
\subseteq \xrightarrow{\delta}; A^{\cup}; S^{\cup}; S; A; S;\ \le \cup & \\
\quad A^{\cup}; S^{\cup}; \sigma^{\infty}; S; \le & \{A,\ S \text{ commut and } S; \sigma^{\infty} \subseteq \sigma^{\infty}\} \\
\subseteq \xrightarrow{\delta}; A^{\cup}; S^{\cup}; S; A; S;\ \le \cup & \\
\quad A^{\cup}; S^{\cup}; (\xrightarrow{\delta} \cup \sigma^{\infty}); S; \le & \{\text{set theory}\} \\
\subseteq \xrightarrow{\delta}; A^{\cup}; S^{\cup}; S; A; S;\ \le \cup & \\
\quad \xrightarrow{\delta}; A^{\cup}; S^{\cup}; S; \le & \{A, S \text{ commut}\} \\
\subseteq \xrightarrow{\delta}; A^{\cup}; S^{\cup}; S; A; S;\ \le & \{\le\ \subseteq (A; S; \le)\}.
\end{array}$$

**Lemma 6.3.1(2).** *C respects S and A.*

**Proof.** First we are going to show that $(C^{\cup}; S^{\cup}; S; C; S;\ \le) \subseteq \le$

$$\begin{array}{ll}
\quad C^{\cup}; S^{\cup}; S; C; S;\ \le; \xrightarrow{\delta} & \\
= C^{\cup}; S^{\cup}; S; C; (\xrightarrow{\delta} \cup \sigma^{\infty});\ S; \le & \{\le, S\text{commut}\} \\
= C^{\cup}; S^{\cup}; (\xrightarrow{\delta} \cup \sigma^{\infty}); S; C; S;\ \le & \{C; \xrightarrow{\sigma} = \{\} \text{ and } C, S \text{ commut}\} \\
\subseteq \xrightarrow{\delta}; C^{\cup};\ S^{\cup}; S; C; S;\ \le & \{C, S \text{ commut}\} \\
\quad C^{\cup}; S^{\cup}; S; C; S;\ \le; \xrightarrow{\tau} & \\
= C^{\cup}; S^{\cup}; S; (\xrightarrow{\tau} \cup \xrightarrow{\sigma}); C; S; \le & \{\le, S, C \text{ commut}\} \\
\subseteq C^{\cup}; (\xrightarrow{\tau} \cup \xrightarrow{\sigma}); S^{\cup}; S; C; S;\ \le & \{S \text{ commut}\} \\
\subseteq \xrightarrow{\tau}; C^{\cup}; S^{\cup}; S; C; S;\ \le & \{C \text{ commut}\}.
\end{array}$$

In the following we will show that $(C^{\cup}; A^{\cup}; A; C; A; \le) \subseteq \le$

$$\begin{array}{ll}
\quad C^{\cup}; A^{\cup}; A; C; A;\ \le; \xrightarrow{\tau} & \\
= C^{\cup}; A^{\cup}; A; C; (\xrightarrow{\delta} \cup \xrightarrow{\tau}); A; \le & \{\le, A\ \text{ commut}\} \\
= C^{\cup}; A^{\cup}; A; (\xrightarrow{\sigma} \cup \xrightarrow{\delta} \cup \xrightarrow{\tau}); C; A; \le & \{C \text{ commut}\} \\
= C^{\cup}; A^{\cup}; (\xrightarrow{\sigma} \cup \xrightarrow{\delta} \cup \xrightarrow{\tau}); A; C; A;\ \le & \{A \text{ commut}\} \\
\subseteq C^{\cup}; (\xrightarrow{\sigma} \cup \xrightarrow{\tau}); A^{\cup}; A; C; A;\ \le & \{A \text{ commut}\} \\
\subseteq \xrightarrow{\tau}; C^{\cup}; A^{\cup}; A; C; A;\ \le & \{C \text{ commut}\}.
\end{array}$$

**Lemma 6.3.1(3).** *S respects C.*

**Proof.** We will only show that $(C; S; \le) \subseteq (C; S; C; \le)$. The reverse inequation can be established in a similar way.

$$\begin{array}{ll}
\quad C^{\cup}; S^{\cup}; C^{\cup}; C; S; \xrightarrow{\tau} & \\
= C^{\cup}; S^{\cup}; C^{\cup}; (\xrightarrow{\sigma} \cup \xrightarrow{\tau}); C; S; \le & \{\le, C, S \text{ commut}\} \\
\subseteq C^{\cup}; \xrightarrow{\tau}; S^{\cup}; C^{\cup}; C; S; \le & \{C, S \text{ commut}\} \\
\subseteq C^{\cup}; (\xrightarrow{\sigma} \cup \xrightarrow{\tau}); S^{\cup}; C^{\cup}; C; S; \le & \{\text{set theory}\} \\
\subseteq \xrightarrow{\tau}; C^{\cup}; S^{\cup}; C^{\cup}; C; S; \le & \{C \text{ commut}\} \\
\quad C^{\cup}; S^{\cup}; C^{\cup}; C; S; \le; \xrightarrow{\delta} & \\
= C^{\cup}; S^{\cup}; C^{\cup}; C; (\xrightarrow{\delta} \cup \sigma^{\infty}); S; \le & \{\le, S \text{ commut}\} \\
\subseteq C^{\cup}; S^{\cup}; \xrightarrow{\delta}; C^{\cup}; C; S; \le & \{C \text{ commut and } C; \sigma^{\infty} = \{\}\} \\
\subseteq C^{\cup}; S^{\cup}; (\xrightarrow{\delta} \cup \sigma^{\infty}); C^{\cup}; C; S; \le & \{\text{set theory}\} \\
\subseteq \xrightarrow{\delta}; C^{\cup}; S^{\cup}; C^{\cup}; C; S; \le & \{C, S \text{ commut}\}.
\end{array}$$

**Lemma 6.3.2(1).** *R respects D.*

**Proof.** First we are going to show that $(D;\, R;\, D;\ \leq) \subseteq (D;\, R;\, \leq)$

$$\begin{array}{ll}
\quad R^{\cup};\, D^{\cup};\, D;\, R;\, D;\ \leq;\, \xrightarrow{\tau} & \\
= R^{\cup};\, D^{\cup};\, D;\, R;\, f_D^{\tau};\, D;\, \leq & \{\leq, D \text{ commut}\} \\
\subseteq R^{\cup};\, D^{\cup};\, D;\, (\xrightarrow{\tau} \cup \xrightarrow{\delta} \cup \xrightarrow{\sigma});\, R;\, D;\, \leq & \{R \text{ commut and } D;\, \sigma^{\infty} = \{\}\} \\
= R^{\cup};\, D^{\cup};\, f_D^{\tau};\, D;\, R;\, D;\ \leq & \{D \text{ commut}\} \\
\subseteq \xrightarrow{\tau};\, R^{\cup};\, D^{\cup};\, D;\, R;\, D;\ \leq & \{R, D \text{ commut}\}.
\end{array}$$

The reverse inequation $(D;\, R;\, \leq) \subseteq (D;\, R;\, D;\ \leq)$ can be proved in the same way as in Lemma 6.3.2(1).

**Lemma 6.3.2(2).** *W respects D.*

**Proof.** We are going to show that $(D;\, W;\, D;\ \leq) \subseteq (D;\, W;\, \leq)$

$$\begin{array}{ll}
\quad W^{\cup};\, D^{\cup};\, D;\, W;\, D;\ \leq;\, \xrightarrow{\tau} & \\
= W^{\cup};\, D^{\cup};\, D;\, W;\, f_D^{\tau};\, D;\, \leq & \{\leq, D \text{ commut}\} \\
= W^{\cup};\, D^{\cup};\, D;\, (\overset{\tau}{\Longrightarrow} \cup \overset{\delta}{\Longrightarrow});\, W;\, D;\, \leq & \{W \text{ commut and } D;\, \xrightarrow{\sigma} = \{\}\} \\
= W^{\cup};\, D^{\cup};\, (f_D^{\tau})^*;\, D;\, W;\, D;\ \leq & \{D \text{ commut}\} \\
\subseteq W^{\cup};\, \overset{\tau}{\Longrightarrow};\, D^{\cup};\, D;\, W;\, D;\ \leq & \{D\ \ \text{commut}\} \\
\subseteq \xrightarrow{\tau};\, W^{\cup};\, D^{\cup};\, D;\, W;\, D;\ \leq\ . &
\end{array}$$

Similar to Lemma 6.3.2(1) we can also show that $(D;\, W;\, \leq) \subseteq (D;\, W;\, D;\ \leq)$. □

## References

[1] Samson Abramsky, Observation equivalence as a testing equivalence, Theoret. Comput. Sci. 53 (1987) 225–241.
[2] Bisimulation can't be traced, J. ACM 31 (3) (1995) 560–599.
[3] S.D. Brookes, On the Relationship of CCS and CSP, in: LNCS, vol. 154, 1983.
[4] S.D. Brookes, C.A.R. Hoare, A.W. Roscoe, A theory of communicating sequential processes, J. ACM 31 (3) (1984) 560–599.
[5] J.A. Bergstra, J.W. Klop, Algebra of communicating processes with abstraction, Theoret. Comput. Sci. 37 (1) (1985) 77–121.
[6] R. Cleaveland, J. Parrow, B. Steffen, The Concurrency Workbench, in: LNCS, vol. 407, 1989, pp. 24–37.
[7] Rance Cleaveland, Matthew Hennessy, Testing equivalence as a bisimulation equivalence, FAC 2 (4) (1993) 1–20.
[8] P. Gardiner, Power simulation and its relation to traces and failures refinement, Theoret. Comput. Sci. 309 (1) (2003) 157–176.
[9] Matthew Hennessy, Robin Milner, Algebraic laws for non-determinism and concurrency, J. ACM 32 (1) (1985) 137–161.
[10] J.F. Groote, F. Vaandrager, Structured operational semantics and bisimulation as a congruence, Inform. Comput. 100 (2) (1992) 202–260.
[11] C.A.R. Hoare, He Jifeng, Unifying Theories of Programming, Prentice Hall, 1998.
[12] A.W. Roscoe, J.N. Reed, Je Sinclair, Responsiveness and stable revivals, FAC 19 (3) (2007).
[13] Bigraphs and mobile processes (revised), UCAM-CL-TR-580, 2004.
[14] K. Larsen, A. Skou, Bisimulation through probabilistic testing, Inform. Comput. 94 (1) (1991) 1–28.
[15] A.J.R.G. Milner, Communication and Concurrency, Prentice Hall, 1985.
[16] R. Milner, Communicating and Mobile Systems: The $\pi$-Calculus, Cambridge University Press, 1999.
[17] R. Milner, D. Sangiorgi, Barbed Bisimulation, in: Springer LNCS, vol. 623, 1992.
[18] I. Phillips, Refusal Testing, in: Springer LNCS, vol. 226, 1986, pp. 304–313.
[19] G.D. Plotkin, A structural approach to operational semantics, DAIMI-FN-19, Aarhus University, Denmark, 1981.
[20] Sriram K. Rajamani, Jakob Rehof, Shaz Qadeer, Yichen Xie, Tony Andrews, Zing: A model checker for concurrent software, in: Proceedings CAV 2004, in: LNCS, vol. 3114, 2004, pp. 484–487.
[21] A.W. Roscoe, The Theory and Practice of Concurrency, Prentice Hall, 1998.
[22] B. Meyer, J. Woodcock (Eds.), Verified Software: Theories, Tools, Experiments, in: LNCS, vol. 4171, 2008.
[23] Dana Scott, Data types as lattices, SIAM J. Comput. 5 (1976) 522–587.
[24] R.J. van Glabbeek, Notes on the methodology of CCS and CSP, Theoret. Comput. Sci. 177 (2) (1997) 329–349.
[25] R.J. van Glabbeek, The linear time — Branching time spectrum I, in: Handbook of Process Algebra, Elsevier, 2001, pp. 3–39.
[26] R.J. van Glabbeek, The linear time — Branching time spectrum II; The semantics of sequential processes with silent moves, in: E. Best (Ed.), Proceedings CONCUR'93, Hildesheim, Germany, August 1993, in: LNCS, vol. 715, Springer-Verlag, 1993, pp. 66–81 (extended abstract).