Finite Axiom Systems for Testing Preorder and De Simone Process Languages

Irek Ulidowski
Research Institute for Mathematical Sciences, Kyoto University, Japan
Email: irek@kurims.kyoto-u.ac.jp, tel:+ 8175-753-7238

Abstract

We prove that testing preorder of De Nicola and Hennessy is preserved by all De Simone process operators. Based on this result we propose an algorithm for generating axiomatisations of testing preorder for arbitrary De Simone process languages. The axiom systems produced by our algorithm are finite and complete for processes with finite behaviour. In order to achieve completeness for a subclass of processes with infinite behaviour we use one infinitary induction rule. The usefulness of our results is illustrated in specification and verification of small concurrent systems, where suspension, resumption and alternation of execution of component systems occur. We argue that better specifications can be more easily written in custom-made De Simone process languages which contain both the standard operators as well as new, specially tailored for the task in hand De Simone operators. Moreover, the automatically generated axiom systems for such specification languages make the verification straightforward.

1 Introduction

Process languages, for example CCS of Milner and CSP of Hoare, together with their semantic theories are reasonably successful in specification and verification of small concurrent systems. Both specifications, of varying level of abstractness, and the implementation of a given system are represented by processes—the terms in a process language. The task of proving that a more concrete specification refines an abstract one, and that the implementation meets some specification is then performed. There is a number of different proof methods which can be employed. One can define a denotational meaning of processes and then argue that the abstract specifications, concrete specifications and the implementation have increasingly better (in some sense) denotations. Alternatively, an operational preorder may be defined which relates the abstract with concrete specifications and the concrete specifications with the implementation. The third method, which is solely based on the syntax of processes, uses equational reasoning within some axiom system.

The major process languages (algebras) and their theories were developed in [Hoa85, Mil89a, Hen88, BW90] where, among others, the above described approach to specification and verification of concurrent systems was originally presented. More recently, due to Plotkin style *structured operational semantics* (SOS), these particular process languages and their numerous extensions were generalized to whole classes of process languages. Moreover, the theories for these languages and specification and verification techniques were extended to apply to such classes of languages. For example, a number of important preorders and equivalences on processes were shown to be congruences for certain classes of

process languages [BIM88, BIM95, GV90, Gro90, Vaa91, Uli92, Blo95a, Uli94]. Axiomatic systems and denotational models for these classes of process languages were proposed in [ABV92, ABV94, Ace94, Uli95] and [AI95] respectively. Also, two classes of languages were used in [Blo95b] as general specification languages.

This paper continues the research related to general process languages. We consider a class of process languages generated by the De Simone format of rules with a special treatment of silent actions. This class, denoted by τDeS, was defined in [Uli95]. Out of many preorders (equivalences) proposed in the concurrency literature we consider *testing preorder* of De Nicola and Hennessy [NH84, Nic87, Hen88], which is based on an intuitive notion of comparing responses of processes to tests.

The first contribution of this paper is the proof that testing preorder is a precongruence for all τDeS languages. This result is the foundation on which our axiomatisation algorithm is based. It guarantees *compositionality* of specification and verification when working with τDeS process languages. For example, it allows to split the verification of a complex system into the verifications of its components. In this paper we employ equational reasoning as the method of proof for verification.

Since arbitrary τDeS operators are to be used in our specifications we require a general algorithm for deriving axiomatisations for these operators. Our second contribution is such an algorithm. It produces sound and complete axiomatisations of testing preorder for of τDeS languages generating well-founded processes. In contrast to axiomatisations of refusal simulation preorder for well-founded τDeS languages [Uli95] the axiomatisations of testing preorder for the same class of process languages are *finite*. Our algorithm is a modification of the original axiomatisation algorithm developed by Aceto, Bloom and Vaandrager [ABV92, ABV94] for strong bisimulation and GSOS process languages.

We support the argument of Bloom in [Blo95b] that classes of process languages, where one can use any operators within the class, are more superior for specification purposes than the standard languages like CCS and CSP. This is because simpler and easier to understand specifications can be written using both newly invented, specially tailored operators and the standard operators. As our third contribution we present examples of systems, where *suspension* and *resumption* of execution is important. Such systems usually do not have simple specifications in standard languages, although the addition of extra features like prioritized actions [CH90] may help. However, clear and intuitive specifications of such systems can be written using new, invented for that purpose operators. We give both abstract and concrete specifications. Equational reasoning, based on axiomatisation generated by our procedure, is then used to show that the specifications are equivalent.

The paper is organized as follows. We recall, in Section 2, the definitions of τDeS process languages, labelled transition systems and testing preorder. In Section 3 we show that (a characterisation of) testing preorder is a precongruence for any τDeS language. We begin Section 4 by restating a completeness result for a simple τDeS language and testing preorder. Then, we present an algorithm for generating sound axiomatisations for arbitrary τDeS languages. These axiomatisations are proved to be finite and head normalising for certain classes of τDeS languages. The section finishes with a completeness result. In Section 5 we present two examples illustrating the application of our results. The last section contains a conclusion and a discussion of related work. A number of proofs concerning soundness, head normalisation and completeness of axiom systems produced by our algorithm appears in Appendices A and B.

2 Basic Concepts

In this section we repeat the definitions of τDeS process languages, labelled transition systems and testing preorder. We also define a number of other notions in order to make the paper self-contained. Full definitions and the results concerning the above can be found, for example, in [NH84, Hen88, BIM95, GV90, Uli95, Uli92].

Let Var be a countable set of variables ranged by $X, Y, \ldots$. A signature Σ is a set process operators (f, n), where $f \notin \mathsf{Var}$ and $n \in \mathbb{N}$ is the arity of the operator. Often, when the operator's arity is clear from the context, the operator is simply abbreviated by its name: for example f. The set of open terms over Σ with variables in V is denoted by $\mathbb{T}(\Sigma, V)$. $\mathbb{T}(\Sigma, \mathsf{Var})$, abbreviated by $\mathbb{T}(\Sigma)$, is ranged by $P, Q, \ldots$. The set of closed terms $\mathrm{T}(\Sigma)$ is ranged by $p, q, t, \ldots$. A Σ context with n holes $C[X_1, \ldots, X_n]$, $n \in \mathbb{N}$, is a member of $\mathbb{T}(\Sigma, \{X_1, \ldots, X_n\})$ in which each X_i occurs at most once. If $t_1, \ldots, t_n$ are Σ terms then $C[t_1, \ldots, t_n]$ is a term obtained by replacing each X_i with t_i, $1 \leq i \leq n$. We will often denote a sequence of process terms $P_1, \ldots, P_n$ as $\boldsymbol{P}$. We will also write $\boldsymbol{P}\mathcal{R}\boldsymbol{Q}$, for some relation $\mathcal{R}$, to mean $P_1 \mathcal{R} Q_1, \ldots, P_n \mathcal{R} Q_n$.

A preorder $\sqsubseteq$ on $\mathrm{T}(\Sigma)$ is a precongruence (or substitutive) if for all Σ contexts $C[\boldsymbol{X}]$ we have $\boldsymbol{t} \sqsubseteq \boldsymbol{t'}$ implies $C[\boldsymbol{t}] \sqsubseteq C[\boldsymbol{t'}]$. A (closed) *substitution* ρ is a mapping from Var to $(\mathrm{T}(\Sigma))$ $\mathbb{T}(\Sigma)$. The substitution ρ extends to a mapping from $\mathbb{T}(\Sigma)$ to $\mathbb{T}(\Sigma)$ in a standard way. We will write $Q[P_1/X_1, \ldots, P_n/X_n]$ to denote the substitution which assigns each X_i to P_i. $\mathsf{Act} \cup \{\tau\}$, ranged by $\alpha, \ldots$, is a finite set of actions, where Act consists of visible actions $a, b, \ldots$, and $\tau \notin \mathsf{Act}$ is the silent action.

2.1 τDeS Process Languages

We define τDeS format of SOS rules and τDeS process languages both of which are based on the structured transition rules originally proposed by De Simone [dS85].

Definition 2.1 [Uli95] A De Simone (structured transition) rule for (f, n) has the following form:

$$\frac{\{\ X_i \xrightarrow{a_i} X_i'\ \}_{i \in I}}{f(X_1, \ldots, X_n) \xrightarrow{\alpha} C[\boldsymbol{Y}],} \tag{1}$$

where X_i, X_i' are all different variables and $I \subseteq \{1, \ldots, n\}$. $C[\boldsymbol{Y}]$ contains at most the variables $Y_1, \ldots, Y_n$, where $Y_i = X_i'$ if $i \in I$ and $Y_i = X_i$ otherwise.

A τ-rule for the above De Simone rule is any of the rules with the following form, where $i \in I$:

$$\frac{X_i \xrightarrow{\tau} X_i'}{f(X_1, \ldots, X_i, \ldots, X_n) \xrightarrow{\tau} f(X_1, \ldots, X_i', \ldots, X_n).}$$

The set of all τ-rules for rule (1) is called the set of *associated* τ-rules. A set of rules is in the De Simone format, written as τDeS, if it consists of De Simone rules, their associated τ-rules and no other rules. A process operator is τDeS if the set of its defining rules is in τDeS. The set of associated τ-rules for a τDeS operator consists of all τ-rules associated with all De Simone rules defining the operator. A De Simone process language, or simply τDeS language, is a triple $(\Sigma, A \cup \{\tau\}, R)$, where Σ is a finite set of τDeS operators, $A \subseteq \mathsf{Act}$ and R is a finite set of rules defining the operators in Σ.

Most of the operators in process algebras are τDeS except, for example, the CCS and ACP choice operator $+$ and the ACP left-merge and communication merge operators: their

sets of defining rules do not contain the required τ-rules. τDeS is a subformat of the GSOS format [BIM88, BIM95, Blo90a] and thus it enjoys all the properties of the GSOS format. In particular, τDeS operators preserve (strong) bisimulation [BIM88, BIM95]. Moreover, τDeS is a subformat of the ISOS format [Uli92, Uli94] which itself is a minor restriction of the GSOS format with a natural treatment of silent actions. Since a version of weak bisimulation, sometimes called *delay* bisimulation, is preserved by ISOS operators [Uli94] it is also preserved by τDeS operators.

Below, we define a number of notions related to De Simone rules.

Definition 2.2 Let $(\Sigma, \mathsf{Act} \cup \{\tau\}, R)$ be τDeS language and $r \in R$ has the form (1). Then, $\{X_i \stackrel{a_i}{\rightarrow} X_i'\}_{i \in I}$ is the set of *antecedents* of r, written as $ante(r)$, and $f(X_1, \ldots, X_n) \stackrel{\alpha}{\rightarrow} C[\boldsymbol{Y}]$ is the *consequent*, written as $cons(r)$. In general, expressions $T \stackrel{\alpha}{\rightarrow} T'$ with $T, T' \in \mathbb{T}(\Sigma)$ and $\alpha \in \mathsf{Act} \cup \{\tau\}$ are called *transitions*. T is the *source* and T' is the *target* of $T \stackrel{\alpha}{\rightarrow} T'$. f is the *operator* of r and α is the *action*. The *trigger* of r is an n-tuple $(\lambda_1, \ldots, \lambda_n)$, where $\lambda_i = a_i$, if $i \in I$, and $\lambda_i = *$ otherwise. The α trigger for f is a trigger of one of De Simone rules for f with the action α. We will use $trig_\alpha(f)$ for the set of all α triggers for f. We say that i-th argument of f is *active* if there is a De Simone rule for f with the antecedent $X_i \stackrel{a_i}{\rightarrow} X_i'$, for some a_i. $active(f)$ denotes the set of all i such that the ith argument of f is active.

2.2 Labelled Transition Systems

Next, we define labelled transition systems for τDeS languages. We will use a special nullary operator Ω with no defining rules to represent divergence of processes due to underspecification (lack of information about behaviour). Later, we will also discuss another form of divergence, namely the ability to perform an infinite sequence of silent actions.

Definition 2.3 Let $G = (\Sigma_G, \mathsf{Act}_G \cup \{\tau\}, R_G)$ be a τDeS language. The *labelled transition system with divergence* for G is a structure $(\mathrm{T}(\Sigma_G), \mathsf{Act}_G \cup \{\tau\}, \rightarrow_G, \uparrow_G)$, where $\mathrm{T}(\Sigma_G)$ is a set of *processes* or process *terms*, $\rightarrow_G \subseteq \mathrm{T}(\Sigma_G) \times \mathsf{Act}_G \cup \{\tau\} \times \mathrm{T}(\Sigma_G)$ is the unique *transition relation* generated by G[1] and $\uparrow_G \subseteq \mathrm{T}(\Sigma_G)$ is the (postfix) *divergence predicate*. $\uparrow_G$ is defined as the least relation satisfying (a) $\Omega\uparrow_G$ if $\Omega \in \Sigma_G$ and (b) $f(\boldsymbol{p})\uparrow_G$, for any $f \in \Sigma_G$ and $\boldsymbol{P}$, if $p_i\uparrow_G$ for $i \in active(f)$.

The above defined transition system will be often called the basic transition system for G, and the subscript G will be omitted when it is obvious from the context. We will use the following abbreviations: $p \stackrel{a}{\rightarrow} q$ denotes $(p, a, q) \in \rightarrow$; $p \stackrel{a}{\rightarrow}$ is equivalent to $\exists q \in \mathrm{T}(\Sigma).\ p \stackrel{a}{\rightarrow} q$; and $p \stackrel{t}{\rightarrow}$ stands for $p \stackrel{a_1}{\rightarrow} \ldots \stackrel{a_n}{\rightarrow}$, where $t = a_1 \ldots a_n \in \mathsf{Act}^*$. ε is the empty sequence in Act^* and $s \leq t$, for $s, t \in \mathsf{Act}^*$, means that s is an initial subsequence of t. With the notation as in the previous definition, we say that transition $t \stackrel{\alpha}{\rightarrow} t'$, for $t, t' \in \mathrm{T}(\Sigma_G)$, is *valid* (*holds*) in $\rightarrow_G$ if $(t, \alpha, t') \in \rightarrow_G$.

Since the τDeS format is a subformat of the GSOS format the finite branching property of (basic) transition systems generated by the GSOS languages [BIM88, BIM95] also holds for basic transition systems of τDeS languages. Namely, $\{q \mid p \stackrel{\alpha}{\rightarrow} q,\ \text{any } \alpha \in \mathsf{Act} \cup \{\tau\}\}$ is finite, for each process p in a given τDeS language.

The derived transition system, which abstracts away the silent actions and captures divergence due to infinite sequences of silent actions, is defined below.

[1] Full details of the construction of transition relation for a process language defined by a general format of rules can be found in, for example, [BIM88, BIM95, GV90].

Definition 2.4 Given the basic transition system $(\mathrm{T}(\Sigma), \mathsf{Act}\cup\{\tau\}, \rightarrow, \uparrow)$ for a τDeS language a structure $(\mathrm{T}(\Sigma), \mathsf{Act} \cup \{\varepsilon\}, \Rightarrow, \Uparrow)$ is the *derived* transition system with divergence. The transition relation $\Rightarrow \subseteq \mathrm{T}(\Sigma) \times \mathsf{Act} \cup \{\varepsilon\} \times \mathrm{T}(\Sigma)$ and the divergence predicate $\Uparrow \subseteq \mathrm{T}(\Sigma) \times \mathsf{Act}^*$ are defined as follows, where $\stackrel{\tau^*}{\rightarrow}$ is the reflexive and transitive closure of $\stackrel{\tau}{\rightarrow}$:

$$p \stackrel{\varepsilon}{\Rightarrow} q \equiv p \stackrel{\tau^*}{\rightarrow} q, \qquad p \stackrel{a}{\Rightarrow} q \equiv p \stackrel{\tau^*}{\rightarrow} \stackrel{a}{\rightarrow} \stackrel{\tau^*}{\rightarrow} q,$$

$$p\Uparrow\varepsilon \equiv \exists q.(p \stackrel{\tau^*}{\rightarrow} q \ \wedge \ q\uparrow) \ \vee \ p \stackrel{\tau^\omega}{\rightarrow}, \qquad p\Uparrow as \equiv p\Uparrow\varepsilon \ \vee \ \exists q.(p \stackrel{a}{\Rightarrow} q \ \wedge \ q\Uparrow s).$$

We say 'p converges on s', written as $p\Downarrow s$, if $\neg\, p\Uparrow s$. Expressions $p \stackrel{a}{\Rightarrow}$ and $p \stackrel{t}{\Rightarrow}$ are defined correspondingly as $p \stackrel{a}{\rightarrow}$ and $p \stackrel{t}{\rightarrow}$. Expression $p \Rightarrow q$ means $p \stackrel{\alpha}{\Rightarrow} q$, for some $\alpha \in \mathsf{Act} \cup \{\varepsilon\}$. The *language* of p, written as $\mathcal{L}(p)$, is $\{s \mid p \stackrel{s}{\Rightarrow}\}$. The set of initial visible actions a process can perform $\mathcal{S}(p)$ is $\{a \mid p \stackrel{a}{\Rightarrow}\}$. The set of t descendants of process p is defined as $\mathcal{D}(p,t) \equiv \{p' \mid p \stackrel{t}{\Rightarrow} p'\}$.

We distinguish the following classes of processes and τDeS languages according to the properties of transition systems they are related to.

Definition 2.5 Assume the basic and derived transition systems for an arbitrary τDeS process language.
(a) A process p is *well-founded* if and only if there exists no infinite sequence $p_0, \alpha_0, p_1, \alpha_1, \ldots$ with $p \equiv p_0$ and $p_i \stackrel{\alpha_i}{\rightarrow} p_{i+1}$ for all $i \geq 1$. A τDeS language is well-founded if and only if all its processes are.
(b) A process p is *finite branching* if and only if $\{q \mid p' \Rightarrow q\}$ is finite, for every $t \in \mathsf{Act}^*$ and p' in $\mathcal{D}(p,t)$.
(c) A process p is *τ-divergent* if and only if $q \stackrel{\tau^\omega}{\rightarrow}$, for some $t \in \mathsf{Act}^*$ and q in $\mathcal{D}(p,t)$. A process p is *initially τ-divergent* if and only if $p \stackrel{\tau^\omega}{\rightarrow}$. A process is *$\tau$-convergent* (*initially τ-convergent*) if and only if it is not τ-divergent (not initially τ-divergent). A τDeS language is τ-convergent if and only if all its processes are τ-convergent.

Example 2.6 Not all τDeS process languages are finite branching. Consider a τDeS language with prefixing with a, the deadlocked process $\mathbf{0}$ and the operators g and f defined by

$$g \stackrel{a}{\rightarrow} g \qquad \frac{X \stackrel{a}{\rightarrow} X'}{f(X,Y) \stackrel{\tau}{\rightarrow} f(X', a.Y)} \qquad f(X,Y) \stackrel{\tau}{\rightarrow} Y.$$

Consider term $f(g,\mathbf{0})$. We have $f(g,\mathbf{0}) \stackrel{\tau}{\rightarrow} f(g,a.\mathbf{0}) \stackrel{\tau}{\rightarrow} \ldots \stackrel{\tau}{\rightarrow} f(g,a^n.\mathbf{0}) \stackrel{\tau}{\rightarrow}$ and $f(g,a^n.\mathbf{0}) \stackrel{\tau}{\rightarrow} a^n.\mathbf{0}$, for all n. Process $f(g,\mathbf{0})$ is not finite branching since $f(g,\mathbf{0}) \stackrel{\varepsilon}{\Rightarrow} a^n.\mathbf{0}$, for all n.

Finally, we define disjoint extensions of τDeS languages.

Definition 2.7 A τDeS language H is a *disjoint extension* of a τDeS language G, written as $G \leq H$, if the signature and rules of H include those of G, and H introduces no new rule for the operators of G.

2.3 Basic τDeS Language B

Throughout this work we will consider a simple τDeS language B which is essentially a finite version of the language considered in [NH87, Hen88]. It has the two CSP choice operators $\boxplus$ and $\oplus$, the *external* and *internal* choice operators respectively, instead of the CCS choice

operator $+$ and prefixing with τ. B is a τDeS process language $(\Sigma, \mathsf{Act} \cup \{\tau\}, R)$, where $\Sigma = \{(\mathbf{0},0),(\Omega,0)\} \cup \{(a,1) \mid a \in \mathsf{Act}\} \cup \{(\boxplus,2),(\oplus,2)\}$ and R is given below. Operators $(\mathbf{0},0)$ and $(\Omega,0)$ are abbreviated by $\mathbf{0}$ and Ω; they do not have the defining rules. The precedence of operators is given by $a > \boxplus > \oplus$.

prefixing	$a.X \xrightarrow{a} X, \qquad a \in \mathsf{Act}$	
external choice	$\dfrac{X \xrightarrow{a} X'}{X \boxplus Y \xrightarrow{a} X'}$	$\dfrac{Y \xrightarrow{a} Y'}{X \boxplus Y \xrightarrow{a} Y'}$
	$\dfrac{X \xrightarrow{\tau} X'}{X \boxplus Y \xrightarrow{\tau} X' \boxplus Y}$	$\dfrac{Y \xrightarrow{\tau} Y'}{X \boxplus Y \xrightarrow{\tau} X \boxplus Y'}$
internal choice	$X \oplus Y \xrightarrow{\tau} X$	$X \oplus Y \xrightarrow{\tau} Y$

We will call B the basic τDeS process language. Sometime it will be convenient to consider sub-languages of B which only have some of the prefixing operators of B. We will write $\mathsf{B}(A)$ for a language which contains prefixing operators a for all $a \in A$, the two choice operators as well as $\mathbf{0}$ and Ω. Clearly $\mathsf{B}(\mathsf{Act}) = \mathsf{B}$.

B generates the basic transition system $B = (\mathrm{T}(\Sigma), \mathsf{Act} \cup \{\tau\}, \rightarrow, \uparrow)$. Note that all processes in $\mathrm{T}(\Sigma)$ are well-founded and τ-convergent. We have $\Omega\uparrow$. Also, if $p\uparrow$ then $(p \boxplus q)\uparrow$ and $(q \boxplus p)\uparrow$. But, for example, $p\uparrow$ does not imply $(p \oplus q)\uparrow$. Given B we can define $(\mathrm{T}(\Sigma), \mathsf{Act} \cup \{\varepsilon\}, \Rightarrow, \Uparrow)$ in a straightforward way. One can check that the divergence predicate $\Uparrow$ is the least relation satisfying the following: $\Omega\Uparrow$ and if $p\Uparrow$ then $(p \boxplus q)\Uparrow$, $(q \boxplus p)\Uparrow$, $(p \oplus q)\Uparrow$ and $(q \oplus p)\Uparrow$.

2.4 Testing Preorder

Testing was introduced by De Nicola and Hennessy in [NH84, Nic87, Hen88]. The preorder based on testing, testing preorder, is defined in terms of the responses of processes to tests, which themselves are special processes with a set of actions containing a *success* action w. The responses of a process to a test are determined by observing the computations of a system consisting of the test 'applied' to the process.

This paper concerns itself with testing in τDeS process languages. For technical reasons we will limit our attention to a certain subclass of τDeS process languages which we describe below. First, we define an auxiliary notion.

Definition 2.8 A τDeS process language G with the set of visible actions A is called *self-encoding* if $\mathsf{B}(A) \leq G$.

Given any τDeS language G let G_T denote a disjoint extension of G with an extra prefixing operator $(w,1)$ not in Σ_G and the test application operator $(|,2)$ defined by the τDeS rules below.

$$\frac{X \xrightarrow{a} X' \quad Y \xrightarrow{a} Y'}{X|Y \xrightarrow{\tau} X'|Y'} \qquad \frac{X \xrightarrow{\tau} X'}{X|Y \xrightarrow{\tau} X'|Y} \qquad \frac{Y \xrightarrow{\tau} Y'}{X|Y \xrightarrow{\tau} X|Y'}$$

In what follows we will consider testing in τDeS languages G with the set of actions $A \cup \{\tau\}$ only if $\mathsf{B}(A)_T \leq G$. In other words, only if G is self-encoding and it contains the operators $(w,1)$ and $(|,2)$ described above. We will give such τDeS languages a special name.

Definition 2.9 A τDeS process language G is called *self-testing* if $\mathsf{B}_T(A) \leq G$.

When we consider testing we assume that the processes are those terms over G which do not involve prefixing with w and the tests (varied by $t, t', \ldots$) are the terms which contain w. The operator $|$ is used to express the application of a test t to a process p, written as $p \mid t$. Sequence $p_0 \mid t_0 \stackrel{\tau}{\rightarrow} p_1 \mid t_1 \ldots \stackrel{\tau}{\rightarrow} p_k \mid t_k \ldots$ is called a *computation* if it is maximal, i.e. it is infinite or it is finite with the last element $p_k \mid t_k$ such that $p_k \mid t_k \stackrel{\tau}{\rightarrow} p' \mid t'$ for no pair p', t'.

Next, we restate the definitions of relations 'may satisfy' and 'must satisfy' between processes and tests, and of testing preorders $\sqsubseteq_{\mathsf{may}}$, $\sqsubseteq_{\mathsf{must}}$ and $\sqsubseteq$ as in [NH84, Nic87, Hen88].

Definition 2.10 Let G be a self-testing τDeS process language. Given a process p and a test t over G

1. $p \,\mathsf{may}\, t$ if $p \mid t \stackrel{\tau^*}{\rightarrow} p' \mid t'$ and $t' \stackrel{w}{\rightarrow}$,
2. $p \,\mathsf{must}\, t$ if whenever $(p \mid t =) p_0 \mid t_0 \stackrel{\tau}{\rightarrow} p_1 \mid t_1 \stackrel{\tau}{\rightarrow} \ldots$ is a computation then (a) $\exists n \geq 0$ such that $t_n \stackrel{w}{\rightarrow}$, and (b) if $(p_k \mid t_k)\!\uparrow$ then $t_{k'} \stackrel{w}{\rightarrow}$, for some $k' \leq k$.

Definition 2.11 For processes p, q over the language G as in the previous definition we have

$$\begin{array}{lcl} p \sqsubseteq_{\mathsf{may}} q & \equiv & \forall \text{ tests } t.\ p \,\mathsf{may}\, t \ \text{ implies } \ q \,\mathsf{may}\, t, \\ p \sqsubseteq_{\mathsf{must}} q & \equiv & \forall \text{ tests } t.\ p \,\mathsf{must}\, t \ \text{ implies } \ q \,\mathsf{must}\, t, \\ p \sqsubseteq q & \equiv & p \sqsubseteq_{\mathsf{may}} q \ \wedge \ p \sqsubseteq_{\mathsf{must}} q. \end{array}$$

It is easily checked that $\sqsubseteq_{\mathsf{may}}$, $\sqsubseteq_{\mathsf{must}}$ and $\sqsubseteq$ are preorders. Testing equivalence, written as $\approx$, is defined by $p \approx q \equiv p \sqsubseteq q \wedge q \sqsubseteq p$. The equivalences arising from $\sqsubseteq_{\mathsf{may}}$ and $\sqsubseteq_{\mathsf{must}}$, denoted by $\approx_{\mathsf{may}}$ and $\approx_{\mathsf{must}}$, are defined correspondingly.

Instead of working directly with $\sqsubseteq_{\mathsf{may}}$, $\sqsubseteq_{\mathsf{must}}$ and $\sqsubseteq$, we will employ their alternative characterisations as described in [NH84, Hen88].

Definition 2.12 With the notation as in the previous definition

$$\begin{array}{lcl} p \ll_{\mathsf{may}} q & \equiv & \mathcal{L}(p) \subseteq \mathcal{L}(q), \\ p \ll_{\mathsf{must}} q & \equiv & \forall s \in \mathsf{Act}^*.\ p \Downarrow s \text{ implies} \\ & & \qquad \text{(a)} \ \ q \Downarrow s \ \wedge \\ & & \qquad \text{(b)} \ \ \forall q'.\ q \stackrel{s}{\Rightarrow} q' \text{ implies } \exists p'.\ p \stackrel{s}{\Rightarrow} p' \wedge \mathcal{S}(p') \subseteq \mathcal{S}(q'), \\ p \ll q & \equiv & p \ll_{\mathsf{may}} q \ \wedge \ p \ll_{\mathsf{must}} q. \end{array}$$

Note that our definition of $\ll$ is almost the same as the corresponding Definition 4.4.5 in [Hen88]. The only difference is that instead of condition (b) the condition $\mathcal{A}(q,s) \subset\subset \mathcal{A}(p,s)$ is used in [Hen88]. We remind that $\mathcal{A}(q,s)$ is $\{S(p') \mid p \stackrel{s}{\Rightarrow} p'\}$ and $\mathcal{A} \subset\subset \mathcal{B}$ if for every $A \in \mathcal{A}$ there is some $B \in \mathcal{B}$ such that $B \subseteq A$. It is easily checked that condition (b) and $\mathcal{A}(q,s) \subset\subset \mathcal{A}(p,s)$ are logically equivalent.

The equivalences based on $\ll_{\mathsf{may}}, \ll_{\mathsf{must}}$ and $\ll$, denoted by $\simeq_{\mathsf{may}}, \simeq_{\mathsf{must}}$ and $\simeq$, are defined correspondingly as $\approx_{\mathsf{may}}, \approx_{\mathsf{must}}$ and $\approx$.

Finally, we have our version of a characterisation result.

Theorem 2.13 $p \sqsubseteq q$ if and only if $p \ll q$, for all processes p and q over any self-testing τDeS process language.

The proof of this theorem follows similar lines as the proofs of characterisation results proposed by the De Nicola and Hennessy, in particular the proofs in [Hen88] of Lemma 4.4.9, which corresponds to the 'only if' part of Theorem 2.13, and Lemma 4.4.13, which corresponds to the 'if' part of Theorem 2.13. A careful examination of the proof of Lemma 4.4.9 will show that it does not depend on the process language considered. Hence, the proof can be easily adapted for any τDeS language. The same is true of Lemma 4.4.13 provided that the following two conditions are satisfied by the considered process language. Firstly, the language has to have appropriate operators to encode the essential tests. This condition is trivially satisfied by self-testing τDeS languages. Secondly, a certain finite branching condition needs to be satisfied by the derived transition system of the considered τDeS language. As we have seen in Example 2.6 not all τDeS processes are finite branching. But, the proof of Lemma 4.4.13 (in particular, the proof of auxiliary Lemma 4.4.12) only requires τ-convergent processes to be finite branching. More precisely, the following property must hold: for all processes p in the considered language, if $p\Downarrow t$ and $s \leq t$ then $\{q \mid p' \Rightarrow q\}$ is finite, for all $p' \in \mathcal{D}(p,s)$. Since for any τDeS language its basic transition system is finite branching it is easy to show that the above property is indeed valid. Hence, $\ll$ can be used instead of $\sqsubseteq$ in every self-testing τDeS process language.

In the rest of the paper we will need the following notational agreements. We will use $\mathsf{Test}(G)$ to denote the quotient algebra of elements of $\mathbb{T}(\Sigma_G)$ factored by $\ll$ and write $\mathsf{Test}(G) \models P \sqsubseteq Q$ to mean $\rho(P) \ll \rho(Q)$, for all P and Q and all ground substitutions ρ over G. Since we will talk about axioms which are preserved by taking disjoint extensions of τDeS languages we will need $\mathsf{TEST}(G)$: a class of all algebras $\mathsf{Test}(G')$, where G' is a disjoint extension of G. For $P, Q \in \mathbb{T}(\Sigma_G)$ we have

$$\mathsf{TEST}(G) \models P \sqsubseteq Q \quad \text{iff} \quad (\forall G'.G \leq G' \text{ implies } \mathsf{Test}(G') \models P \sqsubseteq Q).$$

3 Congruence Theorem

In this section we prove the congruence theorem which is the foundation of our axiomatisation algorithm. Then, we briefly comment on congruence results for other formats with a special treatment of silent actions.

Theorem 3.1 (Congruence) $\ll$ is a precongruence for all τDeS languages.

Its proof, like the proofs of congruence theorems for other weak equivalences and classes of process languages [Blo95a, Uli94], is rather long, technical and relies on checking many cases. Before we present the details of the proof we will show that $\ll$ is not in general preserved by non-τDeS operators. We consider only two classes of such operators: the operators defined by rules with 'copying' and the operators defined by rules with negative premises.

Example 3.2 Consider the operator $\theta_{b>a}$ which gives a priority to action b over action a. The defining rules, which clearly are not τDeS, are as follows:

$$\frac{X \xrightarrow{b} X'}{\theta_{b>a}(X) \xrightarrow{b} \theta_{b>a}(X')}\ b \neq a \qquad\qquad \frac{X \xrightarrow{a} X' \quad X \overset{b}{\not\rightarrow}}{\theta_{b>a}(X) \xrightarrow{a} \theta_{b>a}(X')}$$

Let $p = a(b \boxplus a\Omega) \boxplus aa$ and $q = a(b \boxplus a\Omega) \boxplus a(a \boxplus a\Omega)$. Clearly $p \simeq q$ but not $\theta_{b>a}(p) \simeq \theta_{b>a}(q)$. For $\theta_{b>a}(p)\Downarrow aa$, but $\theta_{b>a}(q)\Uparrow aa$ since $\theta_{b>a}(q) \overset{aa}{\not\Rightarrow} \Omega$.

Example 3.3 Consider operators f and f' defined by the following rules.

$$\frac{X \stackrel{a}{\rightarrow} X'}{f(X) \stackrel{\tau}{\rightarrow} f'(X', X')}\ b \neq a \qquad\qquad \frac{X \stackrel{b}{\rightarrow} X' \quad Y \stackrel{c}{\rightarrow} Y'}{f'(X, Y) \stackrel{c}{\rightarrow} \mathbf{0}}$$

Let $p = ab \boxplus ac$ and $q = a(b \oplus c)$. Clearly $p \simeq q$ but not $f(p) \simeq f(q)$: $f(q) \stackrel{c}{\Rightarrow}$ but $f(X) \stackrel{c}{\not\Rightarrow}$.

Now, we turn the attention to the proof of Theorem 3.1. We will introduce a number of helpful notions. In order to improve the analysis of the behavior of processes in arbitrary contexts we will employ *ruloids* and *Ruloid Theorem* for τDeS languages—these are based on Bloom's work on ruloids for GSOS languages [Blo90a, Blo95a].

Ruloids are like τDeS rules except that the consequent has the form $C[\boldsymbol{X}] \stackrel{\alpha}{\rightarrow} C'[\boldsymbol{Y}]$, where both $C[\boldsymbol{X}]$ and $C'[\boldsymbol{Y}]$ are contexts and the elements of $\boldsymbol{Y}$ are defined as in Definition 2.1. For example, the following is a ruloid for the context $(X \boxplus b.Y)\backslash a$, where $\backslash a$ is the usual CSP hiding operator (it will be formally defined in Section 4.1), and action b not equal to a

$$\frac{X \stackrel{b}{\rightarrow} X'}{(X \boxplus b.Y)\backslash a \stackrel{b}{\rightarrow} X'\backslash a} \tag{2}$$

Given a τDeS language G, Ruloid Theorem says that given a term $C[\boldsymbol{p}]$ and an action $\alpha \in \mathsf{Act} \cup \{\tau\}$ it is possible to construct a set ruloids which describe all the circumstances when $C[\boldsymbol{p}]$ performs α.

Theorem 3.4 Let G be a τDeS language, $C[\boldsymbol{X}]$ a context over Σ and α an action in $\mathsf{Act} \cup \{\tau\}$. Then, there exists a finite set $\mathcal{R}$ of ruloids for $C[\boldsymbol{X}]$ and α such that

1. the consequent of each $r \in \mathcal{R}$ has the form $C[\boldsymbol{X}] \stackrel{\alpha}{\rightarrow} D[\boldsymbol{Y}]$, for some context $D[\boldsymbol{Y}]$ with $\boldsymbol{Y}$ as in Definition 2.1,

2. $\mathcal{R}$ explains all α moves of all instances of $C[\boldsymbol{X}]$. Namely, if $\rho(C[\boldsymbol{X}] \stackrel{\alpha}{\rightarrow} D[\boldsymbol{Y}])$ is valid, for some Σ ground substitution ρ, then (a) there exists $r \in \mathcal{R}$ such that $\rho(cons(r)) = \rho(C[\boldsymbol{X}] \stackrel{\alpha}{\rightarrow} D[\boldsymbol{Y}])$ and (b) $\rho(ante(r))$ is valid.

3. all ruloids in $\mathcal{R}$ are valid. Namely, for any $r \in \mathcal{R}$ and Σ ground substitution ρ, if $\rho(ante(r))$ is valid then $\rho(cons(r))$ is valid.

For example, the set of ruloids for $(X \boxplus b.Y)\backslash a$ and action b as above consists of the ruloid (2) and the ruloid $(X \boxplus b.Y)\backslash a \stackrel{b}{\rightarrow} Y \backslash a$. In general, there are also *τ-ruloids*, the notion corresponding to τ-rules. In our example, there is only one τ-ruloid since only the argument X is active (active arguments of ruloids are defined in the same way as for rules):

$$\frac{X \stackrel{\tau}{\rightarrow} X'}{(X \boxplus b.Y)\backslash a \stackrel{\tau}{\rightarrow} (X' \boxplus b.Y)\backslash a}$$

Also it will be beneficial to use a certain notation for ruloids similar to ruloids in *ϵ-presentation* [Blo95a], which are described first. A ruloid for $C[\boldsymbol{X}]$ is ϵ-presented if it has one antecedent $X_i \stackrel{\mu}{\rightarrow} X_i'$ for each variable in $\boldsymbol{X}$, where $\mu \in A \cup \{\tau\} \cup \{\epsilon\}$ and $\epsilon \notin \mathsf{Act}$. An ordinary ruloid for $C[\boldsymbol{X}]$ and α, call it r, can be made into a corresponding ϵ-presented ruloid by adding to antecedents the transition $X_i \stackrel{\epsilon}{\rightarrow} X_i'$ for each variable X_i of $\boldsymbol{X}$ which does not appear in the

antecedents of r. The antecedent $X_i \stackrel{\epsilon}{\to} X_i'$ represents a null transition: when a ruloid with such an antecedent is used then X_i and X_i' are bound to the same term. For example, the following is the set of ϵ-presented ruloids for $(X \boxplus b.Y)\backslash a$ and b, where because of the form of antecedents the second ruloid could be called 'axiom' ruloid.

$$\frac{X \stackrel{b}{\to} X' \quad Y \stackrel{\epsilon}{\to} Y'}{(X \boxplus b.Y)\backslash a \stackrel{b}{\to} X'\backslash a} \qquad \frac{X \stackrel{\epsilon}{\to} X' \quad Y \stackrel{\epsilon}{\to} Y'}{(X \boxplus b.Y)\backslash a \stackrel{b}{\to} Y'\backslash a}$$

The first of the above ϵ-presented ruloids and the ruloid for $X\backslash a$ and action c not equal to a can be written using the following abstract notation

$$\frac{X \stackrel{b}{\to} X' \quad Y \stackrel{\epsilon}{\to} Y'}{C[X,Y] \stackrel{b}{\to} C'[X',Y']} \qquad \frac{X \stackrel{c}{\to} X' \quad Y \stackrel{\epsilon}{\to} Y'}{C'[X,Y] \stackrel{c}{\to} C'[X',Y']}$$

where $C[X,Y]$ and $C'[X,Y]$ stand for the contexts $(X \boxplus b.Y)\backslash a$ and $X\backslash a$ respectively. Note that the second ruloid has antecedents for both X and Y although $C'[X,Y]$ actually stands for $X\backslash a$ in which Y does not appear. Note that since no process variable can be copied in De Simone rules or ruloids, i.e. used more than once in the target of the consequent, it is correct to use the same number of process variables in both the source and target of the consequent. This abstract notation for ϵ-presented ruloids will be extensively used in the proof of the congruence theorem.

Proof of Theorem 3.1. Let $G = (\Sigma, \mathsf{Act} \cup \{\tau\}, R)$ be a τDeS process language. Assume $\boldsymbol{p} \ll \boldsymbol{q}$, where $\boldsymbol{p}$ and $\boldsymbol{q}$ are vectors of closed Σ terms. We need to prove $C[\boldsymbol{p}] \ll C[\boldsymbol{q}]$, for any n-ary Σ context $C[\boldsymbol{X}]$. The last can be subdivided, by Definition 2.12, into three parts:

1. $\mathcal{L}(C[\boldsymbol{p}]) \subseteq \mathcal{L}(C[\boldsymbol{q}])$,
2. $\forall s \in \mathsf{Act}^*.\ C[\boldsymbol{p}]\Downarrow s$ implies $C[\boldsymbol{q}]\Downarrow s$,
3. $\forall s \in \mathsf{Act}^*.\ C[\boldsymbol{p}]\Downarrow s$ implies $(\forall D[\boldsymbol{X}], \boldsymbol{q}'.\ C[\boldsymbol{q}] \stackrel{s}{\Rightarrow} D[\boldsymbol{q}']$ implies
$\exists \boldsymbol{p}'.(C[\boldsymbol{p}] \stackrel{s}{\Rightarrow} D[\boldsymbol{p}'] \ \wedge\ \mathcal{S}(D[\boldsymbol{p}']) \subseteq \mathcal{S}(D[\boldsymbol{q}']))$.

Part 1. Let $\alpha_1 \ldots \alpha_k \in (\mathsf{Act} \cup \{\tau\})^*$ be such that after removing actions τ we obtain $s \in \mathsf{Act}^*$. We will show that $C[\boldsymbol{p}] \stackrel{s}{\Rightarrow}$ implies $C[\boldsymbol{q}] \stackrel{s}{\Rightarrow}$. $C[\boldsymbol{p}] \stackrel{\alpha_1}{\to} \ldots \stackrel{\alpha_k}{\to}$ can be represented as $C[\boldsymbol{p}] \equiv C_0[\boldsymbol{p}_0] \stackrel{\alpha_1}{\to} \ldots \stackrel{\alpha_k}{\to} C_k[\boldsymbol{p}_k]$, for some n-ary contexts $C_j[\boldsymbol{X}]$ and n-ary vectors $\boldsymbol{p}_j$. Note that some contexts $C_j[\boldsymbol{X}]$ may in fact stand for open terms where not all elements of $\boldsymbol{X}$ appear. For each $0 \leq j < k$, $C_j[\boldsymbol{p}_j] \stackrel{\alpha_{j+1}}{\longrightarrow} C_{j+1}[\boldsymbol{p}_{j+1}]$ gives rise, by Ruloid Theorem and the above discussion on the notation for ϵ-presented ruloids, to the following:

$$p_{1j} \stackrel{\mu_{1j+1}}{\longrightarrow} p_{1j+1}, \quad \ldots \quad , p_{nj} \stackrel{\mu_{nj+1}}{\longrightarrow} p_{nj+1}, \tag{3}$$

where each $\mu_{ij+1} \in \mathsf{Act} \cup \{\tau\} \cup \{\epsilon\}$. Sequence (3) specifies the behavior of the components p_{ij} of $\boldsymbol{p}_j$, where $1 \leq i \leq n$, which produces $C_j[\boldsymbol{p}_j] \stackrel{\alpha_{j+1}}{\longrightarrow} C_{j+1}[\boldsymbol{p}_{j+1}]$.

Sequence (3) can be abbreviated by $\boldsymbol{p}_j \stackrel{A_{j+1}}{\to} \boldsymbol{p}_{j+1}$, where A_{j+1} is the column vector of μ_{ij+1}'s. We can say more about the chemistry of vectors A_l. When A_l contains no visible action then it consists either of a single τ and $n-1$ labels ϵ or it consists of n labels ϵ. These vectors arise when applying ϵ-presented τ-ruloids and 'axiom' ruloids respectively. The first of these vectors is called a *τ-column*. Putting the individual transitions of all $\boldsymbol{p}_j$ together we

obtain $\boldsymbol{p}_0 \stackrel{A_1}{\rightarrow} \ldots \stackrel{A_k}{\rightarrow} \boldsymbol{p}_k$, or abbreviating even further $\boldsymbol{p}_0 \stackrel{\boldsymbol{A}}{\rightarrow} \boldsymbol{p}_k$, where $\boldsymbol{A}$ is a vector of column vectors A_j. In other words, $\boldsymbol{A}$ is an $n \times k+1$ matrix. Note that the ith row of $\boldsymbol{A}$ is $\mu_{i1} \ldots \mu_{ik}$ and it represents the behavior the ith component p_{i0} of $\boldsymbol{p}_0$. Let s_i be $\mu_{i1} \ldots \mu_{ik}$ with all labels τ and ϵ removed, thus $p_{i0} \stackrel{s_i}{\Rightarrow}$. Performing the same for other components of $\boldsymbol{p}_0$ we obtain $\boldsymbol{p}_0 \stackrel{\boldsymbol{s}}{\Rightarrow}$, where $\boldsymbol{s}$ is the column vector $s_1, \ldots, s_n$. Since $\boldsymbol{p} \ll \boldsymbol{q}$ and $\boldsymbol{p} \equiv \boldsymbol{p}_0$ we have $\boldsymbol{q}(\equiv \boldsymbol{q}_0) \stackrel{\boldsymbol{s}}{\Rightarrow}$.

If we could repeat the above argument in reverse for $\boldsymbol{q}_0$, that is from $\boldsymbol{q}_0 \stackrel{\boldsymbol{s}}{\Rightarrow}$ deduce $\boldsymbol{q}_0 \stackrel{\boldsymbol{A}}{\rightarrow}$, then we would derive $C[\boldsymbol{q}_0] \stackrel{s}{\Rightarrow}$ as required. However, $\boldsymbol{q}_0 \stackrel{\boldsymbol{A}}{\rightarrow}$ is not valid in general since the components of $\boldsymbol{q}_0$ and $\boldsymbol{p}_0$ may vary in the number of actions τ they can perform and the places of their occurrence. But, it can be easily shown that given $\boldsymbol{q}_0 \stackrel{\boldsymbol{s}}{\Rightarrow}$ (and keeping $\boldsymbol{A}$ in mind) we can construct an $n \times l+1$ matrix $\boldsymbol{B}$ such that it only differs from $\boldsymbol{A}$ in the number and positions of τ-columns. Having this we can deduce $C[\boldsymbol{q}_0] \stackrel{\nu_1}{\rightarrow} \ldots \stackrel{\nu_l}{\rightarrow}$, where the sequence $\nu_1 \ldots \nu_l$ differs from $\mu_1 \ldots \mu_k$ in the number and places of occurrences of actions τ. Hence, $(C[\boldsymbol{q}] \equiv) C[\boldsymbol{q}_0] \stackrel{s}{\Rightarrow}$ follows.

Part 2. We will show that for every $s \in \mathsf{Act}^*$ if $C[\boldsymbol{q}] \Uparrow s$ then $C[\boldsymbol{p}] \Uparrow s$. Assume $C[\boldsymbol{q}] \Uparrow s$, for some s. This means that for some minimal $t \leq s$, $C'[\boldsymbol{X}]$ and $\boldsymbol{q}'$ such that $C[\boldsymbol{q}] \stackrel{t}{\Rightarrow} C'[\boldsymbol{q}']$ we either have $C'[\boldsymbol{q}'] \uparrow$ or $C'[\boldsymbol{q}'] \stackrel{\tau^\omega}{\rightarrow}$. (The minimality of t means that for all proper subsequences t' of t we have $C[\boldsymbol{q}] \Downarrow t'$). We only consider the first case $C'[\boldsymbol{q}'] \uparrow$, the proof of the second is similar. $C'[\boldsymbol{q}'] \uparrow$ implies there is a ruloid for $C'[\boldsymbol{X}]$ and some α which has the antecedent $X_i \stackrel{\mu}{\rightarrow} X_i'$ with $\mu \neq \epsilon$ such that $q_i' \uparrow$. From $C[\boldsymbol{q}] \stackrel{t}{\Rightarrow} C'[\boldsymbol{q}']$ we obtain, using Ruloid Theorem, $q_i \stackrel{t_i}{\Rightarrow} q_i'$ for some $t_i \in \mathsf{Act}^*$ and thus $q_i \Uparrow t_i$. Since $\boldsymbol{p} \ll \boldsymbol{q}$ we easily obtain $p_i \Uparrow t_i$. Thus, there exist p_i' and $t_i' \leq t_i$, such that $p_i \stackrel{t_i'}{\Rightarrow} p_i'$ and $p_i' \uparrow$ or $p_i' \stackrel{\tau^\omega}{\rightarrow}$. Using Ruloid Theorem for $C[\boldsymbol{X}]$ and the other contexts involved in $C[\boldsymbol{q}] \rightarrow \ldots \rightarrow C'[\boldsymbol{q}']$ we obtain: (a) $C[\boldsymbol{p}] \stackrel{t'}{\Rightarrow} C^*[\boldsymbol{p}']$, where $C^*[\boldsymbol{X}]$ is some context which appears in $C[\boldsymbol{q}] \rightarrow \ldots \rightarrow C'[\boldsymbol{q}']$, (b) the ith argument of $C^*[\boldsymbol{X}]$ is active and the ith component of $\boldsymbol{p}'$ is p_i' and (c) $t' \leq t$. Thus, $C^*[\boldsymbol{p}'] \uparrow$ or $C^*[\boldsymbol{p}'] \stackrel{\tau^\omega}{\rightarrow}$. This implies $C[\boldsymbol{p}] \Uparrow t'$ and so $C[\boldsymbol{p}] \Uparrow s$.

Part 3. Assume $C[\boldsymbol{p}] \Downarrow s$ and $C[\boldsymbol{q}] \stackrel{s}{\Rightarrow} D[\boldsymbol{q}']$, for some $s \in \mathsf{Act}^*$, $\boldsymbol{q}'$ and $D[\boldsymbol{X}]$. We need to show that (a) there exists $\boldsymbol{p}'$ such that $C[\boldsymbol{p}] \stackrel{s}{\Rightarrow} D[\boldsymbol{p}']$ and (b) $\mathcal{S}(D[\boldsymbol{p}']) \subseteq \mathcal{S}(D[\boldsymbol{q}'])$. Using Ruloid Theorem and the technique from Part 1 we establish the behavior of $\boldsymbol{q}$ which gives rise to $C[\boldsymbol{q}] \stackrel{s}{\Rightarrow} D[\boldsymbol{q}']$. We write it as $\boldsymbol{q} \stackrel{\boldsymbol{s}}{\Rightarrow} \boldsymbol{q}'$, where $\boldsymbol{s}$ is a vector of $s_i \in \mathsf{Act}^*$. By Part 2 we obtain $C[\boldsymbol{q}] \Downarrow s$ and thus, overloading the notation, $\boldsymbol{q} \Downarrow \boldsymbol{s}$. $C[\boldsymbol{p}] \Downarrow s$ means that for all $t \leq s$ if $C[\boldsymbol{p}] \stackrel{t}{\Rightarrow}$ then, by Ruloid Theorem, $\boldsymbol{p} \stackrel{\boldsymbol{t}}{\Rightarrow}$ and $\boldsymbol{p} \Downarrow \boldsymbol{t}$. Clearly $\boldsymbol{t} \leq \boldsymbol{s}$, i.e. $t_i \leq s_i$ for all i. We wish to prove that $\boldsymbol{p} \stackrel{\boldsymbol{s}}{\Rightarrow}$. Assume the opposite: $p_i \stackrel{t_i}{\Rightarrow}$ for all $t_i \leq s_i$ described above and $p_i \stackrel{s_i}{\not\Rightarrow}$. The last gives $p_i \Downarrow s_i$ which, together with $p_i \ll q_i$ and $q_i \stackrel{s_i}{\Rightarrow}$, produces $p_i \stackrel{s_i}{\Rightarrow}$: a contradiction. Thus, for all i and p_i' we get $p_i \stackrel{s_i}{\Rightarrow} p_i'$. Hence, $\boldsymbol{p} \stackrel{\boldsymbol{s}}{\Rightarrow} \boldsymbol{p}'$. Similarly as in Part 1, we deduce $C[\boldsymbol{p}] \stackrel{s}{\Rightarrow} D[\boldsymbol{p}']$ which is the required part (a). Since for each i we have $\mathcal{S}(p_i') \subseteq \mathcal{S}(q_i')$ and since the behavior of $D[\boldsymbol{p}']$ and $D[\boldsymbol{q}']$ depends on the actions of $\boldsymbol{p}'$ and $\boldsymbol{q}'$ we deduce that if $D[\boldsymbol{p}'] \stackrel{a}{\Rightarrow}$, for some a, then $D[\boldsymbol{q}'] \stackrel{a}{\Rightarrow}$. Thus, $\mathcal{S}(D[\boldsymbol{p}']) \subseteq \mathcal{S}(D[\boldsymbol{q}'])$ as required. □

As an immediate consequence we have

Corollary 3.5 $\sqsubseteq$ is a precongruence in all self-testing τDeS languages.

Concluding this section we wish to underline how simple and intuitive τDeS operators are. They preserve testing preorder as well as many other weak equivalences and preorders. Since

Figure 1: Axiomatization of Testing Preorder.

$E1$	$X \boxplus \mathbf{0} = X$		
$E2$	$X \boxplus X = X$	$I2$	$X \oplus X = X$
$E3$	$X \boxplus Y = Y \boxplus X$	$I3$	$X \oplus Y = Y \oplus X$
$E4$	$X \boxplus (Y \boxplus Z) = (X \boxplus Y) \boxplus Z$	$I4$	$X \oplus (Y \oplus Z) = (X \oplus Y) \oplus Z$
$C1$	$a.X \boxplus a.Y = a.X \oplus a.Y$		
$C2$	$a.X \boxplus a.Y = a.(X \oplus Y)$	$C3$	$X \oplus Y \sqsubseteq X \boxplus Y$
$D1$	$X \boxplus (Y \oplus Z) = (X \boxplus Y) \oplus (X \boxplus Z)$	$\Omega 1$	$\Omega \sqsubseteq X$
$D2$	$X \oplus (Y \boxplus Z) = (X \oplus Y) \boxplus (X \oplus Z)$	$\Omega 2$	$X \boxplus \Omega \sqsubseteq X \oplus \Omega$

the τDeS format is a subformat of the ISOS format τDeS operators preserve refusal simulation preorder [Uli92, Uli94] and delay bisimulation [Uli94]. It is thus somewhat surprising that such well behaved operators do *not* preserve the standard version of weak bisimulation as defined, for example, in [Mil89a]. It was argued in [vG90] that useful operators such as *action refinement* operators do not preserve weak bisimulation. In [PU96] a simple τDeS version of action refinement operator is defined which does not preserve weak bisimulation. However, there is a format (similar in the expressive power to the ISOS format) proposed by Bloom [Blo95a] which preserves weak bisimulation, but in our opinion it suffers from a complicated definition. Bloom [Blo95a] also proposes formats of rules for branching bisimulation and the 'rooted' versions of both weak and branching bisimulations.

4 Finite Axiom Systems for τDeS Languages

In this section we show how to generate an axiomatisation of an arbitrary τDeS language. The strategy is as follows. Given a τDeS language G we disjointly extend it with the basic process language B to obtain G'. We propose sound laws (axiom schemas) for all G' operators which are not in B. The axiom system for G (in fact for G') consists of the instances of these laws together with the axiom system for B. If G is well-founded then the proposed axiom system is sufficient to rewrite arbitrary process terms over G into equivalent terms over B. This is possible because well-founded terms over G can be put into head normal form by using the axiom system. Hence, the problem of completeness for well-founded processes over G is reduced to the already solved completeness problem for B. In order to achieve completeness for a certain subclass of not well-founded processes it is enough to extend the axiom system with one infinitary induction rule.

We start by recalling the axiom system T_{B} for $\ll$ over the language B. T_{B} consists of equations and inequations given in Figure 1 as well as the usual properties of precongruences: reflexivity, transitivity, substitutivity (if $\boldsymbol{p} \sqsubseteq \boldsymbol{q}$ then $f(\boldsymbol{p}) \sqsubseteq f(\boldsymbol{q})$), instantiation and inequations as, for example, in Figure 1.2 in [Hen88]. We will write $\mathrm{T} \vdash P \sqsubseteq Q$, $(P = Q)$ to mean that $P \sqsubseteq Q$, $(P = Q)$ can be derived from the axiomatic system T.

Lemma 4.1 [Hen88] $\mathrm{T}_{\mathsf{B}} \vdash P \sqsubseteq Q$ if and only if $\mathsf{Test}(\mathsf{B}) \vDash P \sqsubseteq Q$.

Since $\boxplus$ and $\oplus$ are associative the summation notations $\sum_{i \in I} X_i$ and $\bigoplus_{j \in J \neq \emptyset} Y_j$ are justified.

Finally, we remind the definitions of *head normal forms*, hnf for short, from [NH84, Hen88].

Definition 4.2 Let $\mathcal{A}$ be a finite non-empty set of subsets of Act. We say that $\mathcal{A}$ is *saturated* if and only if (a) $A, B \in \mathcal{A}$ implies $A \cup B \in \mathcal{A}$ and (b) $A, B \in \mathcal{A}$, $A \subseteq C \subseteq B$ implies $C \in \mathcal{A}$. Head normal forms are defined as follows: $\mathbf{0}$ is in hnf; if $\mathcal{A}$ is saturated and $\mathcal{A} \neq \{\emptyset\}$ then $\bigoplus_{A \in \mathcal{A}} \sum_{a \in A} aP_a$ is in hnf; if A is a subset of Act then $\sum_{a \in A} aP_a \boxplus \Omega$ is in hnf.

Hence, Ω, $(ap \boxplus bq) \oplus cr$ and $ap \boxplus \Omega$ are in hnf but $(ap \boxplus \Omega) \oplus cr$ and $ap \boxplus aq$ are not. An important property of terms in hnf is that they have unique a descendants. For example, $(ap + bq) \oplus ar$ is in hnf if and only if $p \equiv r$. This property is vital for obtaining *finite* axiomatisations of $\ll$. Terms can be put in hnf using system T_{B} and the following derived axioms.

$$
\begin{array}{lrcl}
Der1 & X \oplus \Omega & = & X \boxplus \Omega \\
Der2 & X \oplus (Y \boxplus \Omega) & = & X \boxplus Y \boxplus \Omega \\
Der3 & X \oplus Y & = & X \oplus Y \oplus (X \boxplus Y) \\
Der4 & X \oplus (X \boxplus Y \boxplus Z) & = & X \oplus (X \boxplus Y) \oplus (X \boxplus Y \boxplus Z) \\
Der5 & (aX \boxplus Z) \oplus (aY \boxplus V) & = & (a(X \oplus Y) \boxplus Z) \oplus (a(X \oplus Y) \ \boxplus V)
\end{array}
$$

4.1 Axiomatisation Algorithm

This subsection introduces a number of *laws* or axiom schemas which allow rewriting process terms into hnf. We define an algorithm which uses these laws to generate sound axiomatisations of arbitrary τDeS languages. The usefulness of our algorithm is illustrated by generating axiomatisations of the language B extended with the CSP hiding operator and the CCS parallel composition operator.

We will need to distinguish two subclasses of τDeS operators, namely *distinctive* [ABV92] and τ*-introducing* operators [Uli95]. A τDeS operator (f, n) is *distinctive* if for each argument i either all rules for f, which are not τ-rules, test i or none of them does, and also for each pair of different non-τ-rules for f there is an argument for which both rules have the same antecedents, but with a different action. A τDeS operator (f, n) is τ*-introducing* if τ is the action of one of its De Simone rules.

Lemma 4.3 [Uli95] (**Action Laws**). Let G be a self-encoding τDeS process language. Let (f, n) be a distinctive operator of G with a rule of the form (1). Also assume $P_i \equiv a_i.X_i'$ for $i \in I$ and $P_i \equiv X_i$ otherwise. If $\alpha \in \mathsf{Act}$ then, for an appropriate $\boldsymbol{P'}$, we have

$$\mathsf{TEST}(G) \vDash f(\boldsymbol{P}) = \alpha.C[\boldsymbol{P'}] \tag{4}$$

else (i.e. f is τ-introducing), for some $\boldsymbol{P'}$, we have

$$\mathsf{TEST}(G) \vDash f(\boldsymbol{P}) = C[\boldsymbol{P'}]. \tag{5}$$

(**Inaction Laws**). Let (f, n) be an operator of G such that for each rule for f of the form (1) there is an index $i \in I$ such that either $P_i \equiv \mathbf{0}$ or $P_i \equiv a.X_i'$, for some $a \neq a_i$. Then

$$\mathsf{TEST}(G) \vDash f(\boldsymbol{P}) = \mathbf{0}. \tag{6}$$

(**Divergence Laws**). Let (f, n) be an operator of G. Suppose that there is a rule for f of the form (1) such that for some $i \in I$ we have $P_i \equiv \Omega$, and, for all other rules for f of the form

$$\frac{\{\ X_i \xrightarrow{a_i} X_i'\ \}_{i \in I'}}{f(X_1, \ldots, X_n) \xrightarrow{\alpha} C'[\boldsymbol{Y}],}$$

there is an index $i \in I'$ such that $P_i \equiv \mathbf{0}$, $P_i \equiv \Omega$ or $P_i \equiv a.X'_i$, for some $a \neq a_i$. Then

$$\mathsf{TEST}(G) \models f(\boldsymbol{P}) \sqsubseteq \Omega. \tag{7}$$

Since we also have $\Omega \sqsubseteq X$ (axiom $\Omega 1$), for all X, we could substitute (7) by $f(\boldsymbol{P}) = \Omega$.

Proof. The above laws are sound with respect to refusal simulation preorder [Uli95] and since refusal simulation preorder refines $\ll$ we obtain that the laws are also sound for $\ll$. □

Next, we define the distributivity over $\oplus$ and $\boxplus$ laws.

Lemma 4.4 (Distributivity over $\oplus$ laws). Let G be a self-encoding τDeS process language. Let (f, n) be an operator of G such that its ith argument is active. If $X_i \equiv X \oplus Y$ then

$$\mathsf{TEST}(G) \models f(\boldsymbol{X}) = f(\boldsymbol{X})[X/X_i] \ \oplus \ f(\boldsymbol{X})[Y/X_i]. \tag{8}$$

It is important to note that $\oplus$ laws hold, unlike the corresponding laws in [Uli95] and the distributivity over $+$ laws in [ABV92], without the requirement that each rule for f has an antecedent for the ith argument. The proof of Lemma 4.4 can be found in Appendix A.

Notice that by left-to-right application of the derived axiom $X \boxplus \Omega = X \oplus \Omega$ and the instances of distributivity over $\oplus$ laws a process $f(\boldsymbol{p})$ with active arguments in hnf can be rewritten as $\bigoplus_I f(\boldsymbol{p}_i)$, where all active components of each $\boldsymbol{p}_i$ are either Ω or of the form $\sum_A a.p_a$, where $A \subseteq \mathsf{Act}$. This suggests that the distributivity over $\boxplus$ laws need only be given for $f(\boldsymbol{X})$ with all active arguments of the form $\sum_A a.X_a$ or Ω. This is, as will be explained later, one of the two reasons why our axiomatisations are finite.

We will use the following notation to express the form of active arguments. Let $\boldsymbol{A}$ be an n-ary vector whose i-th element A_i is $*$ if $i \notin \mathit{active}(f)$ and it is $*$ or $A_i \subseteq \mathsf{Act}$ otherwise. Then, $\boldsymbol{AX}$ (as in $f(\boldsymbol{AX})$ below) denotes an n-ary vector of open terms such that its i-th element, written as $A_i\boldsymbol{X}_i$, is any term if $i \notin \mathit{active}(f)$. Otherwise $A_i\boldsymbol{X}_i$ is Ω if $A_i = *$ and $\sum_{a_j \in A_i} a_j.X_{ij}$ if $A_i \subseteq \mathsf{Act}$. In the last two cases we will write $(A_i\boldsymbol{X}_i)_j$ to denote Ω if $A_i = *$ and $a_j.X_{ij}$ otherwise. Moreover, given trigger χ we say that χ *fires* from $\boldsymbol{A}$ (or $\chi \in \boldsymbol{A}$) if whenever χ_i, the ith element of χ, is not $*$ then $A_i \neq *$ and $\chi_i \in A_i$. Also, let $\Omega(f(\boldsymbol{AX}))$ be true if one of the active arguments of $f(\boldsymbol{AX})$ is Ω.

Lemma 4.5 (Distributivity over $\boxplus$ laws) Let (f, n) be an operator of a self-encoding τDeS process language G. Suppose that i is such that each rule for f has an antecedent for X_i. Then the following laws are valid in $\mathsf{TEST}(G)$:

$$\begin{aligned} f(\boldsymbol{AX}) \ &= \ \sum_j f(\boldsymbol{AX})[(A_i\boldsymbol{X}_i)_j/A_i\boldsymbol{X}_i] \\ &\quad \text{if } \ \Omega(f(\boldsymbol{AX})) \ \vee \\ &\qquad \neg\Omega(f(\boldsymbol{AX})) \wedge \forall \chi \in \mathit{trig}_\tau(f).\ \chi \notin \boldsymbol{A} \end{aligned} \tag{9}$$

$$\begin{aligned} f(\boldsymbol{AX}) \ &= \ \sum_{A_i} f(\boldsymbol{AX})[a_j.X_{ij}/A_i\boldsymbol{X}_i] \ \oplus \bigoplus_{C_i \subseteq A_i} f(\boldsymbol{AX})[c_j.X_{ij}/A_i\boldsymbol{X}_i] \\ &\quad \text{if } \ \neg\Omega(f(\boldsymbol{AX})) \ \wedge \ \mathit{trig}_\tau(f) \neq \emptyset \ \wedge \\ &\quad \exists \text{minimal } C_i, \forall \chi \in \mathit{trig}_\tau(f).\ \text{if } \chi \in \boldsymbol{A} \text{ then } \chi \in \boldsymbol{A}[C_i/A_i] \end{aligned} \tag{10}$$

Next, we briefly explain the syntactic conditions of the laws. The condition for law (9) says that either one of the active arguments (not necessarily the jth) is Ω or none of the active

Figure 2: Axioms for hiding.

$$
\begin{array}{rcll}
(X \oplus Y)\backslash a & = & X\backslash a \oplus Y\backslash a & \\
(\sum_{b\in B} b.X_b)\backslash a & = & \sum_{b\in B}(b.X_b)\backslash a & \text{if } a \notin B \\
(\sum_{b\in B} b.X_b \boxplus a.X)\backslash a & = & (\sum_{b\in B}(b.X_b)\backslash a \boxplus (a.X)\backslash a) \oplus (a.X)\backslash a & \text{if } a \notin B \\
(b.X)\backslash a & = & b.(X\backslash a) & \text{if } b \neq a \\
(a.X)\backslash a & = & X\backslash a & \\
\mathbf{0}\backslash a & = & \mathbf{0} & \\
\Omega\backslash a & \sqsubseteq & \Omega &
\end{array}
$$

arguments is Ω and no τ triggers for f can fire from $\boldsymbol{A}$. The right conjunct of the condition for law (10) says that f is τ-introducing and there exists the minimal (non-empty) $C_i \subseteq A_i$ such that whenever any τ trigger for f can fire from $\boldsymbol{A}$ then it can fire from $\boldsymbol{A}[C_i/A_i]$.

The proof of Lemma 4.5 can be found in Appendix A.

Next, consider the extension of B with the CSP hiding operator $\backslash a$ defined by the following rules:

$$\frac{X \xrightarrow{a} X'}{X\backslash a \xrightarrow{\tau} X'\backslash a} \qquad \frac{X \xrightarrow{b} X'}{X\backslash a \xrightarrow{b} X'\backslash a}\ b \neq a \qquad \frac{X \xrightarrow{\tau} X'}{X\backslash a \xrightarrow{\tau} X'\backslash a.}$$

Since $\backslash a$ is distinctive it clearly satisfies the condition in Lemma 4.5. The axioms obtained by Lemmas (4.3)–(4.5) are given in Figure 2. Note that the second axiom *does not* hold without the side condition. Each of the first two equations is an axiom schema but, as Act is finite, it represents a finite set of axioms—one for each subset B of Act. Hence, the axiomatisation is *finite*. Similarly, the axiomatisations of other distinctive operators, for example the restriction or renaming of CCS, will also be finite.

The following lemma show how non-distinctive operators can be axiomatised with the help of certain auxiliary distinctive operators.

Lemma 4.6 (Non-distinctive operator laws) Let a non-distinctive (f, n) be an operator of a self-encoding τDeS process language G. Then there is a disjoint extension G' of G with (possibly zero) n-ary distinctive non-τ-introducing operators f_k, where $k \in K \subset_{fin} \mathbb{N}$, and (possibly zero) n-ary distinctive τ-introducing operators f^l, $l \in L \subset_{fin} \mathbb{N}$, such that G' is a τDeS language and the following laws hold in $\mathsf{TEST}(G')$:

$$
\begin{array}{rcll}
f(\boldsymbol{AX}) & = & \sum_K f_k(\boldsymbol{AX}) \boxplus \sum_L f^l(\boldsymbol{AX}) & \\
 & & \text{if } \Omega(f(\boldsymbol{AX})) \vee & \\
 & & \neg\Omega(f(\boldsymbol{AX})) \wedge \forall \chi \in trig_\tau(f).\ \chi \notin \boldsymbol{A} & (11) \\
 & & & \\
f(\boldsymbol{AX}) & = & (\sum_K f_k(\boldsymbol{AX}) \boxplus \sum_L f^l(\boldsymbol{AX})) \oplus \bigoplus_N f^n(\boldsymbol{AX}) & \\
 & & \text{if } \neg\Omega(f(\boldsymbol{AX})) \wedge trig_\tau(f) \neq \emptyset \wedge \exists N \subseteq L, \forall \chi \in trig_\tau(f). & \\
 & & \qquad \text{if } \chi \in \boldsymbol{A} \text{ then } \exists n \in N.\chi \in trig_\tau(f^n) & (12)
\end{array}
$$

The condition of (11) is the same as the one for (9). The condition of (12) says that none of the active arguments of $f(\boldsymbol{AX})$ is Ω, f is τ-introducing and there exists a (non-empty since f is τ-introducing) subset N of L such that whenever a τ trigger for f fires from $\boldsymbol{A}$ then this trigger is a τ trigger for some $f^n, n \in N$.

The sketch of proof of Lemma 4.6 can be found in Appendix A.

As an example of axiomatisation of a non-distinctive and τ-introducing operator we consider B extended with the CCS parallel composition operator $\|$. Assume that for each $a \in \mathsf{Act}$ we have $\bar{a} \in \mathsf{Act}$ and $\bar{\bar{a}} = a$. By Lemma 4.6, the $\|$ operator is axiomatised in terms of three auxiliary operators. These operators might be called the left-merge, written $\lfloor\!\lfloor$, the right-merge, written $\rfloor\!\rfloor$, and the communication merge, written $|$, since they are very similar to the same named original operators discussed in [BK84, BK85, BW90, ABV92]. They are defined by the following De Simone rules together with the associated τ-rules which are not shown. Note that, unlike their original versions, these operators are τDeS.

$$\frac{X \xrightarrow{a} X'}{X \lfloor\!\lfloor Y \xrightarrow{a} X' \| Y} \qquad \frac{Y \xrightarrow{a} Y'}{X \rfloor\!\rfloor Y \xrightarrow{a} X \| Y'} \qquad \frac{X \xrightarrow{a} X' \quad Y \xrightarrow{\bar{a}} Y'}{X \,|\, Y \xrightarrow{\tau} X' \| Y'}$$

As before AX and BY stand for $\sum_{a \in A} aX_a$ and $\sum_{b \in B} bX_b$ respectively, where $A, B \neq \emptyset$. The axioms for $\|$ and the auxiliary operators obtained by applying Lemmas (4.3)–(4.6) are given in Figure 4. For the same reason as with hiding the axiomatisation of the CCS parallel is finite.

We can remove some redundancy among the axioms in Figure 4 by adding, for example, valid and derivable (from the axioms generated by our method) reflexivity axioms for $\|$ and $|$ and for closed terms. Such an addition would make the second, the fourth and five other easily identifiable axioms redundant. However, it can be shown employing a similar argument as in [ABV94] that the reflexivity axioms for $\|$ and $|$ and for open terms are *not* in general derivable from the axioms generated by our method.

Next, in Figure 3 we present algorithm A which generates a sound axiom system for $\ll$ and an arbitrary τDeS process language. Moreover, the axiom system generated by A is head normalising for initially τ-convergent processes in the considered language.

Theorem 4.7 (**Soundness**) Let G be a τDeS process language and G' be a disjoint extension of G produced by algorithm A. Moreover, let T be the axiom system produced by algorithm A. Then $\mathsf{TEST}(G') \models \mathrm{T}$.

Proof. $\mathsf{TEST}(G') \models \mathrm{T}$ follows from Lemmas (4.3)–(4.6). □

4.2 Finiteness and Head Normalisation

In this subsection we argue that the axiomatisations produced by algorithm A are finite. Moreover, they are head normalising for initially τ-convergent processes.

Theorem 4.8 (**Finiteness**) Axiom systems produced by A are finite.

Proof. Given a τDeS process language G with the set of visible actions A the following are true about the resulting language G' and the axiomatisation T produced by algorithm A.

1. Since G has finitely many operators there are only finitely many instances of law (8) in T.

Input: τDeS language G with the set of visible actions A.

1. If G is not self-encoding then add disjointly to G a copy of $\mathsf{B}(A)$. Call the result G''.

2. For each operator $f \notin \Sigma_{\mathsf{B}(A)}$ in G'' add to $\mathrm{T}_{\mathsf{B}(A)}$ all the instances of the distributivity over $\oplus$ laws (8) and call the resulting system T''.

3. For each non-distinctive operator $f \notin \Sigma_{\mathsf{B}(A)}$ in G'' apply the construction of Lemma 4.6 to obtain the language G'. Add to T'' all the resulting instances of axioms (11)–(12) and call the resulting system T'.

4. For each distinctive operator $f \notin \Sigma_{\mathsf{B}(A)}$ in G' add to T' the following axioms: an action law (4) or (5) for each rule for f; all the inaction laws (6); all the divergence laws (7); for each active argument of f the distributivity over $\boxplus$ laws (9)–(10). Call the resulting system T.

Output: τDeS language G' such that $G < G'$ and the axiom system T such that $\mathsf{TEST}(G') \models \mathrm{T}$.

Figure 3: Axiomatisation algorithm A.

2. G can only have finitely many non-distinctive operators. Since each τDeS operator has only finitely many defining rules there will only be finitely many auxiliary distinctive operators required by Lemma 4.6. Thus, G' has a finite number of distinctive operators.

3. It is clear, from the remarks in point 2 above, that laws (4)–(7) give rise to finitely many instances for each distinctive operator in G'.

4. There are only finitely many different instances of laws (9) and (10) in T, up to a change of variable names. This is because there is only a finite number of different $\boldsymbol{A}$ we can use in $f(\boldsymbol{AX})$: for A_i is either $*$ or Ω or one of the finitely many subsets of finite A.

5. For the same reason as for laws (9) and (10), each of laws (11) and (12) contributes a finite set of instances to T.

Since T consists of a finite $\mathrm{T}_{\mathsf{B}(A)}$ and the sets of all instances of laws (4)–(12), each of them finite, we conclude that T is finite. □

In the next subsection we will argue that the axiomatisations produced by algorithm A are not only finite but also *complete* for well-founded τDeS languages. Since $\mathsf{B}_{||}$, the basic language B extended with the CCS parallel composition operator, is well-founded the axiomatisation of testing preorder over $\mathsf{B}_{||}$ is both finite and complete, and so is the axiomatisation of B extended with the hiding operator.

The next theorem is essential in proving completeness of the generated axiomatisations.

Theorem 4.9 (**Head Normalisation**) Let G be a τDeS process language and G' be a disjoint extension of G produced by algorithm A. Moreover, let T be the axiom system produced by algorithm A. Then T is head normalising for all initially τ-convergent processes in $\mathrm{T}(\Sigma_{G'})$.

Proof. We will prove by induction the following property denoted by $(\star)$:

> if $p \in \mathrm{T}(\Sigma_{G'})$ and p is initially τ-convergent then there exists $q \in \mathrm{T}(\Sigma_{G'})$ such that q is in hnf and $\mathrm{T} \vdash p = q$.

The induction is on $d_\varepsilon(p)$ which is defined as follows. Firstly, we define a function $d : \mathrm{T}(\Sigma) \to \mathbb{N}$ which specifies the depth of operator names in ground terms over some signature Σ. Function d is defined inductively as follows: $d(t) = 0$ if t is a constant, and $d(g(t_1, \ldots, t_r)) = 1 + \max\{d(t_i) \mid 1 \leq i \leq r\}$. Function d_ε is defined only for initially τ-convergent processes. It specifies the maximal operator depth of ε-descendants of processes: for τ-convergent p we define $d_\varepsilon(p)$ as $\max\{d(p') \mid p \stackrel{\varepsilon}{\Rightarrow} p'\}$.

In the base case and the inductive step we will demonstrate how to rewrite initially τ-convergent process terms into terms with a certain form by using laws (4)–(12) and some of the axioms and derived axioms for B. The mentioned form is called *sum of prefixed terms* form, sptf for short, and it is defined inductively as follows. A term p is in sptf if

- $p \equiv \mathbf{0}$, $p \equiv \Omega$ or $p \equiv a.r$, for some action a and term r,
- $p \equiv p' \boxplus p''$ or $p \equiv p' \oplus p''$, for p' and p'' in sptf.

It is routine to prove, using the rewriting techniques developed in [NH84, Hen88], that terms in sptf can be transformed into terms in hnf.

We begin with the base case. Assume that p is initially τ-convergent $\Sigma_{G'}$ process and $d_\varepsilon(p) = 0$. This means that p is a constant and for each p' such that $p \stackrel{\varepsilon}{\Rightarrow} p'$ p' is a constant too. We transform p into sptf by repeatedly performing the following steps:

1. If p is distinctive, which means that there is at most one defining rule for p, then p can be rewritten using action or inaction laws.
2. If p is not distinctive then it can be rewritten using law (11) or law (12). The resulting term will contain only distinctive constants and each of them will be rewritten using action and inaction laws if required.

It may happen that after performing steps 1 and 2 the resulting term t is in sptf, then we are finished. If t is not in sptf (this will happen when p is τ-introducing) we rewrite each of the constants in t, which is not prefixed by an action and it is neither $\mathbf{0}$ nor Ω, according to steps 1 and 2 above. We continue the transformation of the resulting term as long as it contains other than $\mathbf{0}$ and Ω constants which are not prefixed with actions. In the transformation we use the above steps 1 and 2. Notice that each τ transition of a constant corresponds to the rewriting by action law (5) or by law (12) together with (5). Since p is initially τ-convergent the transformation will eventually terminate with the resulting process term in sptf. Finally, the axioms and the derived axioms for B can be used to rewrite it into hnf.

Next, we assume that the property $(\star)$ holds for all q such that $d_\varepsilon(q) \leq n$ and show that it also holds for any p with $d_\varepsilon(p) = n + 1$. As in the base case it is enough to show how to rewrite p into sptf. Since p is not necessarily a constant the steps corresponding to 1 and 2 above are more complicated. They are as follows:

1′. If the operator of p is distinctive and moreover p is a constant then rewrite it using action and inaction laws if required. However, if $p \equiv f(\boldsymbol{p})$ then since $d_\varepsilon(p_i) \leq n$, for each p_i of $\boldsymbol{p}$, the term p_i can be put into hnf p_i' by inductive hypothesis. Thus, we continue with $f(\boldsymbol{p'})$.

   (a) If any active argument of $f(\boldsymbol{p'})$ has the form $r \boxplus \Omega$, for r other than $\mathbf{0}$, then each such argument is rewritten as $r \oplus \Omega$ by the derived axiom $Der1$. As the result we obtain $f(\boldsymbol{p''})$, where each active argument p_i'' has one of the following forms: Ω, $\mathbf{0}$, $\bigoplus_{K \neq \emptyset} p_{ik}''$ or $p_{ik}'' \oplus \Omega$, where p_{ik}'' is an external choice of terms prefixed with actions. If p_i'' is not active then it equals p_i'.

(b) If any of the active arguments of $f(\boldsymbol{p}'')$ have the form of an internal sum then we repeatedly use law (8) from left to right to rewrite $f(\boldsymbol{p}'')$. As the result we obtain $\bigoplus_M f(\boldsymbol{p}''_m)$, where each active argument of each $f(\boldsymbol{p}''_m)$ has one of the forms Ω, $\mathbf{0}$ or $\sum_{I\neq\emptyset} a_i.r_i$, for some actions a_i and terms r_i.
Remark. Note that Lemmas (4.5)–(4.6) give the distributivity laws for terms $f(\boldsymbol{AX})$ with active arguments of the form: Ω, $\mathbf{0}$ or $\sum_{I\neq\emptyset} a_i.r_i$.

(c) If any of the active arguments of any $f(\boldsymbol{p}''_m)$ have the form of an external sum then use the distributivity over $\boxplus$ laws for such arguments of all such $f(\boldsymbol{p}''_m)$. The resulting term will be a sum (external and/or internal) of terms $f(\boldsymbol{p}'''_n)$, where each active argument is either Ω, $\mathbf{0}$ or $a_{ni}.r_{ni}$, for some a_{ni} and r_{ni}. Finally, rewrite each of the subterms $f(\boldsymbol{p}'''_n)$ using action, inaction and divergence laws as appropriate.

2′. If the operator of p is not distinctive then p can be rewritten using law (11) or law (12). The resulting term will be a sum (external and/or internal) of terms with outermost operators being distinctive. Each of these terms will be rewritten as in step 1′.

At this stage the resulting term is almost in sptf—it is a sum (external and/or internal) of terms of the following form: Ω, $\mathbf{0}$, $a_{ni}.r_{ni}$ or $g(\boldsymbol{r})$, where g is neither the action prefixing operator, the internal choice nor the external choice. The last form of term appears when f is τ-introducing and there is a De Simone rule for f with the consequent $f(\boldsymbol{X}) \xrightarrow{\tau} g(\boldsymbol{Y})$ and a ground substitution ρ such that $\rho(f(\boldsymbol{X}) \xrightarrow{\tau} g(\boldsymbol{Y})) \equiv f(\boldsymbol{p}) \xrightarrow{\tau} g(\boldsymbol{r})$. Since $d_\varepsilon(g(\boldsymbol{r}))$ can be $n+1$ we cannot use the inductive hypothesis to rewrite it to hnf.

If after rewriting p according to steps 1′ and 2′ the resulting term is not in sptf yet then we continue to apply steps 1′ and 2′ to all subterms $g(\boldsymbol{r})$ (as described in the previous paragraph) and all the resulting from them similar subterms. Since p is initially τ-convergent the rewriting of p will eventually terminate with the resulting process term in sptf. As in the base case, the axioms and the derived axioms for B can be used to rewrite it into hnf. □

4.3 Completeness

Given any τDeS process language G with the set of visible actions A algorithm A produces axiom system T which allows conversion of initially τ-convergent processes into head normal form. For well-founded processes this implies completeness of T: it is possible to rewrite arbitrary process terms over G into terms over $\mathsf{B}(A)$ by applying the reduction to head normal form a finite number of times. However, in order to obtain completeness for (some) process terms representing infinite behaviours we need to augment T with some (infinitary) induction rule. These two ways of achieving completeness are described below.

We start with the completeness result for well-founded processes. Its proof together with a number of auxiliary results appears in Appendix B.

Theorem 4.10 Let G be a well-founded self-encoding τDeS process language. Let G' and T denote the disjoint extension and the axiom system produced by the algorithm in Figure 3 respectively. Then for all $p, q \in \mathrm{T}(\Sigma_{G'})$ if $\mathsf{Test}(G') \vDash p \sqsubseteq q$ then $\mathrm{T} \vdash p \sqsubseteq q$.

Although well-foundedness of τDeS process languages has been defined semantically (Definition 2.5) it can also be characterised in a syntactical manner. We follow the approach developed in [ABV94]. There, *syntactical* well-foundedness of GSOS languages is introduced (Definition 6.6) in terms of a certain weight function, if it exists. The authors prove that syntactically well-founded and *linear* GSOS languages are well-founded. Also, they argue that

although (semantic) well-foundedness is not in general decidable syntactic well-foundedness is. Since τDeS languages are a subclass of linear GSOS languages every syntactically well-founded τDeS language is well-founded. However, as noticed in [ABV94], not all well-founded GSOS (and thus τDeS) languages are syntactically well-founded.

Axiomatisations of process languages with terms representing non-finite behaviour require some form of induction rule in order to obtain completeness. For example, a version of AIP is used in [ABV92, ABV94] to achieve completeness for arbitrary GSOS languages. A version of ω-Induction is suggested in [Uli95] as the induction rule needed for completeness of axiomatisations of refusal simulation preorder and τDeS process languages. A similar from of ω-Induction rule is employed here. Our rule is based on the ω-Induction rule as discussed in [NH84, Hen88]. It represents a principle that if two processes are testing related at all finite depths then they are testing related.

$$\frac{X/b^n\Omega \sqsubseteq Y \quad \text{all } n}{X \sqsubseteq Y} \qquad (\omega\text{-Induction})$$

The notation $b^n\Omega$ denotes $b \ldots b.\Omega$, where b appears n times. A binary infix operator $/$, taken from [ABV92, ABV94], is defined by the following rule

$$\frac{X \xrightarrow{a} X' \quad Y \xrightarrow{b} Y'}{X/Y \xrightarrow{a} X/Y'}$$

together with the associated τ-rules. Since $/$ is defined by a single De Simone rule it is distinctive. Moreover, it is not τ-introducing. Hence, using our axiomatisation method we obtain the following intuitive axioms for $/$, where AX and BY stand for $\sum_A a.X_a$ and $\sum_B b.Y_b$ respectively with $A, B \neq \emptyset$.

$$\begin{array}{rcl}
(X \oplus Y)/Z & = & X/Z \oplus Y/Z \\
X/(Y \oplus Z) & = & X/Y \oplus X/Z \\
(\sum_A a.X_a)/BY & = & \sum_A (a.X_a/BY) \\
AX/(\sum_B b.Y_b) & = & \sum_B (AX/b.Y_b) \\
a.X/b.Y & = & a.(X/Y) \\
\mathbf{0}/X & = & \mathbf{0} \\
X/\mathbf{0} & = & \mathbf{0} \\
\Omega/X & \sqsubseteq & \Omega \\
X/\Omega & \sqsubseteq & \Omega
\end{array}$$

We let $G_/$ denote the disjoint extension of a τDeS language G with the operator $/$. The soundness of ω-Induction and some useful properties of $/$ are proved in Appendix B.

Finally, we have the soundness and completeness result for τ-convergent but not necessarily well-founded τDeS processes.

Theorem 4.11 Given τDeS language G let G' and T denote the disjoint extension of $G_/$ and the axiom system respectively produced by the algorithm in Figure 3. Then, for all τ-convergent $p, q \in \mathrm{T}(\Sigma_{G'})$, $\mathsf{Test}(G') \vDash p \sqsubseteq q$ if and only if T, ω-Induction $\vdash p \sqsubseteq q$.

The proof for Theorem 4.11 can be found in Appendix B.

Our ω-Induction rule is very strong and in many cases (for example, the systems to be discussed in the next section) a weaker rule would be sufficient, for example the *Recursive Specification Principle* (RSP) defined in [BK88, BW90] and used in [Mil84, Mil89b, vG93, Ace94].

It is expected that for a suitably defined class of regular τDeS languages the techniques described in [Mil84, Mil89b, Ace94], which do not rely on infinitary rules, would be applicable. Moreover, weaker but effective induction rules, for example Scott Induction as discussed in [Hen88], may be sufficient for some interesting subclasses of τDeS languages. This will be considered elsewhere.

5 Applications

We argue that τDeS languages and their automatically generated axiomatisations can make the task of specification and verification of concurrent systems easier and more intuitive. Freedom to use arbitrary τDeS operators allows one to write intuitive and clear specifications without worrying about the implementation. It is advisable, in the specification process, to write a series of specifications of increasing concreteness. Ideally, the most concrete specification would suggest a good method of implementation. We prove that the more concrete specifications refine the less concrete ones. The proofs are done within the axiom systems as introduced in the last section.

The common theme in our examples is the control and scheduling of the execution of several systems acting in parallel. The situations we describe are:

- interruption or abortion, suspension and resumption of a system by its own or other system's actions;
- alternate execution of two systems, where the control of execution is passed between the systems by 'seizing' and 'releasing' actions.

Example 5.1 Consider a system which repeatedly performs a until action s takes place, after which it performs repeatedly b until the next occurrence of s, after which the system behaves as originally. The abstract way to specify this system is by listing its states and describing the transitions between them. The system has two states which we represent by nullary τDeS operators S_1 and S_1'. The operational meaning of S_1 and S_1' is defined by the following rules:

$$S_1 \xrightarrow{a} S_1 \qquad S_1 \xrightarrow{s} S_1' \qquad S_1' \xrightarrow{b} S_1' \qquad S_1' \xrightarrow{s} S_1$$

Note that S_1 (and S_1') is non-distinctive: it fails the second part of the distinctiveness definition because it has two rules with the same (i.e. none) antecedents. The axiomatisation for S_1 and S_1' produced by A gives a more familiar description of S_1 and S_1':

$$S_1 = a.S_1 \boxplus s.S_1' \qquad \text{and} \qquad S_1' = b.S_1' \boxplus s.S_1$$

How can this system be constructed? One way is to take two subsystems specified by P and Q, with $P = a.P \boxplus s.P$ and $Q = b.Q \boxplus s.Q$, and combine them by means of the operator $\gg$ defined as follows, where $\alpha \in \{a, b, \tau\}$:

$$\frac{X \xrightarrow{\alpha} X'}{X \gg Y \xrightarrow{\alpha} X' \gg Y} \qquad \frac{Y \xrightarrow{s} Y'}{X \gg Y \xrightarrow{s} Y' \gg X} \qquad \frac{Y \xrightarrow{\tau} Y'}{X \gg Y \xrightarrow{\tau} X \gg Y'}$$

Note that the process P in $P \gg Q$ executes freely until it is suspended by Q, then the process Q in $Q \gg P$ proceeds freely until P restarts itself (or suspends Q).

It is easy to derive axioms for $\gg$ and prove that $P \gg Q = a.(P \gg Q) \boxplus s.(Q \gg P)$ and $Q \gg P = b.(Q \gg P) \boxplus s.(P \gg Q)$. Thus, by ω-Induction, we obtain $S_1 = P \gg Q$ and $S_1' = Q \gg P$. Thus, we conclude that $P \gg Q$ is a more concrete version of S_1.

Example 5.2 Imagine a multi-media player which offers an interactive video performance. The viewer can *start* the performance, view the offered items, suspend and resume the performance by using the *pause* function and *choose* one of two alternatives. The viewer controls the player by a remote control, which has the buttons *start* (to start or restart the performance), *pause*, and the buttons 1 and 2 for making a choice. As in the previous example, we could represent the states of the system by the nullary operators, define the transitions between these states by certain rules and then derive the equation(s) describing their specification. But, we omit the details and just give the specifications. Let M and RC represent the multi-media player and the remote control systems respectively. M and RC are such that

$$\begin{aligned} M &= s.M' \\ M' &= c_1.(c_2.M'' \boxplus M) \boxplus M \\ M'' &= choice.(one.m_1 \boxplus two.m_2 \boxplus M) \boxplus M \\ RC &= start.s.r.RC \boxplus pause.pause.RC \boxplus \\ &\quad 1.one.r.RC \boxplus 2.two.r.RC \end{aligned}$$

where s is the signal RC sends to M to start the performance afresh; c_1, c_2 and m_1, m_2 are commercials and movies respectively; *one* and *two* are the signals by which RC instructs M of the viewers' choice and r is the actions by which RC returns the control to M.

M and RC make up the subsystem P with the intended behavior described as follows:

$$\begin{aligned} P &= start.s.r.P' \boxplus pause.pause.P \\ P' &= c_1.(c_2.P'' \boxplus P) \boxplus P \\ P'' &= choice.(1.one.r.m_1 \boxplus 2.two.r.m_2 \boxplus P) \boxplus P \end{aligned}$$

The whole system S consists of the viewer V together with M and RC. The concrete specification of the system is $V \parallel_D op(M, RC)$, where D consists of *start*, c_1, c_2, *choice*, $1, 2$, *pause*, m_1, m_2 and $\parallel_D$ is a CSP-like parallel composition operator with synchronisation on all action in D. Instead of defining the specific *op* we will use a general operator $\langle A \rangle_B \rangle_C$ called the *controlled parallel* composition, which has three parameters $A, B, C \subset \mathsf{Act}$ such that $(A \cup C) \cap B = \emptyset$. The defining rules for the operator are given below. It binds weaker than the prefixing. but stronger than the choice operators.

$$\frac{X \xrightarrow{a} X'}{X \langle A \rangle_B \rangle_C Y \xrightarrow{a} X' \langle A \rangle_B \rangle_C Y} \; a \notin A \qquad \frac{X \xrightarrow{a} X'}{X \langle A \rangle_B \rangle_C Y \xrightarrow{a} Y \langle A \rangle_B \rangle_C X'} \; a \in A$$

$$\frac{X \xrightarrow{b} X' \quad Y \xrightarrow{b} Y'}{X \langle A \rangle_B \rangle_C Y \xrightarrow{b} X' \langle A \rangle_B \rangle_C Y'} \; b \in B \qquad \frac{Y \xrightarrow{c} Y'}{X \langle A \rangle_B \rangle_C Y \xrightarrow{c} Y' \langle A \rangle_B \rangle_C X} \; c \in C$$

$$\frac{X \xrightarrow{\tau} X'}{X \langle A \rangle_B \rangle_C Y \xrightarrow{\tau} X' \langle A \rangle_B \rangle_C Y} \qquad \frac{Y \xrightarrow{\tau} Y'}{X \langle A \rangle_B \rangle_C Y \xrightarrow{\tau} X \langle A \rangle_B \rangle_C Y'}$$

Next, we briefly explain these rules. In $P \langle A \rangle_B \rangle_C Q$ the control of execution is with the first argument: see the first rule. The parameter A contains the actions by which the first argument transfers the control to the second argument: the second rule. The parameter B contains

the actions on which both the arguments must synchronize: the third rule. The parameter C consists of all actions by which the second argument takes over the control from the first one: the fourth rule. Finally, the last two rules are the associated τ-rules.

The set of axioms for $\langle A \rangle_B \rangle_C$ can be easily derived using algorithm A. Similarly as for the CCS parallel operator the axiomatisation of $\langle A \rangle_B \rangle_C$ uses three auxiliary operators. Using the derived axioms it is routine to show the following, for $A = \{choice, pause, r\}$, $B = \{s, one, two\}$ and $C = \{start, pause, r\}$.

$$\begin{array}{lcl} M\ {}_A\langle\!\langle_B\rangle_C\ RC & = & start.s.r.(M'\ {}_A\langle\!\langle_B\rangle_C\ RC) \ \boxplus\ pause.pause.(M\ {}_A\langle\!\langle_B\rangle_C\ RC) \\ M'\ {}_A\langle\!\langle_B\rangle_C\ RC & = & c_1.(c_2.(M''\ {}_A\langle\!\langle_B\rangle_C\ RC) \ \boxplus\ M\ {}_A\langle\!\langle_B\rangle_C\ RC) \ \boxplus\ M\ {}_A\langle\!\langle_B\rangle_C\ RC \\ M''\ {}_A\langle\!\langle_B\rangle_C\ RC & = & choice.(1.one.r.m_1 \boxplus 2.two.r.m_2 \ \boxplus\ M\ {}_A\langle\!\langle_B\rangle_C\ RC) \ \boxplus\ M\ {}_A\langle\!\langle_B\rangle_C\ RC \end{array}$$

Finally, applying ω-Induction rule we prove $P = M\ {}_A\langle\!\langle_B\rangle_C\ RC$.

6 Conclusions and Related Work

The De Simone format was introduced in [dS85] as a format which preserves strong bisimulation. However, when weak equivalences are considered the generality of the original De Simone rules permits to define some undesirable operators, namely those which do not preserve the considered equivalences. In order to correct this, one usually imposes various restrictions on the form of De Simone rules and uses τ-rules as, for example, in [Blo90b, Vaa91, Uli92, Blo95a, Uli94]. The τDeS format is a subformat of the ISOS format [Uli92, Uli94]. The ISOS rules may have negative antecedents and copying in the target of the consequent.

Regarding the congruence theorem, most of our proofs are based on the proofs in [Hen88] although neither De Simone nor other generally defined classes of process languages are considered there. The only directly related result is due to Vaandrager who argues in [Vaa91] that testing preorder, in the setting without divergence, is a precongruence for any language in a class of languages similar to τDeS[2].

The idea of generating axiomatisations of arbitrary process languages from their operational specifications (given by SOS rules) is due to Aceto, Bloom and Vaandrager [ABV92, ABV94]. They defined two procedures for obtaining axiomatisations of strong bisimulation for arbitrary GSOS languages. Such axiomatisations are finite for well-founded languages. The finiteness is achieved by extending the considered language with useful, and indeed required [Mol90], auxiliary operators. This approach is a generalisation of the method originally used by Bergstra and Klop in [BK84, BK85], where interleaving parallel composition (no communication) operator is finitely axiomatised with the help of auxiliary left-merge operator. Aceto [Ace94] proposed an extension of the algorithm from [ABV92] which generates finite axiomatisations of strong bisimulation for regular GSOS languages, i.e. languages in which certain infinite behaviours can be represented. However, if one tries to adapt the procedures from [ABV92, ABV94] to any of weak equivalences one finds that the introduced auxiliary operators do not preserve many of weak equivalences. In [Blo95a] this difficulty is overcome by considering *rooted* versions of weak equivalences and further restricting the (GSOS) format of rules. An alternative approach is presented in [Uli95], where a general procedure is defined for deriving axiomatisations of refusal simulation preorder and coarser

[2]Vaandrager allows silent actions in the antecedents of De Simone rules, provides an extra "idling" axiom and, separately, provides recursion.

(weak) preorders. The auxiliary operators introduced by the procedure in [Uli95] preserve the considered weak preorders and equivalences.

The procedure from [Uli95] can be adjusted, by adding a number of extra axioms, to produce axiomatisations of testing preorder. But, the procedure presented in this paper has two important advantages over the adjusted procedure:

- it uses only nine laws instead of twenty of the adjusted procedure,
- it always gives finite axiomatisations for well-founded processes.

One of the motivations for theories concerning classes of process languages, which are defined by formats of SOS rules, is to facilitate the task of specification. However, except for some simple examples where new operators were used to make the specification easier [BIM88, BIM95, GV90, Gro90, Uli94] this theme has not been studied in detail. A recent exception is the work by Bloom [Blo95b], where the author gives several specifications of the job protecting an arbitrary server by a checkpoint/backup scheme. As for the verification method both the equational reasoning and process equivalence techniques are used.

Acknowledgements

The pioneering work of Luca Aceto, Bard Bloom and Frits Vaandrager [ABV92, ABV94] was the inspiration for the present (and the previous [Uli95]) work on axiom generation for weak process preorders, for which I am greatly indebted to them. I would like to thank the referees and the participants of AMAST '96 for their helpful comments and suggestions. Thanks are also due to Paul Taylor for his LaTeX macros.

A Soundness of Head Normalising Laws

This appendix contains soundness proofs of the distributivity over $\boxplus$ and $\oplus$ laws and the non-distinctive operator laws.

Proof of Lemma 4.4. Let $X_i \equiv X \oplus Y$. We need to show $\rho(f(\boldsymbol{X}) \simeq \rho(f(\boldsymbol{X})[X/X_i] \oplus f(\boldsymbol{X})[Y/X_i])$, for any closed substitution ρ over G' with $G \leq G'$. We will only show $\rho(f(\boldsymbol{X}) \ll \rho(f(\boldsymbol{X})[X/X_i] \oplus f(\boldsymbol{X})[Y/X_i])$ as the converse is proved similarly. By Theorem 2.13 it is enough to prove

$$\rho(f(\boldsymbol{X})) \ll_{\mathsf{may}} \rho(f(\boldsymbol{X})[X/X_i] \oplus f(\boldsymbol{X})[Y/X_i])$$

and

$$\rho(f(\boldsymbol{X})) \ll_{\mathsf{must}} \rho(f(\boldsymbol{X})[X/X_i] \oplus f(\boldsymbol{X})[Y/X_i]).$$

For $\ll_{\mathsf{may}}$ case let $\rho(f(\boldsymbol{X}) \stackrel{s}{\Rightarrow}$, where s is a non-empty sequence of visible actions over G', and assume that s is the sequence $\alpha_1 \ldots \alpha_n$ with all τ removed. Thus,

$$(\rho(f(\boldsymbol{X}) \equiv)\rho(C_0[\boldsymbol{X}_0]) \stackrel{\alpha_1}{\rightarrow} \rho(C_1[\boldsymbol{X}_1]) \ldots \stackrel{\alpha_n}{\rightarrow} \rho(C_n[\boldsymbol{X}_n])$$

for some n-ary contexts $C_i[\boldsymbol{X}_i]$. We need to consider two cases depending on whether or not the computation of the above sequence depends on the value of the i-th argument. If it does not then it is clear that $\rho(f(\boldsymbol{X})[X/X_i]) \stackrel{s}{\Rightarrow}$ as well as $\rho(f(\boldsymbol{X})[Y/X_i]) \stackrel{s}{\Rightarrow}$, and we are finished. Otherwise, let $1 \leq k \leq n$ be the smallest k such that $\rho(C_k[\boldsymbol{X}_k]) \stackrel{\alpha_{k+1}}{\rightarrow} \rho(C_{k+1}[\boldsymbol{X}_{k+1}])$ depends on the behaviour of $\rho(X_i)$. Since $\rho(X_i)$ is $\rho(X) \oplus \rho(Y)$ we deduce, by the Ruloid Theorem, that

in fact $\alpha_{k+1} = \tau$ and $\rho(C_{k+1}[\boldsymbol{X}_{k+1}])$ equals $\rho(C_k[\boldsymbol{X}][X/X_i])$ or $\rho(C_k[\boldsymbol{X}][Y/X_i])$. Depending on which of the last two terms it is, we obtain either $\rho(f(\boldsymbol{X})[X/X_i]) \stackrel{s}{\Rightarrow}$ or $\rho(f(\boldsymbol{X})[Y/X_i]) \stackrel{s}{\Rightarrow}$ as required.

For $\ll_{\mathsf{must}}$ case assume $\rho(f(\boldsymbol{X})) \stackrel{s}{\Rightarrow}$, for s as above, and $\rho(f(\boldsymbol{X}))\Downarrow s$. We need to show (a) $(\rho(f(\boldsymbol{X})[X/X_i] \oplus f(\boldsymbol{X})[Y/X_i])\Downarrow s$ and (b) for any open term T if $(\rho(f(\boldsymbol{X})[X/X_i] \oplus f(\boldsymbol{X})[Y/X_i]) \stackrel{s}{\Rightarrow} T$ then $\rho(f(\boldsymbol{X})) \stackrel{s}{\Rightarrow} T'$ and $\mathcal{S}(T') \subseteq \mathcal{S}(T)$, for some open term T'. As $X_i \equiv X \oplus Y$ we obtain, by the definition of the convergence predicate, $\rho(f(\boldsymbol{X})) \stackrel{\varepsilon}{\Rightarrow} \rho(f(\boldsymbol{X})[X/X_i])$ and $\rho(f(\boldsymbol{X})[X/X_i])\Downarrow s$. Similarly, we get $\rho(f(\boldsymbol{X})[Y/X_i])\Downarrow s$. Hence, putting these together we obtain $(\rho(f(\boldsymbol{X})[X/X_i] \oplus f(\boldsymbol{X})[Y/X_i])\Downarrow s$ as required. For part (b) we show, using a similar argument as for $\ll_{\mathsf{may}}$ part of this proof, that for each T we have $T \equiv T'$ and thus $\mathcal{S}(T') \subseteq \mathcal{S}(T)$. □

In the proof of Lemma 4.5 and Lemma 4.6 the following result, based on Lemma 4.1.3 in [NH84], will be helpful.

Result A.1 If $\mathcal{D}(p,\mu) = \mathcal{D}(q,\mu)$, for every $\mu \in \mathsf{Act} \cup \{\varepsilon\}$, then $p \simeq_{\mathsf{may}} q$. If $\mathcal{D}(p,\mu) = \mathcal{D}(p,\mu)$, for every $\mu \in \mathsf{Act} \cup \{\varepsilon\}$, and $p\downarrow$ if and only if $q\downarrow$ then $p \simeq q$.

Proof of law (9) of Lemma 4.5. We need to show

$$\rho(f(\boldsymbol{AX})) \simeq \rho(\sum_j f(\boldsymbol{AX})[(A_i\boldsymbol{X}_i)_j/A_i\boldsymbol{X}_i])$$

provided that the condition of law (9) holds for any closed substitution ρ over any disjoint extension of G. The condition requires us to check two cases. In what follows we will use LHS and RHS to stand for the left and the right hand sides of the above expression respectively.

Firstly, we assume that the second disjunct of the condition holds: no active argument of $f(\boldsymbol{AX})$ is Ω and $\rho(f(\boldsymbol{AX})) \stackrel{\tau}{\not\rightarrow}$, for all ρ. Thus, $(A_i\boldsymbol{X}_i)_j \equiv \sum_{A_i} a_j X_{ij}$. Clearly, by Definition 2.3, LHS$\downarrow$ and RHS$\downarrow$. Thus, by Result A.1, we only need to show $\mathcal{D}(\text{LHS},\mu) = \mathcal{D}(\text{RHS},\mu)$. We notice that $f(\boldsymbol{AX})$ cannot perform τ because f is not τ-producing and no τ-rule can be applied (due to the structure of the active arguments). We will show that LHS $\stackrel{a}{\rightarrow} t$ if and only if RHS $\stackrel{a}{\rightarrow} t$, for all closed terms t and visible actions a. Assume $\rho(f(\boldsymbol{AX})) \stackrel{a}{\rightarrow} t$. This means that there is a De Simone rule r

$$\frac{\{\ X_k \stackrel{a_k}{\rightarrow} X'_k\ \}_{k\in K}}{f(\boldsymbol{X}) \stackrel{a}{\rightarrow} C[\boldsymbol{Y}],}$$

where $Y_k = X'_k$ if $k \in K$ and $Y_k = X_k$ otherwise, such that $\rho(C[\boldsymbol{Y}]) = t$ and $\rho(X_k) \stackrel{a_k}{\rightarrow} \rho(X'_k)$ are valid transitions, for all k. Note that since each rule for f has an antecedent for i we have $i \in K$. Hence, $\rho(A_iX_i) \stackrel{a_i}{\rightarrow} \rho(X'_i)$, so $a_i \in A_i$ and $\rho(a_i.X_{ij}) \stackrel{a_i}{\rightarrow} \rho(X_{ij})$. This implies that the ith component of $\boldsymbol{Y}$ is $X'_i = X_{ij}$. Next, we notice that rule r can be applied to derive $\rho(f(\boldsymbol{AX})[a_i.X_{ij}/A_iX_i]) \stackrel{a}{\rightarrow} \rho(C[\boldsymbol{Y'}])$. Moreover, $\boldsymbol{Y'} = \boldsymbol{Y}$. Thus, $\rho(f(\boldsymbol{AX})[a_i.X_{ij}/A_iX_i]) \stackrel{a}{\rightarrow} t$ and so RHS $\stackrel{a}{\rightarrow} t$. The converse, RHS $\stackrel{a}{\rightarrow} t$ implies LHS $\stackrel{a}{\rightarrow} t$ follows similarly.

Secondly, assume that the other half of the condition holds, namely $\Omega(f(\boldsymbol{AX}))$. We deduce, by Definition 2.3, LHS$\uparrow$ and RHS$\uparrow$. This implies LHS $\simeq_{\mathsf{must}}$ RHS and we only need to show LHS $\simeq_{\mathsf{may}}$ RHS, i.e. $\mathcal{L}(\text{LHS}) = \mathcal{L}(\text{RHS})$. By the same argument as above we can show that LHS $\stackrel{a}{\rightarrow} t$ implies RHS $\stackrel{a}{\rightarrow} t$, for all closed terms t and visible actions a. So, if we could prove that for every t such that LHS $\stackrel{\tau}{\rightarrow} t$ there is t' such that RHS $\stackrel{\tau}{\rightarrow} t'$ and $\mathcal{L}(t) \subseteq \mathcal{L}(t')$ then we would have proved a part of our goal: $\mathcal{L}(\text{LHS}) \subseteq \mathcal{L}(\text{RHS})$.

Let $\textsc{lhs} \xrightarrow{\tau} t$. Since every rule for f has an antecedent for X_i we can assume that the antecedent used to derive $\textsc{lhs} \xrightarrow{\tau} t$ is $X_i \xrightarrow{a_i} X_i'$. Because $\textsc{rhs}$ is an external sum (choice) of terms the silent move of one the summands will not resolve the choice. We easily see that $\textsc{rhs} \xrightarrow{\tau} t \boxplus \rho(\sum_{A_i \setminus \{a_i\}} f(\boldsymbol{AX})[a_j.X_{ij}/A_iX_i])$. By denoting the target of the last transition as t' we deduce that $\mathcal{L}(t) \subseteq \mathcal{L}(t')$ as required.

Finally, in order to obtain $\mathcal{L}(\textsc{rhs}) \subseteq \mathcal{L}(\textsc{lhs})$ it is enough to prove

$$\mathcal{L}(\rho(f(\boldsymbol{AX})[a_j.X_{ij}/A_iX_i])) \subseteq \mathcal{L}(\rho(f(\boldsymbol{AX}))$$

for each $a_j \in A_i$. This follows using a similar argument as in the first case. □

Proof of law (10) of Lemma 4.5. Let $\neg\Omega(f(\boldsymbol{AX}))$ and let f be τ-introducing. Assume there exists the minimal (non-empty) $C_i \subseteq A_i$ such that whenever any τ trigger for f can fire from $\boldsymbol{A}$ then it can fire from $\boldsymbol{A}[C_i/A_i]$. We need to show

$$\rho(f(\boldsymbol{AX})) \simeq \rho(\sum_{A_i} f(\boldsymbol{AX})[a_j.X_{ij}/A_i\boldsymbol{X}_i] \oplus \bigoplus_{C_i \subseteq A_i} f(\boldsymbol{AX})[c_j.X_{ij}/A_i\boldsymbol{X}_i])$$

for any closed substitution ρ over any disjoint extension of G. As before we will use $\textsc{lhs}$ and $\textsc{rhs}$ to abbreviate the left and the right hand sides of the above expression.

We can prove, using a similar argument as for law (9), $\mathcal{D}(\textsc{lhs}, a) = \mathcal{D}(\textsc{rhs}, a)$ for all relevant visible actions a. Thus, $\mathcal{L}(\textsc{lhs}) = \mathcal{L}(\textsc{rhs})$ and so $\textsc{lhs} \simeq_{\mathsf{may}} \textsc{rhs}$. Next, we prove $\textsc{lhs} \ll_{\mathsf{must}} \textsc{rhs}$. The converse follows similarly. Assume $\textsc{lhs}\Downarrow s$, for some sequence s. If $s = \varepsilon$ then $t\Downarrow s$ for every t such that $\textsc{lhs} \xrightarrow{\tau} t$. Also, notice that $\{t' \mid \rho(\sum_{A_i} f(\boldsymbol{AX})[a_j.X_{ij}/A_i\boldsymbol{X}_i]) \xrightarrow{\tau} t'\}$ is a subset of $\{t \mid \textsc{lhs} \xrightarrow{\tau} t\}$. Moreover, every t' such that $\rho(f(\boldsymbol{AX})[c_j.X_{ij}/A_iX_i]) \xrightarrow{\tau} t'$ is also a member of $\{t \mid \textsc{lhs} \xrightarrow{\tau} t\}$. Thus, $\textsc{rhs}\Downarrow\varepsilon$. If $s \neq \varepsilon$ then we deduce $\textsc{rhs}\Downarrow s$ from $\mathcal{D}(\textsc{lhs}, a) = \mathcal{D}(\textsc{rhs}, a)$, for all a, and $\textsc{lhs}\Downarrow s$.

Next, assume also $\textsc{rhs} \stackrel{s}{\Rightarrow} q$. If $s \neq \varepsilon$ then, by the above remarks about transitions of $\textsc{lhs}$ and $\textsc{rhs}$, we obtain $\textsc{lhs} \stackrel{s}{\Rightarrow} p$, $p \equiv q$ and so $\mathcal{S}(p) \subseteq \mathcal{S}(q)$. If $s = \varepsilon$ then we have a number of cases:

1. $q \equiv \rho(\sum_{A_i} f(\boldsymbol{AX})[a_j.X_{ij}/A_i\boldsymbol{X}_i])$,
2. $\rho(\sum_{A_i} f(\boldsymbol{AX})[a_j.X_{ij}/A_i\boldsymbol{X}_i]) \xrightarrow{\tau} q' \stackrel{\varepsilon}{\Rightarrow} q$, for some q',
3. $q \equiv \rho(\bigoplus_{C_i \subseteq A_i} f(\boldsymbol{AX})[c_j.X_{ij}/A_i\boldsymbol{X}_i])$,
4. $\rho(\bigoplus_{C_i \subseteq A_i} f(\boldsymbol{AX})[c_j.X_{ij}/A_i\boldsymbol{X}_i]) \xrightarrow{\tau} q' \stackrel{\varepsilon}{\Rightarrow} q$, for some q'.

In case (1) we have $\textsc{lhs} \stackrel{\varepsilon}{\Rightarrow} \textsc{lhs}$ and $\mathcal{S}(\textsc{lhs}) \subseteq \mathcal{S}(q)$. In case (2) we notice that, because silent moves do not resolve external choice, q is like $\rho(\sum_{A_i} f(\boldsymbol{AX})[a_j.X_{ij}/A_i\boldsymbol{X}_i])$ except that at least one of its summands is replaced by a term to which it (the summand) evolves by performing at least one action τ. Let q^* be such that $\rho(f(\boldsymbol{AX})[a_k.X_{ik}/A_i\boldsymbol{X}_i]) \xrightarrow{\tau} q^*$, for some $a_k \in A_i$. Since $\textsc{lhs} \xrightarrow{\tau} q^*$ we have $\mathcal{S}(\textsc{lhs}) \subseteq \mathcal{S}(q)$. In case (3) we consider any p such that $\textsc{lhs} \xrightarrow{\tau} p$. Since $q \xrightarrow{\tau}\xrightarrow{\tau} p$ we have $\mathcal{S}(p) \subseteq \mathcal{S}(q)$. In the last case let a_k be such that $f(\boldsymbol{AX})[c_k.X_{ik}/A_i\boldsymbol{X}_i]) \xrightarrow{\tau} q'$. Hence, $\textsc{lhs} \xrightarrow{\tau} q'$ and so there exists $p(\equiv q)$ such that $\textsc{lhs} \stackrel{\varepsilon}{\Rightarrow} p$ and $\mathcal{S}(p) \subseteq \mathcal{S}(q)$. □

Now, we are only left with a sketch of proof of Lemma 4.6.

Proof of Lemma 4.6. Firstly, we show how to construct the disjoint extension G' of G required by the lemma. After removing all τ-rules from R, the set of rules for G, we divide the resulting set into R_a, the set of all non-τ-introducing rules, and R_τ, the set of all τ-introducing rules. Next, we split R_a and R_τ into $R_{a1}, \ldots, R_{an}$ and $R_{\tau 1}, \ldots, R_{\tau m}$ respectively in such a way that for any two rules r and r$'$ in R_a (R_τ) we have r, r$' \in R_{ai}$ ($R_{\tau j}$) whenever r and r$'$ have the same active arguments. Each of the resulting sets contains rules which, together with the associated τ-rules, define an operator which satisfies the first requirement of the distinctiveness (for each active argument either all or none of the rules test it). However, it may happen that in some set there are different rules but with the same trigger. Hence, we split further each of R_{ai} and $R_{\tau j}$ into $R_{ai1}, \ldots, R_{ain(i)}$ and $R_{\tau j1}, \ldots, R_{\tau jm(j)}$ respectively in a way that none of the created sets has such rules. Now, it is clear that each of the new sets, together with the associated τ-rules, defines a distinctive operator. Next, we change the name of the operator in each of the rules in the sets $R_{a11}, \ldots, R_{a1n(1)}, \ldots, R_{an1}, \ldots, R_{ann(n)}$ to $f_1, \ldots, f_k$ respectively, for some $k \in \mathbb{N}$. It is easily checked that each f_i is distinctive and non-τ-introducing. We also change the name of operator in each of the rules in $R_{\tau 11}, \ldots, R_{\tau 1m(1)}, \ldots, R_{\tau m1}, \ldots, R_{\tau mm(m)}$ to $g_1, \ldots, g_l$ respectively, for some $l \in \mathbb{N}$. It can be shown that each g_l is distinctive and τ-introducing. The signature of G' is defined by $\Sigma_{G'} = \Sigma_G \cup \{(f_i, n) \mid i \in K\} \cup \{(g_j, n) \mid j \in L\}$, where $K = \{1, \ldots, k\}$ and $J = \{1, \ldots, l\}$. The set of rules for G' is obtained by extending the set of rule for G with $R_{ai1}, \ldots, R_{ain(i)}$, for all appropriate i, then with $R_{\tau j1}, \ldots, R_{\tau jm(j)}$, for all appropriate j, and finally with all the τ-rules associated with the operators in $\Sigma_{G'}$. We easily check that G' is a disjoint extension of G.

Finally, laws (11) and (12) are proved sound in a similar fashion as laws (9) and (10) respectively by a case analysis based on the conditions of the laws. This is possible since the conditions of the corresponding laws are identical or very similar. □

B Completeness Proofs

Lemma B.1 Well-founded τDeS process languages are finitely branching.

Proof. Well-foundedness of a τDeS language and the fact that the basic transition system generated by the language is finitely branching [BIM88, BIM95] imply that the relevant derived transition system, and thus the language itself, is also finitely branching. □

We will show that, given any well-founded self-encoding τDeS process language G, the disjoint extension G' of G produced by the algorithm in Figure 3 is also well-founded. The following lemma will be useful.

Lemma B.2 Let G be a τDeS language with the set of visible actions A such that $\mathsf{B}(A) \leq G$. Assume that T, a theory that extends the axiom system for $\mathsf{B}(A)$, is head normalising and that $\mathsf{Test}(G) \models \mathrm{T}$. Then, for each well-founded Σ_G term p there exists a $\Sigma_{\mathsf{B}(A)}$ term q such that $\mathrm{T} \vdash p = q$.

Proof. We use induction on the action depth of process terms: this is well defined since well-founded τDeS process terms are finitely branching. Since T is head normalising we have $\mathrm{T} \vdash p = \bigoplus_{B \in \mathcal{A}} \sum_{a \in B} a.p_a$ or $\mathrm{T} \vdash p = \sum_{a \in B} a.p_a \boxplus \Omega$, for some $B \subseteq A$. One can easily show that $\simeq$ preserves the action depth of well-founded process terms. Thus, as the action depth of each p_a is smaller than that of p we obtain, by inductive hypothesis, a $\Sigma_{\mathsf{B}(A)}$ term q_a such that $\mathrm{T} \vdash p_a = q_a$. We let $q \equiv \bigoplus_{B \in \mathcal{A}} \sum_{a \in B} a.q_a$ or $q \equiv \sum_{a \in B} a.q_a \boxplus \Omega$ depending on the form of p. By the substitutivity of prefixing and the choice operators it is clear that $\mathrm{T} \vdash p = q$. □

Lemma B.3 Suppose G is a well-founded self-encoding τDeS process language. Then the disjoint extension G' of G produced by algorithm A is also well-founded.

Proof. Let G and G' be τDeS languages as in the lemma with the set of actions A. We will show by structural induction that every $\Sigma_{G'}$ term is well-founded. Assume $p \equiv h(\boldsymbol{p})$. There are two cases depending on the origin of h.

1. $h \in \Sigma_G$. By the inductive hypothesis, all p_j of $\boldsymbol{p}$ are well-founded, thus finitely branching. By Lemma B.2 and the soundness theorem we deduce that for each p_j there exists a $\Sigma_{\mathsf{B}(A)}$ term q_j such that $p_j \simeq q_j$. Since $\simeq$ is a congruence for all τDeS languages we get $h(\boldsymbol{p}) \simeq h(\boldsymbol{q})$. Moreover, $h(\boldsymbol{q})$ is a Σ_G term so it is well-founded. But, as $\simeq$ preserves well-foundedness we deduce that $h(\boldsymbol{p})$ is also well-founded.

2. $h \in \Sigma_{G'} \setminus \Sigma_G$, so h results from applying Lemma 4.6. Let q be a α-descendant of $h(\boldsymbol{p})$ obtained by applying some De Simone rule, say r, after possibly a number of applications of τ-rules for h. Thus, $h(\boldsymbol{p}) \stackrel{\varepsilon}{\Rightarrow} h(\boldsymbol{p'}) \stackrel{\alpha}{\rightarrow} q$. Hence, there is a ground $\Sigma_{G'}$ substitution ρ such that $\rho(\mathit{cons}(r)) \equiv \rho(h(\boldsymbol{X}) \stackrel{\alpha}{\rightarrow} C[\boldsymbol{Y'}]) \equiv h(\boldsymbol{p'}) \stackrel{\alpha}{\rightarrow} C[\boldsymbol{q}]$ with $C[\boldsymbol{q}] \equiv q$. This implies that $\mathit{ante}(r)$ is valid, i.e. $\rho(X_i \stackrel{a_i}{\rightarrow} X_i') \equiv p_i' \stackrel{a_i}{\rightarrow} q_i$, for all active i, so $q_i \equiv p_i'$ for non-active i. Note that, by the construction of the rules for h (as described in the proof of Lemma 4.6), $C[\boldsymbol{Y'}]$ is some Σ_G context. Using the inductive hypothesis we find that all p_i, and thus all q_i, are well-founded. Next, by a similar argument as in case 1, there are $\Sigma_{\mathsf{B}(A)}$ terms u_i such that $u_i \simeq q_i$. Since $\simeq$ is a congruence for the language G' we get $q \equiv C[\boldsymbol{q}] \simeq C[\boldsymbol{u}]$. But, $C[\boldsymbol{u}]$ is a Σ_G term, so it is well-founded. Since $\simeq$ preserves well-foundedness we deduce that q is also well-founded. As the above result is for any α-descendant of $h(\boldsymbol{p})$, it follows that $h(\boldsymbol{p})$ itself is well-founded.

This completes the proof. □

Proof of Theorem 4.10. Let A be the set of visible actions of G'. Assume that for any $\Sigma_{G'}$ terms p and q we have $\mathsf{Test}(G') \vDash p \sqsubseteq q$. By Lemma refw-f-dis-ext p and q are well-founded, hence initially τ-convergent. Thus, T is head normalising for p and q. Hence, using Lemma B.2, we find $\Sigma_{\mathsf{B}(A)}$ terms p' and q' such that $\mathrm{T} \vdash p = p'$ and $\mathrm{T} \vdash q = q'$ respectively. The soundness theorem gives $\mathsf{Test}(G') \vDash p = p'$ and $\mathsf{Test}(G') \vDash q = q'$. Thus, $\mathsf{Test}(G') \vDash p' \sqsubseteq q'$. Finally, by completeness of the axiom system for the language $\mathsf{B}(A)$, we obtain $\mathrm{T} \vdash p' \sqsubseteq q'$, and thus $\mathrm{T} \vdash p \sqsubseteq q$ as required. □

The following lemma describes some of the properties of the operator $/$.

Lemma B.4 Given a process p, a well-founded process q over G' such that $G_/ \leq G'$ and the action prefixing with $b \in \Sigma_{G'}$ we have

- $p/b^n\Omega \ll p$,
- if $q \ll p$ then $q \ll p/b^n\Omega$, for some $n \in \mathbb{N}$.

We show that ω-Induction is sound for all τDeS process languages.

Theorem B.5 Suppose that G is a τDeS language. Then, $\mathsf{TEST}(G_/) \vDash$ ω-Induction.

Proof. Let G' be τDeS language such that $G_/ \leq G'$. Let P, Q be open terms over $\Sigma_{G'}$ and let ρ be a $\Sigma_{G'}$ ground substitution. Assume $\rho(P/b^n\Omega) \ll \rho(Q)$ for all $n \in \mathbb{N}$ and some action b. We need to prove $\rho(P) \ll \rho(Q)$. We only show $\rho(P) \ll_{\mathsf{must}} \rho(Q)$, the may part follows similarly. Let s be a sequence of visible actions over G'. Also, let $\rho(P)\Downarrow s$. Clearly

$\rho(P/b^m\Omega)\Downarrow s$, where m is greater than the length of s. By the assumption above ($\rho(P/b^n\Omega) \ll \rho(Q)$) we obtain $\rho(Q)\Downarrow s$. Next, assume $\rho(P)\Downarrow s$ and $\rho(Q) \stackrel{s}{\Rightarrow} q'$, where $q' = \rho(Q')$ for some open term Q' over $\Sigma_{G'}$. Again using the above assumption we deduce $\rho(P/b^m\Omega) \stackrel{s}{\Rightarrow} p'$, where m is greater than the length of s, and $\mathcal{S}(p') \subseteq \mathcal{S}(q')$. Moreover, the term p' has the form $\rho(P'/b^k\Omega)$ for some $k > 0$. Thus, $\rho(P) \stackrel{s}{\Rightarrow} \rho(P')$. Since $\rho(P')$ and $\rho(P'/b^k\Omega)$ have the same initial actions we obtain $\mathcal{S}(\rho(P')) \subseteq \mathcal{S}(q')$ as required. □

Proof of Theorem 4.11. The soundness part follows immediately from Theorems 4.7 and B.5. For the completeness part, assume that p and q are τ-convergent $\Sigma_{G'}$ closed terms and $\mathsf{Test}(G') \vDash p \sqsubseteq q$. The last means $p \ll q$. Also, let A be the set of visible actions of G'. By Lemma B.4 we have $p/b^m\Omega \ll q$, for all $m \in \mathbb{N}$. Since p is τ-convergent $p/b^m\Omega$ is well-founded. Thus, by Lemma B.2 there exists, for each m, a $\Sigma_{\mathsf{B}(A)}$ term p_m such that T $\vdash p_m = p/b^m\Omega$. By soundness we obtain $p_m \simeq p/b^m\Omega$, hence $p_m \ll q$. Since p_m is well-founded there is, by Lemma B.4, $k \in \mathbb{N}$ such that $p_m \ll q/b^k\Omega$. Since q is τ-convergent $q/b^k\Omega$ is well-founded, so by Lemma B.4 we can construct, for each k, a $\Sigma_{\mathsf{B}(A)}$ term q_k such that T $\vdash q_k = q/b^k\Omega$. By soundness we obtain $q_k \simeq q/b^k\Omega$, hence $p_m \ll q_k$. But, both p_m and q_k are Σ_{B} terms, so by the completeness result for B we have T $\vdash p_m \sqsubseteq q_k$, Thus, T $\vdash p/b^m\Omega \sqsubseteq q/b^k\Omega$. Since T $\vdash q/b^k\Omega \sqsubseteq q$ we get T $\vdash p/b^m\Omega \sqsubseteq q$. Finally, using ω-Induction rule we obtain T, ω-Induction $\vdash p \sqsubseteq q$. □

References

[ABV92] L. Aceto, B. Bloom, and F.W. Vaandrager. Turning SOS rules into equations. In *Proceedings of the 7th Annual IEEE Symposium on Logic in Computer Science*, Santa Cruz, California, 1992.

[ABV94] L. Aceto, B. Bloom, and F.W. Vaandrager. Turning SOS rules into equations. *Information and Computation*, 111:1–52, 1994.

[Ace94] L. Aceto. Deriving complete inteference systems for a class of GSOS languages generating regular behaviours. In *Proceedings of the 5th International Conference* CONCUR'94, Uppsala, Sweden, 1994. LNCS 789.

[AI95] L. Aceto and A. Ingólfdóttir. CPO models for a class of GSOS languages. In P.D. Mosses, M. Nielsen, and M.I. Schwartzbach, editors, *Proceedings of TAPSOFT '95*. Springer-Verlag, 1995. LNCS 915.

[BIM88] B. Bloom, S. Istrail, and A.R. Meyer. Bisimulation can't be traced: preliminary report. In *Conference Record of the 15th ACM Symposium on Principles of Programming Languages*, San Diego, California, 1988.

[BIM95] B. Bloom, S. Istrail, and A.R. Meyer. Bisimulation can't be traced. *Journal of ACM*, 42(1), 1995. Also appeared as Technical Report TR 90-1150, Cornell, 1990.

[BK84] J.A. Bergstra and J.W. Klop. Process algebra for synchronous communication. *Information and Computation*, 60:109–137, 1984.

[BK85] J.A. Bergstra and J.W. Klop. Algebra of communicating processes with abstraction. *Theoretical Computer Science*, 37, 1985.

[BK88] J.A. Bergstra and J.W. Klop. A complete inference system for regular processes with silent moves. In F.R. Drake and J.K. Truss, editors, *Logic Colloquium '86*, pages 21–81. North-Holland, 1988. First appeared as Report CS-R8420, CWI, 1984.

[Blo90a] B. Bloom. *Ready simulation, bisimulation, and the semantics of CCS-like languages.* PhD thesis, MIT, 1990.

[Blo90b] B. Bloom. Strong process equivalence in the presence of hidden moves. Preliminary report, MIT, 1990.

[Blo95a] B. Bloom. Structural operational semantics for weak bisimulations. *Theoretical Computer Science*, 146:27–68, 1995. Also appeared as Technical Report TR 93-1373, Cornell, 1993.

[Blo95b] B. Bloom. Stuctured operational semantics as a specification language. In *Conference Record of the 22nd ACM Symposium on Principles of Programming Languages*, San Francisco, California, 1995.

[BW90] J.C.M Baeten and W.P Weijland. *Process Algebra*. Cambridge Tracts in Theoretical Computer Science, 1990.

[CH90] R. Cleaveland and M. Hennessy. Priorities in process algebra. *Information and Computation*, 87:58–77, 1990.

[dS85] R. de Simone. Higher-level synchronising devices in MEIJE-SCCS. *Theoretical Computer Science*, 37:245–267, 1985.

[Gro90] J.F. Groote. Transition system specifications with negative premises. In J.C.M. Baeten and J.W. Klop, editors, *Proceedings of* CONCUR'90, Amsterdam, 1990. Springer-Verlag. LNCS 458.

[GV90] J.F. Groote and F. Vaandrager. Structured operational semantics and bisimulation as a congruence. *Information and Computation*, 100:202–260, 1990.

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

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[Mil84] R. Milner. A complete inference system for a class of regular behaviours. *Journal of Computer System Sciences*, 28:439–466, 1984.

[Mil89a] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[Mil89b] R. Milner. A complete axiomatisation for observational congruence of finite-state behaviours. *Information and Computation*, 81:227–247, 1989.

[Mol90] F. Moller. The importance of the left merge operator in process algebras. In M. Paterson, editor, *Proceedings of 17th* ICALP, *Warwick*. Springer-Verlag, 1990. LNCS 443.

[NH84] R. De Nicola and M. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.

[NH87] R. De Nicola and M. Hennessy. CCS without τ's. In H. Ehrig, R. Kowalski, G. Levi, and U. Montanari, editors, *TAPSOFT '87*, Berlin, 1987. Springer-Verlag. LNCS 250.

[Nic87] R. De Nicola. Extensional equivalences for transition systems. *Acta Informatica*, 24:212–237, 1987.

[PU96] I.C.C. Phillips and I. Ulidowski. Ordered SOS rules and weak bisimulation. In A. Adalat, S. Jourdan, and G. McCusker, editors, *Theory and Formal Methods 1996*, London, 1996. Imperial College Press.

[Uli92] I. Ulidowski. Equivalences on observable processes. In *Proceedings of the 7th Annual IEEE Symposium on Logic in Computer Science*, Santa Cruz, California, 1992.

[Uli94] I. Ulidowski. *Local Testing and Implementable Concurrent Processes*. PhD thesis, Imperial College, University of London, 1994.

[Uli95] I. Ulidowski. Axiomatisations of weak equivalences for De Simone languages. In I. Lee and S.A. Smolka, editors, *Proceedings of the 6th International Conference on Concurrency Theory* CONCUR'95, Philadelphia, 1995. Springer-Verlag. LNCS 962.

[Vaa91] F.W. Vaandrager. On the relationship between process algebra and input/output automata. In *Proceedings of the 6th Annual IEEE Symposium on Logic in Computer Science*, Amsterdam, 1991.

[vG90] R.J. van Glabbeek. *Comparative Concurrency Semantics and Refinement of Actions*. PhD thesis, CWI, 1990.

[vG93] R.J. van Glabbeek. A complete axiomatisation for branching bisimulation congruence of finite-state behaviours. In A.M. Borzyszkowski and S. Sokołowski, editors, *Proceedings of MFCS 1993*. Springer-Verlag, 1993. LNCS 711.

Figure 4: Axioms for CCS parallel composition.

$$(X \oplus Y) \parallel Z = X \parallel Z \;\oplus\; Y \parallel Z$$

$$X \parallel (Y \oplus Z) = X \parallel Y \;\oplus\; X \parallel Z$$

$$AX \parallel \Omega = AX \lfloor\!\lfloor \Omega \;\boxplus\; \Omega \lfloor\!\lfloor AX \boxplus\; AX \mid \Omega$$

$$\Omega \parallel AX = \Omega \lfloor\!\lfloor AX \;\boxplus\; AX \lfloor\!\lfloor \Omega \boxplus\; \Omega \mid AX$$

$$AX \parallel BY = AX \lfloor\!\lfloor BY \;\boxplus\; BY \lfloor\!\lfloor AX \boxplus\; AX \mid BY \qquad \text{if } A \cap \bar{B} = \emptyset$$

$$AX \parallel BY = AX \lfloor\!\lfloor BY \;\boxplus\; BY \lfloor\!\lfloor AX \oplus\; AX \mid BY \qquad \text{if } A \cap \bar{B} \neq \emptyset$$

$$(X \oplus Y) \lfloor\!\lfloor Z = X \lfloor\!\lfloor Z \;\oplus\; Y \lfloor\!\lfloor Z$$

$$(\sum_A a.X_a) \lfloor\!\lfloor Y = \sum_A (a.X_a \lfloor\!\lfloor Y)$$

$$a.X \lfloor\!\lfloor_A Y = a.(X \parallel Y)$$

$$\mathbf{0} \lfloor\!\lfloor X = \mathbf{0}$$

$$\Omega \lfloor\!\lfloor X \sqsubseteq \Omega$$

$$(X \oplus Y) \mid Z = X \mid Z \;\oplus\; Y \mid Z$$

$$X \mid (Y \oplus Z) = X \mid Y \;\oplus\; X \mid Z$$

$$(\sum_A a.X_a) \mid BY = \sum_A (a.X_a \mid BY) \qquad \text{if } A \cap \bar{B} \neq \emptyset$$

$$AX \mid (\sum_B bY_b) = \sum_B (AX \mid bY_b) \qquad \text{if } A \cap \bar{B} \neq \emptyset$$

$$(\sum_A a.X_a) \mid BY = \sum_{A \setminus C} (a.X_a \mid BY) \oplus \bigoplus_C (c.X_c \mid BY) \qquad \text{if } A \cap \bar{B} = C \neq \emptyset$$

$$AX \mid (\sum_B bY_b) = \sum_{B \setminus C} (AX \mid bY_b) \oplus \bigoplus_C (AX \mid cX_c) \qquad \text{if } \bar{A} \cap B = C \neq \emptyset$$

$$a.X \mid \bar{a}.Y = X \parallel Y$$

$$a.X \mid b.Y = \mathbf{0} \qquad \text{if } a \neq \bar{b}$$

$$\mathbf{0} \mid X = \mathbf{0}$$

$$X \mid \mathbf{0} = \mathbf{0}$$

$$\Omega \mid X \sqsubseteq \Omega$$

$$X \mid \Omega \sqsubseteq \Omega$$