# Finite Axiom Systems for Testing Preorder and De Simone Process Languages

Irek Ulidowski*

Research Institute for Mathematical Sciences,
Kyoto University,
Kitashirakawa, Sakyo-ku, Kyoto 606-01, Japan

**Abstract**

We prove that testing preorder of De Nicola and Hennessy is preserved by all De Simone process operators. Building upon 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 written more easily in customised De Simone process languages which contain both the standard operators as well as new De Simone operators specially tailored for the task in hand. 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 concurrent systems. Both specifications, of varying level of abstractness, and the implementation of a concurrent system are expressed as processes (process terms) in a process language. We verify that the specifications and the implementation describe in fact the same system by proving that the concrete specifications refine the abstract specifications, and that the implementation refines one of the concrete specifications. There are several 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 specifications with the concrete specifications, and one of the the concrete specifications with the implementation. The third proof method is solely based on the syntax of processes and 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

*The address for correspondence is School of Informatics and Multimedia Technology, University of North London, 166–220 Holloway Road, London N7 8DB, U.K., Email I.Ulidowski@unl.ac.uk.

and verification of concurrent systems was introduced. More recently, benefiting from *Structured Operational Semantics* (SOS) approach of Plotkin [Plo81], these particular process languages and their numerous extensions were generalised 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, GV92, Gro93, Vaa91, Uli92, Blo95a, Uli94, UP97]. Axiomatic systems and denotational models for these classes of process languages were proposed in [ABV92, ABV94, Ace94, Uli95, Uli96] and [AI95] respectively. Also, two classes of languages were used in [Blo95b] as general specification languages. Moreover, uniform methods for extending general process languages with the notion of *time* were proposed in [NS91, UY97b, UY97a].

This paper continues the research concerning general process languages. We consider process languages generated by the De Simone format of rules [dS85] with a special treatment of silent actions; these languages are simply called $\tau$DeS process languages. The class of $\tau$DeS languages was developed in [Uli94, Uli95, Uli96]. 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 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 $\tau$DeS languages. This result is the foundation on which our axiomatisation algorithm is based. It guarantees *compositionality* of specification and verification when working with $\tau$DeS 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 in our specifications we intend to use new, customised $\tau$DeS operators we need to be able to derive automatically axiomatisations for such operators so that the verification can be performed. The second contribution of our paper is an algorithm which produces sound and complete axiomatisations of testing preorder for $\tau$DeS languages. In contrast to the axiomatisations of refusal simulation preorder for well-founded $\tau$DeS languages [Uli95], the axiomatisations of testing preorder for the same class of languages are *finite*. The finiteness of our axiomatisations is achieved by assuming that the sets of actions of the considered languages are finite, and by extending the languages with useful, and indeed required [Mol90], auxiliary operators. Our axiomatisation algorithm is a modification of the original algorithm developed by Aceto, Bloom and Vaandrager [ABV92, ABV94] for strong bisimulation and GSOS process languages. The latter algorithm is in turn a generalisation of the technique used by Bergstra and Klop [BK84, BK85] to derive a finite axiomatisation for the parallel composition operator by using the auxiliary left-merge operator.

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 actions with priorities [CH90] may help. However, clear and intuitive specifications of such systems can be produced by using new operators which are invented for that purpose. We give both abstract and concrete specifications of these systems, and then use axiomatisations generated by our algorithm to show that the specifications are equivalent.

The paper is organised as follows. We recall, in Section 2, the definitions of $\tau$DeS process languages, labelled transition systems and testing preorder. In Section 3 we show that (a characterisation of) testing preorder is a precongruence for any $\tau$DeS language. We begin Section 4 by restating a completeness result for testing preorder and a simple $\tau$DeS language. Then, we present an algorithm for generating sound axiomatisations for arbitrary $\tau$DeS languages. These axiomatisations are proved to be finite and head normalising for certain classes of $\tau$DeS languages. The section finishes with a completeness result. In Section 5 we give two examples which illustrate the application of our work. The last section contains a conclusion and a discussion of related work. Several proofs of essential results concerning alternative characterisation of testing preorders, ruloids, soundness and completeness of axiom systems produced by our algorithm appear in Appendices A–D.

# 2 Basic Concepts

In this section we recall the definitions of $\tau$DeS 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, GV92, Uli95, Uli92].

Let $\mathsf{Var}$ be a countable set of variables ranged over by $X, Y, \ldots$. A signature $\Sigma$ 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 $\Sigma$ with variables in $V$ is denoted by $\mathbb{T}(\Sigma, V)$. $\mathbb{T}(\Sigma, \mathsf{Var})$, abbreviated by $\mathbb{T}(\Sigma)$, is ranged over by $P, Q, \ldots$. The set of closed terms $\mathrm{T}(\Sigma)$ is ranged over by $p, q, t, \ldots$. A $\Sigma$ context with $n$ holes $C[X_1, \ldots, X_n]$, for $n \in \mathbb{N}$, 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 $\Sigma$ 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 $\Sigma$ contexts $C[\boldsymbol{X}]$ we have $\boldsymbol{t} \sqsubseteq \boldsymbol{t'}$ implies $C[\boldsymbol{t}] \sqsubseteq C[\boldsymbol{t'}]$. Similarly, an operator $f \in \Sigma_n$ preserves $\sqsubseteq$ if, $\boldsymbol{p} \sqsubseteq \boldsymbol{q}$ implies $f(\boldsymbol{p}) \sqsubseteq f(\boldsymbol{q})$. Consequently, a format preserves $\sqsubseteq$ if all operators definable in the format preserve $\sqsubseteq$.

A (closed) *substitution* $\rho$ is a mapping from $\mathsf{Var}$ to $(\mathrm{T}(\Sigma))$ $\mathbb{T}(\Sigma)$. Substitutions extend to mappings 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 over by $\alpha, \ldots$, is a finite set of actions, where $\mathsf{Act}$ consists of visible actions $a, b, \ldots$, and $\tau \notin \mathsf{Act}$ is the silent action. We will also consider special actions $w$ and $1$ not in $\mathsf{Act} \cup \{\tau\}$, which will be used to encode tests. Let $\mathsf{Vis}$ be $\mathsf{Act} \cup \{w, 1\}$ and $A, B, \ldots$ range over $\mathsf{Vis}$.

## 2.1 $\tau$DeS Process Languages

We define the $\tau$DeS format of transition rules and $\tau$DeS process languages.

**Definition 2.1** [Uli95] A De Simone (transition) rule for $(f, n)$ is an expression of the form

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

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

A $\tau$-rule for the above De Simone rule is any expression of the following form, where $i \in I$.

$$\frac{X_i \stackrel{\tau}{\to} X_i'}{f(X_1,\ldots,X_i,\ldots,X_n) \stackrel{\tau}{\to} f(X_1,\ldots,X_i',\ldots,X_n)}$$

The set of all $\tau$-rules for rule (1) is called its set of *associated* $\tau$-rules. A set of rules is in the De Simone format, abbreviated as the $\tau$DeS format, if it consists of De Simone rules, their associated $\tau$-rules and no other rules. A process operator is $\tau$DeS if the set of its defining rules is in the $\tau$DeS format. The set of associated $\tau$-rules for a $\tau$DeS operator consists of all $\tau$-rules associated with all De Simone rules defining the operator.

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

**Definition 2.2** Let $(\Sigma, A \cup \{\tau\}, R)$ be $\tau$DeS language. Given $f \in \Sigma$, $\mathit{rules}(f) \subseteq R$ and $\mathit{rules}(f,\alpha) \subseteq R$ are the sets of rules for $f$ and rules for $f$ with the action $\alpha$ respectively. Assume $r \in R$ has the form (1). Then, $\{X_i \stackrel{a_i}{\to} X_i'\}_{i\in I}$ is the set of *antecedents* of $r$, written as $\mathit{ante}(r)$, and $f(X_1,\ldots,X_n) \stackrel{\alpha}{\to} C[\boldsymbol{Y}]$ is the *consequent*, written as $\mathit{cons}(r)$. In general, expressions $T \stackrel{\alpha}{\to} T'$ with $T, T' \in \mathbb{T}(\Sigma)$ and $\alpha \in A \cup \{\tau\}$ are called *transitions*. $T$ is the *source* and $T'$ is the *target* of $T \stackrel{\alpha}{\to} T'$. Moreover, $f(\boldsymbol{X})$ is the source of $r$, written as $\mathit{source}(r)$, and $C[\boldsymbol{Y}]$ is the target of $r$, written as $\mathit{target}(r)$. $f$ is the *operator* of $r$ and $\alpha$ is its *action*. The *trigger* of $r$, written $\mathit{trigger}(r)$, is an $n$-tuple $(\lambda_1,\ldots,\lambda_n)$, where $\lambda_i = a_i$ if $i \in I$, and $\lambda_i = *$ otherwise. The $\alpha$ trigger for $f$ is $\mathit{trigger}(r)$ for $r \in \mathit{rules}(f,\alpha)$. We will use $\mathit{trig}_\alpha(f)$ for the set of all $\alpha$ triggers for $f$. We say that the $i$-th argument, $X_i$, is active in $r$ if $X_i \stackrel{a_i}{\to} X_i' \in \mathit{ante}(r)$ for some $a_i$ and $X_i'$. $\mathit{active}(r)$ is the set of all $i$ such that $X_i$ is active in $r$. Moreover, we say that the $i$-th argument of $f$ is *active* if $i \in \mathit{active}(r)$ for some De Simone rule $r$ for $f$.

**Definition 2.3** A De Simone process language, or simply $\tau$DeS process language, is a triple $(\Sigma, A \cup \{\tau\}, R)$, where $\Sigma$ is a finite set of $\tau$DeS operators, $A \subseteq \mathsf{Vis}$ and $R$ is a finite set of defining rules for the operators in $\Sigma$ such that, for each rule $r$ in $R$, the target of $r$ is some $\Sigma$ context and all actions in $r$ belong to $A \cup \{\tau\}$ .

Most of the operators in process languages are $\tau$DeS 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 $\tau$-rules. The $\tau$DeS format is a subformat of the GSOS format [BIM88, BIM95, Blo90a], where GSOS rules are the same as $\tau$DeS rules except that they may have *negative antecedents* (expressions like $X \stackrel{a}{\not\to}$) and *copying* (multiple use of identical process variables). Thus, $\tau$DeS format enjoys all the properties of the GSOS format. In particular, $\tau$DeS operators preserve (strong) bisimulation [BIM88, BIM95]. Moreover, $\tau$DeS is a subformat of the ISOS format [Uli92, Uli94]. Since a version of weak bisimulation studied in [Mil81, Abr87, Wal90, Uli94, UP97] is preserved by ISOS operators [Uli94] it is also preserved by $\tau$DeS operators.

## 2.2 Labelled Transition Systems

Next, we define labelled transition systems for $\tau$DeS languages. We will use a special nullary operator $\Omega$, which has no defining rules, to represent divergence of processes due to under-specification (lack of information about behaviour). Later, we will also consider another form of divergence, namely the ability to perform an infinite sequence of silent actions.

**Definition 2.4** Let $G = (\Sigma_G, A_G \cup \{\tau\}, R_G)$ be a $\tau$DeS language. The *labelled transition system with divergence* for $G$ is a structure $(\mathrm{T}(\Sigma_G), A_G \cup \{\tau\}, \rightarrow_G, \uparrow_G)$, where $\mathrm{T}(\Sigma_G)$ is a set of *processes* or process *terms*, $\rightarrow_G \subseteq \mathrm{T}(\Sigma_G) \times A_G \cup \{\tau\} \times \mathrm{T}(\Sigma_G)$ is the unique *transition relation* generated by $G$ and $\uparrow_G \subseteq \mathrm{T}(\Sigma_G)$ is the (postfix) *divergence predicate*. $\uparrow_G$ is defined as the least relation which satisfies (a) $\Omega\uparrow_G$ if $\Omega \in \Sigma_G$, and i (b) for all $f \in \Sigma_G$ and $\boldsymbol{p}$ if $p_i\uparrow_G$, for $i \in active(f)$, then $f(\boldsymbol{p})\uparrow_G$.

The above transition system will be called the *basic* transition system for $G$, and the subscript $G$ will be omitted when it is obvious from the context. The full details of how to construct the transition relation for a process language defined by a general format of rules can be found in, for example, [BIM88, BIM95, GV92]. We will use the following abbreviations: Expression $p \xrightarrow{a} q$ denotes $(p, a, q) \in \rightarrow$ and $p \xrightarrow{a}$ is equivalent to $p \xrightarrow{a} q$ for some $q \in \mathrm{T}(\Sigma)$. Moreover, $p \rightarrow q$ means $p \xrightarrow{a} q$ for some $a \in \mathsf{Vis} \cup \{\tau\}$. $p \xrightarrow{t}$ stands for $p \xrightarrow{a_1} \ldots \xrightarrow{a_n}$, where $t = a_1 \ldots a_n \in A^*$. $\varepsilon$ is the empty sequence in $A^*$ and $s \leq t$, for $s, t \in A^*$, means that $s$ is an initial subsequence of $t$. The concatenation of sequences $s$ and $u$ is denoted by $su$. The set of $t$ *descendants* of $p$, $d(p,t)$, is defined as $\{p' \mid p \xrightarrow{t} p'\}$. With the notation as in the previous definition, we say that transition $t \xrightarrow{\alpha} t'$, for $t, t' \in \mathrm{T}(\Sigma_G)$, is *valid* (*holds*) in $\rightarrow_G$ if $(t, \alpha, t') \in \rightarrow_G$.

Next, we define the transition system which abstracts away the silent actions and captures divergence due to infinite sequences of silent actions.

**Definition 2.5** Given the transition system $(\mathrm{T}(\Sigma), A \cup \{\tau\}, \rightarrow, \uparrow)$ for a $\tau$DeS language $G$, the structure $(\mathrm{T}(\Sigma), A \cup \{\varepsilon\}, \Rightarrow, \Uparrow)$ is the *derived* transition system (with divergence) for $G$. The transition relation $\Rightarrow \subseteq \mathrm{T}(\Sigma) \times A \cup \{\varepsilon\} \times \mathrm{T}(\Sigma)$ and the divergence predicate $\Uparrow \subseteq \mathrm{T}(\Sigma) \times A^*$ are defined as follows, where $\xrightarrow{\tau^*}$ is the reflexive and transitive closure of $\xrightarrow{\tau}$:

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

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

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 $p \xrightarrow{a}$ and $p \xrightarrow{t}$. Expression $p \Rightarrow q$ means $p \stackrel{\alpha}{\Rightarrow} q$, for some $\alpha \in \mathsf{Vis} \cup \{\varepsilon\}$. The *language* of $p$, written as $\mathcal{L}(p)$, is $\{s \mid p \stackrel{s}{\Rightarrow}\}$. $\mathcal{S}(p)$, the set of initial visible actions a process $p$ can perform, is $\{a \mid p \stackrel{a}{\Rightarrow}\}$. The set of $t$ descendants of $p$ in the derived transition system, $\mathcal{D}(p,t)$, is defined as $\{p' \mid p \stackrel{t}{\Rightarrow} p'\}$.

We distinguish the following classes of $\tau$DeS process languages, and the classes of their processes, depending on the properties of transition systems they generate.

**Definition 2.6** Assume the basic and derived transition systems for an arbitrary $\tau$DeS 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 $p_i \xrightarrow{\alpha_i} p_{i+1}$ for all $i \geq 1$. A $\tau$DeS language is well-founded if and only if all

its processes are well-founded.
(b) A process $p$ is *finite branching* in the basic (derived) transition system if and only if $\{q \mid p' \to q\}$ ($\{q \mid p' \Rightarrow q\}$) is finite, for every $t \in \mathsf{Act}^*$ and $p'$ in $d(p,t)$ ($\mathcal{D}(p,t)$). A $\tau$DeS language is finite branching if and only if all its processes are finite branching in the derived transition system.
(c) A process $p$ is *$\tau$-divergent* if and only if $q \xrightarrow{\tau^\omega}$ for some $t \in \mathsf{Vis}^*$ and $q$ in $\mathcal{D}(p,t)$. A process $p$ is *initially $\tau$-divergent* if and only if $p \xrightarrow{\tau^\omega}$. A process is *$\tau$-convergent* (*initially $\tau$-convergent*) if and only if it is not $\tau$-divergent (not initially $\tau$-divergent). A $\tau$DeS language is $\tau$-convergent if and only if all its processes are $\tau$-convergent.

**Lemma 2.7** All well-founded $\tau$DeS process languages are $\tau$-convergent.

Since the $\tau$DeS 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 $\tau$DeS languages:

**Lemma 2.8** All processes over $\tau$DeS process languages are finite branching in their basic transition systems.

But, not all $\tau$DeS process languages are finite branching:

**Example 2.9** Consider a $\tau$DeS language with prefixing with $a$, the deadlocked process $\mathbf{0}$ and operators $g$ and $f$ defined by the following De Simone rules

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

together with the associated $\tau$-rules. Consider term $f(g,\mathbf{0})$. We have $f(g,\mathbf{0}) \xrightarrow{\tau} f(g,a.\mathbf{0}) \xrightarrow{\tau} \ldots \xrightarrow{\tau} f(g,a^n.\mathbf{0}) \xrightarrow{\tau}$ and $f(g,a^n.\mathbf{0}) \xrightarrow{\tau} a^n.\mathbf{0}$, for all $n$, where $a^n.\mathbf{0}$ means $\overbrace{a.\ldots.a.}^{n}\mathbf{0}$. Process $f(g,\mathbf{0})$ is not finite branching since $f(g,\mathbf{0}) \stackrel{\varepsilon}{\Rightarrow} a^n.\mathbf{0}$ for all $n$.

However, we have the following weaker result.

**Lemma 2.10** All $\tau$-convergent processes over $\tau$DeS process languages are finite branching.

**Proof.** Let $p$ be a $\tau$-convergent process over some $\tau$DeS process language. Assume $p' \in \mathcal{D}(p,t)$ for some $t$. We need to show $\{q \mid p' \Rightarrow q\}$ is finite. We have $\{q \mid p' \Rightarrow q\} = \bigcup_{a \in \mathsf{Vis}}\{q \mid p' \stackrel{a}{\Rightarrow} q\} \cup \{q \mid p' \stackrel{\varepsilon}{\Rightarrow} q\}$. It follows by Lemma 2.8 that $\{r' \mid r \stackrel{\varepsilon}{\Rightarrow} r'\}$ is finite for any initially $\tau$-convergent process $r$. Hence, $\{q \mid p' \stackrel{\varepsilon}{\Rightarrow} q\}$ is finite. We have $\{q \mid p' \stackrel{a}{\Rightarrow} q\} = \{q \mid p' \stackrel{\varepsilon}{\Rightarrow}\xrightarrow{a}\stackrel{\varepsilon}{\Rightarrow}\}$. So, by Lemma 2.8 and the fact that $\{r' \mid r \stackrel{\varepsilon}{\Rightarrow} r'\}$ is finite for any initially $\tau$-convergent $r$, we deduce that $\{q \mid p' \stackrel{a}{\Rightarrow} q\}$ is finite for each $a$. Hence, $\{q \mid p' \Rightarrow q\}$ is finite. ∎

## 2.3 Basic $\tau$DeS Language B

Throughout this paper we will consider a simple $\tau$DeS language $\mathsf{B}$ 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 $\tau$. $\mathsf{B}$ is a $\tau$DeS 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 $\Omega$, and $\mathbf{0}$ is sometimes omitted. Both of the operators hove no 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 $\mathsf{B}$ the basic $\tau$DeS process language. Sometime it will be convenient to consider sub-languages of $\mathsf{B}$ which only have some of the prefixing operators of $\mathsf{B}$. 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 $\Omega$. Clearly $\mathsf{B}(\mathsf{Act}) = \mathsf{B}$.

**Definition 2.11** A $\tau$DeS language $H = (\Sigma', A', R')$ is a *disjoint extension* of a $\tau$DeS language $G = (\Sigma, A, R)$ with $A, A' \subseteq \mathsf{Vis} \cup \{\tau\}$, written as $G \leq H$, if $\Sigma \subseteq \Sigma'$, $A \subseteq A'$, $R \subseteq R'$ and $R'$ does not contain any new rules for the operators in $\Sigma$.

**Definition 2.12** A $\tau$DeS process language $G$ with the set of visible actions $A$ is called *self-encoding* if $\mathsf{B}(A) \leq G$.

$\mathsf{B}$ generates the basic transition system $B = (\mathrm{T}(\Sigma), \mathsf{Act} \cup \{\tau\}, \to, \uparrow)$. Note that all processes in $\mathrm{T}(\Sigma)$ are well-founded and $\tau$-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 which satisfies $\Omega\Uparrow$ 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 processes that may perform special actions $w$ (indicating success) and 1. 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 $\tau$DeS process languages. For technical reasons we will limit our attention to a certain subclass of $\tau$DeS process languages:

**Definition 2.13** A $\tau$DeS process language with the set of actions $A$ is called *proper* if $A \subseteq \mathsf{Act} \cup \{\tau\}$.

Given any proper $\tau$DeS language $G = (\Sigma, A, R)$, we define the *testing* language for $G$, denoted by $\mathit{Test}(G)$, as follows: If $\mathit{Test}(G) = (\Sigma', A', R')$, then $A' = A \cup \{w, 1\}$ and $\Sigma' = \Sigma \cup \{(\mathbf{0}, 0)\} \cup \{(a, 1) \mid a \in A' \setminus \{\tau\}\} \cup \{(|, 2), (\boxplus, 2)\}$, where the $(a, 1)$'s are prefixing operators and $(|, 2)$ is the test application operator. $R'$ contains $R$ and all the rules for the new prefixing operators, the usual rules for $\boxplus$ and the following rules for the test application operator $|$.

$$\frac{X \xrightarrow{a} X' \quad Y \xrightarrow{a} Y'}{X \,|\, Y \xrightarrow{\tau} X' \,|\, Y'} \qquad \frac{Y \xrightarrow{1} Y'}{X \,|\, Y \xrightarrow{\tau} X \,|\, Y'} \qquad \frac{X \xrightarrow{\tau} X'}{X \,|\, Y \xrightarrow{\tau} X' \,|\, Y} \qquad \frac{Y \xrightarrow{\tau} Y'}{X \,|\, Y \xrightarrow{\tau} X \,|\, Y'}$$

Clearly, $G \leq \mathit{Test}(G)$. We define testing of processes over $G$ in the transition system generated by the testing language $\mathit{Test}(G)$. We assume that processes are the terms over $\Sigma$ and tests are the terms over $\Sigma'$. The tests are ranged over by $t, t', \ldots$. The application of a test $t$ to a process $p$ is written as $p \,|\, t$. The sequence $p_0 \,|\, t_0 \xrightarrow{\tau} p_1 \,|\, t_1 \ldots \xrightarrow{\tau} p_k \,|\, t_k \ldots$ is called a *computation* if it is maximal, i.e. it is infinite or it is finite with the last element $p_k \,|\, t_k$ such that $p_k \,|\, t_k \xrightarrow{\tau} p' \,|\, t'$ 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 2.14** Let $G$ be a proper $\tau$DeS process language and $\mathit{Test}(G)$ be its testing language. For a process $p$ and a test $t$

$$
\begin{array}{lll}
p \,\mathsf{may}\, t & \text{if} & p \,|\, t \xrightarrow{\tau^*} p' \,|\, t' \ \text{ and } \ t' \xrightarrow{w} \\
p \,\mathsf{must}\, t & \text{if} & \text{whenever } (p \,|\, t \equiv) p_0 \,|\, t_0 \xrightarrow{\tau} p_1 \,|\, t_1 \xrightarrow{\tau} \ldots \text{ is a computation then} \\
 & & \quad (a)\ \exists n \geq 0.\ t_n \xrightarrow{w}, \text{ and} \\
 & & \quad (b) \text{ if } \ (p_k \,|\, t_k)\!\uparrow \ \text{ then } t_{k'} \xrightarrow{w}, \text{ for some } k' \leq k
\end{array}
$$

For processes $p$ and $q$ and tests $t$

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

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 alternative characterisations of these preorders as described in [NH84, Hen88].

**Definition 2.15** With the notation as in the previous definition

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

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 recall that $\mathcal{A}(q, s)$ is $\{S(p') \mid p \overset{s}{\Rightarrow} p'\}$ 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 the expression $\mathcal{A}(q, s) \subset\subset \mathcal{A}(p, s)$ 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. The proof can be found in Appendix A.

**Theorem 2.16** (**Alternative Characterisation of Testing Preorders**) For all processes $p$ and $q$ over any proper $\tau$DeS process language

$$\begin{array}{lll} p \ll_{\mathsf{may}} q & \text{if and only if} & p \sqsubseteq_{\mathsf{may}} q \\ p \ll_{\mathsf{must}} q & \text{if and only if} & p \sqsubseteq_{\mathsf{must}} q \\ p \ll q & \text{if and only if} & p \sqsubseteq q \end{array}$$

In the rest of the paper we will use the following notation. We will write $\mathsf{Test}(G)$ for the quotient algebra of elements of $\mathbb{T}(\Sigma_G)$ factored by $\ll$. Hence, $\mathsf{Test}(G) \models P \sqsubseteq Q$ will mean $\rho(P) \ll \rho(Q)$ for all $P$ and $Q$ and all ground substitutions $\rho$ over $G$. Since we will talk about axioms which are preserved by taking disjoint extensions of $\tau$DeS languages we will need $\mathsf{TEST}(G)$, the 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

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

# 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 $\tau$DeS languages.

The proof of Theorem 3.1, similarly as the proofs of congruence theorems for other weak equivalences and classes of process languages [Blo95a, Uli94], is rather technical. Before we give the details we will show that $\ll$ is not in general preserved by non-$\tau$DeS operators. We consider only two classes of such operators: the operators defined by rules negative antecedents and the operators defined by rules with copying. Both of these features, namely negative premises and copying, appear in GSOS rules.

**Example 3.2** Consider the operator $\theta_{b>a}$ which gives a priority to action $b$ over action $a$. The defining rules are as follows:

$$\frac{X \xrightarrow{b} X'}{\theta_{b>a}(X) \xrightarrow{b} \theta_{b>a}(X')}\ b \neq a \qquad\qquad \frac{X \xrightarrow{a} X' \quad X \stackrel{b}{\nrightarrow}}{\theta_{b>a}(X) \xrightarrow{a} \theta_{b>a}(X')}$$

Because of the second rule has the negative antecedent $X \stackrel{b}{\nrightarrow}$ the rules for $\theta_{b>a}$ are not $\tau$DeS. Let $p = a.(b \boxplus a.\Omega) \boxplus a.a$ and $q = a.(b \boxplus a.\Omega) \boxplus a.a.\Omega$. 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 $\theta_{b>a}(q) \stackrel{aa}{\Longrightarrow} \Omega$.

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

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

Notice that because the target of the first rule has two copies of $X'$ the rules for $f$ are not $\tau$DeS. Let $p = a.b \boxplus a.c$ and $q = a.(b \boxplus c)$. Clearly $p \simeq q$ but not $f(p) \simeq f(q)$: $f(q) \stackrel{c}{\Rightarrow}$ but $f(p) \stackrel{c}{\not\Rightarrow}$.

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

Ruloids are expressions that are very similar to $\tau$DeS rules:

$$\frac{\{\ X_i \stackrel{a_i}{\to} X'_i\ \}_{i \in I}}{C(X_1, \ldots, X_n) \stackrel{\alpha}{\to} D[\boldsymbol{Y}]}$$

Unlike in $\tau$DeS rules, the consequent has the form $C[\boldsymbol{X}] \stackrel{\alpha}{\to} D[\boldsymbol{Y}]$, where both $C[\boldsymbol{X}]$ and $D[\boldsymbol{Y}]$ are contexts and $\boldsymbol{Y}$ is defined as in Definition 2.1. Active arguments of ruloids are defined in the same fashion as for $\tau$DeS rules. There are ruloids corresponding to $\tau$-rules, henceforth called *$\tau$-ruloids*. So, for each $i \in I$, the following ruloid is a $\tau$-ruloid associated with the above ruloid.

$$\frac{X_i \stackrel{\tau}{\to} X'_i}{C(X_1, \ldots, X_i, \ldots, X_n) \stackrel{\tau}{\to} C(X_1, \ldots, X'_i, \ldots, X_n)}$$

**Example 3.4** Consider actions $a$ and $b$ with $b \neq a$, and the context $(X \boxplus b.Y)\backslash a$, where $\backslash a$ is the usual CSP hiding operator that will be formally defined in Section 4.1. The ruloids for $(X \boxplus b.Y)\backslash a$ and action $b$, and their associated $\tau$-ruloids are as follows:

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

We have only one $\tau$-ruloid since only the argument $X$ is active.

Given a $\tau$DeS language, a context $C[\boldsymbol{p}]$ and an action $\alpha$ over the language, Ruloid Theorem says that it is possible to construct the set ruloids which describes all transitions $C[\boldsymbol{p}] \stackrel{\alpha}{\to} t$.

**Theorem 3.5** (**Ruloid Theorem**) Let $G = (\Sigma, A \cup \{\tau\}, R)$ be a $\tau$DeS language. Assume $C[\boldsymbol{X}] \notin \mathsf{Var}$ is a context over $\Sigma$ and $\alpha \in A \cup \{\tau\}$. Then, there exists a finite set of ruloids for $C[\boldsymbol{X}]$ and $\alpha$ together with their associated *$\tau$-ruloids*, denoted by $\mathit{Ruloids}(C[\boldsymbol{X}], \alpha)$, such that

1. $\mathit{Ruloids}(C[\boldsymbol{X}], \alpha)$ explains all $\alpha$ moves of all instances of $C[\boldsymbol{X}]$. Namely, if $\rho(C[\boldsymbol{X}] \stackrel{\alpha}{\to} D[\boldsymbol{Y}])$ is valid, for some $\Sigma$ ground substitution $\rho$, then there exists $r \in \mathcal{R}$ such that $\rho(\mathit{cons}(r)) \equiv \rho(C[\boldsymbol{X}] \stackrel{\alpha}{\to} D[\boldsymbol{Y}])$ and the members of $\rho(\mathit{ante}(r))$ are valid.
2. $\mathit{Ruloids}(C[\boldsymbol{X}], \alpha)$ consists of valid ruloids only. Namely, for any $r \in \mathcal{R}$ and $\Sigma$ ground substitution $\rho$, if $\rho(\mathit{ante}(r))$ is valid then $\rho(\mathit{cons}(r))$ is valid.

The proof of the theorem can be found in Appendix B.

It will be beneficial in the forthcoming proof of Theorem 3.1 to use a certain notation for ruloids which is similar to ruloids in *$\epsilon$-presentation* [Blo95a]. A ruloid for $C[\boldsymbol{X}]$ is $\epsilon$-presented if it has one antecedent $X_i \stackrel{\mu}{\to} X'_i$ for each variable in $\boldsymbol{X}$, where $\mu \in A \cup \{\tau\} \cup \{\epsilon\}$ and $\epsilon$ is a new auxiliary action not in $\mathsf{Vis}$. An ordinary ruloid for $C[\boldsymbol{X}]$ and $\alpha$, call it $r$, can be made into a corresponding $\epsilon$-presented ruloid by adding to antecedents the transition $X_i \stackrel{\epsilon}{\to} X'_i$ 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 $\epsilon$-presented ruloids for $(X \boxplus b.Y)\backslash a$ and $b$. The second ruloid can be called an 'axiom' ruloid because of the form of its antecedents.

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

The first of the above $\epsilon$-presented ruloids and the ruloid for $X\backslash a$ and action $c$, with $c \neq a$, can be written using the following abstract notation

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

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 $X\backslash a$ and $Y$ does not appear in $X\backslash a$. 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 $\epsilon$-presented ruloids will be extensively used in the following proof. bash: a: command not found

**Proof of Theorem 3.1.** Let $G = (\Sigma, A \cup \{\tau\}, R)$ be a $\tau$DeS process language. Assume $\boldsymbol{p} \ll \boldsymbol{q}$, where $\boldsymbol{p}$ and $\boldsymbol{q}$ are vectors of closed $\Sigma$ terms. We need to prove $C[\boldsymbol{p}] \ll C[\boldsymbol{q}]$, for any $n$-ary $\Sigma$ context $C[\boldsymbol{X}]$. The last can be subdivided, by Definition 2.15, into three parts:

1. $\mathcal{L}(C[\boldsymbol{p}]) \subseteq \mathcal{L}(C[\boldsymbol{q}])$,
2. $\forall s \in \mathsf{Act}^*.\ C[\boldsymbol{p}]{\Downarrow\!\!\Downarrow} s$ implies $C[\boldsymbol{q}]{\Downarrow\!\!\Downarrow} s$,
3. $\forall s \in \mathsf{Act}^*.\ C[\boldsymbol{p}]{\Downarrow\!\!\Downarrow} s$ 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 $\tau$ 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 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 $0 \leq j < k$ transition $C_j[\boldsymbol{p}_j] \xrightarrow{\alpha_{j+1}} C_{j+1}[\boldsymbol{p}_{j+1}]$ gives rise, by Ruloid Theorem and the discussion concerning $\epsilon$-presented ruloids, to

$$p_{1j} \xrightarrow{\mu_{1j+1}} p_{1j+1}, \quad \ldots \quad , p_{nj} \xrightarrow{\mu_{nj+1}} p_{nj+1}, \tag{2}$$

where each $\mu_{ij+1} \in \mathsf{Act} \cup \{\tau\} \cup \{\epsilon\}$. Sequence (2) specifies the behaviour of the components $p_{ij}$ of $\boldsymbol{p}_j$, where $1 \leq i \leq n$, which produces $C_j[\boldsymbol{p}_j] \xrightarrow{\alpha_{j+1}} C_{j+1}[\boldsymbol{p}_{j+1}]$.

Sequence (2) can be abbreviated by $\boldsymbol{p}_j \xrightarrow{A_{j+1}} \boldsymbol{p}_{j+1}$, where $A_{j+1}$ is the column vector of $\mu_{ij+1}$'s. We can say more about vectors $A_l$. When $A_l$ contains no visible action, then it consists either of a single $\tau$ and $n-1$ labels $\epsilon$ or it consists of $n$ labels $\epsilon$. These vectors arise when applying $\epsilon$-presented $\tau$-ruloids and 'axiom' ruloids respectively. The first of these vectors is called a *$\tau$-column*. Putting the individual transitions of all $\boldsymbol{p}_j$ together we obtain

$\boldsymbol{p}_0 \xrightarrow{A_1} \ldots \xrightarrow{A_k} \boldsymbol{p}_k$, or abbreviating even further $\boldsymbol{p}_0 \xrightarrow{\boldsymbol{A}} \boldsymbol{p}_k$, where $\boldsymbol{A}$ is a vector of column vectors $A_j$. In other words, $\boldsymbol{A}$ is an $n \times k$ matrix. Note that the $i$th row of $\boldsymbol{A}$ is $\mu_{i1} \ldots \mu_{ik}$ and it represents the behaviour the $i$th component $p_{i0}$ of $\boldsymbol{p}_0$. Let $s_i$ be $\mu_{i1} \ldots \mu_{ik}$ with all labels $\tau$ and $\epsilon$ 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 $\boldsymbol{q}_0 \xrightarrow{\boldsymbol{A}}$, then we would derive $C[\boldsymbol{q}_0] \stackrel{s}{\Rightarrow}$ as required. However, $\boldsymbol{q}_0 \xrightarrow{\boldsymbol{A}}$ is not valid in general since the components of $\boldsymbol{q}_0$ and $\boldsymbol{p}_0$ may vary in the number of actions $\tau$ 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 $n \times l$ matrix $\boldsymbol{B}$ such that it only differs from $\boldsymbol{A}$ in the number and positions of $\tau$-columns. Having this we can deduce $C[\boldsymbol{q}_0] \xrightarrow{\nu_1} \ldots \xrightarrow{\nu_l}$, where the sequence $\nu_1 \ldots \nu_l$ differs from $\mu_1 \ldots \mu_k$ in the number and places of occurrences of actions $\tau$. 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 $C'[\boldsymbol{q}'] \xrightarrow{\tau^\omega}$. (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 $\alpha$ which has the antecedent $X_i \xrightarrow{\mu} X'_i$ with $\mu \neq \epsilon$ such that $q'_i \uparrow$. From $C[\boldsymbol{q}] \stackrel{t}{\Rightarrow} C'[\boldsymbol{q}']$ we obtain by 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 $p'_i \xrightarrow{\tau^\omega}$. 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 $i$th argument of $C^*[\boldsymbol{X}]$ is active and the $i$th component of $\boldsymbol{p}'$ is $p'_i$ and (c) $t' \leq t$. Thus, $C^*[\boldsymbol{p}'] \uparrow$ or $C^*[\boldsymbol{p}'] \xrightarrow{\tau^\omega}$. 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 (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 behaviour 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 $\boldsymbol{p} \stackrel{\boldsymbol{t}}{\Rightarrow}$ and $\boldsymbol{p} \Downarrow \boldsymbol{t}$ by Ruloid Theorem. 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: for all $t_i \leq s_i$ described above we have $p_i \stackrel{t_i}{\Rightarrow}$ 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 obtain $p_i \stackrel{s_i}{\Rightarrow} p'_i$. 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 sub-result (a). Since $\mathcal{S}(p'_i) \subseteq \mathcal{S}(q'_i)$ for each $i$, and since the behaviour of $D[\boldsymbol{p}']$ and $D[\boldsymbol{q}']$ depends on the actions of $\boldsymbol{p}'$ and $\boldsymbol{q}'$ respectively 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 3.6** $\sqsubseteq$ is a precongruence for all proper $\tau$DeS languages.

Figure 1: Axiomatisation of Testing Preorder.

| | | | |
|---|---|---|---|
| $E1$ | $X \boxplus \mathbf{0} = X$ | | |
| $E2$ | $X \boxplus X = X$ | $I2$ | $X \oplus X = X$ |
| $E3$ | $X \boxplus Y = Y \boxplus X$ | $I3$ | $X \oplus Y = Y \oplus X$ |
| $E4$ | $X \boxplus (Y \boxplus Z) = (X \boxplus Y) \boxplus Z$ | $I4$ | $X \oplus (Y \oplus Z) = (X \oplus Y) \oplus Z$ |
| $C1$ | $a.X \boxplus a.Y = a.X \oplus a.Y$ | | |
| $C2$ | $a.X \boxplus a.Y = a.(X \oplus Y)$ | $C3$ | $X \oplus Y \sqsubseteq X \boxplus Y$ |
| $D1$ | $X \boxplus (Y \oplus Z) = (X \boxplus Y) \oplus (X \boxplus Z)$ | $\Omega1$ | $\Omega \sqsubseteq X$ |
| $D2$ | $X \oplus (Y \boxplus Z) = (X \oplus Y) \boxplus (X \oplus Z)$ | $\Omega2$ | $X \boxplus \Omega \sqsubseteq X \oplus \Omega$ |

$\tau$DeS process operators preserve testing preorder as well as many other weak equivalences and preorders. Since $\tau$DeS languages are a subclass of ISOS languages $\tau$DeS operators preserve refusal simulation preorder [Uli92, Uli94] and *eager* bisimulation [Uli94, UP96, UP97]. It is thus somewhat surprising that such well behaved operators do *not* preserve the standard version of weak bisimulation [Mil89a]. Following the example of the *action refinement* operator [vG90], which does not preserve weak bisimulation, a simple $\tau$DeS version of this operator was defined in [UP96] thus showing that $\tau$DeS operators do not preserve the standard weak bisimulation. However, Bloom [Blo95a] proposed a class of process languages where weak bisimulation is preserved, but the languages in the class suffer in our opinion from a complicated definition. There are also several other classes of process languages [Blo95a], where branching bisimulation and the *rooted* versions of both weak and branching bisimulations are preserved.

# 4 Finite Axiom Systems for $\tau$DeS Languages

In this section we show how to generate an axiomatisation of an arbitrary $\tau$DeS language. The strategy is as follows: Given a $\tau$DeS language $G$, we disjointly extend it with the basic process language $\mathsf{B}$ to obtain $G'$. We propose sound laws (axiom schemas) for all $G'$ operators which are not in $\mathsf{B}$. The axiom system for $G$ (in fact for $G'$) consists of the instances of these laws together with the axiom system for $\mathsf{B}$. If $G$ is well-founded then the proposed axiom system is sufficient to rewrite arbitrary process terms over $G$ into equivalent terms over $\mathsf{B}$. 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 $\mathsf{B}$. 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 $\mathrm{T}_{\mathsf{B}}$ for $\ll$ and the language $\mathsf{B}$. $\mathrm{T}_{\mathsf{B}}$ 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 $\mathsf{Test}(\mathsf{B}) \models P \sqsubseteq Q$.

Since $\boxplus$ and $\oplus$ are associative we will use the following summation notation. If $\{P_i \mid i \in I = \{i_1, \ldots, i_m\}\}$ is a finite set of process terms, then let $\sum_{i \in I} P_i$ denote the term $P_{i_1} \boxplus \ldots \boxplus P_{i_m}$. When $I$ is in fact empty, then $\sum_{i \in I} P_i$ denotes $\mathbf{0}$. For the internal choice we use $\bigoplus_{i \in I} P_i$ to denote $P_{i_1} \oplus \ldots \oplus P_{i_m}$ only when $I \neq \emptyset$.

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

**Definition 4.2** Let $\mathcal{A}$ denote a finite set of subsets of $\mathsf{Act}$. 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: if $\mathcal{A}$ is saturated and non-empty, then $\bigoplus_{A \in \mathcal{A}} \sum_{a \in A} a.P_a$ is in hnf; if $A$ is a subset of $\mathsf{Act}$, then $\sum_{a \in A} a.P_a \boxplus \Omega$ is in hnf.

Notice that if $\mathcal{A} = \{\emptyset\}$, then $\bigoplus_{A \in \mathcal{A}} \sum_{a \in A} a.P_a \equiv \mathbf{0}$ since $\sum_{a \in \emptyset} a.P_a$ denotes $\mathbf{0}$. Also, if $A = \emptyset$, then $\sum_{a \in A} a.P_a \boxplus \Omega = \mathbf{0} \boxplus \Omega$, and $\mathbf{0} \boxplus \Omega$ can be rewritten by $E1$ as $\Omega$. Hence, we will treat $\mathbf{0}$ and $\Omega$ as terms in hnf. Terms $(a.p \boxplus b.q) \oplus c.r$ and $a.p \boxplus \Omega$ are in hnf but $(a.p \boxplus \Omega) \oplus c.r$ and $a.p \boxplus a.q$ are not. Each term in hnf enjoys an important property: if it has any $a$ descendants, then they are syntactically equal. For example, $(a.p \boxplus b.q) \oplus a.r$ is in hnf if and only if $p \equiv r$. This property is vital in obtaining *finite* axiomatisations of $\ll$. Process terms can be put in hnf by using $\mathrm{T}_{\mathsf{B}}$ and the following derived axioms [Hen88].

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

We will show how to transform process terms into hnf in Section 4.2. Notice the importance of axioms $C1, C2$ and $Der5$ in obtaining terms with unique $a$ descendants.

### 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 $\tau$DeS languages. The usefulness of our algorithm is illustrated by generating axiomatisations of the language $\mathsf{B}$ extended with the CSP hiding operator and the CCS parallel composition operator.

We will need to distinguish two subclasses of $\tau$DeS operators, namely *distinctive* [ABV92] and *$\tau$-introducing* operators [Uli95].

**Definition 4.3** A $\tau$DeS operator $(f, n)$ is *distinctive* if (a) for each argument $i$ either $i$ is active in all De Simone rules for $f$ or in none of them, and (b) for each pair of different De Simone rules for $f$ there is an argument $X_i$ such that both rules have an antecedent for $X_i$ but with a different action. A $\tau$DeS operator $(f, n)$ is *$\tau$-introducing* if $\tau$ is the action of one of its De Simone rules.

**Lemma 4.4** [Uli95] (**Action Laws**). Let $G$ be a self-encoding $\tau$DeS 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

$$\mathsf{TEST}(G) \vDash f(\boldsymbol{P}) = \alpha.C[\boldsymbol{P'}] \tag{3}$$

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

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

(**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,

$$\mathsf{TEST}(G) \vDash f(\boldsymbol{P}) = \mathbf{0}. \tag{5}$$

(**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

$$\mathsf{TEST}(G) \vDash f(\boldsymbol{P}) \sqsubseteq \Omega. \tag{6}$$

Since we also have $\Omega \sqsubseteq X$ (axiom $\Omega 1$), for all $X$, we could substitute (6) 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 4.5** (**Distributivity over $\oplus$ laws**). Let $G$ be a self-encoding $\tau$DeS process language. Let $(f, n)$ be an operator of $G$ such that its $i$th argument is active. If $X_i \equiv X \oplus Y$, then

$$\mathsf{TEST}(G) \vDash f(\boldsymbol{X}) = f(\boldsymbol{X})[X/X_i] \ \oplus\ f(\boldsymbol{X})[Y/X_i]. \tag{7}$$

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 $i$th argument. The proof of Lemma 4.5 can be found in Appendix C.

Notice that by left-to-right application of $X \boxplus \Omega = X \oplus \Omega$ and of 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 $\Omega$ or of the form $\sum_A a.p_a$ with $A \subseteq \mathsf{Act}$. This suggests that the distributivity over $\boxplus$ laws need only be given for terms $f(\boldsymbol{X})$ with all active arguments of the form $\sum_A a.X_a$ or $\Omega$. Since $\mathsf{Act}$ is finite there are only finitely many subsets $A$ of $\mathsf{Act}$, and thus there will only be finitely many instances of the the distributivity over $\boxplus$ laws—this is one of the two reasons why our axiomatisations are finite. In contrast, the distributivity over $\boxplus$ laws for refusal simulation preorder $\precsim_{RS}$ in [Uli95] apply to terms with the active subterms of the form $\sum_{\mathcal{A}} a_i X_{a_i}$, where $\mathcal{A}$ is a multiset of visible actions. Since there are infinitely many multisets $\mathcal{A}$ with actions in $\mathsf{Act}$ axiomatisations of $\precsim_{RS}$ in [Uli95] are not finite. Note that terms $\sum_{\mathcal{A}} a_i X_{a_i}$ can be rewritten to terms $\sum_A a_i.Y_{a_i}$, where $A$ is the set of actions of $\mathcal{A}$, using the axioms for $\ll$ but not the axioms for $\precsim_{RS}$.

We will use the following notation to express the form of active arguments. Let $\boldsymbol{A}$ be a vector with $n$ elements such that the $i$-th element, $A_i$, is $*$ if $i \notin \mathit{active}(f)$, and $*$ or a subset of $\mathsf{Act}$ otherwise. Then, $\boldsymbol{AX}$ (as in $f(\boldsymbol{AX})$ below) denotes a vector of $n$ 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 $\Omega$ 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 $\Omega$ if $A_i = *$ and $a_j.X_{ij}$ otherwise. Moreover, given trigger $\chi$ we say that $\chi$ *fires* from $\boldsymbol{A}$ (or $\chi \in \boldsymbol{A}$) if whenever $\chi_i$, the $i$th element of $\chi$, 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 $\Omega$.

**Lemma 4.6** (**Distributivity over $\boxplus$ laws**) Let $(f, n)$ be an operator of a self-encoding $\tau$DeS 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 $\mathsf{TEST}(G)$.

$$\begin{array}{rcll} f(\boldsymbol{AX}) & = & \sum_j f(\boldsymbol{AX})[(A_i\boldsymbol{X}_i)_j/A_i\boldsymbol{X}_i] & \\ & & \text{if } \Omega(f(\boldsymbol{AX})) \vee & \\ & & \quad \neg\Omega(f(\boldsymbol{AX})) \wedge \forall\chi \in \mathit{trig}_\tau(f).\ \chi \notin \boldsymbol{A} & (8) \end{array}$$

$$\begin{array}{rcll} f(\boldsymbol{AX}) & = & \sum_{A_i} f(\boldsymbol{AX})[a_j.X_{ij}/A_i\boldsymbol{X}_i] \oplus \bigoplus_{C_i} f(\boldsymbol{AX})[c_j.X_{ij}/A_i\boldsymbol{X}_i] & \\ & & \text{if } \neg\Omega(f(\boldsymbol{AX})) \wedge \exists\chi \in \mathit{trig}_\tau(f).\ \chi \in \boldsymbol{A} \wedge & \\ & & \quad C_i = \{\chi_i \mid \chi \in \mathit{trig}_\tau(f) \wedge \chi \in \boldsymbol{A}\} & (9) \end{array}$$

Next, we briefly explain the syntactic conditions of the laws. The condition for law (8) says that either one of the active arguments (not necessarily the $j$th) is $\Omega$ or none of the active arguments is $\Omega$ and no $\tau$ triggers for $f$ can fire from $\boldsymbol{A}$. The condition for law (9) says that none of the active arguments is $\Omega$, $f$ is $\tau$-introducing and set $C_i \subseteq A_i$ is the set of all $i$th elements of all $\tau$ triggers for $f$ that can fire from $\boldsymbol{A}$.

The proof of Lemma 4.6 can be found in Appendix C.

Next, consider the extension of $\mathsf{B}$ with the CSP hiding operator $\backslash a$ which is defined as follows:

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

Since by Definition 4.3 $\backslash a$ is distinctive it clearly satisfies the assumption of Lemma 4.6. The axioms obtained by Lemmas 4.4–4.6 are given in Figure 2. Note that the third axiom *does not* hold without the side condition. The second and the third equations in Figure 2 are axiom schemas. Since $\mathsf{Act}$ is finite these equations represent finite sets of axioms, namely one for each subset $B$ of $\mathsf{Act}$. Hence, the axiomatisation is *finite*. Similarly, the axiomatisations of other distinctive operators, for example the restriction or renaming of CCS, are also be finite.

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

**Lemma 4.7** (**Non-distinctive operator laws**) Let a non-distinctive $(f, n)$ be an operator of a self-encoding $\tau$DeS process language $G$. Then there is a disjoint extension $G'$ of $G$ with (possibly zero) $n$-ary distinctive non-$\tau$-introducing operators $f_k$, for $k \in K \subset_{\mathit{fin}} \mathbb{N}$, and (possibly zero) $n$-ary distinctive $\tau$-introducing operators $f^l$, for $l \in L \subset_{\mathit{fin}} \mathbb{N}$, such that $G'$ is a self-encoding $\tau$DeS language and the following laws hold in $\mathsf{TEST}(G')$:

$$\begin{array}{rcll} f(\boldsymbol{AX}) & = & \sum_K f_k(\boldsymbol{AX}) \boxplus \sum_L f^l(\boldsymbol{AX}) & \\ & & \text{if } \Omega(f(\boldsymbol{AX})) \vee & \\ & & \neg\Omega(f(\boldsymbol{AX})) \wedge \forall\chi \in \mathit{trig}_\tau(f).\ \chi \notin \boldsymbol{A} & (10) \end{array}$$

Figure 2: Axioms for hiding.

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

$$
\begin{array}{rcl}
f(\boldsymbol{AX}) & = & (\sum_K f_k(\boldsymbol{AX}) \boxplus \sum_L f^l(\boldsymbol{AX})) \oplus \bigoplus_N f^n(\boldsymbol{AX}) \\
& & \text{if } \neg\Omega(f(\boldsymbol{AX})) \wedge \exists\chi \in \mathit{trig}_\tau(f).\ \chi \in \boldsymbol{A} \wedge \\
& & \quad N = \{n \mid \chi \in \mathit{trig}_\tau(f) \wedge \chi \in \boldsymbol{A} \wedge \chi \in \mathit{trig}_\tau(f^n)\}
\end{array} \tag{11}
$$

The condition of (10) is the same as the one for (8). The condition of (11) says that none of the active arguments of $f(\boldsymbol{AX})$ is $\Omega$, $f$ is $\tau$-introducing and $N \subseteq L$ is the set of all $n$ such that each $\tau$ trigger for $f$ that can fire from $\boldsymbol{A}$ is also a $\tau$ trigger for $f^n$.

The sketch of a proof of Lemma 4.7 can be found in Appendix C. It is important to notice that the lemma allows us to rewrite process terms $f(\boldsymbol{AX})$, for each non-distinctive operator $f$, using only finitely many auxiliary operators $f_k$ and $f^l$. This is because $f$ has finitely many defining De Simone rules, and each such rule gives rise to at most one auxiliary operator $f_k$ or $f^l$ (the details are described in Appendix C). This is the second reason why our axiomatisations are finite.

As an example of axiomatisation of a non-distinctive and $\tau$-introducing operator we consider $\mathsf{B}$ 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 4.7 the operator $\|$ is axiomatised in terms of three auxiliary operators. We call these operators the left-merge, written $\lfloor\!\lfloor$, the right-merge, written $\rfloor\!\rfloor$, and the communication merge, written $|$, since they are very similar to the operators with the same names discussed in [BK84, BK85, BW90, ABV92]. Our operators are defined by the following De Simone rules together with the associated $\tau$-rules which are not shown. Note that, unlike their original versions, these operators are $\tau$DeS.

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

As before $AX$ and $BY$ stand for $\sum_{a\in A} a.X_a$ and $\sum_{b\in B} b.X_b$ respectively, where $A, B \neq \emptyset$. The axioms for $\|$ and for the auxiliary operators generated by Lemmas 4.4–4.7 are given in Figure 3. For the same reason as with the hiding operator the axiomatisation of the CCS parallel operator is finite.

Figure 3: Axioms for CCS parallel composition.

$$
\begin{array}{ll}
(X \oplus Y) \parallel Z = X \parallel Z \;\oplus\; Y \parallel Z & \\
X \parallel (Y \oplus Z) = X \parallel Y \;\oplus\; X \parallel Z & \\
AX \parallel \Omega = AX \mathbin{\lfloor\!\lfloor} \Omega \;\boxplus\; \Omega \mathbin{\lfloor\!\lfloor} AX \boxplus\; AX \mid \Omega & \\
\Omega \parallel AX = \Omega \mathbin{\lfloor\!\lfloor} AX \;\boxplus\; AX \mathbin{\lfloor\!\lfloor} \Omega \boxplus\; \Omega \mid AX & \\
AX \parallel BY = AX \mathbin{\lfloor\!\lfloor} BY \;\boxplus\; BY \mathbin{\lfloor\!\lfloor} AX \boxplus\; AX \mid BY & \text{if } A \cap \bar{B} = \emptyset \\
AX \parallel BY = AX \mathbin{\lfloor\!\lfloor} BY \;\boxplus\; BY \mathbin{\lfloor\!\lfloor} AX \oplus\; AX \mid BY & \text{if } A \cap \bar{B} \neq \emptyset
\end{array}
$$

$$
\begin{array}{l}
(X \oplus Y) \mathbin{\lfloor\!\lfloor} Z = X \mathbin{\lfloor\!\lfloor} Z \;\oplus\; Y \mathbin{\lfloor\!\lfloor} Z \\
(\sum_A a.X_a) \mathbin{\lfloor\!\lfloor} Y = \sum_A (a.X_a \mathbin{\lfloor\!\lfloor} Y) \\
a.X \mathbin{\lfloor\!\lfloor}_A Y = a.(X \parallel Y) \\
\mathbf{0} \mathbin{\lfloor\!\lfloor} X = \mathbf{0} \\
\Omega \mathbin{\lfloor\!\lfloor} X \sqsubseteq \Omega
\end{array}
$$

$$
\begin{array}{ll}
(X \oplus Y) \mid Z = X \mid Z \;\oplus\; Y \mid Z & \\
X \mid (Y \oplus Z) = X \mid Y \;\oplus\; X \mid Z & \\
(\sum_A a.X_a) \mid BY = \sum_A (a.X_a \mid BY) & \text{if } A \cap \bar{B} \neq \emptyset \\
AX \mid (\sum_B b.Y_b) = \sum_B (AX \mid b.Y_b) & \text{if } A \cap \bar{B} \neq \emptyset \\
(\sum_A a.X_a) \mid BY = \sum_{A \setminus C} (a.X_a \mid BY) \oplus \bigoplus_C (c.X_c \mid BY) & \text{if } A \cap \bar{B} = C \neq \emptyset \\
AX \mid (\sum_B b.Y_b) = \sum_{B \setminus C} (AX \mid b.Y_b) \oplus \bigoplus_C (AX \mid c.X_c) & \text{if } \bar{A} \cap B = C \neq \emptyset \\
a.X \mid \bar{a}.Y = X \parallel Y & \\
a.X \mid b.Y = \mathbf{0} & \text{if } a \neq \bar{b} \\
\mathbf{0} \mid X = \mathbf{0} & \\
X \mid \mathbf{0} = \mathbf{0} & \\
\Omega \mid X \sqsubseteq \Omega & \\
X \mid \Omega \sqsubseteq \Omega &
\end{array}
$$

**Input**: $\tau$DeS 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 (7) and call the resulting system $\mathrm{T}''$.

3. For each non-distinctive operator $f \notin \Sigma_{\mathsf{B}(A)}$ in $G''$ apply the construction of Lemma 4.7 to obtain the language $G'$. Add to $\mathrm{T}''$ all the resulting instances of laws (10)–(11) and call the resulting system $\mathrm{T}'$.

4. For each distinctive operator $f \notin \Sigma_{\mathsf{B}(A)}$ in $G'$ add to $\mathrm{T}'$ the following axioms: an action law (3) or (4) for each rule for $f$; all the inaction laws (5); all the divergence laws (6); and, for each active argument of $f$, the distributivity over $\boxplus$ laws (8)–(9). Call the resulting system T.

**Output**: self-encoding $\tau$DeS language $G'$ such that $G < G'$, and
the axiom system T such that $\mathsf{TEST}(G') \models \mathrm{T}$.

Figure 4: Axiomatisation algorithm $\mathsf{A}$.

We can remove some redundancy among the axioms in Figure 3 by adding, for example, valid and derivable (from the axioms generated by our method) reflexivity axioms for $\|$ and $|$ and for closed terms. Such addition would make the second, the fourth and five other easily identifiable axioms redundant. However, it can be shown by 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 4 we present algorithm $\mathsf{A}$ which generates a sound axiomatisation of $\ll$ for an arbitrary $\tau$DeS process language. Moreover, the axiom system generated by $\mathsf{A}$ is head normalising for initially $\tau$-convergent processes.

**Theorem 4.8** (**Soundness**) Let $G$ be a $\tau$DeS process language and $G'$ be a disjoint extension of $G$ produced by algorithm $\mathsf{A}$. Moreover, let T be the axiom system produced by algorithm $\mathsf{A}$. Then $\mathsf{TEST}(G') \models \mathrm{T}$.

**Proof.** By Lemmas 4.4–4.7. ∎

### 4.2 Finiteness and Head Normalisation

In this subsection we argue that the axiomatisations produced by algorithm $\mathsf{A}$ are finite. Moreover, they are head normalising for initially $\tau$-convergent processes.

**Theorem 4.9** (**Finiteness**) Axiom systems produced by $\mathsf{A}$ are finite.

**Proof.** Given a $\tau$DeS process language $G$ with the set of visible actions $A$, the following properties are true about the resulting language $G'$ and the axiom system T produced by algorithm $\mathsf{A}$. Note that since $\mathsf{Act}$ is finite $A$ is also finite.

1. Since $G$ has finitely many operators T has only finitely many instances of law (7).

2. $G$ can only have finitely many non-distinctive operators. Since each $\tau$DeS operator has only finitely many defining rules there will only be finitely many auxiliary distinctive operators required by Lemma 4.7. Thus, $G'$ has a finite number of distinctive operators.

3. It is clear, from the remarks in point 2 above, that laws (3)–(6) give rise to finitely many instances for each distinctive operator in $G'$.

4. There are only finitely many different instances of laws (8) and (9) 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 $\Omega$ or one of the finitely many subsets of $A$.

5. For the same reason as for laws (8) and (9), each of laws (10) and (11) contributes a finite set of instances to T.

Since T consists of $\mathrm{T}_{\mathsf{B}(A)}$, which is finite, and the sets of all instances of laws (3)–(11), 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 $\tau$DeS languages. Since B extended with the CCS parallel composition operator is well-founded the axiomatisation of testing preorder over the language is both finite and complete. Similarly, the axiomatisation produced by A for B extended with the hiding operator is also finite and complete.

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

**Theorem 4.10** (**Head Normalisation**) Let $G$ be a $\tau$DeS 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 $\tau$-convergent processes in $\mathrm{T}(\Sigma_{G'})$.

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

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

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 $\Sigma$. Function $d$ is defined inductively by $d(t) = 0$ if $t$ is a constant, and $d(g(t_1, \ldots, t_r)) = 1 + \max\{d(t_i) \mid 1 \leq i \leq r\}$ otherwise. Function $d_\varepsilon$ is defined only for initially $\tau$-convergent processes. It specifies the maximal operator depth of $\varepsilon$-descendants of processes: for $\tau$-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 $\tau$-convergent process terms into terms in a certain form by using laws (3)–(11) 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 $r$, and
- $p \equiv p' \boxplus p''$ or $p \equiv p' \oplus p''$, for $p'$ and $p''$ in sptf.

Terms in sptf do not necessarily have unique $a$ descendants, but it is routine to prove, using the rewriting techniques developed in [NH84, Hen88], that terms in sptf can be transformed into terms in hnf. Axioms $C1, C2$ and $Der5$ play the central role in this transformation.

Now we return to the proof of property $(\star)$. We begin with the base case. Assume that $p$ is an initially $\tau$-convergent $\Sigma_{G'}$ process term and $d_\varepsilon(p) = 0$. This means that $p$ is a constant, i.e. an operator with zero arguments, and, for each $p'$ such that $p \stackrel{\varepsilon}{\Rightarrow} p'$, process $p'$ is also a constant. 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 (10) or law (11). 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 $\tau$-introducing), then we rewrite each of the constants in $t$, which is not prefixed by an action and it is neither $\mathbf{0}$ nor $\Omega$, according to steps 1 and 2 above. We continue the above defined transformation of the resulting term as long as it contains constants which are not prefixed with actions and which are not $\mathbf{0}$ or $\Omega$. Notice that each $\tau$ transition of a constant corresponds to rewriting the constant either by law (4) or by law (11) together with (4). Since $p$ is initially $\tau$-convergent the transformation will eventually terminate with the resulting process term in sptf. Finally, the axioms and the derived axioms for $\mathsf{B}$ 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 $p$ 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 *Der*1. As a result we obtain $f(\boldsymbol{p}'')$, where each active argument $p''_i$ has one of the following forms: $\Omega$, $\mathbf{0}$, $\bigoplus_{K \neq \emptyset} p''_{ik}$ or $p''_{ik} \oplus \Omega$, where $p''_{ik}$ is an external choice of terms prefixed with actions. If $p''_i$ is not active, then $p''_i \equiv p'_i$.

   (b) If any of the active arguments of $f(\boldsymbol{p}'')$ have the form of an internal sum, then we repeatedly use law (7) from left to right. So, $f(\boldsymbol{p}'')$ is rewritten to $\bigoplus_M f(\boldsymbol{p}''_m)$, where each active argument of each $f(\boldsymbol{p}''_m)$ has one of the forms $\Omega$, $\mathbf{0}$ or $\sum_{I \neq \emptyset} a_i.r_i$ for some actions $a_i$ and terms $r_i$.
   *Remark.* Note that Lemmas 4.6–4.7 give the distributivity laws for terms $f(\boldsymbol{AX})$ with active arguments of the form $\Omega$, $\mathbf{0}$ 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 we use the distributivity over $\boxplus$ laws for such arguments. We perform this transformation for 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 $\Omega$, $\mathbf{0}$ 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 (10) or law (11). The resulting term will be a sum (external and/or internal) of terms with outermost operators being distinctive. Each of these terms is 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: $\Omega$, $\mathbf{0}$, $a_{ni}.r_{ni}$ or $g(\boldsymbol{r})$, where $g$ is none of the action prefixing, internal or external choice operators. The last situation occurs when $f$ is $\tau$-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 $\rho$ 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 similar subterms which result from $g(\boldsymbol{r})$. Since $p$ is initially $\tau$-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 $\mathsf{B}$ can be used to rewrite it into hnf. ■

### 4.3 Completeness

Given any $\tau$DeS process language $G$ with the set of visible actions $A$, algorithm $\mathsf{A}$ produces axiom system T which allows conversion of initially $\tau$-convergent process terms 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 a certain class of processes representing infinite behaviours we need to augment T with some 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 D.

**Theorem 4.11** Let $G$ be a well-founded $\tau$DeS process language. Let $G'$ and T denote the disjoint extension and the axiom system produced by the algorithm in Figure 4 respectively. Then, for all $p, q \in \mathrm{T}(\Sigma_{G'})$ if $\mathsf{Test}(G') \models p \sqsubseteq q$, then $\mathrm{T} \vdash p \sqsubseteq q$.

Although well-foundedness of $\tau$DeS process languages has been defined semantically (Definition 2.6) 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 $\tau$DeS languages are a subclass of linear GSOS languages every syntactically well-founded $\tau$DeS language is well-founded. However, as noticed in [ABV94], not all well-founded GSOS (and thus $\tau$DeS) languages are syntactically well-founded.

Axiomatisations of non-well-founded process languages require some form of induction rule in order to guarantee completeness. For example, a version of AIP is used in [ABV92, ABV94] to achieve completeness for arbitrary GSOS languages. A similar induction rule is suggested in [Uli95] as the induction rule needed for completeness of axiomatisations of refusal simulation preorder and $\tau$DeS process languages. Here we use the following induction rule, denoted by **Ind**, in order to guarantee completeness for $\tau$-convergent processes.

$$\frac{X/b^n.\Omega \sqsubseteq Y \quad \text{for all } n}{X \sqsubseteq Y} \qquad \textbf{Ind}$$

The rule says that if a process is testing related at all its finite depths to another process, then these processes are testing related. Our rule is somewhat similar to $\omega$-Induction rule in [NH84, Hen88]. It employs a binary infix operator /, taken from [ABV92, ABV94], which is defined by the following rules.

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

We easily check that / is distinctive. Moreover, it is not $\tau$-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 $A, B \neq \emptyset$.

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

Let $G_/$ denote the disjoint extension of a $\tau$DeS language $G$ with the operator /. The soundness of **Ind** and some useful properties of / are proved in Appendix D.

Finally, we have the soundness and completeness result for $\tau$-convergent, and thus finite branching, but not necessarily well-founded $\tau$DeS processes.

**Theorem 4.12** Given $\tau$DeS language $G$, let $G'$ and $T$ denote the disjoint extension of $G_/$ and the axiom system respectively, both produced by the algorithm in Figure 4. Then, for all $\tau$-convergent $p, q \in \mathrm{T}(\Sigma_{G'})$, $\mathsf{Test}(G') \models p \sqsubseteq q$ if and only if T, $\mathbf{Ind} \vdash p \sqsubseteq q$.

The proof for Theorem 4.12 can be found in Appendix D.

Our **Ind** rule is very strong and in many cases (for example, the systems we discuss 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 $\tau$DeS 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 $\tau$DeS languages. This will be considered elsewhere.

# 5 Applications

We argue that $\tau$DeS 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 $\tau$DeS operators allows one to write intuitive and clear specifications without worrying about the implementation. In the specification process it is a good practice to write a series of specifications of increasing concreteness. Ideally, the most concrete specification will the appropriate implementation. We prove that the more concrete specifications refine the less concrete ones using the axiom systems produced as described 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 consider are:

- interruption or abortion, suspension and resumption of a system by its own actions or actions of other systems,
- alternate execution of two systems, where the control of execution is passed between the systems by the '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 $\tau$DeS 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 $S_1'$ are non-distinctive: they fail the second part of the distinctiveness definition because they have two rules with the same (i.e. none) antecedents. The axiomatisation for $S_1$ and $S_1'$ produced by $\mathsf{A}$ 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 **Ind**, 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

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

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 viewer's 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 behaviour described as follows:

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

The whole system $S$ consists of the viewer $V$ together with $P$. 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 operator *op* we will use a general operator ${}_A\rangle_B\rangle_C$ 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.

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

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

$$
\frac{X \xrightarrow{\tau} X'}{X \mathbin{{}_A\rangle_B\rangle_C} Y \xrightarrow{\tau} X' \mathbin{{}_A\rangle_B\rangle_C} Y}
\qquad
\frac{Y \xrightarrow{\tau} Y'}{X \mathbin{{}_A\rangle_B\rangle_C} Y \xrightarrow{\tau} X \mathbin{{}_A\rangle_B\rangle_C} Y'}
$$

Next, we briefly explain these rules. In $P \mathbin{{}_A\rangle_B\rangle_C} Q$ 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 synchronise: 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 $\tau$-rules.

The set of axioms for ${}_A\rangle_B\rangle_C$ can be easily derived using algorithm $\mathsf{A}$. Similarly as for the CCS parallel operator the axiomatisation of ${}_A\rangle_B\rangle_C$ 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 $C = \{start, pause, r\}$.

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

Finally, applying **Ind** rule we prove $P = M\ {}_{A}\rangle\!_{B}\rangle_{C}\ RC$.

## 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 (undesirable) operators which do not preserve the considered equivalences. In order to correct this, one usually imposes certain restrictions on the form of De Simone rules, and uses $\tau$-rules as, for example, in [Blo90b, Vaa91, Uli92, Blo95a, Uli94]. The $\tau$DeS 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, the only directly related result is due to Vaandrager [Vaa91] who argues that testing preorder, in the setting without divergence, is a precongruence for any language in a class of languages similar to $\tau$DeS[1].

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 (weak) preorders. The auxiliary operators from 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 are used to make the specification easier [BIM88, BIM95, GV92, Gro93, Uli94], this theme has not been studied in detail. A recent exception is [Blo95b], where the author gives several specifications of the job protecting an arbitrary server by a checkpoint/backup scheme.

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

**Acknowledgements**

The pioneering work of Luca Aceto, Bard Bloom and Frits Vaandrager [ABV92, ABV94] was the inspiration for the present (and the previous [Uli95]) paper. I would like to thank the referees, the participants of AMAST '96, Mark Harman, Matthew Hennessy, Iain Phillips and Shoji Yuen for their helpful comments and suggestions. Thanks are also due to Paul Taylor for his LaTeXmacros.

# A Alternative Characterisation of Testing Preorders

The proof of Theorem 2.16 follows very closely the proofs of the corresponding characterisation results for testing preorders by De Nicola and Hennessy in [NH84, Hen88]. In order to make the paper self-contained we provide the detailed sketch of the proof.

**Proof of Theorem 2.16**. The proof of the first part relating to 'may' testing is easy. We prove the second part, namely $p \ll_{\mathsf{must}} q$ if and only if $p \sqsubseteq_{\mathsf{must}} q$ for any processes $p$ and $q$ over any proper $\tau$DeS process language. Then, the third part follows immediately from the first two parts.

Let $p$ and $q$ be any processes over a given proper $\tau$DeS process language $G$. Firstly, we show that $p \sqsubseteq_{\mathsf{must}} q$ implies $p \ll_{\mathsf{must}} q$. Assume $p \sqsubseteq_{\mathsf{must}} q$. We must prove $p \ll_{\mathsf{must}} q$. Suppose $p{\Downarrow}s$ for some $s \in \mathsf{Act}^*$. We need to show (a) $q{\Downarrow}s$ and (b) $q \stackrel{s}{\Rightarrow} q'$ implies $p \stackrel{s}{\Rightarrow} p'$ and $\mathcal{S}(p') \subseteq \mathcal{S}(q')$, for some $p'$. Instead of using all the tests in $\mathit{Test}(G)$, we shall only consider the following three types of tests [NH84, Hen88].

- $c(s)$ for any $s \in \mathsf{Act}^*$. The test is defined by $c(\varepsilon) = 1.w$ and $c(a.s) = a.c(s) \boxplus 1.w$. It is easy to show that
$$p \,\mathsf{must}\, c(s) \text{ if and only if } p{\Downarrow}s$$
- $t(s,a) = 1.w \boxplus b_1.(1.w \boxplus \ldots \boxplus b_n.(1.w \boxplus a)\ldots)$, where $s = b_1.\ldots.b_n$. We easily check that
$$p \,\mathsf{must}\, t(s,a) \text{ if and only if } p \stackrel{sa}{\not\Rightarrow}$$
- $t(s,A) = 1.w \boxplus b_1.(1.w \boxplus \ldots \boxplus b_n.(a_1.w \boxplus \ldots \boxplus a_k.w)\ldots)$, where $s$ is as above and $A = \{a_1, \ldots, a_k\}$. We have
$$p \,\mathsf{must}\, t(s,A) \text{ if and only if } \forall p'.(\, p \stackrel{s}{\Rightarrow} p' \text{ implies } \exists a \in A.\ p' \stackrel{a}{\Rightarrow})$$

Since $p{\Downarrow}s$ we deduce $p \,\mathsf{must}\, c(s)$. Thus, $q \,\mathsf{must}\, c(s)$ by $p \sqsubseteq_{\mathsf{must}} q$, and hence $q{\Downarrow}s$. Next, assume $q \stackrel{s}{\Rightarrow} q'$ for some $q'$. We must show $p \stackrel{s}{\Rightarrow} p'$, for some $p'$, and $\mathcal{S}(p') \subseteq \mathcal{S}(q')$. If $s = \varepsilon$, then clearly $p \stackrel{\varepsilon}{\Rightarrow} p$. If $s = s'a$, then $q \stackrel{s'a}{\Rightarrow} q'$. Assume $p \stackrel{s'a}{\not\Rightarrow}$ in order to exhibit a contradiction. $p \stackrel{s'a}{\not\Rightarrow}$ implies $p \,\mathsf{must}\, t(s',a)$, so $q \,\mathsf{must}\, t(s',a)$. This contradicts $q \stackrel{s'a}{\Rightarrow} q'$. Let $p_1, \ldots, p_k$ be an arbitrary enumeration of all $p'$ such that $p \stackrel{s}{\Rightarrow} p'$. We must show $\mathcal{S}(p_k) \subseteq \mathcal{S}(q')$, for some $k$. Suppose this is false. Hence, for $1 \leq i \leq k$ there exists $a_i \in \mathcal{S}(p_i)$ such that $a_i \notin \mathcal{S}(q')$. Hence, $p \,\mathsf{must}\, t(s,A)$, where $A = \{a_1, \ldots, a_k\}$. But clearly $q \,\not{\mathsf{must}}\, t(s,A)$, which contradicts $p \sqsubseteq_{\mathsf{must}} q$.

Secondly, we show that $p \ll_{\mathsf{must}} q$ implies $p \sqsubseteq_{\mathsf{must}} q$. Suppose $p \ll_{\mathsf{must}} q$ and $p \,\mathsf{must}\, t$ for some test $t$. We prove that $q \,\mathsf{must}\, t$. Let

$$(q \,|\, t \equiv) q_0 \,|\, t_0 \stackrel{\tau}{\rightarrow} q_1 \,|\, t_1 \stackrel{\tau}{\rightarrow} \ldots$$

be an arbitrary computation, denoted by $(\star)$, which may not be finite. We need to show that $t_k \xrightarrow{w}$ for some $k \geq 0$. The computation $(\star)$ can be decomposed into two sequences

$$q_0 \stackrel{a_1}{\Rightarrow} q_1 \stackrel{a_2}{\Rightarrow} q_2 \ldots \qquad \text{and} \qquad t_0 \stackrel{a_1}{\Rightarrow} t_1 \stackrel{a_2}{\Rightarrow} t_2 \ldots$$

which are denoted by $(\star_1)$ and $(\star_2)$ respectively, where for convenience $\xrightarrow{1}$ moves are absorbed in $(\star_2)$. Let $u$ denote the possibly infinite sequence $a_1 a_2 \ldots$. Hence, $q \stackrel{u}{\Rightarrow}$. We assume $p \Downarrow u$, which means $p \Downarrow u_k$ for every finite subsequence $u_k$ consisting of the first $k$ elements of $u$. Our assumption is reasonable for otherwise we could construct an unsuccessful computation from $p \,|\, t$ which contradicts $p \,\mathsf{must}\, t$. We have two cases:

1. $u$ is finite, i.e. $u = a_1 \ldots a_k$ and $q \stackrel{u}{\Rightarrow} q_k$ with $t \stackrel{u}{\Rightarrow} t_k$. $p \Downarrow u$ and $q \stackrel{u}{\Rightarrow} q_k$ imply, by $p \ll_{\mathsf{must}} q$, $p \stackrel{u}{\Rightarrow} p'$ and $\mathcal{S}(p') \subseteq \mathcal{S}(q_k)$ for some $p'$. From $\mathcal{S}(p') \subseteq \mathcal{S}(q_k)$ we deduce that there exists $p''$ such that $p' \stackrel{\varepsilon}{\Rightarrow} p'' \stackrel{\tau}{\not\rightarrow}$ and, of course, $\mathcal{S}(p'') \subseteq \mathcal{S}(q_k)$. Since $q_k \,|\, t_k$ cannot be extended, $p'' \,|\, t_k$ cannot be extended. Hence, the sequences $p \stackrel{u}{\Rightarrow} p''$ and $t \stackrel{u}{\Rightarrow} t_k$ can be combined to produce a computation $p \,|\, t \xrightarrow{\varepsilon} p'' \,|\, t_k$. Since $p \,\mathsf{must}\, t$ it follows that $t_n \xrightarrow{w}$ for some $n \leq k$.

2. $u$ is infinite. Lemma 4.4.12 [Hen88] states that if $p \ll_{\mathsf{must}} q$ and $p \Downarrow u$, where $u$ is infinite, then $q \stackrel{u}{\Rightarrow}$ implies $p \stackrel{u}{\Rightarrow}$. By this lemma we deduce $p \stackrel{u}{\Rightarrow}$. Any infinite derivation $p \stackrel{u}{\Rightarrow}$ can be combined with $(\star_2)$ to produce a computation $p \,|\, t = p_0 \,|\, t_0 \xrightarrow{\tau} \ldots$. Since $p \,\mathsf{must}\, t$ it clearly follows that $t_k \xrightarrow{w}$ for some $k \geq 0$.

Hence, in both cases we have $q \,\mathsf{must}\, t$. ■

# B Ruloid Theorem

In order to improve reading, we restate the theorem.

**Theorem 3.5** (**Ruloid Theorem**) Let $G = (\Sigma, A \cup \{\tau\}, R)$ be a $\tau$DeS language. Assume $C[\boldsymbol{X}] \notin \mathsf{Var}$ is a context over $\Sigma$ and $\alpha \in A \cup \{\tau\}$. Then, there exists a finite set of ruloids for $C[\boldsymbol{X}]$ and $\alpha$ together with their associated *$\tau$-ruloids*, denoted by *Ruloids*$(C[\boldsymbol{X}], \alpha)$, such that

1. *Ruloids*$(C[\boldsymbol{X}], \alpha)$ explains all $\alpha$ moves of all instances of $C[\boldsymbol{X}]$. Namely, if $\rho(C[\boldsymbol{X}] \xrightarrow{\alpha} D[\boldsymbol{Y}])$ is valid, for some $\Sigma$ ground substitution $\rho$, then there exists $r \in \mathcal{R}$ such that $\rho(\mathit{cons}(r)) \equiv \rho(C[\boldsymbol{X}] \xrightarrow{\alpha} D[\boldsymbol{Y}])$ and the members of $\rho(\mathit{ante}(r))$ are valid.

2. *Ruloids*$(C[\boldsymbol{X}], \alpha)$ consists of valid ruloids only. Namely, for any $r \in \mathcal{R}$ and $\Sigma$ ground substitution $\rho$ if $\rho(\mathit{ante}(r))$ is valid, then $\rho(\mathit{cons}(r))$ is valid.

Before we prove Theorem 3.5, we define several useful notations.

Context $C[\boldsymbol{X}] \notin \mathsf{Var}$ can be rewritten as $f(\boldsymbol{C}[\boldsymbol{X}])$, where $(f, n) \in \Sigma$ and $\boldsymbol{C}[\boldsymbol{X}]$ is a sequence of $n$ contexts $C_i[\boldsymbol{X}_i]$. It may happen that some $C_i[\boldsymbol{X}_i]$ are variables. Since any variable occurs at most once in $C[\boldsymbol{X}]$ it follows that contexts $C_i[\boldsymbol{X}_i]$ have no variables in common, and there is one to one correspondence between elements in $\boldsymbol{X}$ and in $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_n$.

A ruloid for $f(\boldsymbol{C}[\boldsymbol{X}])$ and $\alpha$ is constructed using one of the rules for $f$ and $\alpha$. Let the rule below, denoted by $r$, be such a rule.

$$\frac{\{\ X_i \xrightarrow{a_i} X_i'\ \}_{i \in I}}{f(X_1, \ldots, X_n) \xrightarrow{\alpha} D[\boldsymbol{Y}]} \tag{12}$$

The construction of a ruloid for $f(\boldsymbol{C}[\boldsymbol{X}])$ and $\alpha$ also requires the ruloids for $C_i[\boldsymbol{X}_i]$ and $a_i$ for some $i$. Clearly, we only need ruloids for those contexts $C_i[\boldsymbol{X}_i]$ which are not variables, and when $i$ is active, i.e. $i \in I$. Let $K = \{i \mid C_i[\boldsymbol{X}_i] \in \mathsf{Var}\}$. So we only need ruloids for $C_i[\boldsymbol{X}_i]$ and $a_i$ when $i \in I \setminus K$. In the rest of this section we let $C_i[\boldsymbol{X}_i] \equiv Y_i$, for $i \in K$, assuming that variables $Y_i$ are new and they do not appear in $\boldsymbol{X}$. Expression $\mathit{ruloid}((\boldsymbol{C}[\boldsymbol{X}]), r, (\boldsymbol{r}))$ denotes one of the ruloids for $f(\boldsymbol{C}[\boldsymbol{X}])$ and $\alpha$, where the sequence $\boldsymbol{r}$ (of length $n$) contains ruloids $r_i$ for $C_i[\boldsymbol{X}_i]$ and $a_i$ when $i \in I \setminus K$. Explicitly, $\mathit{ruloid}((\boldsymbol{C}[\boldsymbol{X}]), r, (\boldsymbol{r}))$ stands for

$$\frac{\{\mathit{ante}(r_i) \mid i \in I \setminus K\} \ \cup \ \{\, Y_i \xrightarrow{a_i} Y_i' \mid i \in I \cap K\}}{f(\boldsymbol{C}[\boldsymbol{X}]) \xrightarrow{\alpha} D[\boldsymbol{Z}]},$$

where $K = \{i \mid C_i[\boldsymbol{X}_i](\equiv Y_i) \in \mathsf{Var}\}$, and $\boldsymbol{Z}$ is defined as follows:

$$\begin{array}{lll} Z_i \equiv \mathit{target}(r_i) & \text{if} & i \in I \setminus K \\ Z_i \equiv Y_i' & \text{if} & i \in I \cap K \\ Z_i \equiv C_i[\boldsymbol{X}_i] & \text{if} & i \notin I \end{array}$$

The structure of $\mathit{ruloid}((\boldsymbol{C}[\boldsymbol{X}]), r, (\boldsymbol{r}))$ suggests that it is a ruloid. In order to be certain, we argue by induction on the structure of contexts that there are no copies of process variables in any of $\mathit{ruloid}((\boldsymbol{C}[\boldsymbol{X}]), r, (\boldsymbol{r}))$. Since there are no variables in common in contexts $C_i[\boldsymbol{X}_i]$, and by induction ruloids for $C_i[\boldsymbol{X}_i]$ do not have copies in the antecedents it follows that $\mathit{ruloid}((\boldsymbol{C}[\boldsymbol{X}]), r, (\boldsymbol{r}))$ does not have copies in the antecedents. Moreover, since there are no copies in $\boldsymbol{Z}$, and by induction there are no copies in the targets of ruloids for $C_i[\boldsymbol{X}_i]$ we deduce that there are no copies in $D[\boldsymbol{Z}]$.

Next, given $\boldsymbol{C}[\boldsymbol{X}]$ and $\lambda$, the trigger of $r$, we describe how to construct the set of sequences $\boldsymbol{r}$. We will denote this set as $\mathit{Aux}(\boldsymbol{C}[\boldsymbol{X}], \lambda)$. Roughly, $\mathit{Aux}(\boldsymbol{C}[\boldsymbol{X}], \lambda)$ is the product of the sets of ruloids for $C_i[\boldsymbol{X}_i]$ and $\lambda(i)$. But, for $1 \leq i \leq n$, $C_i[\boldsymbol{X}_i]$ may be a variable, $\lambda(i)$ may be a $*$, or the set of ruloids for $C_i[\boldsymbol{X}_i]$ and $\lambda(i)$ may be empty. Since $\lambda(i) \neq * \ \wedge \ C_i[\boldsymbol{X}_i] \notin \mathsf{Var}$ is logically equivalent to $i \in I \setminus K$, where $I$ and $K$ are easily definable in terms of $\lambda$ and $\boldsymbol{C}[\boldsymbol{X}]$ respectively, the set $\mathit{Aux}((\boldsymbol{C}[\boldsymbol{X}]), \lambda)$ is defined as follows:

$$\prod_{1 \leq i \leq n} \{\ R_i \ \mid \ \begin{array}{l} R_i = \mathit{Ruloids}(C_i[\boldsymbol{X}_i], \lambda(i)) \ \text{ if } \ i \in I \setminus K \wedge \mathit{Ruloids}(C_i[\boldsymbol{X}_i], \lambda(i)) \neq \emptyset, \ \text{ and} \\ R_i = \{*\} \text{ otherwise } \} \end{array}$$

**Proof of Theorem 3.5**. The proof is by induction on the structure of contexts. Assume that $G = (\Sigma, A \cup \{\tau\}, R)$ is a $\tau$DeS language, $C[\boldsymbol{X}] \notin \mathsf{Var}$ is a context and $\alpha \in A \cup \{\tau\}$.

If $C[\boldsymbol{X}] \equiv f(\boldsymbol{X})$, for some operator $f$, then $\mathit{Ruloids}(f(\boldsymbol{X}), \alpha) = \mathit{rules}(f, \alpha)$. From the way $\rightarrow$ is defined, it is clear that every transition $f(\boldsymbol{p}) \xrightarrow{\alpha} t$ can be derived using the rules in $\mathit{rules}(f, \alpha)$, and that all rules in $\mathit{rules}(f, \alpha)$ are valid.

Next, we must show that the theorem holds for $C[\boldsymbol{X}]$ and $\alpha$, where $C[\boldsymbol{X}] \equiv f(\boldsymbol{C}[\boldsymbol{X}])$ and not all element in $\boldsymbol{C}[\boldsymbol{X}]$ are variables. Inductive hypothesis: Assume that Ruloid Theorem holds for all $C_i[\boldsymbol{X}_i]$, which are not variables, and for all actions. Using this assumption we define the set of ruloids for $C[\boldsymbol{X}]$ and $\alpha$ as

$$\mathit{Ruloids}(C[\boldsymbol{X}], \alpha) = \bigcup_r \{R(\boldsymbol{C}[\boldsymbol{X}], r) \mid C[\boldsymbol{X}] \equiv f(\boldsymbol{C}[\boldsymbol{X}]) \ \wedge \ r \in \mathit{rules}(f, \alpha)\},$$

where

$$R(\boldsymbol{C}[\boldsymbol{X}], r) = \{\mathit{ruloid}((\boldsymbol{C}[\boldsymbol{X}]), r, (\boldsymbol{r})) \ \mid \ \begin{array}{l} (\boldsymbol{r}) \in \mathit{Aux}((\boldsymbol{C}[\boldsymbol{X}]), \mathit{trigger}(r)) \ \wedge \\ r_i \neq *, \text{ for all } i \in I \setminus K\}. \end{array}$$

When $C[\boldsymbol{X}] \equiv f(\boldsymbol{C}[\boldsymbol{X}])$ and $rules(f, \alpha) = \emptyset$, then clearly $Ruloids(C[\boldsymbol{X}], \alpha)$ is empty. The set $R(\boldsymbol{C}[\boldsymbol{X}], r)$ contains ruloids for $C[\boldsymbol{X}]$ and $\alpha$ which arise from $r \in rules(f, \alpha)$. When $Ruloids(C_i[\boldsymbol{X}_i], \lambda(i)) = \emptyset$, for some $i \in I \setminus K$, then $R(\boldsymbol{C}[\boldsymbol{X}], r) = \emptyset$.

Finally, we need to prove that $Ruloids(C[\boldsymbol{X}], \alpha)$ satisfies the two conditions of Theorem 3.5. From the definition of $Ruloids(C[\boldsymbol{X}], \alpha)$ it is clear that it explains all $\alpha$ moves of all instances of $C[\boldsymbol{X}]$. Let $r' \in Ruloids(C[\boldsymbol{X}], \alpha)$ be constructed using rule $r$ for $f$. We know that $r$ is valid and, by inductive hypothesis, all ruloids used in the construction of $r'$ are valid. Hence, $r$ is valid. ∎

## C 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 4.5.** 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 $\rho$ 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 2.16 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 $G'$. Assume that $s$ is the sequence $\alpha_1 \ldots \alpha_m$ with all $\tau$ removed. Thus,

$$(\rho(f(\boldsymbol{X}) \equiv)\rho(C_0[\boldsymbol{X}_0]) \stackrel{\alpha_1}{\to} \rho(C_1[\boldsymbol{X}_1]) \ldots \stackrel{\alpha_m}{\to} \rho(C_m[\boldsymbol{X}_m])$$

for some contexts $C_i[\boldsymbol{X}_i]$ with $n$ holes. We need to consider two cases depending on whether or not the derivation 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 $\rho(C_k[\boldsymbol{X}_k]) \stackrel{\alpha_{k+1}}{\to} \rho(C_{k+1}[\boldsymbol{X}_{k+1}])$ 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 there exists an open term $T'$ such that $\rho(f(\boldsymbol{X})) \stackrel{s}{\Rightarrow} T'$ and $\mathcal{S}(T') \subseteq \mathcal{S}(T)$. 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 $T$ and $T'$ as above we have $T \equiv T'$, and thus $\mathcal{S}(T') \subseteq \mathcal{S}(T)$. ∎

In the proof of Lemma 4.6 and Lemma 4.7 the following result, based on Lemma 4.1.3 in [NH84], will be helpful.

**Result C.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 (8) of Lemma 4.6**. 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 (8) holds for any closed substitution $\rho$ 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 $\Omega$ and $\rho(f(\boldsymbol{AX})) \stackrel{\tau}{\not\rightarrow}$ for all $\rho$. Thus, $(A_i\boldsymbol{X}_i)_j \equiv \sum_{A_i} a_j X_{ij}$. Clearly, LHS$\downarrow$ and RHS$\downarrow$ follow by Definition 2.4. Thus, by Result C.1 we only need to show $\mathcal{D}(\text{LHS},\mu) = \mathcal{D}(\text{RHS},\mu)$. We notice that $f(\boldsymbol{AX})$ cannot perform $\tau$ because $f$ is not $\tau$-producing and no $\tau$-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 $Y_k \equiv X'_k$ if $k \in K$ and $Y_k \equiv X_k$ otherwise, such that $\rho(C[\boldsymbol{Y}]) \equiv t$ 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 $i$ (this is the assumption of Lemma 8) 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 $i$th component of $\boldsymbol{Y}$ is $X'_i \equiv X_{ij}$. 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, $\boldsymbol{Y}' \equiv \boldsymbol{Y}$. 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, i.e. $\Omega(f(\boldsymbol{AX}))$. Thus, 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 LHS $\stackrel{\tau}{\rightarrow} t$. Since every rule for $f$ has an antecedent for $X_i$ we can assume that the antecedent used to derive LHS $\stackrel{\tau}{\rightarrow} t$ is $X_i \stackrel{a_i}{\rightarrow} X'_i$. Because RHS is an external sum (choice) of terms the silent move of one the summands will not resolve the choice. We easily see that RHS $\stackrel{\tau}{\rightarrow} t \boxplus \rho(\sum_{A_i\setminus\{a_i\}} f(\boldsymbol{AX})[a_j.X_{ij}/A_iX_i])$. 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 $\mathcal{L}(\text{RHS}) \subseteq \mathcal{L}(\text{LHS})$ 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 (9) of Lemma 4.6**. Let $\neg\Omega(f(\boldsymbol{AX}))$ and let $f$ be $\tau$-introducing. Assume that there is a $\tau$ trigger for $f$ that can fire from $\boldsymbol{A}$. Also, let $C_i \subseteq A_i$ be the set of the $i$th elements of all $\tau$ triggers for $f$ that can fire from $\boldsymbol{A}$. We need to show

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

for any closed substitution $\rho$ over any disjoint extension of $G$. As before we will use LHS and RHS to abbreviate the left and the right hand side of the above expression respectively.

We prove, using a similar argument as for law (8), $\mathcal{D}(\text{LHS}, a) = \mathcal{D}(\text{RHS}, a)$ for all relevant visible actions $a$. Thus, $\mathcal{L}(\text{LHS}) = \mathcal{L}(\text{RHS})$ and so LHS $\simeq_{\mathsf{may}}$ RHS. Next, we prove LHS $\ll_{\mathsf{must}}$ RHS. The converse follows similarly. Assume LHS$\Downarrow s$, for some sequence $s$. If $s = \varepsilon$, then $t\Downarrow s$ for every $t$ such that 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 $\{t \mid \text{LHS} \xrightarrow{\tau} t\}$. Moreover, every $t'$ such that $\rho(f(\boldsymbol{AX})[c_j.X_{ij}/A_iX_i]) \xrightarrow{\tau} t'$ is also a member of $\{t \mid \text{LHS} \xrightarrow{\tau} t\}$. Thus, RHS$\Downarrow\varepsilon$. If $s \neq \varepsilon$, then we deduce RHS$\Downarrow s$ from $\mathcal{D}(\text{LHS}, a) = \mathcal{D}(\text{RHS}, a)$, for all $a$, and LHS$\Downarrow s$.

Next, assume RHS $\stackrel{s}{\Rightarrow} q$. If $s \neq \varepsilon$, then by the above remarks about transitions of LHS and RHS we obtain LHS $\stackrel{s}{\Rightarrow} p$, $p \equiv q$ and so $\mathcal{S}(p) \subseteq \mathcal{S}(q)$. If $s = \varepsilon$, then we have four 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. $q \equiv \rho(\bigoplus_{C_i} f(\boldsymbol{AX})[c_j.X_{ij}/A_i\boldsymbol{X}_i])$,
4. $\rho(\bigoplus_{C_i} f(\boldsymbol{AX})[c_j.X_{ij}/A_i\boldsymbol{X}_i]) \xrightarrow{\tau} q' \stackrel{\varepsilon}{\Rightarrow} q$, for some $q'$.

In case (1) we have LHS $\stackrel{\varepsilon}{\Rightarrow}$ LHS and $\mathcal{S}(\text{LHS}) \subseteq \mathcal{S}(q)$. In case (2) we notice that, because silent moves do not resolve external choice, $q$ is the same as

$$\rho(\sum_{A_i} f(\boldsymbol{AX})[a_j.X_{ij}/A_i\boldsymbol{X}_i])$$

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 $\tau$. 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 LHS $\xrightarrow{\tau} q^*$ we have $\mathcal{S}(\text{LHS}) \subseteq \mathcal{S}(q)$. In case (3) we consider any $p$ such that 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, LHS $\xrightarrow{\tau} q'$ and so there exists $p(\equiv q)$ such that 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 4.7.

**Proof of Lemma 4.7**. Let $R$ be the set of rules for non-distinctive $n$-ary operator $f$. Firstly, we show how to construct the disjoint extension $G'$ of $G$ required by the lemma. After removing all $\tau$-rules from $R$, we divide the resulting set into $R_a$, the set of all non-$\tau$-introducing rules, and $R_\tau$, the set of all $\tau$-introducing rules. Next, we split $R_a$ and $R_\tau$ 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 $R_a$ $(R_\tau)$ 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 $\tau$-rules, define an operator which satisfies the first part of the distinctiveness property (Definition 4.3: for each $i$th argument either all or none of the rules have active $i$). 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 $\tau$-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-$\tau$-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 $\tau$-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 $\tau$-rules associated with the operators in $\Sigma_{G'}$. We easily check that $G'$ is a disjoint extension of $G$.

Finally, laws (10) and (11) are proved sound in a similar fashion as laws (8) and (9) 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. ■

# D Completeness

Lemmas 2.7 and 2.10 imply the following.

**Lemma D.1** Well-founded $\tau$DeS process languages are finitely branching.

We will show that, given any well-founded self-encoding $\tau$DeS process language $G$, the disjoint extension $G'$ of $G$ produced by the algorithm in Figure 4 is also well-founded. The following lemma will be useful.

**Lemma D.2** Let $G$ be a $\tau$DeS 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 $\mathsf{Test}(G) \vDash \mathrm{T}$. Then, for each well-founded $\Sigma_G$ 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 $\tau$DeS 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 D.3** Suppose $G$ is a well-founded and self-encoding $\tau$DeS process language. Then, the disjoint extension $G'$ of $G$ produced by algorithm $\mathsf{A}$ is also well-founded.

**Proof.** Let $G$ and $G'$ be $\tau$DeS 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 D.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 $\tau$DeS languages we get $h(\boldsymbol{p}) \simeq h(\boldsymbol{q})$. Moreover, $h(\boldsymbol{q})$ is a $\Sigma_G$ 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 4.7. Let $q$ be a $\alpha$-descendant of $h(\boldsymbol{p})$ obtained by applying some De Simone rule, say $r$, after possibly a number of applications of $\tau$-rules for $h$. Thus, $h(\boldsymbol{p}) \stackrel{\varepsilon}{\Rightarrow} h(\boldsymbol{p}') \stackrel{\alpha}{\to} q$. Hence, there is a ground $\Sigma_{G'}$ substitution $\rho$ such that $\rho(cons(r)) \equiv \rho(h(\boldsymbol{X}) \stackrel{\alpha}{\to} C[\boldsymbol{Y}']) \equiv h(\boldsymbol{p}') \stackrel{\alpha}{\to} C[\boldsymbol{q}]$ with $C[\boldsymbol{q}] \equiv q$. This implies that $ante(r)$ is valid, i.e. $\rho(X_i \stackrel{a_i}{\to} X_i') \equiv p_i' \stackrel{a_i}{\to} q_i$, 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 4.7), $C[\boldsymbol{Y}']$ is some $\Sigma_G$ 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 $\Sigma_G$ 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 $\alpha$-descendant of $h(\boldsymbol{p})$, it follows that $h(\boldsymbol{p})$ itself is well-founded.

This completes the proof. ∎

**Proof of Theorem 4.11.** 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 D.3 $p$ and $q$ are well-founded, hence initially $\tau$-convergent. Thus, T is head normalising for $p$ and $q$. Hence, using Lemma D.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. ∎

Finally, we prove the soundness and completeness result for $\tau$-convergent processes. The following lemma describes some of the properties of the operator / which is used in **Ind**.

**Lemma D.4** Let $G'$ be such that $G_{/} \leq G'$ for some $\tau$DeS process language $G$. If $p$ and $q$ are processes over $G'$ such that $q$ is well-founded and $G'$ has prefixing with $b$, then we have

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

We leave the proof of this lemma to th reader. Next, we show that **Ind** is sound for all $\tau$DeS process languages.

**Theorem D.5** Suppose that $G$ is a $\tau$DeS language. Then, $\mathsf{TEST}(G_{/}) \vDash$ **Ind**.

**Proof.** Let $G'$ be $\tau$DeS language such that $G_{/} \leq G'$. Let $P$ and $Q$ be open terms over $\Sigma_{G'}$, and let $\rho$ 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 $\mathsf{may}$ 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 $s$. Since $\rho(P/b^m.\Omega) \ll \rho(Q)$ we obtain $\rho(Q)\Downarrow s$. Next, assume $\rho(P)\Downarrow s$ and $\rho(Q) \stackrel{s}{\Rightarrow} q'$, where $q' \equiv \rho(Q')$ 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 4.12**. The soundness part follows immediately from Theorems 4.8 and D.5. For the completeness part, assume that $p$ and $q$ are $\tau$-convergent $\Sigma_{G'}$ closed terms and $\mathsf{Test}(G') \models p \sqsubseteq q$. The last means $p \ll q$. Also, let $A$ be the set of visible actions of $G'$. Lemma D.4 states that $p/b^n.\Omega \ll p$ holds for all $n \in \mathbb{N}$. Thus, we obtain $p/b^m.\Omega \ll q$ for all $m \in \mathbb{N}$. Since $p$ is $\tau$-convergent $p/b^m.\Omega$ is well-founded. Thus, by Lemma D.2 there exists, for each $m$, a $\Sigma_{\mathsf{B}(A)}$ term $p_m$ such that $\mathrm{T} \vdash p_m = p/b^m.\Omega$. By soundness we obtain $p_m \simeq p/b^m.\Omega$, hence $p_m \ll q$. Since $p_m$ is well-founded we deduce by Lemma D.4 that there exists $k \in \mathbb{N}$ such that $p_m \ll q/b^k.\Omega$. Since $q$ is $\tau$-convergent $q/b^k.\Omega$ is well-founded, so by Lemma D.2 we can construct, for each $k$, a $\Sigma_{\mathsf{B}(A)}$ term $q_k$ such that $\mathrm{T} \vdash q_k = q/b^k.\Omega$. 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 $\Sigma_{\mathsf{B}}$ terms, so by the completeness result for $\mathsf{B}$ we have $\mathrm{T} \vdash p_m \sqsubseteq q_k$, Thus, $\mathrm{T} \vdash p/b^m.\Omega \sqsubseteq q/b^k.\Omega$. Since T $\vdash q/b^k.\Omega \sqsubseteq q$ we get $\mathrm{T} \vdash p/b^m.\Omega \sqsubseteq q$. Finally, using **Ind** obtain our result: T, **Ind** $\vdash p \sqsubseteq q$. ■

## References


[Abr87] S. Abramsky. Observation equivalence as a testing equivalence. *Theoretical Computer Science*, 53:225–241, 1987.

[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. IEEE Computer Science Press.

[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 inference systems for a class of GSOS languages generating regular behaviours. In *Proceedings of the 5th International Conference CONCUR'94*, Uppsala, Sweden, 1994. Springer-Verlag. 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, 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):232–268, 1995.

[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(1):77–121, 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.

[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, volume 18. Cambridge University Press, 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.

[Gro93] J.F. Groote. Transition system specifications with negative premises. *Theoretical Computer Science*, 118(2):263–299, 1993.

[GV92] J.F. Groote and F. Vaandrager. Structured operational semantics and bisimulation as a congruence. *Information and Computation*, 100:202–260, 1992.

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

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

[Mil81] R. Milner. A modal characterisation of observable machine behaviours. In G. Astesiano and C. Böhm, editors, *Proceedings of CAAP'81*, Berlin, 1981. Springer-Verlag. LNCS 112.

[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, U.K., 1990. Springer-Verlag. 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 $\tau$'s. In H. Ehrig, R. Kowalski, G. Levi, and U. Montanari, editors, *Proceedings of TAPSOFT'87*, Berlin, 1987. Springer-Verlag. LNCS 250.

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

[NS91] X. Nicollin and J. Sifakis. An overview and synthesis on timed process algebras. In K.G. Larsen and A. Skou, editors, *Proceedings of CAV'91*, Aalborg, Denmark, 1991. Springer-Verlag. LNCS 575.

[Plo81] G. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University, 1981.

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

[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. LNCS 962.

[Uli96] I. Ulidowski. Finite axiom systems for testing preorder and De Simone process languages. In M. Wirsing and M. Nivat, editors, *Proceedings of the 5th International Conference on Algebraic Methodology and Software Technology AMAST'96*, München, Germany, 1996. Springer. LNCS 1101.

[UP96] I. Ulidowski and I.C.C. Phillips. Formats of ordered SOS rules. Technical report, RIMS, Kyoto University, 1996. Available at http://www.kurims.kyoto-u.ac.jp/~irek/.

[UP97] I. Ulidowski and I.C.C. Phillips. Formats of ordered SOS rules with silent actions. In M. Bidoit and M. Dauchet, editors, *Proceedings of the 7th International Conference on Theory and Practice of Software Development TAPSOFT'97*, Lille, France, 1997. Springer. LNCS 1214.

[UY97a] I. Ulidowski and S. Yuen. Extending process languages with time. In M. Johnson, editor, *Proceedings of the 6th International Conference on Algebraic Methodology and Software Technology AMAST'97*, Sydney, Australia, 1997. Springer. LNCS 1349.

[UY97b] I. Ulidowski and S. Yuen. Towards general timed process languages. Technical report, RIMS, Kyoto University, 1997. Available at http://www.kurims.kyoto-u.ac.jp/~irek/.

[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. IEEE Computer Science Press.

[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'93*. Springer-Verlag, 1993. LNCS 711.

[Wal90] D. Walker. Bisimulation and divergence. *Information and Computation*, 85(2):202–241, 1990.