A format for semantic equivalence comparison

Vashti Galpin

Programme for Highly Dependable Systems
School of Computer Science, University of the Witwatersrand
Private Bag 3, Wits 2050, South Africa
Tel: +27 11 717 6184 Fax: +27 11 717 6199
vashti@cs.wits.ac.za

Abstract

This paper presents a new format for process algebras, the extended *tyft/tyxt* format which generalises the *tyft/tyxt* format of Groote and Vaandrager. The format differs from most previous formats in that the labels on transitions are treated as many-sorted terms. Bisimulation is a congruence for all operators defined by extended transition system specifications in this format.

When one extended transition system specification is summed with another, the resulting bisimulation can either identify more terms (an abstracting extension up to bisimulation) or fewer terms (a refining extension up to bisimulation) than the original bisimulation on the individual system. The notions of abstracting extension and refining extension are defined, and two theorems are presented giving conditions required for achieving each type of extension. These results provide a way to compare different semantic equivalences defined for different process algebras.

Finally, an application of this theory to semantic equivalence comparison is given for a new result relating Castellani's pomset equivalence and Krishnan's multiprocessor equivalence.

Key words: process algebra, extended *tyft/tyxt* format, structured operational semantics, structural operational semantics, comparison of semantic equivalences

1 Introduction

This paper focusses on the metatheory of process algebras; namely theory that considers the general form of rules that make up the structural operational semantics [38] that are used to define process algebras such as CCS [35]. A description of the form that these rules can have is called a format, and a collection of operators and rules in a specific format is called a transition system specification.

A number of formats have been defined and these focus on different aspects of process algebras. Some of this research into formats considers the congruence of a semantic equivalence such as bisimulation, with respect to the operators defined in the format [18,28,8,39,11,12,21,30,9,20,33], while other research studies negative hypotheses and/or predicates in rules [8,27,42,13,40]. Another area of interest is how to produce an equational axiom system from rules in a given format [1,3,2,7,10,20]. The area most related to the research presented here is the work on conservative extensions since it considers the results of combining two transition system specifications [28,41,16,22,4].

As motivation for this research, consider the many extensions to CCS and other process algebras which have been proposed to model different aspects of concurrent computation. This proliferation indicates that the notion of process algebra has wide application and is flexible; however, it is not immediately obvious how a process algebra and its semantic equivalences relate to other process algebras and equivalences. It is important both theoretically and practically to understand the relationships between process algebras and their semantic equivalences.

Additionally, a significant aspect of recent process algebras is the introduction of structured or non-atomic labels; labels that have structure or contain information beyond what the action is. Even in CCS, basic actions are not totally atomic or undifferentiated since complements are required for communication to occur, and the distinguished action τ is required to present an internal action or communication. In transition systems for process algebras that deal with dependencies between actions, tags or markers appear in the labels (and are stored in the process terms) to keep a record of these dependencies, and these introduce structure in the labels. Examples of these are CCS with locations [14], CCS with local and global causes [31] and CCS with causalities [17]. Other process algebras such as multiprocessor CCS [32] and pomset CCS [15] introduce structure by allowing compound labels. Labels also become non-atomic when actions are split so that each label represents either the start of an action or the end of an action – this approach has been taken in action refinement [5,6], ST-bisimulation [29] and split-bisimulation semantics [26].

Hence, this paper is an investigation into how the extended *tyft/tyxt* format can be used in comparison of semantic equivalences based on bisimulation over process algebras whose behaviours are described by structural operational semantics and expressed as labelled transition systems. This format extends the *tyft/tyxt* format [28] in that it deals with complex label structure.

This research uses many-sorted signatures and algebras to extend the notion of format. Most prior work in formats uses a single-sorted signature and corresponding term algebra for processes, and assumes a set of atomic (or uninterpreted) actions [8,12,18,28,27]. Moreover, the actions are treated schemati-

cally; a rule is a scheme that represents a number of rules, each with different actions. This approach becomes unsatisfactory when dealing with more complex labels, especially when the semantic equivalence does not require an exact match on labels; for example, pomset bisimulation [15] and parameterised location bisimulation [14]. In the extended *tyft/tyxt* format, both processes and actions are dealt with syntactically. Process terms have a distinguished sort, and there may be more than one sort for label terms. Allowing label terms to be many-sorted provides a mechanism for comparison. The label terms appear on transitions and can also appear as arguments in process terms. For example, for the prefix operator of CCS, there is essentially one operator for each action. When expressing this in the extended *tyft/tyxt* format, there will be a single prefix operator taking two arguments – an action and a process. Additionally, using labels syntactically gives a full account of how the passing of dependency information is performed. The actual labels of the process algebra are represented as elements in a Σ-algebra since the syntactic form of the labels may make more distinctions than are required. Since there is a unique homomorphism from the term algebra to any Σ-algebra which induces a congruence on the elements of the term algebra, this congruence is then used to equate labels that have the same semantics. Bisimulation is defined with respect to this congruence. Moreover, it is possible to work in a more general manner with a congruence over the labels without considering the specific Σ-algebra, and all the results for the extended *tyft/tyxt* format are presented in this way.

Since the aim of formats is to prove theorems about process algebras based on structural operational semantics in a syntactic manner, the extended *tyft/tyxt* format is a logical extension to the existing notion of format. Moreover, this syntactic approach permits the comparison of semantic equivalences. When one extended transition system specification is summed with another, it becomes possible using the extended *tyft/tyxt* format to give conditions under which the bisimulation over the summed systems is either coarser or finer than the bisimulation on the original system.

The main contributions of this paper are the introduction of the extended *tyft/tyxt* format which deals with labels syntactically as opposed to schematically, the congruence result for this format, and the two extension theorems which describe the conditions under which semantic equivalences such as bisimulation may be refined or abstracted. This paper considers extending the *tyft/tyxt* format. This format does not cover predicates or negative premises. Predicates are discussed in Section 5.3 and the use of negative premises is an issue for further work.

In Section 2, the extended *tyft/tyxt* format is developed and it is shown that this format gives congruence, and how process algebra rules can be expressed in this format. Section 3 introduces the extensions and the results for the

two types of extensions. In Section 4, the results are used to compare two different bisimulations. Section 5 compares the extended *tyft/tyxt* format to the *tyft/tyxt* format, and discusses related work. The last two sections look at further work and conclusions respectively. The appendix gives the proof of one of the main theorems. An earlier version of this paper appeared as [24].

2 A new format

2.1 Definitions

These universal algebra definitions covering many-sorted, signature, term, substitution, algebra, homomorphism and congruence, although standard, are presented here to fix notation for the remainder of the paper.

Definition 1 *For any set S, an* S-sorted set A *is a family $\{A_s\}_{s\in S}$ of sets indexed by S. Intersection, union, difference and subset are defined componentwise. Let $S' \subseteq S$, then $A_{S'}$ is the S'-sorted subset of A.*

Definition 2 *A* signature Σ *is a pair (S, F) where S is a set of* sorts *and F is a set of* function symbols (operators) *such that F is equipped with a mapping, type $: F \to S^* \times S$. If type$(f) = (\epsilon, s)$ for some $s \in S$ where ϵ is the empty string, then f is a* constant symbol. *Write $f : w \to s$ for $f \in F$ with type$(f) = (w, s)$, and $f : s_1 \dots s_n \to s$ if $w = s_1 \dots s_n$, $f :\to s$ if $w = \epsilon$.*

Let V be an S-sorted set of variables disjoint from F. The set of terms over V can be formed. Operators will be written in infix notation when convenient. In the remainder of this subsection, $\Sigma = (S, F)$ is a signature.

Definition 3 *Let W be an S-sorted subset of V. For each $s \in S$, the set $T(\Sigma, W)_s$* of Σ-terms of sort s *is the least set containing*

- *every $w \in W_s$ of sort s and every constant symbol $f :\to s \in F$,*
- *every $f(t_1, \dots, t_n)$ where $f : s_1 \dots s_n \to s$ is a function symbol in F with range s and every t_i $(1 \leqslant i \leqslant n)$ is a term of sort s_i in $T(\Sigma, W)_{s_i}$.*

$T(\Sigma, W)$ denotes $\{T(\Sigma, W)_s\}_{s\in S}$, the set of Σ-terms over W. *$T(\Sigma, \emptyset)$ denotes the set of* closed *or* ground terms, *abbreviated $\mathbf{T}(\Sigma)$. $T(\Sigma, V)$, abbreviated $\mathbb{T}(\Sigma)$, denotes* open terms.

Σ *is* sensible *if it admits at least one ground term for each sort, i.e. for all $s \in S$, $\mathbf{T}(\Sigma)_s \neq \emptyset$. The* variables of sorts $S' \subseteq S$ in a term $t \in \mathbb{T}(\Sigma)$, *$Var_{S'}(t)$ is defined in the standard manner, and $Var_S(t)$ is written $Var(t)$.*

A substitution σ *is a mapping in* $V \to \mathbb{T}(\Sigma)$ *which preserves sorts, i.e.* $\sigma|V_s : V_s \to \mathbb{T}(\Sigma)_s$ *for each* $s \in S$*. A substitution* σ *is extended to a mapping* $\sigma : \mathbb{T}(\Sigma) \to \mathbb{T}(\Sigma)$ *in the standard way.*

Definition 4 *A* Σ-algebra *consists of an* S*-sorted family of non-empty carrier sets* $\{\mathcal{A}_s\}_{s\in S}$*, also denoted* $\mathcal{A}$*; and a total function* $f^{\mathcal{A}} : \mathcal{A}_{s_1} \times \ldots \times \mathcal{A}_{s_n} \to \mathcal{A}_s$ *for each* $f \in F$ *such that* $f : s_1 \ldots s_n \to s$*.*

Definition 5 *Let* $\mathcal{A}$ *and* $\mathcal{B}$ *be two* Σ*-algebras. A* Σ-homomorphism $h : \mathcal{A} \to \mathcal{B}$ *is a family of maps* $\{h_s : \mathcal{A}_s \to \mathcal{B}_s\}_{s\in S}$ *such that for all* $f : s_1 \ldots s_n \to s \in F$ *and* $a_1 \in \mathcal{A}_{s_1}, \ldots, a_n \in \mathcal{A}_{s_n}$*,* $h_s(f^{\mathcal{A}}(a_1, \ldots, a_n)) = f^{\mathcal{B}}(h_{s_1}(a_1), \ldots, h_{s_n}(a_n))$*.*

Definition 6 *Let* $\mathcal{A}$ *be a* Σ*-algebra. A* Σ-congruence *on* $\mathcal{A}$ *is an* S*-sorted equivalence relation* $\equiv$ *which is compatible with all function symbols, i.e.* $\equiv = \{\equiv_s\}_{s\in S}$*, and for all* $s \in S$*,* $\equiv_s \subseteq \mathcal{A}_s \times \mathcal{A}_s$ *is reflexive, symmetric and transitive, and for any* $f : s_1 \ldots s_n \to s \in F$ *and for all* $a_i, b_i \in \mathcal{A}_{s_i}$ *for* $1 \leqslant i \leqslant n$*,* $a_i \equiv_{s_i} b_i \ (1 \leqslant i \leqslant n) \Rightarrow f^{\mathcal{A}}(a_1, \ldots, a_n) \equiv_s f^{\mathcal{A}}(b_1, \ldots, b_n)$*.*

Both $\mathbb{T}(\Sigma)$ and $\mathbf{T}(\Sigma)$ form Σ-algebras and it can be shown that there is a unique homomorphism denoted $i_{\mathcal{A}}$ from $\mathbf{T}(\Sigma)$ to any Σ-algebra $\mathcal{A}$. For each Σ-algebra, there exists a congruence over $\mathbf{T}(\Sigma)$, defined as $t \equiv_{\mathcal{A}} t'$ whenever $i_{\mathcal{A}}(t) = i_{\mathcal{A}}(t')$ for $t, t' \in \mathbf{T}(\Sigma)$.

The notation $(\ s_1, s_2, \ldots;\ f_1, f_2, \ldots;\ g_1, g_2, \ldots\ )$ will be used for a signature where $s_1, s_2, \ldots \in S$; $f_1, f_2, \ldots$ are the constant symbols in F, and $g_1, g_2, \ldots$ are the remaining function symbols from F.

2.2 Extended transition system specifications

A specific kind of sorted set and signature is required to represent the terms that appear in the rules of the format. Let P be a distinguished sort which is understood to be the sort of processes.

Definition 7 *A signature* $\Sigma = (S \cup \{\mathsf{P}\}, F)$ *is* suitable *if* S *does not contain* P *and is non-empty, and for any function symbol* $f \in F$ *such that* $f : s_1 \ldots s_n \to s$*, whenever* $s \neq \mathsf{P}$ *then for all* $1 \leqslant i \leqslant n$*,* $s_i \neq \mathsf{P}$*.*

Hence only process terms can be constructed from process terms. This is reasonable because of the asymmetry in the way in which processes and labels are treated; moreover, if there is a need for a label term to contain a process term, it is possible to define a label term that would represent the process term. An understanding of the expressive power of this approach is an issue for further work.

For convenience, assume that functions with range sort P take the non-P arguments first and then the arguments of sort P; for example, $f : s_1 \dots s_m \mathsf{P} \dots \mathsf{P} \to \mathsf{P}$ with $n \geqslant 0$ arguments with sort P. S will be used to denote the label (non-P) sorts. For process variables, $x, y, \dots$ range over V_{P} and for label variables, $z, z_s, \dots$ range over V_s for $s \in S$. For open terms in general, $t, t', \dots$ range over $\mathbb{T}(\Sigma)$. For open process terms, $p, q, \dots$ range over $\mathbb{T}(\Sigma)_{\mathsf{P}}$ and for closed process terms, $u, v, \dots$ range over $\mathbf{T}(\Sigma)_{\mathsf{P}}$. For open label terms, $\lambda, \eta, \dots$ range over $\mathbb{T}(\Sigma)_S$, and for closed label terms, $\alpha, \beta, \mu, \nu, \dots$ range over $\mathbf{T}(\Sigma)_S$.

Next, the notion of an extended transition system specification is defined. This definition extends the prior definition [28] by allowing a richer structure for the labels. For the rest of this section, $\Sigma = (S \cup \{\mathsf{P}\}, F)$ indicates a sensible, suitable signature. However, this will not hold when considering extensions in Section 3 where the concepts suitable and sensible will be used separately.

Definition 8 *An* extended transition system specification (eTSS) *is a pair* $\mathcal{E} = (\Sigma, R)$ *and* R *a set of rules of the form*

$$\frac{\{p_i \xrightarrow{\lambda_i} p_i' \mid i \in I\}}{p \xrightarrow{\lambda} p'}$$

where I *is an index set,* $p_i, p_i', p, p' \in \mathbb{T}(\Sigma)_{\mathsf{P}}$*, and* $\lambda_i, \lambda \in \mathbb{T}(\Sigma)_S$ *for* $i \in I$*.*

If r *is a rule in the format above, then the elements of* $\{p_i \xrightarrow{\lambda_i} p_i' \mid i \in I\}$ *are the* premises *or* hypotheses *of* r*, and* $p \xrightarrow{\lambda} p'$ *is the* conclusion *of* r*. A rule with* $I = \emptyset$ *is an* axiom *and is written* $p \xrightarrow{\lambda} p'$*. An expression of the form* $p \xrightarrow{\lambda} p'$ *with* $\lambda \in \mathbb{T}(\Sigma)_S$ *and* $p, p' \in \mathbb{T}(\Sigma)$ *is a* transition (labelled with λ)*;* p *is the* source*, and* p' *is the* target *of the transition.* $\phi, \psi, \chi, \dots$ *are used to range over transitions. The notions of closed, substitution and Var can be extended to transitions and rules in the obvious way.*

Definition 9 *A* proof *of a transition* ψ *from* $\mathcal{E} = (\Sigma, R)$ *is a well-founded, upwardly branching tree of which the nodes are labelled by transitions* $p \xrightarrow{\lambda} p'$ *with* $\lambda \in \mathbb{T}(\Sigma)_S$ *and* $p, p' \in \mathbb{T}(\Sigma)_{\mathsf{P}}$*, such that*

- *the root is labelled with* ψ*,*
- *if* χ *is the label of a node* π *and* $\{\chi_i \mid i \in I\}$ *is the set of labels of the nodes directly above* π*, then there is a rule* $\frac{\{\phi_i \mid i \in I\}}{\phi}$ *in* R *and a substitution* $\sigma : V \to \mathbb{T}(\Sigma)$ *such that* $\chi = \sigma(\phi)$ *and* $\chi_i = \sigma(\phi_i)$ *for all* $i \in I$*.*

If a proof ψ *from* $\mathcal{E}$ *exists,* ψ *is* provable *from* $\mathcal{E}$*, notation* $\mathcal{E} \vdash \psi$*. A proof is* closed *if it only contains closed transitions.*

Lemma 10 *Let* $\lambda \in \mathbf{T}(\Sigma)_S$*,* $u, u' \in \mathbf{T}(\Sigma)_{\mathsf{P}}$ *such that* $\mathcal{E} \vdash u \xrightarrow{\lambda} u'$*. Then* $u \xrightarrow{\lambda} u'$ *is provable by a closed proof.*

$$\frac{}{z_{\mathsf{A}}.x \xrightarrow{\mathsf{act}(z_{\mathsf{A}},z_{\mathsf{K}})} x} \qquad \frac{x \xrightarrow{z_{\mathsf{Act}}} y}{x + x' \xrightarrow{z_{\mathsf{Act}}} y} \qquad \frac{x \xrightarrow{z_{\mathsf{Act}}} y}{x' + x \xrightarrow{z_{\mathsf{Act}}} y}$$

$$\frac{x \xrightarrow{z_{\mathsf{Act}}} y}{x \mid x' \xrightarrow{z_{\mathsf{Act}}} y \mid x'} \qquad \frac{x \xrightarrow{z_{\mathsf{Act}}} y}{x' \mid x \xrightarrow{z_{\mathsf{Act}}} x' \mid y} \qquad \frac{x \xrightarrow{z_{\mathsf{Act}}} y \quad x' \xrightarrow{z'_{\mathsf{Act}}} y'}{x \mid x' \xrightarrow{\mathsf{comb}(z_{\mathsf{Act}},z'_{\mathsf{Act}})} y \mid y'}$$

Fig. 1. Rules for $\mathcal{E}_{\mathrm{MP}}$

The following is a running example which will be used throughout this section. It will be used in Section 2.6 to express a subset of multiprocessor CCS [32]. Note that instead of a schematic prefix operator $a.x$ taking a single argument x as used in *tyft/tyxt* format, a prefix operator with two arguments, $z.x$ is used, hence there is only one prefix operator. This illustrates how the schematic approach differs from the syntactic.

Example 11 *Let* A *be a set of actions and let* K *be a set of labels, both disjoint from previously defined sets.* A *and* K *will also be used as sort names. Consider the signature* $\Sigma_{\mathrm{MP}} = (\ \mathsf{A}, \mathsf{K}, \mathsf{Act}, \mathsf{P};\ \{\mathsf{a}\}_{\mathsf{a}\in\mathsf{A}}, \{\mathsf{k}\}_{\mathsf{k}\in\mathsf{K}}, \mathsf{nil};\ .\,, +, \mid, \mathsf{act}, \mathsf{comb}\ )$ *with*

$$\begin{array}{llll} \mathsf{a} :\rightarrow \mathsf{A} \quad \forall \mathsf{a} \in \mathsf{A} & \mathsf{act} : \mathsf{A}, \mathsf{K} \rightarrow \mathsf{Act} & \mathsf{nil} :\rightarrow \mathsf{P} & . : \mathsf{A}, \mathsf{P} \rightarrow \mathsf{P} \\ \mathsf{k} :\rightarrow \mathsf{K} \quad \forall \mathsf{k} \in \mathsf{K} & \mathsf{comb} : \mathsf{Act}, \mathsf{Act} \rightarrow \mathsf{Act} & + : \mathsf{P}, \mathsf{P} \rightarrow \mathsf{P} & \mid : \mathsf{P}, \mathsf{P} \rightarrow \mathsf{P} \end{array}$$

The rules R_{MP} *are given in Figure 1. Let* $\mathcal{E}_{\mathrm{MP}} = (\Sigma_{\mathrm{MP}}, R_{\mathrm{MP}})$*. Clearly this is a sensible and suitable signature. The transition* $\mathsf{a.nil} + \mathsf{b.nil} \xrightarrow{\mathsf{act(b,k)}} \mathsf{nil}$ *can be proved by constructing the appropriate proof tree.*

Since this work relates to labelled transition systems, and it is clear that the labels of the transition will be many-sorted, a suitably modified definition of labelled transition system is required.

Definition 12 *An* S-sorted labelled transition system (LTS) *is a labelled transition system* $\mathcal{L} = (States, Actions, \rightarrow)$ *where Actions is an* S*-sorted set of transition labels. Write* $s \xrightarrow{a} s'$ *for* $(s, a, s') \in \rightarrow$*.*

Definition 13 *The* S-sorted labelled transition system $TS(\mathcal{E})$ specified by the eTSS $\mathcal{E}$ *is given by* $TS(\mathcal{E}) = (\mathbf{T}(\Sigma)_{\mathsf{P}}, \mathbf{T}(\Sigma)_S, \rightarrow)$ *where* $\rightarrow \subseteq \mathbf{T}(\Sigma)_{\mathsf{P}} \times \mathbf{T}(\Sigma)_S \times \mathbf{T}(\Sigma)_{\mathsf{P}}$ *is defined by* $u \xrightarrow{\alpha} u' \iff \mathcal{E} \vdash u \xrightarrow{\alpha} u'$*.*

Definition 14 *Two eTSSs* $\mathcal{E}$ *and* $\mathcal{E}'$ *are* transition equivalent *if* $TS(\mathcal{E}) = TS(\mathcal{E}')$*.*

The standard definition of bisimulation [34] is not useful since it would only consider syntactically-equal labels. Assume that there is an S-sorted equiva-

lence that identifies terms that are to be considered the same.

Definition 15 *Let $\mathcal{L} = (States, Actions, \rightarrow)$ be an S-sorted LTS, and let $\equiv$ be an S-sorted equivalence relation on Actions. A* strong bisimulation with respect to an equivalence relation $\equiv$ *is a binary relation $\mathcal{R} \subseteq States \times States$ such that $(s,t) \in \mathcal{R}$ only if for all $a \in Actions$*

(1) whenever $s \xrightarrow{a} s'$, then there exists $t' \in States$ and $b \in Actions$ such that $t \xrightarrow{b} t'$, $a \equiv b$ and $(s',t') \in \mathcal{R}$
(2) whenever $t \xrightarrow{a} t'$, then there exists $s' \in States$ and $b \in Actions$ such that $s \xrightarrow{b} s'$, $a \equiv b$ and $(s',t') \in \mathcal{R}$.

Two states, s and t are strongly bisimilar with respect to $\equiv$, *$s \sim_{\equiv} t$, if there exists a strong bisimulation $\mathcal{R}$ with respect to $\equiv$ such that $(s,t) \in \mathcal{R}$. The relation $\sim_{\equiv} = \bigcup\{\mathcal{R} \mid \mathcal{R}$ is a strong bisimulation with respect to $\equiv\}$ is the largest strong bisimulation with respect to $\equiv$ and is an equivalence relation, hence the name* strong equivalence with respect to $\equiv$.

This definition means that only transitions with labels of the same sort are compared and this, as will be shown in the rest of the paper, is a powerful mechanism for comparing process algebras and their semantic equivalences. Note that for this definition, the equivalence is only required over label terms $\mathbf{T}(\Sigma)_S$, hence the definition of the equivalence over process terms $\mathbf{T}(\Sigma)_{\mathsf{P}}$ is irrelevant.

When expressing process algebras in the extended *tyft/tyxt* format, a Σ-algebra will be used to define the semantics of the actual labels and this will induce a congruence over the term algebra – this will equate the label terms that are considered as the same. However, it is possible to consider the congruence of bisimulation in a general manner, and without considering the specific Σ-algebra used. The issue of Σ-algebras and induced congruences will be discussed further in Section 2.6 which considers how process algebras can be expressed in the extended *tyft/tyxt* format.

2.3 Extended tyft/tyxt *format*

The general definition of an eTSS is now made more specific to ensure desirable properties, such as congruence. In the rest of this section, this work follows much the same path as Groote and Vaandrager [28] in showing congruence, although the definitions and results require more care because of the new way of dealing with labels and additional conditions are required. The format presented here differs from the *tyft/tyxt* format [28] in the fact that the schematic labels on the transitions are replaced with label terms, and the function in the source of the conclusion takes label terms as arguments as well

as process variables. Additionally, the introduction of label terms requires conditions on the relationships between the variables that appear in these terms. For a further discussion of the differences, see Section 5.1.

Informally, the relationship between the variables in this definition can be described as follows: all the x_j's and y_i's are distinct. λ and p can contain any variables; however, the λ_i's must have distinct variables from each other and from the p_i's and the η_k's. Also the η_k's must have distinct variables from each other. Example 11 is in extended *tyft/tyxt* format.

Definition 16 *A rule in R is in* extended tyft format *if it has the form*

$$\frac{\{p_i \xrightarrow{\lambda_i} y_i \mid i \in I\}}{f(\eta_1, \ldots, \eta_m, x_1, \ldots, x_n) \xrightarrow{\lambda} p}$$

with I an index set, $f : s_1 \ldots s_m \mathsf{P} \ldots \mathsf{P} \to \mathsf{P} \in F$, x_j $(1 \leqslant j \leqslant n)$ and y_i $(i \in I)$ all different variables from V_{P}, $p \in \mathbb{T}(\Sigma)_{\mathsf{P}}$, $\lambda \in \mathbb{T}(\Sigma)_S$,

- *$\eta_k \in \mathbb{T}(\Sigma)_{s_k}$ such that $Var_S(\eta_k) \subset V_S - \bigcup_{\substack{1 \leqslant l \leqslant m \\ l \neq k}} Var_S(\eta_l)$ for $1 \leqslant k \leqslant m$,*
- *$\lambda_i \in \mathbb{T}(\Sigma)_S$ for $i \in I$ such that $Var_S(\lambda_i) \subset V_S - (\bigcup_{l \in I, l \neq i} Var_S(\lambda_l) \cup \bigcup_{1 \leqslant k \leqslant m} Var_S(\eta_k))$ for all $i \in I$,*
- *$p_i \in \mathbb{T}(\Sigma)_{\mathsf{P}}$ such that $Var_S(p_i) \subset V_S - \bigcup_{l \in I} Var_S(\lambda_l)$ for $i \in I$.*

A rule in R is in extended tyxt format *if it has the form*

$$\frac{\{p_i \xrightarrow{\lambda_i} y_i \mid i \in I\}}{x \xrightarrow{\lambda} p}$$

with I an index set, x and y_i $(i \in I)$ all different variables from V_{P}, $p \in \mathbb{T}(\Sigma)_{\mathsf{P}}$, $\lambda \in \mathbb{T}(\Sigma)_S$,

- *$\lambda_i \in \mathbb{T}(\Sigma)_S$ such that $Var_S(\lambda_i) \subset V_S - \bigcup_{l \in I, l \neq i} Var_S(\lambda_l)$ for all $i \in I$,*
- *$p_i \in \mathbb{T}(\Sigma)_{\mathsf{P}}$ such that $Var_S(p_i) \subset V_S - \bigcup_{l \in I} Var_S(\lambda_l)$ for $i \in I$.*

$\mathcal{E}$ is in extended tyft/tyxt format *(abbreviated extended* tyft/tyxt*) if every rule in R is either in extended* tyft *format or extended* tyxt *format. A labelled transition system L is extended* tyft/tyxt specifiable *if there exists an eTSS $\mathcal{E}$ in extended* tyft/tyxt *format with $L = TS(\mathcal{E})$.*

2.4 Congruence

An important property for any format is that bisimulation is a congruence with respect to the operators defined by rules of that format. Some additional

definitions are required before this can be shown for the extended *tyft/tyxt* format. After the main congruence result, counter-examples will be given to show that the requirements cannot be further relaxed without losing congruence.

First, a technical condition is required to describe the type of congruence over label terms that will allow the congruence result to be obtained. This technical condition allows an appropriate substitution to be found, and hence a matching transition to be obtained. If the format were simpler in the sense that only variables could appear in the positions of the p_i's, λ_i's and η_k's then any congruence could be used, but the approach taken here allows more generality. In what follows, $\mathcal{E} = (\Sigma, R)$ indicates an eTSS in extended *tyft/tyxt* format.

Definition 17 *Let $\equiv$ be an S-sorted congruence on $\mathbf{T}(\Sigma)_S$. $\equiv$ is* compatible *with $\eta \in \mathbb{T}(\Sigma)_S$ if whenever $\sigma(\eta) \equiv \mu$ for $\mu \in \mathbf{T}(\Sigma)_S$, there exists a substitution σ' such that $\mu = \sigma'(\eta)$ and $\sigma(z) \equiv \sigma'(z)$ for all $z \in Var_S(\eta)$.*

$\equiv$ is compatible *with a rule $r \in R$ if it is compatible with any $\eta \in \mathbb{T}(\Sigma)_S$ that appears as the label on the transition of a premise of r or as an argument to the function on the left hand side of the conclusion of r. $\equiv$ is* compatible *with $\mathcal{E}$ if $\equiv$ is compatible with all rules in R.*

Note that if $\eta = z$, then the required condition is always fulfilled. Furthermore, note that the occurrence of repeated variables in η results in the only compatible congruence being syntactic equivalence. Compatibility can be obtained by imposing a few conditions on the functions of the Σ-algebra used to represent the process algebra labels, and hence is a reasonable technical condition. For further discussion of the compatibility requirement, see Section 2.5.

The following definition is required to ensure that there are no cycles of variable references appearing in the premises, and hence to prove congruence. It has been shown that for the original *tyft/tyxt* format that any rule can be written in a well-founded form [21]. See the related work section for further discussion on this.

Definition 18 *Let $\mathcal{E} = (\Sigma, R)$ be an eTSS. Let $U = \{p_i \xrightarrow{\lambda_i} p'_i \mid i \in I\}$ be a set of transitions of $\mathcal{E}$. The* dependency graph *of U is a directed (unlabelled) graph with*

- *Nodes:* $\bigcup_{i \in I} Var_{\mathsf{P}}(p_i \xrightarrow{\lambda_i} p'_i)$,
- *Edges:* $\{\langle x, y\rangle \mid x \in Var_{\mathsf{P}}(p_i), y \in Var_{\mathsf{P}}(p'_i)$ *for some* $i \in I\}$.

A set of transitions is well-founded *if any backward chain of edges in the dependency graph of these transitions is finite. A rule is* well-founded *if the set of its premises is so. Finally, an eTSS is* well-founded *if all of its rules are well-founded.*

Some additional results are required to work with eTSSs conveniently, and it is necessary to prove that both compatibility and well-foundedness are preserved. For the rest of this section, let $\equiv$ be a congruence on $\mathbf{T}(\Sigma)$ compatible with R.

Lemma 19 *There is a well-founded eTSS $\mathcal{E}' = (\Sigma, R')$ in extended* tyft *format which is transition equivalent to $\mathcal{E} = (\Sigma, R)$ such that $\equiv$ is compatible with R'.*

Proof. Define R' to consist of every extended *tyft* rule of R, as well as the extended *tyft* rules r_f ($f \in F$) created from each extended *tyxt* rule $r \in R$ by replacing x with $f(z_1, \ldots, z_m, x_1, \ldots, x_n)$ where the variables do not appear in r.

The rules in R' are well-founded, since none of the premises have been changed. $\equiv$ is compatible with R' since only terms of the form $f(z_1, \ldots, z_m, x_1, \ldots, x_n)$ have been added. It can be shown that a closed proof of a transition from $\mathcal{E}$ is a proof of that transition from $\mathcal{E}'$, and *vice versa*. $\square$

Definition 20 *Let $r \in R$. A variable in $Var_{\mathsf{P}}(r)$ is* free *if it does not occur in the left hand side of the conclusion or in the right hand side of a premise. A rule r is* pure *if it is well-founded and contains no free variables from V_{P}. $\mathcal{E}$ is* pure *if all its rules are pure.*

Definition 21 *Let $r \in R$. A variable in $Var_S(r)$ is* label-free *if it does not occur in the label of a premise or in the left hand side of the conclusion. A rule r is* label-pure *if it contains no label-free variables from V_S. $\mathcal{E}$ is* label-pure *if all its rules are label-pure.*

Lemma 22 *There is a pure, label-pure eTSS $\mathcal{E}' = (\Sigma, R')$ in extended* tyft *format which is transition equivalent to $\mathcal{E} = (\Sigma, R)$ such that $\equiv$ is compatible with R'.*

Proof. From Lemma 19, $\mathcal{E}$ is in extended *tyft* format. Let R' consist of every pure and label-pure rule of R as well as extended *tyft* rules created from each non-pure or non-label-pure rule in R where every possible substitution of closed terms is applied to the free variables from V.

$\mathcal{E}'$ is well-founded, since for each rule only edges have been removed from its dependency graph. $\equiv$ is compatible with R' since only free and label-free variables have been modified. R' is in extended *tyft* format since no left hand side of any conclusion has been modified, and only closed terms have been used in the substitutions. Every closed proof for a transition from $\mathcal{E}$ is also a proof for the transition from $\mathcal{E}'$ and *vice versa*. $\square$

The eTSS in Example 11 is well-founded and pure. However, it is not label-pure because of the axiom. Any congruence over the label terms is compatible with the rules since all terms of interest are single variables.

Before the congruence result, it is interesting to consider the proof technique which will be used for most major results in this paper, and by presenting it here, the proofs will be shorter. This proof technique is based on the approach taken in [28] but is more complex because of the use of label variables. Before the technique is presented, the following result is required to partition the variables which appear in a extended *tyft* rule. A similar result can be obtained for a extended *tyxt* rule.

Lemma 23 *Let r be a well-founded extended* tyft *rule of the form*

$$\frac{\{p_i \xrightarrow{\lambda_i} y_i \mid i \in I\}}{f(\eta_1, \ldots, \eta_m, x_1, \ldots, x_n) \xrightarrow{\lambda} p}$$

Let $depth(x)$ for $x \in Var_{\mathsf{P}}(r)$ be the length of the maximal backward chain of edges in the dependency graph rooted at x. Then the variables of r can be classified as follows

- $X = \{x_i \mid 1 \leqslant i \leqslant n\}$
- $Y = \{y_i \mid i \in I\}$ $\quad Y_d = \{y \in Y \mid depth(y) = d\}$ *for* $d \geqslant 0$
- $Y_f = Var_{\mathsf{P}}(r) - (X \cup Y)$
- $Z = \bigcup_{1 \leqslant k \leqslant m} Var_S(\eta_k)$
- $Z' = \bigcup_{i \in I} Var_S(\lambda_i)$ $\quad Z'_d = \bigcup_{y_i \in Y_d} Var_S(\lambda_i)$ *for* $d \geqslant 0$
- $Z'' = \bigcup_{i \in I} Var_S(p_i) - Z$
- $Z_f = Var_S(\lambda) \cup Var_S(p) - (Z \cup Z' \cup Z'')$.

where X, Y, Y_f partition $Var_{\mathsf{P}}(r)$ and Z, Z', Z'', Z_f partition $Var_S(r)$. Additionally the sets Y_d partition Y and the sets Z'_d partition Z'. If r is pure then $Y_f = \emptyset$. If r is label-pure then $Z_f = \emptyset$ and $Z'' = \emptyset$.

Proof. Consider the dependency graph G of the premises of r. Because r is in extended *tyft* format, each node in G has at most finitely many incoming nodes, since each y_i is distinct and each p_i is a finite term. Hence G is a finitely branching tree, since it can have no cycles. For each node x of G, its subgraph is a finitely branching tree, since there are no cycles. If this graph were infinite, then by Koenig's Lemma, there would exist an infinite backward chain, contradicting the well-foundedness of G. Hence this graph is finite, and it is possible to define $\text{depth}(x) \in \mathbb{N}$ as the length of the maximal backward chain of edges in the subgraph associated with x. Hence $\text{depth}(x)$ is well-defined and the sets Y_d form a partition of Y. $\square$

To summarise, X contains the process variables that appear in the source

of the conclusion, Y the process variables that appear in the targets of the premises and Y_f all other process variables that appear in the rule (these are the free variables of the rule). Z contains all label variables that appear in the source of the conclusion, Z' all label variables which appear in the labels of the transitions, Z'' all label variables which appear in the sources of the premises but not in the source of the conclusion and Z_f all other label variables that appear in the rule. Z'' and Z_f together contain all label-free variables. The general proof technique used in this paper can be then described as follows:

- The aim is to prove that given a transition, a different transition with certain properties exists (or alternatively the given transition has certain properties).
- Given a transition, there is a closed proof of this transition (and sometimes it can be assumed that the rules are pure, label-pure extended *tyft*).
- Work by induction on the depth of the proof. First show that the required transition can be found by using a different substitution on an axiom and show that the required properties hold of it.
- Next show that the required transition can be found for transitions with proofs involving more than an axiom.
  - Assume appropriate transitions can be found for all transitions with shorter proofs.
  - Let r be the well-founded extended *tyft* rule which is the last used in the proof together with σ a substitution.
  - Use the well-foundedness property to classify the variables in r as in Lemma 23. Perform induction on d to construct the appropriate substitution σ' from σ for each premise in turn.
  - Once σ' is constructed for all variables in r, use the inductive hypothesis (for the depth of the proof) on the premises of r, and then use the properties of σ' to argue that this is a proof of the required transition.

The following result states that two terms of sort P with subterms that are related (either by the congruence if they have sort S or by bisimulation up to the congruence if they have sort P) are bisimilar up to the congruence. Hence it can be concluded that the bisimulation up to the congruence is a congruence itself.

Theorem 24 *Let $\Sigma = (S \cup \{\mathsf{P}\}, F)$ be a sensible, suitable signature, and $\mathcal{E} = (\Sigma, R)$ be a well-founded extended transition system specification in extended* tyft/tyxt *format. Let $\equiv$ be a congruence on $\mathbf{T}(\Sigma)$ compatible with R. For all $f \in F$ such that $f : s_1 \dots s_m \mathsf{P} \dots \mathsf{P} \to \mathsf{P}$, for all terms $\mu_k, \nu_k \in \mathbf{T}(\Sigma)_S$ $(1 \leqslant k \leqslant m)$, and for all terms $u_j, v_j \in \mathbf{T}(\Sigma)_{\mathsf{P}}$ $(1 \leqslant j \leqslant n)$,*

$$\mu_k \equiv \nu_k \ (1 \leqslant k \leqslant m) \text{ and } u_j \sim_{\equiv} v_j \ (1 \leqslant j \leqslant n) \Rightarrow \\ f(\mu_1, \dots, \mu_m, u_1, \dots, u_n) \sim_{\equiv} f(\nu_1, \dots, \nu_m, v_1, \dots v_n).$$

Proof. Let $\mathcal{R}$ be the least relation satisfying

- $\sim_{\equiv} \subseteq \mathcal{R}$,
- for all $f \in F$ such that $f : s_1 \dots s_m \mathsf{P} \dots \mathsf{P} \to \mathsf{P}$, for all terms $\mu_k, \nu_k \in \mathbf{T}(\Sigma)_S$ and for all terms $u_j, v_j \in \mathbf{T}(\Sigma)_{\mathsf{P}}$,
  $\mu_k \equiv \nu_k$ $(1 \leqslant k \leqslant m)$ and $u_j \,\mathcal{R}\, v_j$ $(1 \leqslant j \leqslant n) \Rightarrow$
  $f(\mu_1, \dots, \mu_m, u_1, \dots, u_n) \,\mathcal{R}\, f(\nu_1, \dots, \nu_m, v_1, \dots v_n)$.

It is enough to show $\mathcal{R}$ is a bisimulation since $\sim_{\equiv} \subseteq \mathcal{R}$. Assume $u \,\mathcal{R}\, v$. There are two cases – the first is simple since $u \sim_{\equiv} v$. The second requires it to be shown that whenever $\mathcal{E} \vdash f(\mu_1, \dots, \mu_m, u_1, \dots, u_n) \xrightarrow{\alpha} u'$, $\mu_k \equiv \nu_k$ for all k and $u_j \,\mathcal{R}\, v_j$ for all j then there is a $v' \in \mathbf{T}(\Sigma)_{\mathsf{P}}$ and $\alpha' \in \mathbf{T}(\Sigma)_S$ such that $\mathcal{E} \vdash f(\nu_1, \dots, \nu_m, v_1, \dots, v_n) \xrightarrow{\alpha'} v', \alpha \equiv \alpha'$ and $u' \,\mathcal{R}\, v'$.

By the lemmas, there is a proof T of $f(\mu_1, \dots, \mu_m, u_1, \dots, u_n) \xrightarrow{\alpha} u'$ that contains only closed transitions, and the rules in R are pure, label-pure and in extended *tyft* format. Let r be the last rule used in T, with a substitution σ. Assume r has the form given in Lemma 23. Then $\sigma(\eta_k) = \mu_k$ for all k, $\sigma(x_j) = u_j$ for all j, $\sigma(p) = u'$ and $\sigma(\lambda) = \alpha$. The base case (r an axiom) for the induction on the proof tree is straightforward since there are no variables from premises. Assume that the proof of the transition involves more than an axiom and that the required property holds for all transitions with shorter proofs.

The variables in r can be classified as in Lemma 23. Y_f, Z_f and Z'' are empty. Define a substitution σ' that satisfies the following properties.

(1) $\sigma'(x_j) = v_j$ for $1 \leqslant j \leqslant n$
(2) $\sigma(y) \,\mathcal{R}\, \sigma'(y)$ for $y \in X \cup Y$
(3) $\mathcal{E} \vdash \sigma'(p_i \xrightarrow{\lambda_i} y_i)$ for $i \in I$
(4) $\sigma'(z) \equiv \sigma(z)$ for $z \in Z \cup Z'$.

To start, let $\sigma'(x_j) = v_j$ for all j, $\sigma'(y) = \sigma(y)$ for $y \in V_{\mathsf{P}} - (X \cup Y)$, $\sigma'(z) = \sigma(z)$ for $z \in V_S - (Z \cup Z')$.

First consider Z. Since $\sigma(\eta_k) = \mu_k \equiv \nu_k$ and $\equiv$ is compatible with R, there is a substitution σ'' such that $\sigma''(\eta_k) = \nu_k$ and for all $z \in \mathrm{Var}_S(\eta_k)$, $\sigma(z) \equiv \sigma''(z)$. Let $\sigma'(z) = \sigma''(z)$. This can be done for all η_k since there are no variables shared between the terms.

The proof proceeds by induction on d. When σ' is defined for $y \in X \cup Y_0 \cup \ldots \cup Y_d$ and $z \in Z \cup Z'_0 \cup \ldots \cup Z'_d$ $(d \geqslant 0)$, then $\beta(d)$, $\gamma(d)$ and $\delta(d)$ will hold.

- $\beta(d) : \sigma(y) \,\mathcal{R}\, \sigma'(y)$ for $y_i \in X \cup Y_0 \cup \ldots \cup Y_d$
- $\gamma(d) : \mathcal{E} \vdash \sigma'(p_i \xrightarrow{\lambda_i} y_i)$ for $y_i \in Y_0 \cup \ldots \cup Y_d$
- $\delta(d) : \sigma'(z) \equiv \sigma(z)$ for $z \in Z \cup Z'_0 \cup \ldots \cup Z'_d$.

First, show that $\beta(0)$, $\delta(0)$ and $\gamma(0)$ hold. Since $\sigma(x_j) = u_j$ and $\sigma'(x_j) = v_j$, $\sigma(x_j)\,\mathcal{R}\,\sigma'(x_j)$ for $x_j \in X$. Next consider some $y_i \in Y_0$. and the transition $p_i \xrightarrow{\lambda_i} y_i$. The base case is a simpler version of the inductive step.

Let $d > 0$, and suppose that σ' has been defined for all variables in $X \cup Y_0 \cup \ldots Y_{d-1}$ and $Z \cup Z'_0 \cup \ldots \cup Z'_{d-1}$ such that $\beta(d-1)$, $\gamma(d-1)$ and $\delta(d-1)$ hold. Consider $y_i \in Y_d$ and the transition $p_i \xrightarrow{\lambda_i} y_i$. By the definition of Y_d and the fact that r is pure, $\mathrm{Var}_{\mathsf{P}}(p_i) \subseteq X \cup Y_0 \cup \ldots \cup Y_{d-1}$ so $\sigma(y)\,\mathcal{R}\,\sigma'(y)$ for $y \in \mathrm{Var}_{\mathsf{P}}(p_i)$ by $\beta(d-1)$. Also since r is label-pure, $\mathrm{Var}_S(p_i) \subseteq Z$ so $\sigma(z) \equiv \sigma'(z)$ for all $z \in \mathrm{Var}_S(p_i)$. The following fact is required.

Fact Let $p \in \mathbb{T}(\Sigma)_{\mathsf{P}}$ and let ρ, ρ' be substitutions such that for all $x \in \mathrm{Var}_{\mathsf{P}}(t)$, $\rho(x)\,\mathcal{R}\,\rho'(x)$ and for all $z \in \mathrm{Var}_S(p)$, $\rho(z) \equiv \rho'(z)$. Then $\rho(p)\,\mathcal{R}\,\rho'(p)$.

Hence by this fact $\sigma(p_i)\,\mathcal{R}\,\sigma'(p_i)$. There are two cases to consider

- $\sigma(p_i) \sim_{\equiv} \sigma'(p_i)$. Since $\mathcal{E} \vdash \sigma(p_i) \xrightarrow{\sigma(\lambda_i)} \sigma(y_i)$, there are $w \in \mathbf{T}(\Sigma)_{\mathsf{P}}$ and $\alpha_i \in \mathbf{T}(\Sigma)_S$ such that $\mathcal{E} \vdash \sigma'(p_i) \xrightarrow{\alpha_i} w$, $\sigma(\lambda_i) \equiv \alpha_i$ and $\sigma(y_i)\,\mathcal{R}\,w$. Define $\sigma'(y_i) = w$. Moreover, by the compatibility of $\equiv$, suitable values of $\sigma'(z)$ for $z \in \mathrm{Var}_S(\lambda_i)$ can be found. (Since $\mathrm{Var}_S(p_i) \cap \mathrm{Var}_S(\lambda_i) = \emptyset$, this does not affect values assigned to $\mathrm{Var}_S(p_i)$. Also since λ_i has distinct variables, there is no effect on any other transition.)
- there is a function symbol $h \in F$ such that $h : s'_1 \ldots s'_m \mathsf{P} \ldots \mathsf{P} \to \mathsf{P}$ with $\sigma(p_i) = h(\mu'_1, \ldots, \mu'_{m'}, w_1, \ldots, w_{n'})$ and $\sigma'(p_i) = h(\nu'_1, \ldots, \nu'_{m'}, w'_1, \ldots, w'_{n'})$ where $\mu'_{k'} \equiv \nu'_{k'}$ and $w_{i'}\,\mathcal{R}\,w'_{i'}$. Now the induction hypothesis can be applied. Since $\mathcal{E} \vdash h(\mu'_1, \ldots, \mu'_{m'}, w_1, \ldots, w_{n'}) \xrightarrow{\sigma(\lambda_i)} \sigma(y_i)$, there exist a w, and α_i such that $\mathcal{E} \vdash h(\nu'_1, \ldots, \nu'_{m'}, w'_1, \ldots, w'_{n'}) \xrightarrow{\alpha_i} w$, $\sigma(\lambda_i) \equiv \alpha_i$, and $\sigma(y_i)\,\mathcal{R}\,w$. Again define $\sigma'(y_i) = w$. Moreover, by the compatibility of $\equiv$, suitable values of $\sigma'(z)$ for $z \in \mathrm{Var}_S(\lambda_i)$ can be found.

This can be done for all $y \in Y_d$. From this it can be seen that $\beta(d)$, $\gamma(d)$ and $\delta(d)$ hold for all $d \geqslant 0$. Hence the second and third properties hold, and the fourth property holds for $Z \cup Z'$. The first property holds by definition.

$\mathcal{E} \vdash f(\nu_1, \ldots, \nu_m, v_1, \ldots, v_n) \xrightarrow{\sigma'(\lambda)} \sigma'(p)$ since for all $i \in I$, $\mathcal{E} \vdash \sigma'(p_i) \xrightarrow{\sigma'(\lambda_i)} \sigma'(y_i)$. $\sigma'(\lambda) \equiv \sigma(\lambda) = \alpha$, since for all $z \in \mathrm{Var}_S(\lambda)$, $\sigma(z) \equiv \sigma'(z)$ and $\equiv$ is a congruence. $u = \sigma(p)\,\mathcal{R}\,\sigma'(p)$, by the fact above since for all $x \in \mathrm{Var}_{\mathsf{P}}(p)$, $\sigma(x)\,\mathcal{R}\,\sigma'(x)$ and for all $z \in \mathrm{Var}_S(p)$, $\sigma(z) \equiv \sigma'(z)$. $\square$

From this theorem, it can be seen that $\sim_{\equiv}$ is a congruence with respect to terms from $\mathbf{T}(\Sigma)_{\mathsf{P}}$ for all function symbols with sort P. This definition may seem unusual since some arguments are related by $\equiv$. But since the arguments of sort P represent processes, whereas the other sorts represent actions or information that is stored in process terms, this is not unreasonable. Consider the location set prefix $X :: p$ in Kiehn's local/global cause bisimulation [31].

If there are two equivalent processes p and q, then it is reasonable for $X_1 :: p$ and $X_2 :: p$ to be equivalent if $X_1 = X_2$. In [33], a similar effect is obtained by using the idea of given sorts which are sorts with existing semantics.

2.4.1 Counter-examples

In this section, counter-examples will be given to show that the constraints on the extended *tyft/tyxt* format cannot be relaxed for Theorem 24. Note that the relationship between the variables from V_{P} in a rule are the same as those in the paper by Groote and Vaandrager [28] where counter-examples are given to show the constraints on these variables are necessary. So here, the focus is on the relationship between the variables from V_S.

Example 25 *Let* $\Sigma = (\ \mathsf{P}, s, s';\ nil, ok, \{g_a \mid a \in A\};\ g, k, f, \|, h, h'\ )$ *with*

$ok :\rightarrow s'$ $\quad g : s \rightarrow s$ $\quad nil :\rightarrow \mathsf{P}$ $\quad k : \mathsf{P} \rightarrow \mathsf{P}$ $\quad \|: \mathsf{P}, \mathsf{P} \rightarrow \mathsf{P}$

$g_a :\rightarrow s\ \ \forall\, a \in A$ $\quad f : s \rightarrow \mathsf{P}$ $\quad h' : s, s, \mathsf{P} \rightarrow \mathsf{P}$ $\quad h : s, \mathsf{P} \rightarrow \mathsf{P}$

and the axiom $f(z_s) \xrightarrow{z_s} nil$. *This axiom can be seen as the BPA axiom* $a \xrightarrow{a} nil$. *Let* $\equiv$ *be the smallest congruence containing* (g_a, g_b) *and* $(g(g_c), g(g_d))$.

In the following counter-examples, the eTSSs under consideration consist of the signature and axiom given above as well as additional rules. Note that $\equiv$ is not compatible with g_a or g_b.

Counter-example 26 (Repeated label variables appear across the labels of premises) *With the following rule which has non-distinct variables in the labels of the premises,* $f(g_a) \parallel f(g_a) \not\sim_{\equiv} f(g_a) \parallel f(g_b)$, *although* $f(g_a) \sim_{\equiv} f(g_b)$.

$$\frac{x \xrightarrow{z_s} y \ \ x' \xrightarrow{z_s} y'}{x \parallel x' \xrightarrow{ok} nil}.$$

Counter-example 27 (Repeated label variables appear in the source of the conclusion) *With the axiom* $h'(z_s, z_s, x) \xrightarrow{ok} nil$ *where the variables in the source of the conclusion are not distinct,* $h'(g_a, g_a, nil) \not\sim_{\equiv} h'(g_a, g_b, nil)$.

Counter-example 28 (Repeated label variables appear across the source of the conclusion and the labels of premises) *With the following rule where the same variable appears in the source of the conclusion and in a label of a premise,* $h(g_a, f(g_a)) \not\sim_{\equiv} h(g_a, f(g_b))$

$$\frac{x \xrightarrow{z_s} y}{h(z_s, x) \xrightarrow{ok} nil}.$$

Counter-example 29 (The congruence is not compatible with terms in the rule) $\equiv$ *is not compatible with the the rule*

$$\frac{x \xrightarrow{g_a} y}{k(x) \xrightarrow{ok} nil}$$

because g_a *appears in the label of a premise (and* g_a *is not compatible with* $\equiv$*). Assuming the congruence of bisimulation, it would be expected that* $k(f(g_a)) \sim_{\equiv} k(f(g_b))$ *because* $g_a \equiv g_b$*. However, it is not possible using this rule and the given axiom to show that* $k(f(g_b)) \xrightarrow{ok} nil$ *to match the transition* $k(f(g_a)) \xrightarrow{ok} nil$*. Hence without the compatibility condition, the congruence theorem does not hold. A similar counter-example can be shown with the terms* $h(g_a, nil)$ *and* $h(g_b, nil)$*, and the axiom* $h(g_a, x) \xrightarrow{ok} nil$ *where a label term not compatible with* $\equiv$ *appears in the source of the conclusion.*

To see that this does not only apply when constants appear in rules, consider the following examples. $\equiv$ *is not also compatible with the rule*

$$\frac{x \xrightarrow{g(z_s)} y}{k(x) \xrightarrow{z_s} nil}$$

because it is not compatible with $g(z_s)$ *which appears in the premise. Compatibility requires that if* $\sigma(g(z_s)) = g(g_c)$*, there exists* σ' *such that* $\sigma'(g(z_s)) = g(g_d)$ *and* $\sigma(z_s) \equiv \sigma'(z_s)$*, since it is necessary to consider all terms* $\mu \equiv g(g_c)$*. A substitution* σ' *can be found such that* $\sigma'(g(z_s)) = g(g_d)$*, namely the substitution which maps* z_s *to* g_d*, however, it is not the case that* $\sigma(z_s) \equiv \sigma'(z_s)$ *since* $g_c \not\equiv g_d$*. So although* $f(g(g_c)) \sim_{\equiv} f(g(g_d))$*, it is not possible to show* $k(f(g(g_c))) \sim_{\equiv} k(f(g(g_d)))$ *using this rule and the given axiom.* $\equiv$ *is also not compatible with the axiom* $h(g(z_s), x) \xrightarrow{z_s} nil$ *because* $g(z_s)$ *appears in the source of the conclusion and by a similar argument,* $h(g(g_c), nil) \not\sim_{\equiv} h(g(g_d), nil)$*.*

Finally in this section, the requirement that the variables that appear in the source of a premise should be distinct from the variables that appear in the labels of the premises is considered. With the proof technique used in Theorem 24 which is based on well-foundedness, the ordering on premises may make it impossible to find the right substitution. Hence the requirement that variables in sources of premises should be distinct from those in labels of premises is retained. It may be possible to find a unification style technique such as that in [21]. However, even if the well-foundedness condition is not required for congruence, it is still open as to whether a counter-example can be found to show that sharing variables between source of premises and labels of premises breaks congruence.

As shown in Counter-example 29, this technical condition is required for the congruence result, and as will be seen, is required for one of the extension results. As shown in [25], if the following conditions are satisfied, compatibility of equivalence with respect to a term is guaranteed. The result requires that the term in question contains no repeated variables. This is not a serious limitation as when there are repeated variables, the only compatible congruence is the identity relation.

Proposition 30 *Let $\Sigma = (S, F)$ be a signature, and let $\mathcal{A}$ be a Σ-algebra. Let $\lambda \in \mathbb{T}(\Sigma)$ be a term with no repeated variables, then $\equiv_{\mathcal{A}}$ is compatible with λ if for all function symbols g that appear in λ with $g^{\mathcal{A}} : \mathcal{A}_{s_1} \times \ldots \times \mathcal{A}_{s_n} \to \mathcal{A}_s$,*

- *$Im(g^{\mathcal{A}}) \cap Im(g_1^{\mathcal{A}}) = \emptyset$ for all $g_1^{\mathcal{A}} : \mathcal{A}_{s'_1} \times \ldots \times \mathcal{A}_{s'_m} \to \mathcal{A}_s$,*
- *$g^{\mathcal{A}}$ is injective.*

Proof. Let $\alpha \equiv_{\mathcal{A}} \sigma(\lambda)$ for some $\alpha \in \mathbf{T}(\Sigma)$ and a closed substitution σ. It is necessary to find a substitution σ' such that $\sigma'(\lambda) = \alpha$ and $\sigma(z) \equiv_{\mathcal{A}} \sigma'(z)$ for all $z \in \mathrm{Var}(\lambda)$. The proof will proceed by induction on the structure of λ. For the base case, assume $\lambda = z$ for $z \in V$ and define σ' as $\sigma'(z') = \alpha$ if $z' = z$, and $\sigma(z')$ otherwise, then the conditions for compatibility are satisfied.

Next consider $\lambda = g(\lambda_1, \ldots, \lambda_n)$ where $g : s_1 \ldots s_n \to s$. Hence $i_{\mathcal{A}}(\alpha) = i_{\mathcal{A}}(\sigma(g(\lambda_1, \ldots, \lambda_n))) = g^{\mathcal{A}}(i_{\mathcal{A}}(\sigma(\lambda_1)), \ldots, i_{\mathcal{A}}(\sigma(\lambda_n)))$. $g^{\mathcal{A}}(a_1, \ldots, a_n) = i_{\mathcal{A}}(\alpha)$ implies that $\alpha = g(\mu_1, \ldots, \mu_n)$ for some $\mu_1, \ldots, \mu_n$ since $Im(g^{\mathcal{A}}) \cap Im(g_1^{\mathcal{A}}) = \emptyset$ for all $g_1^{\mathcal{A}} : \mathcal{A}_{s'_1} \times \ldots \times \mathcal{A}_{s'_m} \to \mathcal{A}_s$. Hence $g^{\mathcal{A}}(i_{\mathcal{A}}(\sigma(\lambda_1)), \ldots, i_{\mathcal{A}}(\sigma(\lambda_n))) = i_{\mathcal{A}}(g(\mu_1, \ldots, \mu_n)) = g^{\mathcal{A}}(i_{\mathcal{A}}(\mu_1), \ldots, i_{\mathcal{A}}(\mu_n))$. Therefore $i_{\mathcal{A}}(\sigma(\lambda_i)) = i_{\mathcal{A}}(\mu_i)$ for all $1 \leqslant i \leqslant n$ (since $g_{\mathcal{A}}$ is injective) and $\sigma(\lambda_i) \equiv_{\mathcal{A}} \mu_i$ for all $1 \leqslant i \leqslant n$.

By the inductive hypothesis, there are substitutions σ_i such that for each $1 \leqslant i \leqslant n$, $\sigma_i(\lambda_i) = \mu_i$, and $\sigma(z) \equiv_{\mathcal{A}} \sigma_i(z)$ for all $z \in \mathrm{Var}(\lambda_i)$. Construct σ' by defining $\sigma'(z) = \sigma_i(z)$ if $z \in \mathrm{Var}(\lambda_i)$, and $\sigma(z)$ otherwise. This is well-defined since there are no repeated variables. Therefore $\sigma'(\lambda) = \sigma'(g(\lambda_1, \ldots, \lambda_n)) = g(\sigma'(\lambda_1), \ldots, \sigma'(\lambda_n)) = g(\sigma_1(\lambda_1), \ldots, \sigma_n(\lambda_n)) = g(\mu_1, \ldots, \mu_n) = \alpha$. Also for $z \in \mathrm{Var}(\lambda)$, $\sigma'(z) = \sigma_i(z)$ for a unique $1 \leqslant i \leqslant n$, and $\sigma_i(z) \equiv_{\mathcal{A}} \sigma(z)$, hence $\sigma'(z) \equiv_{\mathcal{A}} \sigma(z)$. $\square$

Using this proposition, rules that are sufficient for compatibility meet the following: whenever a function symbol appears in the label of a premise or in the source of the conclusion, its corresponding function in the Σ-algebra must be injective and have a suitably disjoint image.

Hence, function symbols that lead to unnecessary distinctions between label terms when considering the underlying semantics of the labels, should not appear in these positions in the rule. This is because these function symbols are precisely the ones with which non-injective functions would be associated. This is not a major limitation as in many process algebras these positions in rules have single variables, and the more complex terms are likely to appear in the label on the conclusion or in the target of the conclusion, positions which are not affected by the definition of compatibility.

The condition that images of functions must be disjoint is also not a major limitation for the same reason. It is also the case that often images are disjoint because one function symbol deals with the initial construction of the label, and a second function symbol with the same range combines labels in a way that makes them distinct from the initial form of labels. An example is multi-processor CCS [32] which is expressed in the extended *tyft/tyxt* format in the next section. For these reasons, Proposition 30 gives a useful characterisation of compatibility.

2.6 Expressing process algebras in extended tyft/tyxt *format*

As mentioned earlier, when expressing process algebras in this format, a Σ-algebra $\mathcal{A}$ will give the semantics of the actual labels and this will induce a congruence $\equiv_{\mathcal{A}}$ over the term algebra. For an example of the utility of this approach, consider two label terms that are constructed by two applications of an operation that is viewed as commutative, but where the applications have occurred in different orders in each label term. In the process algebra, these labels are considered identical because of the commutativity of the operation, hence when considering bisimulation these labels would match. For this match to occur when this process algebra is described in the extended *tyft/tyxt* format, a congruence is required over label terms to ensure that the matching in bisimulation occurs. An appropriate Σ-algebra will define this congruence.

This can be expressed as a requirement that the transition system of the process algebra is isomorphic to the transition system $\left(\mathbf{T}(\Sigma)_{\mathsf{P}}/\equiv, \mathbf{T}(\Sigma)_S/\equiv, \mathcal{T}\right)$ where $\mathcal{T} = \{S \xrightarrow{A} S' \mid S, S' \in \mathbf{T}(\Sigma)_{\mathsf{P}}/\equiv, A \in \mathbf{T}(\Sigma)_S/\equiv, \forall s \in S, s' \in S', a \in A, s \xrightarrow{a} s'\}$.

The congruence over label terms can also incorporate other relationships between label terms. For example, in parameterised location bisimulation [14], a relation is given over locations and this is used in the matching of labels consisting of actions and location strings in bisimulation.

Before proceeding with the example, some general comments about expressing process algebras in the extended *tyft/tyxt* format are required. First, rule

schemas are introduced as a way in which to specify infinite sets of rules, to describe constant processes with defining equations and to match on labels within a rule, as for instance with the CCS communication rule. In the earlier work involving formats, most authors have used schemas, although not explicitly. In this work, because of the need to give an account of how information is passed from action terms to process terms, these concerns are dealt with in a more explicit manner. Assume a set of schema variables $\mathcal{V}$ disjoint from any other sets.

Definition 31 *A rule schema has the form*

$$\Big\{ \frac{\{p_i \xrightarrow{\lambda_i} p_i' \mid i \in I\}}{p \xrightarrow{\lambda} p'} \mid C_1, \ldots, C_n \Big\}$$

where I is an index set, $p_i, p_i', p, p' \in T(\Sigma, V \cup \mathcal{V})_{\mathsf{P}}$, and $\lambda_i, \lambda \in T(\Sigma, V \cup \mathcal{V})_S$ for $i \in I$. $C_1, \ldots, C_n$ are conditions on the schema variables involving equality and inequality, whether terms are open or closed and the sorts of terms. A rule schema represents the set of rules created by replacing each schema variable by any closed term in $\mathbf{T}(\Sigma)$ in accordance with the conditions.

A rule schema is in extended tyft/tyxt *format, if it obeys the conditions for the extended* tyft/tyxt *format with the exception that any term which is not required to be a variable is in $T(\Sigma, V \cup \mathcal{V})$.*

Proposition 32 *Let $\Sigma = (S \cup \mathsf{P}, F)$ be a suitable signature. Given a rule schema in extended* tyft/tyxt *format then the rules generated by the schema are in extended* tyft/tyxt *format. Moreover, if the rule schema is well-founded then all the rules generated are well-founded.*

It is not possible to achieve a similar result for compatibility since a schema label variable may be replaced by a term.

As examples of rule schemas consider the following. Here $X, Y, \ldots$ denote schema variables of sort P, and $Z_s, \ldots$ denote schema variables of sort $s \in S$. Constants can be defined by the rule schema

$$\Big\{ \quad \frac{X \xrightarrow{z} y}{\mathsf{Cn} \xrightarrow{z} y} \quad \mid \quad \mathsf{Cn} \stackrel{\text{def}}{=} X \text{ for } X \in \mathbf{T}(\Sigma) \quad \Big\}$$

where it is assumed that each constant is assigned a closed term. Matching required by the communication rule cannot be defined by a single rule, since the conditions for the format require that the variables in the labels of the premises are disjoint. The two rule schemas define the same set of communi-

cation rules.

$$\left\{ \quad \frac{x \xrightarrow{\mathsf{act}(Z_{\mathsf{A}})} y \quad x' \xrightarrow{\overline{\mathsf{act}}(Z'_{\mathsf{A}})} y'}{x \mid x \xrightarrow{\tau} y \mid y'} \mid Z_{\mathsf{A}} = Z'_{\mathsf{A}} \quad \right\} \quad \left\{ \quad \frac{x \xrightarrow{\mathsf{act}(Z_{\mathsf{A}})} y \quad x' \xrightarrow{\overline{\mathsf{act}}(Z_{\mathsf{A}})} y'}{x \mid x \xrightarrow{\tau} y \mid y'} \quad \right\}$$

In a different approach to matching for communication, transitions from a rule of the form

$$\frac{x \xrightarrow{z_{\mathsf{Act}}} y \quad x' \xrightarrow{z'_{\mathsf{Act}}} y'}{x \mid x' \xrightarrow{\mathsf{comb}(z_{\mathsf{Act}}, z'_{\mathsf{Act}})} y \mid y'}$$

can have the term $\mathsf{comb}(z_{\mathsf{Act}}, z'_{\mathsf{Act}})$ mapped to τ if the arguments are complementary, or some distinguished value $\perp$ when the arguments are not. Then in the definition of bisimulation, transitions with labels equivalent to $\perp$ are not considered. This has the effect of moving side conditions such as those in the CCS restriction rule, into the semantics of the label terms. A similar approach is taken in [20].

CCS [34], CCS with locations [14], multiprocessor equivalence [32] and pomset bisimulation [15] can be expressed in the extended *tyft/tyxt* format, and in each of these cases the congruence theorem can be applied. For the last three, one eTSS is used with different algebras to obtain the different process algebras and semantic equivalences. For details of these see [23].

2.6.1 Multiprocessor CCS

Consider $\mathcal{E}_{\mathrm{MP}}$ where Σ_{MP} has three additional constants added to the definition given in Example 11: $\perp_{\mathsf{A}} :\rightarrow \mathsf{A}$, $\perp_{\mathsf{K}} :\rightarrow \mathsf{K}$ and $\perp_{\mathsf{Act}} :\rightarrow \mathsf{Act}$ and where R_{MP} is the same as in Example 11.

Next, consider a subset of multiprocessor CCS without a restriction operator and communication between processes. Let A be a set of actions and $P ::= a.P \mid \mathbf{0} \mid P + P \mid P|P$ for $a \in A$. Let $\mathcal{O}_n$ denote the set of n-tuples over $A \cup \{\delta\}$, and let $Allocate(a) = \{O \in \mathcal{O}_n \mid \exists i, O(i) = a, \forall j \neq i, O(j) = \delta\}$. Also define the partial function $+_n$ on $\mathcal{O} \times \mathcal{O} \rightarrow \mathcal{O}$ as $O_1 +_n O_2 = O$ where for all $1 \leqslant i \leqslant n$

$$O(i) = \begin{cases} O_1(i) & \text{if } O_2(i) = \delta \\ O_2(i) & \text{if } O_1(i) = \delta \end{cases}$$

The rules are given in Figure 2. An n *multiprocessor bisimulation* $\mathcal{R}$ is a binary relation such that for any $(p, q) \in \mathcal{R}$ and $O \in \mathcal{O}_n$, the following holds

(1) $p \xrightarrow{O} p'$ implies there exists q' such that $q \xrightarrow{O} q'$ and $(p', q') \in \mathcal{R}$,
(2) $q \xrightarrow{O} q'$ implies there exists p' such that $p \xrightarrow{O} p'$ and $(p', q') \in \mathcal{R}$.

$$\frac{}{a.p \xrightarrow{O} p} \quad \forall O \in Allocate(a) \qquad \frac{p \xrightarrow{O} p'}{p + q \xrightarrow{O} p'} \qquad \frac{p \xrightarrow{O} p'}{q + p \xrightarrow{O} p'}$$

$$\frac{p \xrightarrow{O} p'}{p \mid q \xrightarrow{O} p' \mid q} \qquad \frac{p \xrightarrow{O} p'}{q \mid p \xrightarrow{O} q \mid p'} \qquad \frac{p \xrightarrow{O} p' \quad q \xrightarrow{O'} q'}{p \mid q \xrightarrow{O +_n O'} p' \mid q'}$$

Fig. 2. Rules for n multiprocessor CCS

$$\mathsf{a}^{\mathcal{A}_n} = a \quad \forall \mathsf{a} \in \mathsf{A} \qquad \mathsf{k}^{\mathcal{A}_n} = num(\mathsf{k}) \quad \forall \mathsf{k} \in \mathsf{K}$$

$$\bot_{\mathsf{A}}^{\mathcal{A}_n} = \bot_{\mathsf{A}} \qquad \bot_{\mathsf{K}}^{\mathcal{A}_n} = 0 \qquad \bot_{\mathsf{Act}}^{\mathcal{A}_n} = \bot_{\mathsf{Act}}$$

$$\mathsf{act}^{\mathcal{A}_n}(\zeta_1, \zeta_2) = \begin{cases} (\overbrace{\delta, \ldots, \delta}^{\zeta_2 - 1}, \zeta_1, \overbrace{\delta, \ldots, \delta}^{n - \zeta_2}) & \text{if } \zeta_1 \in A \cup \{\tau\} \text{ and } \zeta_2 \in \mathbb{N}^+ \\ & \text{where } \zeta_1 \text{ is in the } \zeta_2\text{-th} \\ & \text{position of the vector} \\ \bot_{\mathsf{Act}} & \text{otherwise} \end{cases}$$

$$\mathsf{comb}^{\mathcal{A}_n}(\zeta_1, \zeta_2) = \begin{cases} \zeta_1 +_n \zeta_2 & \text{if } \zeta_1, \zeta_2 \in A_n - \{\bot_{\mathsf{Act}}\} \text{ and } \zeta_1 +_n \zeta_2 \text{ defined} \\ \bot_{\mathsf{Act}} & \text{otherwise} \end{cases}$$

Fig. 3. Functions for A_n

To express this process algebra in the format, a Σ_{MP}-algebra is required. Consider the algebra $\mathcal{A}_n$ given in Figure 3, where the carrier sets for A, K and Act are $A \cup \{\bot_{\mathsf{A}}\}$, $\mathbb{N}$ and $A_n \cup \{\bot_{\mathsf{Act}}\}$ respectively, where

$$A_n = \overbrace{(A \cup \{\delta\}) \times \ldots \times (A \cup \{\delta\})}^{n \text{ times}}.$$

It is assumed for each a in A there is a corresponding a in A and also that there is a total order on K with $num : \mathsf{K} \to \mathbb{N}$. Additionally, let the partial function $(a_1, \ldots, a_n) +_n (b_1, \ldots, b_n)$ be defined as equal to $(c_1, \ldots, c_n)$ where for all $1 \leqslant i \leqslant n$, $c_i = a_i$ if $b_i = \delta$ or $c_i = b_i$ if $a_i = \delta$.

This Σ_{MP}-algebra induces a congruence on $\mathbf{T}(\Sigma_{\mathrm{MP}})$. Note that for the correct definition of bisimulation, the only transitions of interest are those that are not labelled with terms of sort Act which are equivalent to $\bot_{\mathsf{Act}}$. However, the notation $\sim^{\mathcal{E}_{\mathrm{MP}}}_{\equiv_{\mathcal{A}_n}}$ will still be used for this congruence. Hence, the labelled transition system under consideration is $(\mathbf{T}(\Sigma_{\mathrm{MP}})_{\mathsf{P}}, \mathbf{T}(\Sigma_{\mathrm{MP}})_S, \mathcal{T})$ where $\mathcal{T} = \{p \xrightarrow{\lambda} p' \mid \lambda \in \mathbf{T}(\Sigma_{\mathrm{MP}})_{\mathsf{Act}}, \lambda \not\sim^{\mathcal{E}_{\mathrm{MP}}}_{\equiv_{\mathcal{A}_n}} \bot_{\mathsf{Act}}\}$.

Consider the processes $\mathsf{a.nil} \mid \mathsf{b.nil}$ and $\mathsf{b.nil} \mid \mathsf{a.nil}$. For convenience, assume that each element of K is subscripted in accordance with num, i.e. $\mathsf{k}_i^{\mathcal{A}_n} = i$.

Clearly for any n, $\mathsf{a.nil} \mid \mathsf{b.nil} \sim^{\mathcal{E}_{\mathrm{MP}}}_{\equiv_{\mathcal{A}_n}} \mathsf{b.nil} \mid \mathsf{a.nil}$. Next consider $n = 2$, then $\mathsf{a.nil} \mid \mathsf{b.nil} \xrightarrow{\mathsf{act(a,k_1)}} \mathsf{nil} \mid \mathsf{b.nil}$, $\mathsf{a.nil} \mid \mathsf{b.nil} \xrightarrow{\mathsf{act(a,k_2)}} \mathsf{nil} \mid \mathsf{b.nil}$, $\mathsf{a.nil} \mid \mathsf{b.nil} \xrightarrow{\mathsf{act(a,k_3)}} \mathsf{nil} \mid \mathsf{b.nil}$, $\mathsf{a.nil} \mid \mathsf{b.nil} \xrightarrow{\mathsf{act(a,k_4)}} \mathsf{nil} \mid \mathsf{b.nil}$, etc. The first transition label is mapped by $i_{\mathcal{A}_2}$ to (a, δ), the second to (δ, a), but all the rest are mapped to $\bot_{\mathsf{Act}}$.

Since $\mathcal{E}_{\mathrm{MP}}$ is in extended *tyft/tyxt* form, and $\equiv_{\mathcal{A}_n}$ is compatible with the rules, bisimulation up to $\equiv_{\mathcal{A}_n}$ is a congruence for all operators in F_{MP}. Hence it can be concluded, for example, that $\mathsf{c.}(\mathsf{a.nil} \mid \mathsf{b.nil}) \sim^{\mathcal{E}_{\mathrm{MP}}}_{\equiv_{\mathcal{A}_n}} \mathsf{c.}(\mathsf{b.nil} \mid \mathsf{a.nil})$.

3 Extensions and semantic equivalence comparison

This section considers extensions up to bisimulation, and investigates under which conditions it is possible to achieve various relationships between the original semantic equivalences and the semantic equivalences of the combined eTSSs. There are two approaches to this and both are presented. These are new results which are not extensions of existing results, and hence full counter-examples for these results are given.

3.1 Sums of eTSSs and conservative extensions

The notion of the asymmetric sum of two eTSSs is required to define the notion of an extension. The second summand does not necessarily involve a sensible signature, and so the definition is asymmetric. Therefore this definition differs from that of Groote and Vaandrager [28]. Since $\mathcal{E}_0 \oplus\!\!\!\triangleright \mathcal{E}_1$ is considered as an extension of $\mathcal{E}_0$, there is an inherent lack of symmetry. For additional results on sums of signatures, algebras and eTSSs, as well as the notion of sort-similarity, see [25].

Definition 33 *Let $\Sigma_i = (S_i \cup \{\mathsf{P}\}, F_i)$ for $i = 0, 1$ be two suitable signatures such that $f \in F_0 \cap F_1$ implies that $type_0(f) = type_1(f)$. The* sum *of Σ_0 and Σ_1, $\Sigma_0 \oplus \Sigma_1$ is the signature $\Sigma_0 \oplus \Sigma_1 = (S_0 \cup S_1 \cup \{\mathsf{P}\}, F_0 \cup F_1)$. When Σ_0 and $\Sigma_0 \oplus \Sigma_1$ are sensible, then $\Sigma_0 \oplus \Sigma_1$ is called* asymmetric *and denoted $\Sigma_0 \oplus\!\!\!\triangleright \Sigma_1$.*

Definition 34 *Let $\mathcal{E}_i = (\Sigma_i, R_i)$ for $i = 0, 1$ be two eTSSs with $\Sigma_0 \oplus \Sigma_1$ defined. The* sum *of $\mathcal{E}_0$ and $\mathcal{E}_1$, $\mathcal{E}_0 \oplus \mathcal{E}_1$, is the eTSS $\mathcal{E}_0 \oplus \mathcal{E}_1 = (\Sigma_0 \oplus \Sigma_1, R_0 \cup R_1)$. If $\Sigma_0 \oplus \Sigma_1$ is asymmetric, then $\mathcal{E}_0 \oplus \mathcal{E}_1$ is* asymmetric *and denoted $\mathcal{E}_0 \oplus\!\!\!\triangleright \mathcal{E}_1$.*

In what follows, let $\Sigma_i = (S_i \cup \{\mathsf{P}\}, F_i)$ for $i = 0, 1$ be two signatures with $\Sigma = \Sigma_0 \oplus\!\!\!\triangleright \Sigma_1$ defined, and let $\mathcal{E}_i = (\Sigma_i, R_i)$ for $i = 0, 1$ be two eTSSs in extended *tyft/tyxt* format with $\mathcal{E} = \mathcal{E}_0 \oplus\!\!\!\triangleright \mathcal{E}_1$ defined and let $\mathcal{E} = (\Sigma, R)$. Additionally, let $S = S_0 \cup S_1$. Let $\equiv_i$ be congruences over $\mathbf{T}(\Sigma_i)_{S_i}$ compatible with $\mathcal{E}_i$ for $i = 0, 1$, and let $\equiv$ be an congruence over $\mathbf{T}(\Sigma_0 \oplus\!\!\!\triangleright \Sigma_1)_S$ compatible with $\mathcal{E}$.

The following definition is standard and the implication $\mathcal{E} \vdash t_0 \xrightarrow{\alpha} t \Leftarrow \mathcal{E}_0 \vdash t_0 \xrightarrow{\alpha} t$ holds trivially because transitions can only be added (this would not be the case if negative transitions were permitted).

Definition 35 *$\mathcal{E}$ is a* conservative extension of *$\mathcal{E}_0$ if for all $t_0 \in \mathbf{T}(\Sigma_0)_\mathsf{P}$, $\alpha \in \mathbf{T}(\Sigma_0)_{S_0}$, and $t \in \mathbf{T}(\Sigma)_\mathsf{P}$, then $\mathcal{E} \vdash t_0 \xrightarrow{\alpha} t \iff \mathcal{E}_0 \vdash t_0 \xrightarrow{\alpha} t$.*

It is possible to achieve a conservative extension result for the extended *tyft/tyxt* format. See [23] for the details of the results and a comparison with Verhoef's result for the *panth* format [41].

3.2 Extensions up to bisimulation

A notion of a conservative extension up to bisimulation has been defined by Groote and Vaandrager [28]. The following is a modification of this definition to take into account the more general definition of bisimulation.

Definition 36 *$\mathcal{E}$ is a* conservative extension of *$\mathcal{E}_0$* up to bisimulation with respect to *$\equiv$ if for all $t_0, u_0 \in \mathbf{T}(\Sigma_0)_\mathsf{P}$, $t_0 \sim_{\equiv}^{\mathcal{E}} u_0 \iff t_0 \sim_{\equiv}^{\mathcal{E}_0} u_0$.*

Clearly a conservative extension is a conservative extension up to bisimulation with respect to an equivalence.

The next two definitions describe the relationships between semantic equivalences, and are the basic definitions for the extension results. The proposition follows immediately.

Definition 37 *$\mathcal{E}$ is a* refining extension of *$\mathcal{E}_0$* up to bisimulation with respect to *$\equiv$ if for all $t_0, u_0 \in \mathbf{T}(\Sigma_0)_\mathsf{P}$, $t_0 \sim_{\equiv}^{\mathcal{E}} u_0 \Rightarrow t_0 \sim_{\equiv_0}^{\mathcal{E}_0} u_0$.*

Definition 38 *$\mathcal{E}$ is an* abstracting extension of *$\mathcal{E}_0$* up to bisimulation with respect to *$\equiv$ if for all $t_0, u_0 \in \mathbf{T}(\Sigma_0)_\mathsf{P}$, $t_0 \sim_{\equiv_0}^{\mathcal{E}_0} u_0 \Rightarrow t_0 \sim_{\equiv}^{\mathcal{E}} u_0$*

Proposition 39 *$\mathcal{E}$ is both a refining and an abstracting extension up to bisimulation with respect to $\equiv$ if and only if it is a conservative extension up to bisimulation with respect to $\equiv$.*

In the following, conditions under which these types of extensions can be achieved are presented. As this is fairly complex, a number of definitions to capture these conditions will be given first, as well as a general lemma required for the results about congruences and sums of eTSSs. Note that refining (abstracting) extension will be used synonymously with refining (abstracting) extension up to bisimulation in the remainder of this paper.

Definition 40 *$\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ is* type-1 *if*

- *there is no extended* tyft *rule in* R_1 *containing a function symbol from* F_0 *in the source of the conclusion that has a conclusion label with a sort from* S_0*, and*
- *there is no extended* tyxt *rule in* R_1 *that has a conclusion label of a sort from* S_0*.*

Definition 41 $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ *is* type-0 *if it is type-1 and no extended* tyft *rule in* R_1 *contains a function symbol from* F_0*.*

Clearly, if $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ is type-0 then it is also type-1. The following lemma is required before the main results of this section can be proved. This lemma states that given a type-1 sum and a proof of a transition where the last rule used came from R_0 then the transition can be proved in $\mathcal{E}_0$.

Lemma 42 *Let* $\mathcal{E}_0$ *and* $\mathcal{E}_1$ *be in extended* tyft/tyxt *format. Let* $\mathcal{E}_0$ *be pure and label-pure, let* $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ *be type-1. Let* $\mathcal{E} \vdash t_0 \xrightarrow{\alpha} t$ *with* $t_0 \in \mathbf{T}(\Sigma_0)_{\mathsf{P}}$*. If the last rule used in the proof of* $\mathcal{E} \vdash t_0 \xrightarrow{\alpha} t$ *is an extended* tyft/tyxt *rule from* R_0 *then* $\mathcal{E}_0 \vdash t_0 \xrightarrow{\alpha} t$*.*

Proof. Let T be a proof of $t_0 \xrightarrow{\alpha} t$ from $\mathcal{E}$ with the last step of T involving an extended *tyft/tyxt* rule from R_0. To show that $\mathcal{E}_0 \vdash t_0 \xrightarrow{\alpha} t$, it will be demonstrated that that T is also a proof of $t_0 \xrightarrow{\alpha} t$ from $\mathcal{E}_0$ and that $\alpha \in \mathbf{T}(\Sigma_0)_{S_0}$ and $t \in \mathbf{T}(\Sigma_0)_{\mathsf{P}}$.

Let r be the last rule used in T. Suppose r is pure, label-pure extended *tyft* (the case that r is pure, label-pure extended *tyxt* is proved in a similar fashion). The case for r an axiom is a simpler version of the induction step. Assume that the proof of the transition consists of more than an axiom and that for all shorter proofs, it has been shown that the proof is a proof in $\mathcal{E}_0$.

Suppose that r has the form given in Lemma 23 and let σ be the substitution. The variables in r can be classified as in Lemma 23 with that Y_f, Z_f and Z'' empty, and $\sigma(f(\eta_1, \ldots, \eta_m, x_1, \ldots, x_n)) = t_0$, $\sigma(\lambda) = \alpha$ and $\sigma(p) = t$.

With induction on d, it will be proven that $\sigma(x) \in \mathbf{T}(\Sigma_0)_{\mathsf{P}}$ for all $x \in X \cup Y$, and that $\sigma(z) \in \mathbf{T}(\Sigma_0)_{S_0}$ for all $z \in Z \cup Z'$. Clearly this holds immediately for X and Z, since $\sigma(f(\eta_1, \ldots, \eta_m, x_1, \ldots, x_n)) = t_0 \in \mathbf{T}(\Sigma_0)_{\mathsf{P}}$.

Let $y_i \in Y_d$ for $d > 0$, then by the inductive hypothesis on d, for $x \in X \cup Y_0 \cup \ldots \cup Y_{d-1}$, $\sigma(x) \in \mathbf{T}(\Sigma_0)_{\mathsf{P}}$ and for $z \in Z \cup Z'_0 \cup \ldots \cup Z'_{d-1}$, $\sigma(z) \in \mathbf{T}(\Sigma_0)_{S_0}$.

Consider $\sigma(p_i) \xrightarrow{\sigma(\lambda_i)} \sigma(y_i)$. Since r is pure and by the definition of Y_d, $\mathrm{Var}_{\mathsf{P}}(p_i) \subseteq X \cup Y_0 \cup \ldots \cup Y_{d-1}$, and since r is label-pure, $\mathrm{Var}_S(p_i) \subseteq Z$, therefore $\sigma(p_i) \in \mathbf{T}(\Sigma_0)_{\mathsf{P}}$. The transition $\sigma(p_i) \xrightarrow{\sigma(\lambda_i)} \sigma(y_i)$ must be generated by a proof whose last step involves a rule from R_0, because the sort of $\sigma(\lambda_i)$ is from S_0 since $r \in R_0$, and $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ is type-1. Hence, by the induction hypothesis on proofs,

$\sigma(\lambda_i) \in \mathbf{T}(\Sigma_0)_{S_0}$, $\sigma(y_i) \in \mathbf{T}(\Sigma_0)_{\mathsf{P}}$ and $\mathcal{E}_0 \vdash \sigma(p_i) \xrightarrow{\sigma(\lambda_i)} \sigma(y_i)$.

This is true for all $d \in \mathbb{N}$, so $\sigma(x) \in \mathbf{T}(\Sigma_0)_{\mathsf{P}}$ for all $x \in X \cup Y$ and $\sigma(z) \in \mathbf{T}(\Sigma_0)_S$ for all $z \in Z \cup Z'$. Since r is pure and label-pure, $\mathrm{Var}(p) \subseteq X \cup Y \cup Z \cup Z'$, therefore $t = \sigma(p) \in \mathbf{T}(\Sigma_0)_{\mathsf{P}}$, and $\mathrm{Var}(\lambda) \subseteq Z \cup Z'$, so $\alpha = \sigma(\lambda) \in \mathbf{T}(\Sigma_0)_{S_0}$ and $\mathcal{E}_0 \vdash t_0 \xrightarrow{\alpha} t$. $\square$

A way to combine congruences is required, and for one of the main results, it is necessary to have a specific relation between the original congruences and their sum. An additional lemma makes the proofs simpler.

Definition 43 *The* sum of $\equiv_0$ and $\equiv_1$ ($\equiv_0 \oplus \equiv_1$) *is the smallest congruence over* $\mathbf{T}(\Sigma_0 \oplus \Sigma_1)$ *containing both* $\equiv_0$ *and* $\equiv_1$.

Definition 44 *The sum of congruences* $\equiv_0 \oplus \equiv_1$ *is* conservative with respect to $\equiv_i$ *if* $(\equiv_0 \oplus \equiv_1)|\mathbf{T}(\Sigma_i)_{S_i} = \equiv_i|\mathbf{T}(\Sigma_i)_{S_i}$.

Lemma 45 *For* $t_0, u_0 \in \mathbf{T}(\Sigma_0)_{\mathsf{P}}$

(1) $t_0 \sim^{\mathcal{E}_0}_{\equiv_0} u_0 \Rightarrow t_0 \sim^{\mathcal{E}_0}_{\equiv_0 \oplus \equiv_1} u_0$,
(2) if $\equiv$ *is conservative with respect to* $\equiv_0$ *then* $t_0 \sim^{\mathcal{E}_0}_{\equiv_0 \oplus \equiv_1} u_0 \Rightarrow t_0 \sim^{\mathcal{E}_0}_{\equiv_0} u_0$.

Two theorems are now proved – one for refining extensions and one for abstracting extensions. Both of these results rely on the fact that the congruence respects sorts, and hence transitions with different sorts cannot be matched.

Theorem 46 *Let* $\mathcal{E}_0$ *and* $\mathcal{E}_1$ *be in extended* tyft/tyxt *format. Let* $\equiv_0 \oplus \equiv_1$ *be conservative with respect to* $\equiv_0$. *If* $\mathcal{E}_0$ *is pure and label-pure, and* $\mathcal{E}_0 \oplus\!\!\gg \mathcal{E}_1$ *is type-1, then* $\mathcal{E}_0 \oplus\!\!\gg \mathcal{E}_1$ *is a refining extension of* $\mathcal{E}_0$.

Proof. Let $t_0, u_0 \in \mathbf{T}(\Sigma_0)$ and let $\equiv = \equiv_0 \oplus \equiv_1$. It is necessary to show that $t_0 \sim^{\mathcal{E}}_{\equiv} u_0 \Rightarrow t_0 \sim^{\mathcal{E}_0}_{\equiv_0} u_0$. Assume $t_0 \not\sim^{\mathcal{E}_0}_{\equiv_0} u_0$. Hence (without loss of generality), from t_0 or some derivative of t_0, there is a transition that cannot be matched by a transition from u_0 or by any transitions from a possible corresponding derivative of u_0. Since $\equiv$ is conservative with respect to $\equiv_0$, no more transitions can be equated in $\mathcal{E}_0$ under $\equiv$ than under $\equiv_0$, hence only the new transitions generated under $\mathcal{E}$ need be considered. None of the new transitions can match this transition. Consider the last rule used in the proof of the transition. One of the following cases applies.

- The transition was generated by an extended *tyft/tyxt* rule from R_0. Then by Lemma 42, this transition is not a new transition.
- The transition was generated by an extended *tyxt* rule from R_1. Then because $\mathcal{E}_0 \oplus\!\!\gg \mathcal{E}_1$ is type-1, the transition label has a sort not in S_0 and hence cannot be a matching transition.
- The transition was generated by an extended *tyft* rule from R_1. Then be-

cause $\mathcal{E}_0 \oplus\!\!\triangleright \mathcal{E}_1$ is type-1 and since $t_0 \in \mathbf{T}(\Sigma_0)$, the transition label has a sort not in S_0 and hence cannot be a matching transition.

Since the transition cannot be matched by any new transitions, $t_0 \not\sim^{\mathcal{E}}_{\equiv} u_0$. □

Stronger conditions are required to show a similar result for abstracting extensions.

Theorem 47 *Let $\mathcal{E}_0$ and $\mathcal{E}_1$ be in extended* tyft/tyxt *format. Let $\equiv_0 \oplus \equiv_1$ be compatible with $\mathcal{E}_0 \oplus\!\!\triangleright \mathcal{E}_1$. If $\mathcal{E}_0$ is pure and label-pure, $\mathcal{E}_1$ is well-founded, and $\mathcal{E}_0 \oplus\!\!\triangleright \mathcal{E}_1$ is type-0, then $\mathcal{E}_0 \oplus\!\!\triangleright \mathcal{E}_1$ is an abstracting extension of $\mathcal{E}_0$.*

Proof. The details of this proof are included in Appendix A. It proceeds in a similar fashion to Theorem 24, but with greater intricacy since the relation that is to be shown a bisimulation is more complex. □

Corollary 48 *Let $\mathcal{E}_0$ and $\mathcal{E}_1$ be in extended* tyft/tyxt *format. Let $\equiv_0 \oplus \equiv_1$ be conservative with respect to $\equiv_0$ and compatible with $\mathcal{E}_0 \oplus\!\!\triangleright \mathcal{E}_1$. If $\mathcal{E}_0$ is pure, label-pure and well-founded, $\mathcal{E}_1$ is well-founded, and $\mathcal{E}_0 \oplus\!\!\triangleright \mathcal{E}_1$ is type-0, then $\mathcal{E}_0 \oplus\!\!\triangleright \mathcal{E}_1$ is a conservative extension of $\mathcal{E}_0$.*

Counter-examples to show that the conditions are necessary are given in Appendix B. For the refining extension results, counter-examples are given for all conditions except some of the pureness and label-pureness conditions, and for the abstracting extension results, counter-examples are given for all conditions except an aspect of compatibility.

3.3 A different approach to extensions up to bisimulation

This section will describe how it is possible to change the label-pureness requirement by giving additional requirements on Σ_1. In the counter-examples, a new constant with sort from S_0 was introduced in Σ_1 which allowed the required result to be lost. It is possible then to impose restrictions on Σ_1 disallowing such function symbols and hence remove the label-pureness requirement in the results under discussion.

Definition 49 Σ_1 *is* safe for S_0 *if no function symbol in $F_1 - F_0$ has a range with a sort from the set S_0.*

Note that if $(S_0 \cap S_1) \neq \emptyset$ and Σ_1 is safe for S_0, then Σ_1 may not be a sensible signature since there may be no closed terms in $\mathbf{T}(\Sigma_1)_{S_1}$ for the sorts in the intersection. This is why the notion of asymmetric sum is required. The following technical lemma proved by induction on term structure, shows that all closed terms in the sum with a sort from S_0 are in $\mathbf{T}(\Sigma_0)_{S_0}$.

Lemma 50 *If Σ_1 is safe for S_0, then for all $t \in \mathbf{T}(\Sigma_0 \oplus\!\!\triangleright \Sigma_1)_S - \mathbf{T}(\Sigma_0)_{S_0}$, the sort of t is in $S_1 - S_0$; namely for all $s \in S_0$, $t \in \mathbf{T}(\Sigma_0 \oplus\!\!\triangleright \Sigma_1)_s \Rightarrow t \in \mathbf{T}(\Sigma_0)_s$.*

The lemma and the refining, abstracting and conservative extension up to bisimulation results can all be rephrased in a similar way. It may appear that it is possible in the next lemma to remove the condition that no extended *tyxt* rule from R_1 has a label in the conclusion with a sort from S_0. This, however, is not correct since terms from $\mathbb{T}(\Sigma_1)_{S_1}$ must be considered and this includes variables. This could be fixed by insisting that $(S_0 \cap S_1) = \emptyset$, but this is too strong because then no sorts could be shared between the two eTSSs which would reduce the applicability of the results.

Lemma 51 *Let $\mathcal{E}_0$ and $\mathcal{E}_1$ be in extended* tyft/tyxt *format and let Σ_1 be safe for S_0. Let $\mathcal{E}_0$ be pure, and let $\mathcal{E}_0 \oplus\!\!\triangleright \mathcal{E}_1$ be type-1. Let $\mathcal{E} \vdash t_0 \xrightarrow{\alpha} t$ with $t_0 \in \mathbf{T}(\Sigma_0)_{\mathsf{P}}$. If the last rule used in the proof of $\mathcal{E} \vdash t_0 \xrightarrow{\alpha} t$ is an extended* tyft/tyxt *rule from R_0 then $\mathcal{E}_0 \vdash t_0 \xrightarrow{\alpha} t$.*

Proof. The proof is similar to Lemma 42. However, instead of the argument that no new terms are introduced because there are no label-free variables, Lemma 50 is used to argue that the only terms that can be substituted into the free label variables in rules from R_0 are those from $\mathbf{T}(\Sigma_0)$, since terms with sorts from S_0 are in $\mathbf{T}(\Sigma_0)_{S_0}$. □

Theorems 46 and 47 can also be rephrased. The proofs are almost identical to before, but Lemma 51 is used instead of Lemma 42.

Theorem 52 *Let $\mathcal{E}_0$ and $\mathcal{E}_1$ be in extended* tyft/tyxt *format and let Σ_1 be safe for S_0. Let $\equiv_0 \oplus \equiv_1$ be conservative with respect to $\equiv_0$. If $\mathcal{E}_0$ is pure, and $\mathcal{E}_0 \oplus\!\!\triangleright \mathcal{E}_1$ is type-1, then $\mathcal{E}_0 \oplus\!\!\triangleright \mathcal{E}_1$ is a refining extension.*

Theorem 53 *Let $\mathcal{E}_0$ and $\mathcal{E}_1$ be in extended* tyft/tyxt *format and let Σ_1 be safe for S_0. Let $\equiv_0 \oplus \equiv_1$ be compatible with $\mathcal{E}_0 \oplus\!\!\triangleright \mathcal{E}_1$. If $\mathcal{E}_0$ is pure and well-founded, $\mathcal{E}_1$ is well-founded and $\mathcal{E}_0 \oplus\!\!\triangleright \mathcal{E}_1$ is type-0, then $\mathcal{E}_0 \oplus\!\!\triangleright \mathcal{E}_1$ is an abstracting extension.*

Corollary 54 *Let $\mathcal{E}_0$ and $\mathcal{E}_1$ be in extended* tyft/tyxt *format and let Σ_1 be safe for S_0. Let $\equiv_0 \oplus \equiv_1$ be conservative with respect to $\equiv_1$ and compatible with $\mathcal{E}_0 \oplus\!\!\triangleright \mathcal{E}_1$. If $\mathcal{E}_0$ is pure and well-founded, $\mathcal{E}_1$ is well-founded, and $\mathcal{E}_0 \oplus\!\!\triangleright \mathcal{E}_1$ is type-0, then $\mathcal{E}_0 \oplus\!\!\triangleright \mathcal{E}_1$ is a conservative extension.*

As will be seen in the next section, safety plays an important rôle in comparing semantic equivalences. All the counter-examples in Appendix B for label-pureness are also counter-examples for safety, since they involve adding new function symbols with a range sort of the first eTSS. All the other counter-examples in Appendix B apply to the other conditions in the theorems since

$$\frac{}{a : p \xrightarrow{a:\mathrm{NIL}} p} \qquad \frac{p \xrightarrow{u} p'}{a : p \xrightarrow{a:u} p'} \qquad \frac{p \xrightarrow{u} p'}{p + q \xrightarrow{u} p'} \qquad \frac{p \xrightarrow{u} p'}{q + p \xrightarrow{u} p'}$$

$$\frac{p \xrightarrow{u} p'}{p \mid q \xrightarrow{u} p' \mid q} \qquad \frac{p \xrightarrow{u} p'}{q \mid p \xrightarrow{u} q \mid p'} \qquad \frac{p \xrightarrow{u} p' \quad q \xrightarrow{v} q'}{p \mid q \xrightarrow{u|v} p' \mid q'}$$

Fig. 4. Rules for pomset CCS

all adhere to the safety condition.

4 Comparing two semantic equivalences

In this section, a comparison of the multiprocessor equivalence of Krishnan [32] and the pomset bisimulation of Castellani [15] will be presented. An eTSS to represent pomset CCS is required. This will be done by summing $\mathcal{E}_{\mathrm{MP}}$ from Section 2.6.1 with a new eTSS. $\mathcal{E}_{\mathrm{MP}}$ together with the Σ_{MP}-algebra $\mathcal{A}_n$ which induces the congruence $\equiv_{\mathcal{A}_n}$ represent n multiprocessor CCS and n multiprocessor equivalence. A new eTSS $\mathcal{E}_{\mathrm{MPExt}}$ is introduced together with the congruence $\equiv_{\mathcal{A}_p}$ obtained from the Σ_{MPExt}-algebra $\mathcal{A}_{pom}$ and some axioms on label terms. Then $\mathcal{E}_{\mathrm{MP}} \oplus\!\!> \mathcal{E}_{\mathrm{MPExt}}$ together with $\equiv_{\mathcal{A}_n} \oplus \equiv_{\mathcal{A}_p}$ represent pomset CCS and pomset equivalence. As an intermediate step, a second congruence $\equiv_{\mathcal{A}_{pn}}$ obtained from the Σ_{MPExt}-algebra $\mathcal{A}_{pomn}$ and the same axioms, over $\mathcal{E}_{\mathrm{MPExt}}$ is introduced, and the congruence $\equiv_{\mathcal{A}_n} \oplus \equiv_{\mathcal{A}_{pn}}$ is considered over $\mathcal{E}_{\mathrm{MP}} \oplus\!\!> \mathcal{E}_{\mathrm{MPExt}}$. It will then be possible to apply the refining extension theorem.

4.1 Pomset CCS

First consider the definition of pomset CCS (restricted here to the prefix operator, non-deterministic choice, parallel composition and the nil process). Let A be a set of actions and consider the grammar $P ::= a : P \mid \mathrm{NIL} \mid P + P \mid P|P$ with $a \in A$. General actions are defined by $B ::= a : B \mid \mathrm{NIL} \mid B|B$ with $a \in A$. These actions are referred to as Act. There are two axioms over Act, $u|v = v|u$ and $u|(v|w) = (u|v)|w$ which define an equivalence $\equiv$. The rules are given in Figure 4. A *pomset bisimulation* $\mathcal{R}$ is a binary relation such that for any $(p, q) \in \mathcal{R}$ and $u \in Act$, the following holds

(1) $p \xrightarrow{u} p'$ implies there exist q', v such that $q \xrightarrow{v} q'$, $u \equiv v$ and $(p', q') \in \mathcal{R}$,
(2) $q \xrightarrow{u} q'$ implies there exist p', v such that $p \xrightarrow{v} p'$, $u \equiv v$ and $(p', q') \in \mathcal{R}$.

$$\frac{x \xrightarrow{z_{\mathsf{Act}}} y}{\mathsf{pref}(z_{\mathsf{A}}, x) \xrightarrow{\mathsf{concat}(z_{\mathsf{A}}, z_{\mathsf{Act}})} x} \qquad \frac{x \xrightarrow{z_{\mathsf{Act_{new}}}} y}{\mathsf{pref}(z_{\mathsf{A}}, x) \xrightarrow{\mathsf{concat}(z_{\mathsf{A}}, z_{\mathsf{Act_{new}}})} x}$$

$$\frac{x \xrightarrow{z_{\mathsf{Act_{new}}}} y}{\mathsf{plus}(x, x') \xrightarrow{z_{\mathsf{Act_{new}}}} y} \qquad \frac{x \xrightarrow{z_{\mathsf{Act_{new}}}} y}{\mathsf{plus}(x', x) \xrightarrow{z_{\mathsf{Act_{new}}}} y}$$

$$\frac{x \xrightarrow{z_{\mathsf{Act_{new}}}} y}{\mathsf{par}(x, x') \xrightarrow{z_{\mathsf{Act_{new}}}} \mathsf{par}(y, x')} \qquad \frac{x \xrightarrow{z_{\mathsf{Act_{new}}}} y}{\mathsf{par}(x', x) \xrightarrow{z_{\mathsf{Act_{new}}}} \mathsf{par}(x', y)}$$

$$\frac{x \xrightarrow{z_{\mathsf{Act}}} y \quad x' \xrightarrow{z'_{\mathsf{Act}}} y'}{\mathsf{par}(x, x') \xrightarrow{\mathsf{comb_{new}}(z_{\mathsf{Act}}, z'_{\mathsf{Act}})} \mathsf{par}(y, y')} \qquad \frac{x \xrightarrow{z_{\mathsf{Act_{new}}}} y \quad x' \xrightarrow{z'_{\mathsf{Act}}} y'}{\mathsf{par}(x, x') \xrightarrow{\mathsf{comb_{new}}(z_{\mathsf{Act_{new}}}, z'_{\mathsf{Act}})} \mathsf{par}(y, y')}$$

$$\frac{x \xrightarrow{z_{\mathsf{Act}}} y \quad x' \xrightarrow{z'_{\mathsf{Act_{new}}}} y'}{\mathsf{par}(x, x') \xrightarrow{\mathsf{comb_{new}}(z_{\mathsf{Act}}, z'_{\mathsf{Act_{new}}})} \mathsf{par}(y, y')} \qquad \frac{x \xrightarrow{z_{\mathsf{Act_{new}}}} y \quad x' \xrightarrow{z'_{\mathsf{Act_{new}}}} y'}{\mathsf{par}(x, x') \xrightarrow{\mathsf{comb_{new}}(z_{\mathsf{Act_{new}}}, z'_{\mathsf{Act_{new}}})} \mathsf{par}(y, y')}$$

Fig. 5. Rules for MPExt

4.2 Expressing pomset CCS as a sum of eTSSs

Consider the signature Σ_{MPExt} with $(\mathsf{A}, \mathsf{K}, \mathsf{Act}, \mathsf{Act_{new}}, \mathsf{P}; \{\mathsf{a}\}_{\mathsf{a}\in\mathsf{A}}, \{\mathsf{k}\}_{\mathsf{k}\in\mathsf{K}}, \mathsf{nil}, \bot_{\mathsf{A}}, \bot_{\mathsf{Act}}, \bot_{\mathsf{Act_{new}}}, \bot_{\mathsf{K}}; ., +, |, \mathsf{act}, \mathsf{comb}, \mathsf{comb_{new}}, \mathsf{concat})$ with the types of $\{\mathsf{a}\}_{\mathsf{a}\in\mathsf{A}}$, $\{\mathsf{k}\}_{\mathsf{k}\in\mathsf{K}}$, nil, $\bot_{\mathsf{A}}, \bot_{\mathsf{Act}}, \bot_{\mathsf{Act_{new}}}, \bot_{\mathsf{K}}, ., +, |, \mathsf{act}$ and comb as in Σ_{MP} (Example 11 and Section 2.6.1), and

$$\begin{array}{ll}
\mathsf{concat} : \mathsf{A}, \mathsf{Act} \to \mathsf{Act_{new}} & \mathsf{comb_{new}} : \mathsf{Act}, \mathsf{Act} \to \mathsf{Act_{new}} \\
\mathsf{concat} : \mathsf{A}, \mathsf{Act_{new}} \to \mathsf{Act_{new}} & \mathsf{comb_{new}} : \mathsf{Act}, \mathsf{Act_{new}} \to \mathsf{Act_{new}} \\
 & \mathsf{comb_{new}} : \mathsf{Act_{new}}, \mathsf{Act} \to \mathsf{Act_{new}} \\
\bot_{\mathsf{Act_{new}}} :\to \mathsf{Act_{new}} & \mathsf{comb_{new}} : \mathsf{Act_{new}}, \mathsf{Act_{new}} \to \mathsf{Act_{new}}.
\end{array}$$

A new sort $\mathsf{Act_{new}}$ and two new operators concat and $\mathsf{comb_{new}}$ have been introduced. Also notice the overloading of concat and $\mathsf{comb_{new}}$. The rules are given in Figure 5. Let these rules be denoted R_{MPExt}, and the eTSS $\mathcal{E}_{\mathrm{MPExt}} = (\Sigma_{\mathrm{MPExt}}, R_{\mathrm{MPExt}})$.

Next define a Σ_{MPExt}-algebra $\mathcal{A}_{pom}$ to represent the actual process algebra labels. Let $\mathcal{C}$ be defined as $c ::= a \mid a : c \mid c|c$ for $a \in A$ and $\mathcal{C}'$ be defined as

$$\mathsf{a}^{\mathcal{A}_{pom}} = a \quad \forall \mathsf{a} \in \mathsf{A} \qquad \mathsf{k}^{\mathcal{A}_{pom}} = num(\mathsf{k}) \quad \forall \mathsf{k} \in \mathsf{K}$$

$$\bot_{\mathsf{A}}^{\mathcal{A}_{pom}} = \bot_{\mathsf{A}} \quad \bot_{\mathsf{K}}^{\mathcal{A}_{pom}} = 0 \quad \bot_{\mathsf{Act}}^{\mathcal{A}_{pom}} = \bot_{\mathsf{Act}} \quad \bot_{\mathsf{Act_{new}}}^{\mathcal{A}_{pom}} = \bot_{\mathsf{Act_{new}}}$$

$$\mathsf{act}^{\mathcal{A}_{pom}}(\zeta_1, \zeta_2) = \begin{cases} \zeta_1 & \text{if } \zeta_1 \in A \text{ and } \zeta_2 \in \mathbb{N}^+ \\ \bot_{\mathsf{Act}} & \text{otherwise} \end{cases}$$

$$\mathsf{comb}^{\mathcal{A}_{pom}}(\zeta_1, \zeta_2) = \begin{cases} \zeta_1 | \zeta_2 & \text{if } \zeta_1, \zeta_2 \in \mathcal{C}' \\ \bot_{\mathsf{Act}} & \text{otherwise} \end{cases}$$

$$\mathsf{concat}^{\mathcal{A}_{pom}}(\zeta_1, \zeta_2) = \begin{cases} \zeta_1 : \zeta_2 & \text{if } \zeta_1 \in A,\ \zeta_2 \in \mathcal{C} \\ \bot_{\mathsf{Act_{new}}} & \text{otherwise} \end{cases}$$

$$\mathsf{comb_{new}}^{\mathcal{A}_{pom}}(\zeta_1, \zeta_2) = \begin{cases} \zeta_1 | \zeta_2 & \text{if } \zeta_1, \zeta_2 \in \mathcal{C} \\ \bot_{\mathsf{Act_{new}}} & \text{otherwise} \end{cases}$$

Fig. 6. Functions for $\mathcal{A}_{pom}$

$c' ::= a \mid c'|c'$ for $a \in A$. Then $A \subset \mathcal{C}' \subset \mathcal{C}$. Assume there is an congruence $\equiv_{\mathcal{C}}$ on $\mathcal{C}'$ and $\mathcal{C}$ generated by the axioms $c|c' = c'|c$ and $c|(c'|c'') = (c|c')|c''$. The carrier sets for the sorts A, K, Act and $\mathsf{Act_{new}}$ are $A \cup \{\bot_{\mathsf{A}}\}$, $\mathbb{N}$, $\mathcal{C}' \cup \{\bot_{\mathsf{Act}}\}$ and $\mathcal{C} \cup \{\bot_{\mathsf{Act_{new}}}\}$ respectively. The functions for $\mathcal{A}_{pom}$ are given in Figure 6.

Define $\equiv_{\mathcal{A}_p}$ as follows: if $\alpha, \beta \in \mathbf{T}(\Sigma_{\mathrm{MPExt}})_S$, then

$$\alpha \equiv_{\mathcal{A}_p} \beta \iff \begin{cases} i_{\mathcal{A}_{pom}}(\alpha) = i_{\mathcal{A}_{pom}}(\beta) \\ \text{or} \\ i_{\mathcal{A}_{pom}}(\alpha) \equiv_{\mathcal{C}} i_{\mathcal{A}_{pom}}(\beta). \end{cases}$$

Consider $\mathcal{E}_{\mathrm{MP}} \oplus\!\!> \mathcal{E}_{\mathrm{MPExt}}$ together with $\equiv_{\mathcal{A}_n} \oplus \equiv_{\mathcal{A}_p}$. The signature $\Sigma_{\mathrm{MP}} \oplus\!\!> \Sigma_{\mathrm{MPExt}}$ is the same as Σ_{MPExt} given above, and the rules $R_{\mathrm{MP}} \cup R_{\mathrm{MPExt}}$ are the rules given in Figures 1 and 5. It can be seen from the functions of $\mathcal{A}_{pom}$ in Figure 6 how semantically equivalent labels are equated by $\equiv_{\mathcal{A}_p}$ and how this relates to the original definition of the pomset process algebra.

4.3 A refining extension

The aim is to show that $\mathcal{E}_{\mathrm{MP}} \oplus\!\!> \mathcal{E}_{\mathrm{MPExt}}$ together with the congruence $\equiv_{\mathcal{A}_n} \oplus \equiv_{\mathcal{A}_p}$ provide a refining extension of $\mathcal{E}_{\mathrm{MP}}$ – this will show that pomset equivalence is a subset of n multiprocessor equivalence. However, the refining extension theorem cannot be directly applied, since $\equiv_{\mathcal{A}_n} \oplus \equiv_{\mathcal{A}_p}$ is not conservative with respect to $\equiv_{\mathcal{A}_n}$. Conservativity requires that for $\alpha, \beta \in \mathbf{T}(\Sigma_{\mathrm{MP}})_S$, $\alpha \equiv_{\mathcal{A}_n} \oplus \equiv_{\mathcal{A}_p} \beta \iff \alpha \equiv_{\mathcal{A}_p} \beta$. Then $\mathsf{act}(\mathsf{a}, \mathsf{k}_1) \equiv_{\mathcal{A}_n} \oplus \equiv_{\mathcal{A}_p} \mathsf{act}(\mathsf{a}, \mathsf{k}_2)$ but $\mathsf{act}(\mathsf{a}, \mathsf{k}_1)$

$\not\equiv_{\mathcal{A}_n} \mathsf{act}(\mathsf{a}, \mathsf{k}_2)$.

The solution is to work with a different Σ_{MPExt}-algebra over the labels which is conservative, and show that this algebra generates an congruence which will result in the same semantic equivalence as is obtained by using $\equiv_{\mathcal{A}_p}$. This algebra will have a different carrier set for Act since it is only on this sort that the summed congruence is not conservative. This carrier set contains pairs of actions from $\mathcal{C}'$ and sequences of natural numbers. Let N and M be sequences of positive natural numbers without repetition, and let $N \cap M$ indicate the intersection of sequences. Then let the partial function $(a, N) +^n (b, M)$ where $a, b \in \mathcal{C}'$, be defined as equal to $(a|b, NM)$ whenever $N \cap M = \emptyset$. Also define a function $acts : \mathcal{C}' \to \mathbb{N}$ as $acts(c|c') = acts(c) + acts(c')$ and $acts(a) = 1$.

The carrier sets for A, K and $\mathsf{Act}_{\mathsf{new}}$, and the functions for a, k, $\bot_{\mathsf{A}}$, $\bot_{\mathsf{K}}$, $\bot_{\mathsf{Act}}$ and $\bot_{\mathsf{Act}_{\mathsf{new}}}$ are the same as for $\mathcal{A}_{pom}$. The carrier set for Act is $\mathcal{D} \cup \{\bot_{\mathsf{Act}}\}$ where $\mathcal{D}$ is

$$\bigcup_{j \geqslant 1}^{n} \{(c', a_1 \ldots a_j) | c' \in \mathcal{C}', acts(c') = j, a_1, \ldots, a_j \in \{1, \ldots, j\}, \text{pairwise disjoint}\}.$$

The new functions are given in Figure 7. The congruence generated by the axioms $\equiv_{\mathcal{C}}$ can be extended to $\mathcal{D}$ in the obvious manner.

Define $\equiv_{\mathcal{A}_{pn}}$ as follows: if $\alpha, \beta \in \mathbf{T}(\Sigma_{\mathrm{MPExt}})_S$, then

$$\alpha \equiv_{\mathcal{A}_{pn}} \beta \iff \begin{cases} i_{\mathcal{A}_{pomn}}(\alpha) = i_{\mathcal{A}_{pomn}}(\beta) \\ \text{or} \\ i_{\mathcal{A}_{pomn}}(\alpha) \equiv_{\mathcal{C}} i_{\mathcal{A}_{pomn}}(\beta). \end{cases}$$

First, it will be shown that using the congruence $\equiv_{\mathcal{A}_{pn}}$ it is possible to obtain a refining extension to $\mathcal{E}_{\mathrm{MP}}$, then it will be shown that this defines an congruence $\sim_{\equiv_{\mathcal{A}_{pn}}}$ identical to $\sim_{\equiv_{\mathcal{A}_p}}$. Note that since $\mathcal{E}_{\mathrm{MP}}$ is not label-pure (it is the axiom which is not label-pure – see Figure 1), it is necessary to use Theorem 52. Clearly Σ_{MPExt} is safe for S_{MP} since no function in $F_{\mathrm{MPExt}} - F_{\mathrm{MP}}$ (the functions concat and $\mathsf{comb}_{\mathsf{new}}$) has a range with a sort from S_{MP}. $\mathcal{E}_{\mathrm{MP}}$ and $\mathcal{E}_{\mathrm{MPExt}}$ are both in extended *tyft/tyxt* format. $\mathcal{E}_{\mathrm{MP}}$ is pure, and $\mathcal{E}_{\mathrm{MP}} \oplus\!\!> \mathcal{E}_{\mathrm{MPExt}}$ is type-1 since there is no extended *tyft* rule in R_{MPExt} containing a function symbol from F_{MP} with a conclusion label with a sort from S_{MP}; in other words, all extended *tyft* rules with a function symbol from F_{MP} have a conclusion transition label with sort $\mathsf{act}_{\mathsf{new}}$.

It must also be shown that $\equiv_{\mathcal{A}_n} \oplus \equiv_{\mathcal{A}_{pn}}$ is conservative with respect to $\equiv_{\mathcal{A}_n}$ since this is a condition required by Theorem 52. Note that for all $s \in \{\mathsf{K}, \mathsf{A}\}$, $\alpha, \beta \in \mathbf{T}(\Sigma_{\mathrm{MPExt}})_s$, $\alpha \equiv_{\mathcal{A}_n} \oplus \equiv_{\mathcal{A}_{pn}} \beta \iff \alpha \equiv_{\mathcal{A}_n} \beta$. It is not necessary to consider terms with sort $\mathsf{Act}_{\mathsf{new}}$ since these only occur in $\mathbf{T}(\Sigma_{\mathrm{MPExt}})$. Hence it

$$\mathsf{act}^{\mathcal{A}_{pomn}}(\zeta_1,\zeta_2)=\begin{cases}(\zeta_1,\zeta_2) & \text{if } \zeta_1\in A \text{ and } \zeta_2\leqslant n\\ \bot_{\mathsf{Act}} & \text{otherwise}\end{cases}$$

$$\mathsf{comb}^{\mathcal{A}_{pomn}}(\zeta_1,\zeta_2)=\begin{cases}\zeta_1+^n\zeta_2 & \text{if } \zeta_1,\zeta_2\in\mathcal{D}\\ & \text{and } \zeta_1+^n\zeta_2 \text{ defined}\\ \bot_{\mathsf{Act}} & \text{otherwise}\end{cases}$$

$$\mathsf{concat}^{\mathcal{A}_{pomn}}(\zeta_1,\zeta_2)=\begin{cases}\zeta_1:\zeta_{2,1} & \text{if } \zeta_1\in A,\ \zeta_2\in\mathcal{D} \text{ and}\\ & \zeta_2=(\zeta_{2,1},\zeta_{2,2})\\ \zeta_1:\zeta_2 & \text{if } \zeta_1\in A,\ \zeta_2\in\mathcal{C}\\ \bot_{\mathsf{Act_{new}}} & \text{otherwise}\end{cases}$$

$$\mathsf{comb_{new}}^{\mathcal{A}_{pomn}}(\zeta_1,\zeta_2)=\begin{cases}\zeta_{1,1}|\zeta_{2,1} & \text{if } \zeta_1\in\mathcal{D},\\ & \zeta_2\in\mathcal{D} \text{ and}\\ & \zeta_1=(\zeta_{1,1},\zeta_{1,2})\\ & \zeta_2=(\zeta_{2,1},\zeta_{2,2})\\ \zeta_{1,1}|\zeta_2 & \text{if } \zeta_1\in\mathcal{D},\ \zeta_2\in\mathcal{C} \text{ and}\\ & \zeta_1=(\zeta_{1,1},\zeta_{1,2})\\ \zeta_1|\zeta_{2,1} & \text{if } \zeta_1\in\mathcal{C},\ \zeta_2\in\mathcal{D} \text{ and}\\ & \zeta_2=(\zeta_{2,1},\zeta_{2,2})\\ \zeta_1|\zeta_2 & \text{if } \zeta_1,\zeta_2\in\mathcal{C}\\ \bot_{\mathsf{Act_{new}}} & \text{otherwise}\end{cases}$$

Fig. 7. Functions for $\mathcal{A}_{pomn}$

is only necessary to consider terms with sort Act. The proof is omitted for reasons of space, but is straightforward (see [23] for a full proof).

Hence it can be concluded that for $t,u\in\mathbf{T}(\Sigma_{\mathrm{MP}})_{\mathsf{P}}$, $t\sim^{\mathcal{E}_{\mathrm{MP}}\oplus\mathcal{E}_{\mathrm{MPExt}}}_{\equiv_{\mathcal{A}_n}\oplus\equiv_{\mathcal{A}_{pn}}}u \implies t\sim^{\mathcal{E}_{\mathrm{MP}}}_{\equiv_{\mathcal{A}_n}}u$. In other words, on the process terms of $\mathbf{T}(\Sigma_{\mathrm{MPExt}})$, the restricted version of pomset equivalence is a subset of n multiprocessor equivalence. This is a proper subset since for each n there exist processes which are identified by n multiprocessor equivalence, but not by restricted pomset equivalence.

Counter-example 55 *For arbitrary n, with $\sum$ for $+$, and $\prod$ for $|$,*

$$\prod_{k=1}^{n}\mathsf{a}_k.\mathsf{nil}+\sum_{i=1}^{n}\sum_{\substack{j=1\\j\neq i}}^{n}\Big(\mathsf{a}_i.\mathsf{a}_j.\mathsf{nil}\mid\prod_{\substack{k=1\\k\neq i\\k\neq j}}^{n}\mathsf{a}_k.\mathsf{nil}\Big)\quad\sim^{\mathcal{E}_{\mathrm{MP}}}_{\equiv_{\mathcal{A}_{n-1}}}\quad\sum_{i=1}^{n}\sum_{\substack{j=1\\j\neq i}}^{n}\Big(\mathsf{a}_i.\mathsf{a}_j.\mathsf{nil}\mid\prod_{\substack{k=1\\k\neq i\\k\neq j}}^{n}\mathsf{a}_k.\mathsf{nil}\Big)$$

These processes are not equated by the restricted pomset equivalence since the

first process can perform a transition involving all possible actions, namely a transition whose label is mapped by $i_{\mathcal{A}_{pn}}$ to $a_1 \mid \ldots \mid a_n$, whereas the second process does not have a transition that consists of all n actions.

Note that the abstracting extension theorem (Theorem 53) cannot be applied since one condition is not met – the sum is not type-0, namely there are rules in $\mathcal{E}_{\mathrm{MPExt}}$ with functions from F_{MP} in the sources of the conclusions.

Note also that $\mathbf{T}(\Sigma_{\mathrm{MP}})_{\mathsf{P}} = \mathbf{T}(\Sigma_{\mathrm{MP}} \oplus\!\!> \Sigma_{\mathrm{MPExt}})_{\mathsf{P}}$, and hence no additional process terms have been introduced by the extension, only additional transitions.

Finally, it is necessary to show that $\sim^{\mathcal{E}_{\mathrm{MP}} \oplus\!\!> \mathcal{E}_{\mathrm{MPExt}}}_{\equiv_{\mathcal{A}_n} \oplus \equiv_{\mathcal{A}_{pn}}}$ and $\sim^{\mathcal{E}_{\mathrm{MP}} \oplus\!\!> \mathcal{E}_{\mathrm{MPExt}}}_{\equiv_{\mathcal{A}_n} \oplus \equiv_{\mathcal{A}_p}}$ are the same. This proof is straightforward, but tedious and can be found in [23]. Hence, it can be concluded that for $t, u \in \mathbf{T}(\Sigma_{\mathrm{MP}})_{\mathsf{P}}$ $t \sim^{\mathcal{E}_{\mathrm{MP}} \oplus\!\!> \mathcal{E}_{\mathrm{MPExt}}}_{\equiv_{\mathcal{A}_n} \oplus \equiv_{\mathcal{A}_p}} u \implies t \sim^{\mathcal{E}_{\mathrm{MP}}}_{\equiv_{\mathcal{A}_n}} u$. In other words, on the process terms of $\mathbf{T}(\Sigma_{\mathrm{MP}})$, pomset equivalence is a proper subset of n multiprocessor equivalence.

5 Related work

5.1 Comparison with tyft/tyxt *format*

As mentioned earlier, this work does not concern itself with negative premises or predicates, so clearly it is not comparable with formats that involve negative premises and/or predicates. The *tyft/tyxt* format involves a single-sorted algebra for the operators and an infinite set of labels which are used in the rules in a schematic manner and are atomic. *Tyft/tyxt* rules have the following forms:

$$\frac{\{t_i \xrightarrow{a_i} y_i \mid i \in I\}}{f(x_1, \ldots, x_n) \xrightarrow{a} t} \qquad \frac{\{t_i \xrightarrow{a_i} y_i \mid i \in I\}}{x \xrightarrow{a} t}$$

where all the variables are distinct, the t_i's and t are open terms from the term algebra associated with the signature, and the a_i's and a are labels. The labels are understood to be any labels, as long as repeated labels are preserved. Proofs are constructed by finding substitutions for the variables. The definition implicitly permits operators to be created from the elements of the set of labels, such as prefix operators.

To translate from *tyft/tyxt* format to the extended *tyft/tyxt* format, the first step is to start with a many-sorted signature consisting of the process sort P and a label sort A, as well as operators which are not created from labels. The next step is to make all labels, constants of sort A and then to modify all operators created from elements of the label set to take an argument of the label sort.

Finally, each rule needs to be modified. This is where schematic treatment of actions has some advantages, for example in matching labels. First, consider rules where no matching occurs in contravention of the rules for label variables in the extended *tyft/tyxt* format. Here, it is simple to replace each different label with a different variable of the sort A, so for example an axiom such as $\mathsf{a}.x \xrightarrow{\mathsf{a}} x$ will become $z.x \xrightarrow{z} x$. In the case of matching of actions in a way which contravenes the conditions of label variables, there are two approaches as discussed in Section 2.6.

Hence, it is possible to translate a transition system specification in *tyft/tyxt* format into one in extended *tyft/tyxt* format.

The extended *tyft/tyxt* format has some advantages over the *tyft/tyxt* format. First, it can be used to compare different semantic equivalences as shown in Sections 3 and 4. For the *tyft/tyxt* format, it is possible to show that two semantic equivalences are the same when summing transition system specifications by showing that the sum is a conservative extension; however, there are no results for showing other kinds of relationships between semantic equivalences.

Second, the extended *tyft/tyxt* format can be more convenient to express some process algebras, as the following example shows. In CCS with locations [14], the definition of bisimulation can be parameterised by a relation over the set of location strings, hence the strings that are used in the transitions need only match up to the relation. Assume that the relation is the identity relation together with pairs (l_1, l_2) and (l_2, l_1) for some l_1, l_2 with $l_1 \neq l_2$. Also assume that a rule is introduced for the parallel operator involving communication where communication can only take place if the actions occur at the same locations, which is not an unreasonable restriction. This rule can be expressed as

$$\frac{x \underset{u}{\xrightarrow{a}} y \quad x' \underset{u}{\xrightarrow{\overline{a}}} y'}{x \mid x' \xrightarrow{\tau} y \mid y'}$$

This may appear to be a *tyft* rule if the locations are treated in a schematic manner. However, this rule results in an operator that does not preserve congruence. To see this, consider the following rules

$$\frac{}{a.x \underset{l}{\xrightarrow{a}} l :: x} \quad \forall l \in Loc \qquad\qquad \frac{x \underset{u}{\xrightarrow{a}} y}{l :: x \underset{lu}{\xrightarrow{a}} l :: y}$$

and the new parallel communication rule given above. Then $l_1 :: a.nil \approx_l l_2 :: a.nil$ but $l_1 :: a.nil \mid l_1 :: \overline{a}.nil \not\approx_l l_2 :: a.nil \mid l_1 :: \overline{a}.nil$, since the first term can perform a τ action which the second cannot. This could be expressed using the *tyft/tyxt* format, but this would require a different collection of rules for each possible relation over the label set.

Third, relations over a label set are also introduced whenever the labels become more complex and it becomes necessary to equate labels which have been constructed in different ways, but which are to be viewed as semantically identical. It is clear that care must be taken in these cases, and using a syntactic approach such as the extended *tyft/tyxt* format as opposed to the schematic approach contributes to dealing explicitly with the possible complexities.

The extended *tyft/tyxt* format has the advantage that the theory deals explicitly with equating labels in the definition of bisimulation. This requirement for the equation of labels can arise either as a result of definition of the bisimulation for the original process algebra, or through the syntactic treatment of labels. For the *tyft/tyxt* format, the definition of bisimulation is based on an atomic set of labels and requires that matching transitions have identical labels.

5.2 Other formats

Recently, a number of formats have been proposed that allow terms to appear in transition labels [20,9,22,37,19]. Each of these will be discussed and briefly compared to the extended *tyft/tyxt* format with the comparison focussing on the main aspects of the extended *tyft/tyxt* format, namely the syntactic requirements of the format, congruence results, and comparison of semantic equivalences. Detailed comparisons are beyond the scope of this paper.

Ferrari and Montanari [20] propose a format for parameterised structured operational semantics called Algebraic De Simone Format (AdS) which is an extension of De Simone format. The focus of this research is showing congruence, as well as the finite axiomatisation of parameterised bisimulation. Two single-sorted signatures are used, one for processes and one for labels, and a technique is used similar to the one used for extended *tyft/tyxt* format for the interpretation of label terms. A notion of parameterised bisimulation that permits matching on terms that are identified by an algebra as well as ignoring uninteresting or incorrect transitions is proposed – this is similar to the approach taken in this paper.

The main differences between AdS format and extended *tyft/tyxt* format are that label terms are disjoint from process terms, and the number of label variables which can appear in the term on the transition of the conclusion is equal to the number of premises, which is a feature of the De Simone format. Additionally, the binding occurrence of label variables occur in the label of the transition of the conclusions, as opposed to the extended *tyft/tyxt* format where binding occurrences of label variables occur in the source of the conclusion and in the labels of the hypotheses. A second format is introduced called

AdSA (AdS-format with abstractions) where the label signature is modified to be two-sorted, with the second sort having the same operators as the process signature. This permits predicates over the structure of processes in the rules. The evaluation of predicates are determined by the interpretation of label terms. These formats can be used to express process algebras that record dependencies between actions with the use of tags.

It is possible to compare two semantic equivalences over the same transition system specification using the AdS format by considering two different algebras for the label terms, each of which gives rise to a semantic equivalence, together with a homomorphism between these two algebras. The authors briefly propose that conservative extensions can be handled by using general algebraic techniques, which will give a mapping from one transition system specification system to another where a proof of a transition is mapped to a proof of its translation. It is unclear whether this approach could be extended to refining and abstracting extension results for the comparison of bisimulation over process algebras with different syntax, as achieved here for the extended *tyft/tyxt* format.

Bernstein [9] has introduced the *promoted tyft/tyxt* format which uses a single-sorted signature. The focus of this research is a format for defining higher-order languages, such as the π-calculus [36], and a congruence result for this format. An important aspect of this format is that syntactic substitution and free variable tests can be defined using the rules in the *promoted tyft/tyxt* as opposed to being a part of the meta-theory. This format is not used to compare equivalences defined for different process algebras.

In the *promoted tyft/tyxt* format, terms may appear on transitions, removing the distinction between process and label. The conditions for variables are very different to those of the extended *tyft/tyxt* format. In the conclusion, the same variable may not appear both in the source and label. Neither may the same variable appear in the label of the conclusion and the target of a premise. Furthermore, there must be at most one function symbol in the label of the conclusion (or a single variable), and at least one function symbol in the labels of premises. Additionally the binding occurrences of the variables appear either in the source of the conclusion or the label of the conclusion. This means that it is possible to show a suitable transition exists whenever two terms are equivalent. In the congruence result, this means that for a given transition, it is possible to find a matching transition for any term that is equivalent to the label term on the given transition. This differs from extended *tyft/tyxt* where a more general condition is used in the congruence result, namely that for a given transition it is possible to find a matching transition with a label term equivalent to the label term on the given transition. Since the form of the *promoted tyft/tyxt* format allows matching transitions to be found, it may be possible to obtain an abstracting extension result. However, the technique

used in this paper to obtain a refining extension uses the fact that there are multiple sorts, and this cannot be applied to the *promoted tyft/tyxt* format since it is based on a single sort.

When expressing rules in *promoted tyft/tyxt* format, a prefix axiom of the form $a.x \xrightarrow{a} x$, requires that a must be a constant term, and hence this rule is a schema for a collection of prefix axioms [9]. Similarly, the rules for action prefix and location prefix from CCS with locations (see Section 5.1) need to be expressed with constants – variables cannot be used because they could be instantiated by terms representing processes as well as terms representing actions and locations. Hence, one collection of constants is required to represent actions, together with a collection to represent locations and a collection to represent pairs of actions and strings of locations. Hence it appears for process algebras that record dependencies between actions by tags, such as CCS with locations [14], it is not possible to use the presence of terms and variables in labels provided by the *promoted tyft/tyxt* format to express these languages, whereas this is straightforward in the extended *tyft/tyxt* format. This is understandable since these process algebras were not the motivation for the *promoted tyft/tyxt* format.

Fokkink and Verhoef [22] introduce a very general format to consider the notion of conservative extension. The signature is many-sorted and terms can appear on transitions. To deal with variable binding, formal variables, substitution harnesses and formal rules are introduced. These are mapped to actual rules by the use of substitutions that map from formal variables to open actual terms. The format has no restrictions on where variables can appear in a rule since they are not concerned with congruence results. Additionally, they introduce a more liberal form of the notion 'pure and well-founded', namely source dependency. The focus is on conservative extensions without considering semantic equivalences, and hence there are no congruence results. It may be possible to extend this approach to obtain results for conservative extensions up to bisimulation as well as for refining and abstracting extensions, but as these involve semantic equivalences, it would require further restrictions on variables on rules to prove these results.

Mosses [37] has introduced the notion of modular structural operational semantics (MSOS). The underlying labelled transition system has labels that are arrows from a specific type of category, and labels on adjacent transitions are required to be composable. Label transformers are used to extend the labels of an MSOS without changing the semantics of the existing language constructs, after which new rules for new constructs can be defined that operate on the new components of the labels. An example is given of extending the functional constructs of ML with imperative features and concurrent processes. MSOS focusses on extending languages with constructs that introduce information that is independent of that already present in the semantics, and

hence is not applicable to process algebras where new label information is strongly related to the existing label information. For example, consider CCS with locations [14], where the locations are strongly related to the actions – they describe where the actions have occurred. Additionally, MSOS does not deal with semantic equivalences and hence there are no congruence results.

Degano and Priami [19] consider proved transition systems where labels capture the actions that occurred as well as the proof that was used to obtain the transition. This research does not define a format. A parametric bisimulation is defined where bisimulation is parameterised by the function that extracts the appropriate information from the proof of the deduction of the transition. Different semantics and semantic equivalences can be compared via the functions used to parameterise bisimulation. Proved transition systems can express different semantics covering causality, mobility, location, time and stochastic aspects with little modification to the operators of a process algebra. This approach differs from the approach taken in this paper, in that the focus is on constructing a tool to express systems based on a unique concrete model, whereas the focus in this paper is a new format for the comparison of existing process algebras via the notion of extension.

In summary, there are a number of different approaches to introducing more complex label terms into formats. These approaches differ from the extended *tyft/tyxt* format both in their general aims and in their details. Except for the AdS format and proved transition systems, none of the other approaches have been used to compare semantic equivalences, and the focus of these two approaches has been on comparison via parameterised bisimulation. Additionally, although some of the formats introduce more complex label terms, they are less useful for expressing process algebras that use tags for recording the dependencies between actions.

Two areas for further work relate to the the binding of variables and obtaining extension results for other formats. The binding of label variables in both the AdS and *promoted tyft/tyxt* format are very different from that in the extended *tyft/tyxt* format where binding occurrences occur in the source of the conclusion and in the labels of the hypotheses, and a fuller understanding of the implications of these differences could give further insight into the proof techniques that can be used for formats. Another question of interest is whether abstracting and refining extension results could be proved for these formats – some comments on this are given above.

Baeten and Verhoef [8] have considered a congruence theorem for structured operational semantics with predicates. They work with the notion of a term deduction system which generalises transition system specifications, where instead of relations over (pairs of) states and actions, they work with relation symbols and predicate symbols. It appears that a similar approach could be taken with extended transition system specifications. A brief description follows: let $\mathbb{D} = \{D_1, D_2, \ldots\}$ be a set of predicate symbols disjoint from any collections of variables and function symbols. Extend the notion of extended *tyft/tyxt* rules so that predicate formulae can appear in the premises of a rule in addition to the transitions, in the form $\{D_k p_k \mid k \in K\}$ with K an index set, $D_k \in \mathbb{D}$ and $p_k \in \mathbb{T}(\Sigma)_{\mathsf{P}}$ for $k \in K$ such that the same rules apply to the p_k's as to the p_i's in extended *tyft/tyxt* rules. Additionally, allow two new types of rules: one where the conclusion can have the form Dx for $D \in \mathbb{D}$ and x satisfying the same conditions as in extended *tyxt* rules, and the other where the conclusion can have the form $Df(\eta_1, \ldots, \eta_m, x_1, \ldots, x_n)$ with $D \in \mathbb{D}$ and $\eta_1, \ldots, \eta_m$ and $x_1, \ldots, x_n$ satisfying the same conditions as in extended *tyft* rules.

It appears that it is then possible to apply the same technique as used by Baeten and Verhoef to obtain a congruence theorem. Their approach requires that a term deduction system with predicates is transformed into one with no predicates where the same terms are equated by bisimulation, after which the standard congruence theorem can be used. The transformation involves introducing new constants and relation symbols to replace the predicate symbols and modifying the rules appropriately. This coding was first proposed by Groote and Vaandrager [28].

A similar approach can be taken with the extended *tyft/tyxt* format, by introducing new sorts and constants with these sorts, and then modifying the rules by replacing occurrences of predicates with transitions that have the new constants as labels and targets. Once this transformation has been defined, it would be necessary to prove that the same terms (and no additional terms) are equated by bisimulation, but it appears that this would follow similar lines as the proof by Baeten and Verhoef, although it may have some more complex details. After this has been shown, Theorem 24 can be applied to show congruence.

In the case of the extension theorems (Theorems 46 and 47), it is an issue for further work as to whether the transformation given above can be used to show that these theorems apply to rules with predicates.

Fokkink and van Glabbeek [21] have shown that any transition system specification in *tyft/tyxt* format can be expressed as a transition system specification in tree format. A transition system specification is in tree format when it consists of pure well-founded *xyft* rules (pure, well-founded rules with variables as the source of premises). Hence the well-founded requirement can be dropped for congruence for the *tyft/tyxt* format. The technique they use relies on a different notion of proof and makes use of new results from unification theory. It is not clear whether these results extend to the extended *tyft/tyxt* format. However, it appears that an eTSS in extended *xyft* format (where each premise source must be a variable) is transition equivalent to an eTSS in tree format, so the focus of further work in this direction would be on showing whether an eTSS in extended *tyft/tyxt* format is transition equivalent to one in extended *xyft* format.

Note that well-foundedness is still required for extension results, since transitions obtained from the sum of two transition system specifications in *tyft/tyxt* format may differ from those obtained from the sum of equivalent transition system specifications in tree format. This observation applies also to eTSSs in extended *tyft/tyxt* format. This is because the transformation of a transition system specification to tree format is based on replacing variables with closed terms that are formed by the signature associated with that transition system specification, and this ensures that the same transitions are generated as by the original rules. When summing two transition system specifications, additional closed terms may be formed by the sum of the signatures, and hence additional transitions could be obtained.

6 Conclusions and further work

There are a number of issues for further work, some of which lead automatically from the fact that this is a new format, for example, negative premises, weak bisimulation and axiomatisation. Some issues as yet not resolved include well-foundedness, conditions on variables in sources and labels of premises for congruence, and the conditions of pureness and label-pureness in the refining extension theorem.

Additional areas of further research relate to allowing many-sorted processes and a more applicable form of the abstracting extension result. In general, transition equivalence is not preserved by sum of eTSSs, but it may be the case that reasonable conditions can be found under which transition equivalence is preserved, and this may lead to simpler proofs. Other areas for further work

are considering the more restricted forms of rules that occur in process algebras that use dependencies, and investigating the expressive power of using label terms to represent processes.

This paper has presented a novel manner in which to compare process algebra semantic equivalences by using a format. The extended *tyft/tyxt* format developed here takes a many-sorted syntactic approach as opposed to a schematic approach to transition labels and has desirable properties such as congruence. New notions of extensions have been defined, and conditions have been given under which the two types of extensions can be found. Additionally, a different approach to these theorems has been shown. Finally, it has been shown that these results are applicable to a number of process algebras.

Acknowledgements

Thanks are due to Julian Bradfield, Scott Hazelhurst, Mpho Matlala and András Salamon and anonymous referees for their comments on this paper.

This research was supported in part by a Patrick and Margaret Flanagan Scholarship.

A Proof of Theorem 47

Proof. Let $t_0, u_0 \in \mathbf{T}(\Sigma_0)$ and let $\equiv = \equiv_0 \oplus \equiv_1$. The aim is to show that $t_0 \sim^{\mathcal{E}_0}_{\equiv_0} u_0 \Rightarrow t_0 \sim^{\mathcal{E}}_{\equiv} u_0$. Let $\mathcal{R} \subseteq \mathbf{T}(\Sigma)_{\mathsf{P}} \times \mathbf{T}(\Sigma)_{\mathsf{P}}$ be the least relation satisfying

(1) $\sim^{\mathcal{E}}_{\equiv} \subseteq \mathcal{R}$,
(2) $\sim^{\mathcal{E}_0}_{\equiv_0} \subseteq \mathcal{R}$ (note that $\sim^{\mathcal{E}_0}_{\equiv_0} \subseteq \mathbf{T}(\Sigma_0)_{\mathsf{P}} \times \mathbf{T}(\Sigma_0)_{\mathsf{P}}$),
(3) for all $f \in F$ such that $f : s_1 \dots s_m \mathsf{P} \dots \mathsf{P} \to \mathsf{P}$, for all terms $\mu_k, \nu_k \in \mathbf{T}(\Sigma)_S$ and for all terms $u_j, v_j \in \mathbf{T}(\Sigma)_{\mathsf{P}}$
$\mu_k \equiv \nu_k \ (1 \leqslant k \leqslant m)$ and $u_j \,\mathcal{R}\, v_j \ (1 \leqslant j \leqslant n) \Rightarrow$
$f(\mu_1, \dots, \mu_m, u_1, \dots, u_n) \,\mathcal{R}\, f(\nu_1, \dots, \nu_m, v_1, \dots v_n)$.

It is enough to show $\mathcal{R} \subseteq \sim^{\mathcal{E}}_{\equiv}$, hence it is necessary to show that $\mathcal{R}$ is a bisimulation with respect to $\equiv$ under $\mathcal{E}$. Assume $u \,\mathcal{R}\, v$. There are three cases:

(1) This is simple since $u \sim^{\mathcal{E}}_{\equiv} v$.
(2) It must be shown that whenever $\mathcal{E} \vdash u \xrightarrow{\alpha} u'$ and $u \sim^{\mathcal{E}_0}_{\equiv_0} v$ then there is a $v' \in \mathbf{T}(\Sigma)_{\mathsf{P}}$ such that $\mathcal{E} \vdash v \xrightarrow{\alpha'} v'$ with $\alpha' \equiv \alpha$ and $u' \,\mathcal{R}\, v'$. $(*)$
(3) It must be shown that whenever $\mathcal{E} \vdash f(\mu_1, \dots, \mu_m, u_1, \dots, u_n) \xrightarrow{\alpha} u'$, $\mu_k \equiv \nu_k$ for all k and $u_j \,\mathcal{R}\, v_j$ for all j then there is a $v' \in \mathbf{T}(\Sigma)_{\mathsf{P}}$ such that

$$\mathcal{E} \vdash f(\nu_1, \ldots, \nu_m, v_1, \ldots, v_n)) \xrightarrow{\alpha'} v' \text{ with } \alpha' \equiv \alpha \text{ and } u' \mathcal{R} v'. \qquad (**)$$

These last two cases will be considered together and the proof will proceed by induction on depth of proof.

For $(**)$, $u \sim^{\mathcal{E}_0}_{\equiv_0} v$ with $u, v \in \mathbf{T}(\Sigma_0)$. The transitions from u are either the result of extended *tyft/tyxt* rules from R_0, or extended *tyxt* rules from R_1 (since $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ is type-0).

For extended *tyft/tyxt* rules from R_0, Lemma 42 applies. Therefore, for any transition $u \xrightarrow{\alpha} u'$ then $\alpha \in \mathbf{T}(\Sigma_0)_{S_0}$ and $u' \in \mathbf{T}(\Sigma_0)_{\mathsf{P}}$. Moreover, for any transition of the form $u \xrightarrow{\alpha} u'$ with $\alpha \in \mathbf{T}(\Sigma_0)_{S_0}$ and $u' \in \mathbf{T}(\Sigma_0)_{\mathsf{P}}$, since $u \sim^{\mathcal{E}_0}_{\equiv_0} v$ there exist $\alpha' \in \mathbf{T}(\Sigma_0)_{S_0}$ and $v' \in \mathbf{T}(\Sigma_0)_{\mathsf{P}}$ such that $v \xrightarrow{\alpha'} v'$ and $u' \sim^{\mathcal{E}_0}_{\equiv_0} v'$. Hence $u' \mathcal{R} v'$, as required. For the case of a transition from u being generated by a extended *tyxt* rule from R_1, induction on the depth of proof of the transition is necessary.

The following fact proved by induction on the structure of the term will be used throughout this proof.

Fact Let $p \in \mathbb{T}(\Sigma)_{\mathsf{P}}$ and let $\rho, \rho' : V \to \mathbf{T}(\Sigma)$ be substitutions such that for all x in $\mathrm{Var}_{\mathsf{P}}(t)$, $\rho(x) \mathcal{R} \rho'(x)$ and for all z in $\mathrm{Var}_S(p)$, $\rho(z) \equiv \rho'(z)$. Then $\rho(p) \mathcal{R} \rho'(p)$.

From Lemma 10, there is a closed proof T of $u \xrightarrow{\alpha} u'$. Let r be the last rule used in proof T, in combination with a substitution σ. Assume first that the proof consists of an axiom, then there are the following two cases.

- $u \sim^{\mathcal{E}_0}_{\equiv_0} v$, and $r \in R_1$ has the form $x \xrightarrow{\lambda} p$ with $\sigma(x) = u$, $\sigma(\lambda) = \alpha$ and $\sigma(p) = u'$. Let $\sigma'(x) = v$, $\sigma'(z) = \sigma(z)$ for all $z \in \mathrm{Var}_S(r)$, and $\sigma'(y) = \sigma(y)$ for all $y \in \mathrm{Var}_{\mathsf{P}}(p) - \{x\}$. Then $v \xrightarrow{\sigma'(\lambda)} \sigma'(p)$ with $\sigma'(\lambda) \equiv \alpha$, and by the fact above $\sigma(p) \mathcal{R} \sigma'(p)$ as required.
- Then $f(\mu_1, \ldots, \mu_m, u_1, \ldots, u_n)$ and $f(\nu_1, \ldots, \nu_m, v_1, \ldots v_n)$ with $\mu_i \equiv \nu_i$ for all k and $u_i \mathcal{R} v_i$ for all i.
  - $r \in R_0 \cup R_1$ has the form $x \xrightarrow{\lambda} p$. This is done in a similar fashion to the case above.
  - $r \in R_0 \cup R_1$ has the form $f(\eta_1, \ldots, \eta_m, x_1, \ldots, x_n) \xrightarrow{\lambda} p$ with $\sigma(\eta_k) = \mu_k$ for all k, $\sigma(x_i) = u_i$ for all i, $\sigma(p) = u'$ and $\sigma(\lambda) = \alpha$. Let $\sigma'(x_i) = v_i$ for all i, $\sigma'(z) = \sigma(z)$ for all $z \in \mathrm{Var}_S(r) - \bigcup_{1 \leqslant k \leqslant m} \eta_k$, and $\sigma'(y) = \sigma(y)$ for all $y \in \mathrm{Var}_{\mathsf{P}}(r) - \bigcup_{1 \leqslant i \leqslant n} x_i$. More care is required for η_k. Since for a given k, $\sigma(\eta_k) = \mu_k \equiv \nu_k$ and $\equiv$ is compatible with $R_0 \cup R_1$, there is a substitution σ'' such that $\sigma''(\eta_k) = \nu_k$ and for all $z \in \mathrm{Var}_S(\eta_k)$, $\sigma(z) \equiv \sigma''(z)$. Hence define $\sigma'(z) = \sigma''(z)$. This can be done for all $1 \leqslant k \leqslant m$ since there are no variables shared between the terms. Then $f(\nu_1, \ldots, \nu_m, v_1, \ldots, v_n) \xrightarrow{\sigma'(\lambda)} \sigma'(p)$ with $\sigma'(\lambda) \equiv \alpha$ and by the fact above

$\sigma(p)\,\mathcal{R}\,\sigma'(p)$.

Next assume that the two statements $(*)$ and $(**)$ are true for proofs with n or fewer steps. Only the details of the proof for the case of an extended *tyft* rule from R_0 for $(**)$ are given. The cases of an extended *tyft* rule from R_1 for $(**)$, an extended *tyxt* rule from $R_1 \cup R_0$ for $(**)$ and of an extended *tyxt* rule from R_1 for $(*)$ are proved in a similar manner. Note that when dealing with rules from $R_0 \cup R_1$, they are treated as non-pure, non-label-pure rules, hence the fact that rules from R_0 are pure and label-pure is not used in this part of the proof. The variables in r will be classified as in Lemma 23. Note that Y_f, Z_f and Z'' are not empty. The proof proceeds by induction on the depth of the variables.

A substitution σ' that satisfies the following properties on V_{P} and V_S will be defined

(1) $\sigma'(x_i) = v_i$ for $1 \leqslant i \leqslant n$
(2) $\sigma(y)\,\mathcal{R}\,\sigma'(y)$ for $y \in X \cup Y$
(3) $\mathcal{E} \vdash \sigma'(p_i \xrightarrow{\lambda_i} y_i)$ for $i \in I$
(4) $\sigma'(z) \equiv \sigma(z)$ for $z \in Z \cup Z' \cup Z'' \cup Z_f$.

Substitution σ' will be constructed in stepwise fashion. To begin, let $\sigma'(x_i) = v_i$ for all i, $\sigma'(y) = \sigma(y)$ for $y \in V_{\mathsf{P}} - (X \cup \bigcup_{d \geqslant 0} Y_d)$, $\sigma'(z) = \sigma(z)$ for $z \in V_S - (Z \cup \bigcup_{d \geqslant 0} Z'_d)$. Hence (1), (2) and (4) hold for all these variables. When σ' is defined on $y \in X \cup Y_f \cup Y_0 \cup \ldots \cup Y_d$ and $z \in Z \cup Z'' \cup Z_f \cup Z'_0 \cup \ldots \cup Z'_d$ $(d \geqslant 0)$, then $\beta(d)$, $\gamma(d)$ and $\delta(d)$ will hold.

- $\beta(d) : \sigma(y)\,\mathcal{R}\,\sigma'(y)$ for $y_i \in X \cup Y_f \cup Y_0 \cup \ldots \cup Y_d$
- $\gamma(d) : \mathcal{E} \vdash \sigma'(p_i \xrightarrow{\lambda_i} y_i)$ for $y_i \in Y_0 \cup \ldots \cup Y_d$
- $\delta(d) : \sigma'(z) \equiv \sigma(z)$ for $z \in Z \cup Z'' \cup Z_f \cup Z'_0 \cup \ldots \cup Z'_d$.

It is necessary to show that $\beta(0)$, $\delta(0)$ and $\gamma(0)$ hold. First note that for any $x_i \in X$, since $\sigma(x_i) = u_i$ and $\sigma'(x_i) = v_i$, and $u_i\,\mathcal{R}\,v_i$, $\sigma(x_i)\,\mathcal{R}\,\sigma'(x_i)$. Also $\sigma(y)\,\mathcal{R}\,\sigma'(y)$ for $y \in Y_f$ since $\sigma(y) = \sigma'(y)$.

Then consider Z. Since for a given k, $\sigma(\eta_k) = \mu_k \equiv \nu_k$, by compatibility a suitable substitution can be found so that for all $z \in \mathrm{Var}_S(\eta_k)$, $\sigma(z) \equiv \sigma'(z)$. The conditions $\beta(0)$ and $\delta(0)$ hold on Y_f and Z_f respectively, by definition of σ'. Next consider $y_i \in Y_0$ and the transition $p_i \xrightarrow{\lambda_i} y_i$. This is a simpler version of the inductive step and is omitted.

Now let $d > 0$, and suppose that σ' has been defined for all variables in $X \cup Y_f \cup Y_0 \cup \ldots Y_{d-1}$ and $Z \cup Z'' \cup Z_f \cup Z'_0 \cup \ldots \cup Z'_{d-1}$ such that $\beta(d-1)$, $\gamma(d-1)$ and $\delta(d-1)$ hold. Next define σ' on Y_d and Z'_d such that $\beta(d)$, $\gamma(d)$ and $\delta(d)$ hold.

Consider $y_i \in Y_d$ and the transition $p_i \xrightarrow{\lambda_i} y_i$. Since $y_i \in Y_d$, then by the definition of Y_d, $\mathrm{Var}_{\mathsf{P}}(p_i) \subseteq X \cup Y_f \cup Y_0 \cup \ldots \cup Y_{d-1}$ so $\sigma(y) \mathcal{R} \sigma'(y)$ for $y \in \mathrm{Var}_{\mathsf{P}}(p_i)$ by $\beta(d-1)$, and $\mathrm{Var}_S(p_i) \subseteq Z \cup Z''$ so $\sigma(z) \equiv \sigma'(z)$ for $z \in \mathrm{Var}_S(p_i)$. Hence by the fact above $\sigma'(p_i) \mathcal{R} \sigma(p_i)$. There are three cases to consider:

- $\sigma(p_i) \sim^{\mathcal{E}}_{\equiv} \sigma'(p_i)$. Since $\mathcal{E} \vdash \sigma(p_i) \xrightarrow{\sigma(\lambda_i)} \sigma(y_i)$, there exist $w \in \mathbb{T}(\Sigma)_{\mathsf{P}}$ and $\alpha_i \in \mathbb{T}(\Sigma)_S$ such that $\mathcal{E} \vdash \sigma'(p_i) \xrightarrow{\alpha_i} w$, $\sigma(\lambda_i) \equiv \alpha_i$ and $\sigma(y_i) \mathcal{R} w$. Define $\sigma'(y_i) = w$. Moreover, by the compatibility of $\equiv$, suitable values of $\sigma'(z)$ for $z \in \mathrm{Var}_S(\lambda_i)$ can be found. (Since $\mathrm{Var}_S(p_i) \cap \mathrm{Var}_S(\lambda_i) = \emptyset$, this does not affect values assigned to $\mathrm{Var}_S(p_i)$. Also since the λ_i has distinct variables, this does not affect any other transition.)
- $\sigma(p_i) \sim^{\mathcal{E}_0}_{\equiv_0} \sigma'(p_i)$. There are two cases depending on whether the last rule of the proof of the transition is an extended *tyft/tyxt* rule from R_0 or an extended *tyxt* rule from R_1:
  - If $\sigma(p_i) \xrightarrow{\sigma(\lambda_i)} \sigma(y_i)$ comes from a extended *tyft/tyxt* rule from R_0 then by Lemma 42, there exist $w \in \mathbf{T}(\Sigma_0)_{\mathsf{P}}$ and $\alpha_i \in \mathbb{T}(\Sigma_0)_{S_0}$ such that $\mathcal{E}_0 \vdash \sigma(p_i) \xrightarrow{\alpha_i} w$ with $\alpha_i \equiv_0 \sigma(\lambda_i)$ (hence $\alpha_i \equiv \sigma(\lambda_i)$) and $\sigma(y_i) \mathcal{R} w$. Define $\sigma'(y_i) = w$. By a similar compatibility argument, values can be found for $z \in \mathrm{Var}_S(\lambda_i)$.
  - If $\sigma(p_i) \xrightarrow{\sigma(\lambda_i)} \sigma(y_i)$ comes from a extended *tyxt* rule from R_1, then the induction hypothesis can be used, since it is proved by a shorter proof. Since $(*)$ and $(**)$ hold, there exist $w \in \mathbb{T}(\Sigma)_{\mathsf{P}}$ such that $\mathcal{E} \vdash \sigma'(p_i) \xrightarrow{\alpha_i} w$ with $\sigma(\lambda_i) \equiv \alpha_i$ and $\sigma(y_i) \mathcal{R} w$. Define $\sigma'(y_i) = w$. A similar argument to above defines σ' for $z \in \mathrm{Var}_S(\lambda_i)$.
- there is a function symbol $h \in F$ such that $h : s'_1 \ldots s'_m \mathsf{P} \ldots \mathsf{P} \to \mathsf{P}$ with $\sigma(p_i) = h(\mu'_1, \ldots, \mu'_{m'}, w_1, \ldots, w_{n'})$ and $\sigma'(p_i) = h(\nu'_1, \ldots, \nu'_{m'}, w'_1, \ldots, w'_{n'})$ where $\mu'_{k'} \equiv \nu'_{k'}$ and $w_{i'} \mathcal{R} w'_{i'}$. The induction hypothesis can be applied. Since $\mathcal{E} \vdash h(\mu'_1, \ldots, \mu'_{m'}, w_1, \ldots, w_{n'}) \xrightarrow{\sigma(\lambda_i)} \sigma(y_i)$, there exist a w, and α_i such that $\mathcal{E} \vdash h(\nu'_1, \ldots, \nu'_{m'}, w'_1, \ldots, w'_{n'}) \xrightarrow{\alpha_i} w$, $\sigma(\lambda_i) \equiv \alpha_i$, and $\sigma(y_i) \mathcal{R} w$. Again define $\sigma'(y_i) = w$. Moreover, by the compatibility of $\equiv$, suitable values of $\sigma'(z)$ for $z \in \mathrm{Var}_S(\lambda_i)$ can be found.

Hence for any $y_i \in Y_d$, $\sigma(y_i) \mathcal{R} \sigma'(y_i)$, $\mathcal{E} \vdash \sigma'(p_i \xrightarrow{\lambda_i} y_i)$ and $\sigma(z) \equiv \sigma'(z)$ for all $z \in \mathrm{Var}_S(p_i) \cup \mathrm{Var}_S(\lambda_i)$. Therefore $\beta(d)$, $\gamma(d)$ and $\delta(d)$ hold for all $d \geqslant 0$, and (2), (3) and (4) hold. (1) holds by definition.

$\mathcal{E} \vdash f(\nu_1, \ldots, \nu_m, v_1, \ldots, v_n) \xrightarrow{\sigma'(\lambda)} \sigma'(p)$ since for all $i \in I$, $\mathcal{E} \vdash \sigma'(p_i) \xrightarrow{\sigma'(\lambda_i)} \sigma'(y_i)$. $\sigma'(\lambda) \equiv \sigma(\lambda) = \alpha$, since for all $z \in \mathrm{Var}_S(\lambda)$, $\sigma(z) \equiv \sigma'(z)$ and $\equiv$ is a congruence. $u = \sigma(p) \mathcal{R} \sigma'(p)$, by the fact above since for all $x \in \mathrm{Var}_{\mathsf{P}}(p)$, $\sigma(x) \mathcal{R} \sigma'(x)$ and for all $z \in \mathrm{Var}_S(p)$, $\sigma(z) \equiv \sigma'(z)$. $\square$

B Counter-examples for Theorems 46 and 47

Counter-examples are required to show that the conditions in the two main results cannot be weakened, although as shown in Section 3.3, it is possible to obtain similar results with different conditions.

In the following it will be shown that type-1, pureness, label-pureness and the requirement that the sum of congruences be conservative are necessary conditions for the refining extension result. In each counter-example, two closed terms which are not equivalent are given, and by relaxing the conditions, it is shown that the result is lost.

Counter-example 56 (The sum of congruences is not conservative) *Let* $\Sigma_0 = (\ \mathsf{P}, s, s';\ \ g_1, g_2, g_1', n_1, n_2, n_3;\ \ f_1, f_2, h, h'\ )$ *with* $g_1 :\rightarrow s$, $g_2 :\rightarrow s$, $g_1' :\rightarrow s'$, $n_1 :\rightarrow \mathsf{P}$, $n_2 :\rightarrow \mathsf{P}$, $n_3 :\rightarrow \mathsf{P}$, $f_1 : \mathsf{P} \rightarrow \mathsf{P}$, *and* $f_2 : \mathsf{P} \rightarrow \mathsf{P}$, $h : s' \rightarrow \mathsf{P}$, $h' : s', \mathsf{P} \rightarrow \mathsf{P}$.

Consider $\mathcal{E}_0$ *consisting of* Σ_0, $f_1(x) \xrightarrow{g_1} x$ *and* $f_2(x) \xrightarrow{g_2} x$. *Then* $f_1(n_1) \not\sim^{\mathcal{E}_0}_{\mathbf{Id}} f_2(n_1)$. *Consider summing this with* $\mathcal{E}_0'$ *consisting of* Σ_0 *together with* $\equiv_0$, *the smallest congruence containing* $\{g_1, g_2\}$. *Then* $\mathbf{Id} \oplus \equiv_0$ *is not conservative with respect to* $\mathbf{Id}$, *and* $f_1(n_1) \sim^{\mathcal{E}_0 \oplus \mathcal{E}_0'}_{\mathbf{Id} \oplus \equiv_0} f_2(n_1)$.

Counter-example 57 (The sum is not type-1) *Consider allowing a* tyxt *rule which has a conclusion label sort from the first signature. Let* $\mathcal{E}_1$ *consist of* Σ_0, *as well as the rule* $f_1(x) \xrightarrow{g_1} x$. *Then* $f_1(n_1) \not\sim^{\mathcal{E}_1}_{\mathbf{Id}} n_1$. *However if the sum of* $\mathcal{E}_1$ *and* $\mathcal{E}_1'$ *is formed where* $\mathcal{E}_1'$ *consists of* Σ_0 *and* $x \xrightarrow{g_1} n_1$, *then* $f_1(n_1) \sim^{\mathcal{E}_1 \oplus \mathcal{E}_1'}_{\mathbf{Id}} n_1$.

Next, consider allowing a tyft *rule with a conclusion function symbol and conclusion label sort from the first signature. Let* $\mathcal{E}_2$ *be* Σ_0 *together with* $f_1(x) \xrightarrow{g_1} x$. *Clearly,* $f_1(n_1) \not\sim^{\mathcal{E}_2}_{\mathbf{Id}} f_2(n_1)$. *However if* $\mathcal{E}_2$ *is summed with* $\mathcal{E}_2'$ *consisting of* Σ_0 *and* $f_2(x) \xrightarrow{g_1} x$, *then* $f_1(n_1) \sim^{\mathcal{E}_2 \oplus \mathcal{E}_2'}_{\mathbf{Id}} f_2(n_1)$.

Counter-example 58 (Rules are not pure: a free variable appears in the source of a premise) *Consider* Σ_0 *together with the rules below and call this* $\mathcal{E}_3$. *The second rule is not pure. Clearly* $f_1(n_1) \not\sim^{\mathcal{E}_3}_{\mathbf{Id}} f_2(n_1)$. *Summing with the eTSS* $\mathcal{E}_3'$ *consisting of* Σ_0 *together with* $n_4 :\rightarrow \mathsf{P}$, *and* $n_4 \xrightarrow{g_1} n_1$ *causes* $f_1(n_1) \sim^{\mathcal{E}_3 \oplus \mathcal{E}_3'}_{\mathbf{Id}} f_2(n_1)$.

$$\frac{}{f_1(x) \xrightarrow{g_1} x} \qquad \frac{x_1 \xrightarrow{z_s'} y}{f_2(x) \xrightarrow{g_1} y}$$

Counter-example 59 (Rules are not label-pure: a label-free variable appears in the conclusion) *Consider* $\Sigma' = (\mathsf{P}, s; g, h, nil; f, f')$ *with* $g :\rightarrow s$, $h : s \rightarrow s$, $nil :\rightarrow \mathsf{P}$, $f : \mathsf{P} \rightarrow \mathsf{P}$, *and* $f' : \mathsf{P} \rightarrow \mathsf{P}$, *and the eTSS* $\mathcal{E}'$ *consisting of*

Σ' *and the axiom schema* $f(x) \xrightarrow{Z_s} x$ *and the axiom* $f'(x) \xrightarrow{h(z_s)} x$ *which is not label-pure. The schema describes all rules that have a closed term from* $\mathbf{T}(\Sigma_0)_s$ *in the place of* Z_s*. Note that this is only a mechanism for describing these rules, and must be expanded before summing. Then* $f(nil) \xrightarrow{g} nil$*,* $f(nil) \xrightarrow{h(g)} nil$*,* $f(nil) \xrightarrow{h(h(g))} nil$ *..., and* $f'(nil) \xrightarrow{h(g)} nil$*,* $f'(nil) \xrightarrow{h(h(g))} nil$ *.... Clearly* $f(nil) \not\sim^{\mathcal{E}_0}_{\mathbf{Id}} f'(nil)$*.*

Consider the signature $\Sigma'' = (\ \mathsf{P}, s;\ \ g, g', h, nil;\ \ f, f'\ )$ *with* $g :\rightarrow s$*,* $g' :\rightarrow s$*,* $h : s \rightarrow s$*,* $nil :\rightarrow \mathsf{P}$*,* $f : \mathsf{P} \rightarrow \mathsf{P}$*, and* $f' : \mathsf{P} \rightarrow \mathsf{P}$*. Assume* $\equiv$ *is the smallest congruence containing* $\{(g, h(g')), (h(g), h(h(g'))), (h(h(g)), h(h(h(g')))), \ldots\}$*. Let* $\mathcal{E}'' = (\Sigma'', \emptyset)$*. Then* $\equiv = \mathbf{Id} \oplus \equiv$ *and is conservative with respect to* $\mathbf{Id}$*.*

Then $f(nil) \xrightarrow{g} nil$*,* $f(nil) \xrightarrow{h(g)} nil$*,* $f(nil) \xrightarrow{h(h(g))} nil$ *..., and* $f'(nil) \xrightarrow{h(g)} nil$*,* $f'(nil) \xrightarrow{h(h(g))} nil$*, ..., as well as* $f'(nil) \xrightarrow{h(g')} nil$*,* $f'(nil) \xrightarrow{h(h(g'))} nil$*, ... Now* $f'(nil)$ *can match the* g *transition with* $h(g')$ *under* $\equiv$*, and* $f(nil)$ *can match the new* $h(h(\ldots h(g')))$ *transitions with the equivalent* $h(\ldots h(g))$ *transition under* $\equiv_{\mathcal{E}_0 \oplus \mathcal{E}_1}$*. Hence* $f(nil) \sim^{\mathcal{E}' \oplus \mathcal{E}''}_{\equiv} f'(nil)$*.*

A similar counter-example for the axiom schemas $f(x) \xrightarrow{g} k(Z_s)$ *and the axioms* $f'(x) \xrightarrow{g} k(h(z_s))$ *(which is not label-pure) and* $k(z_s) \xrightarrow{z_s} nil$ *can be shown.*

It has not been shown that all pureness and label-pureness conditions are required for the theorem. However, it can be shown easily that the condition on free label variables in the source of a premise, and free variables in the target of the conclusion are required for Lemma 42.

It will now be shown that pureness, label-pureness, type-0 sum and compatibility are necessary conditions for the abstracting extension result. Note that the earlier comments about well-foundedness apply to this proof also.

Counter-example 60 (Rules are not pure) *A rule is not pure either when there is a free variable in the source of a premise or in the target of the conclusion. Consider* Σ_0 *used in Counter-example 56 together with the rule below which is not pure, and call this* $\mathcal{E}_4$*. Clearly* $n_1 \sim^{\mathcal{E}_4}_{\mathbf{Id}} f_1(n_1)$*. Summing with* $\mathcal{E}'_4$ *consisting of* Σ_0 *with* $n_4 :\rightarrow \mathsf{P}$ *and* $n_4 \xrightarrow{g_1} n_1$ *causes* $n_1 \not\sim^{\mathcal{E}_4 \oplus \mathcal{E}'_4}_{\mathbf{Id}} f_1(n_1)$*.*

$$\frac{x_1 \xrightarrow{z_s} y}{f_1(x) \xrightarrow{z_s} y}$$

Let $\mathcal{E}_5$ *be the eTSS consisting of* $\Sigma_5 = (\mathsf{P}, s;\ g_1, n_1, n_2)$ *where* $g_1 :\rightarrow s$*,* $n_1 :\rightarrow \mathsf{P}$ *and* $n_2 :\rightarrow \mathsf{P}$*, together with* $n_1 \xrightarrow{g_1} n_1$ *and* $n_2 \xrightarrow{g_1} x$*. The second axiom is not pure. Clearly* $n_1 \sim^{\mathcal{E}_5}_{\mathbf{Id}} n_2$*. Let* $\mathcal{E}'_5$ *be the eTSS consisting of* Σ_5 *with* $n_3 :\rightarrow \mathsf{P}$*. Then* $n_1 \not\sim^{\mathcal{E}_5 \oplus \mathcal{E}'_5}_{\mathbf{Id}} n_2$*.*

Counter-example 61 (Rules are not label-pure) *There are three aspects to the label-pureness condition. Consider* $\mathcal{E}_6$ *consisting of* Σ_0, $n_1 \xrightarrow{g'_1} n_3$ *and* $n_2 \xrightarrow{z_{s'}} n_3$ *(which is not label-pure). It is clear that* $n_1 \sim_{\mathbf{Id}}^{\mathcal{E}_6} n_2$. *Consider* $\mathcal{E}'_6$ *consisting of* Σ_0 *as well as the new constant* $g'_2 :\rightarrow s'$, *then* $n_1 \not\sim_{\mathbf{Id}}^{\mathcal{E}_6 \oplus \mathcal{E}'_6} n_2$.

Next, consider the eTSS $\mathcal{E}_7$ *consisting of* Σ_0, $n_1 \xrightarrow{g'_1} h(g'_1)$, $n_2 \xrightarrow{g'_1} h(z_{s'})$ *(which is not label-pure), and* $h(z_{s'}) \xrightarrow{z'_s} n_3$. *Clearly,* $n_1 \sim_{\mathbf{Id}}^{\mathcal{E}_7} n_2$. *Consider* $\mathcal{E}'_7$ *consisting of* Σ_0 *as well as the new constant* $g'_2 :\rightarrow s'$, *hence* $n_1 \not\sim_{\mathbf{Id}}^{\mathcal{E}_7 \oplus \mathcal{E}'_7} n_2$ *since* $n_2 \xrightarrow{g'_1} h(g'_2)$ *and* $h(g'_2) \not\sim_{\mathbf{Id}}^{\mathcal{E}_7 \oplus \mathcal{E}'_7} h(g'_1)$.

Finally consider the eTSS $\mathcal{E}_8$ *consisting of* Σ_0 *and the rules below, the second of which is not label-pure. It can be shown that* $f_1(n_1) \sim_{\mathbf{Id}}^{\mathcal{E}_8} f_2(n_1)$. *Let* $\mathcal{E}'_8$ *be* Σ_0 *together with the new constant* $g'_2 :\rightarrow s'$, *then there is a new transition* $h'(g'_2, n_1) \xrightarrow{g_1} h(g'_2)$, *and hence* $f_1(n_1) \not\sim_{\mathbf{Id}}^{\mathcal{E}_8 \oplus \mathcal{E}'_8} f_2(n_1)$, *since* $f_2(n_1) \xrightarrow{g_1} h(g'_2)$ *and* $h(g'_2) \not\sim_{\mathbf{Id}}^{\mathcal{E}_8 \oplus \mathcal{E}'_8} h(g'_1)$.

$$\frac{}{h'(z_{s'}, x) \xrightarrow{g_1} h(z_{s'})} \qquad \frac{h'(z_{s'}, x) \xrightarrow{g_1} y}{f_1(x) \xrightarrow{g_1} y} \qquad \frac{h'(g'_1, x) \xrightarrow{g_1} y}{f_2(x) \xrightarrow{g_1} y} \qquad \frac{}{h(z_{s'}) \xrightarrow{z_{s'}} n_1}.$$

Counter-example 62 (The sum is not type-0) *Consider* $\mathcal{E}_9$ *consisting of* Σ_0 *and the rules below. It can be shown that* $f_1(n_1) \sim_{\mathbf{Id}}^{\mathcal{E}_9} f_2(n_1)$. *Let* $\mathcal{E}'_9$ *be* Σ_0 *together with* $x \xrightarrow{g'_1} n_1$ *The sum is not type-0 since this axiom has a label sort from the first signature. Then a new transition* $f_2(n_1) \xrightarrow{g'_1} n_2$ *can be derived and* $f_1(n_1) \not\sim_{\mathbf{Id}}^{\mathcal{E}_9 \oplus \mathcal{E}'_9} f_2(n_1)$.

$$\frac{}{f_1(x) \xrightarrow{g_1} x} \qquad \frac{}{f_2(x) \xrightarrow{g_1} x} \qquad \frac{f_2(x) \xrightarrow{z_{s'}} y}{f_2(x) \xrightarrow{z_{s'}} n_2} \qquad \frac{}{n_2 \xrightarrow{g_1} n_3}$$

To see further why type-0 is needed, consider $\mathcal{E}_{10}$ *consisting of* Σ_0 *and* $f_1(x) \xrightarrow{g_1} x$ *and* $f_2(x) \xrightarrow{g_1} x$. *Clearly* $f_1(n_1) \sim_{\mathbf{Id}}^{\mathcal{E}_{10}} f_2(n_1)$. *Also consider the eTSS* $\mathcal{E}'_{10}$ *consisting of* Σ_0, *the new sort* s'', *the new constant* $g'' :\rightarrow s''$ *and* $f_2(x) \xrightarrow{g''} x$. *Then* $\mathcal{E}_{10} \oplus \mathcal{E}'_{10}$ *is type-1 but not type-0 and* $f_1(n_1) \not\sim_{\mathbf{Id}}^{\mathcal{E}_{10} \oplus \mathcal{E}'_{10}} f_2(n_1)$.

Counter-example 63 (Compatibility) *Let* Σ *and* $\equiv$ *be as in Counter-example 25 with axiom* $f(z_s) \xrightarrow{z_s} nil$. *Call this* $\mathcal{E}_{11}$. *Consider the eTSS* $\mathcal{E}'_{11}$ *consisting of* Σ, *a new sort* s_1 *and constant* $ok_1 :\rightarrow s_1$ *and the first rule below.* $\equiv$ *is not compatible with this rule since* $g_a \equiv g_b$ *and hence a suitable substitution cannot be found for the term on the premise. So although* $f(g_a) \sim_{\equiv}^{\mathcal{E}_{11}} f(g_a)$, $f(g_a) \not\sim_{\equiv}^{\mathcal{E}_{11} \oplus \mathcal{E}'_{11}} f(g_b)$. *The second rule below can be used to show that equivalence between the variables under the two substitutions is necessary.*

$$\frac{x \xrightarrow{g_a} y}{x \xrightarrow{ok_1} y} \qquad \frac{x \xrightarrow{g(z_s)} y}{x \xrightarrow{ok_1} y}$$

It is not clear whether there are counter-examples for compatibility with respect to the label terms that appear in the source of the conclusion. However, since this aspect of compatibility is used in the proof of the abstracting extension theorem and since compatibility is required for congruence, it is not an unreasonable condition.

References

[1] L. Aceto. Deriving complete inference systems for a class of GSOS languages generating regular behaviours. In B. Jonsson and J. Parrow, editors, CONCUR '94, Lecture Notes in Computer Science 836, pages 449–464. Springer-Verlag, Berlin, 1994.

[2] L. Aceto. GSOS and finite labelled transition systems. Theoretical Computer Science, 131:181–195, 1994.

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

[4] L. Aceto, W. Fokkink, and C. Verhoef. Conservative extension in structural operational semantics. Bulletin of the European Association for Theoretical Computer Science, 69:110–132, 1999.

[5] L. Aceto and M. Hennessy. Towards action refinement in process algebras. Information and Computation, 103:204–269, 1993.

[6] L. Aceto and M. Hennessy. Adding action refinement to a finite process algebra. Information and Computation, 115:179–247, 1994.

[7] L. Aceto and A. Ingólfsdóttir. CPO models for compact GSOS languages. Information and Computation, 129:107–141, 1996.

[8] J.C.M. Baeten and C. Verhoef. A congruence theorem for structured operational semantics with predicates. In E. Best, editor, CONCUR '93, Lecture Notes in Computer Science 715, pages 477–492. Springer-Verlag, Berlin, 1993.

[9] K.L. Bernstein. A congruence theorem for structured operational semantics of higher-order languages. In LICS 98, pages 153–164. IEEE Computer Society Press, New York, 1998.

[10] B. Bloom. Structured operational semantics for process algebras and equational axiom systems. In E. Best, editor, CONCUR '93, Lecture Notes in Computer Science 715, pages 539–540. Springer-Verlag, Berlin, 1993.

[11] B. Bloom. Structural operational semantics for weak bisimulations. Theoretical Computer Science, 146:25–68, 1995.

[12] B. Bloom, S. Istrail, and A.R. Meyer. Bisimulation can't be traced. Journal of the ACM, 42:232–268, 1995.

[13] R. Bol and J.F. Groote. The meaning of negative premises in transition system specifications. Journal of the ACM, 43:863–914, 1996.

[14] G. Boudol, I. Castellani, M. Hennessy, and A. Kiehn. A theory of processes with localities. Formal Aspects of Computing, 6(2):165–200, 1994.

[15] I. Castellani. Bisimulations for concurrency. PhD thesis, Department of Computer Science, University of Edinburgh, 1988. LFCS report ECS-LFCS-88-51.

[16] P.D. D'Argenio and C. Verhoef. A general conservative extension theorem in process algebras with inequalities. Theoretical Computer Science, 177:351–380, 1997.

[17] P. Darondeau and P. Degano. Causal trees. In G. Ausiello, M. Dezani-Ciancaglini, and S. Ronchi Della Rocca, editors, ICALP 88, Lecture Notes in Computer Science 372, pages 234–248. Springer-Verlag, Berlin, 1989.

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

[19] P. Degano and C. Priami. Enhanced operational semantics: a tool for describing and analyzing concurrent systems. ACM Computing Surveys, 33:135–176, 2001.

[20] G.L. Ferrari and U. Montanari. Parameterized structured operational semantics. Fundamenta Informatica, 34:1–31, 1998.

[21] W. Fokkink and R. van Glabbeek. Ntyft/ntyxt rules reduce to ntree rules. Information and Computation, 126:1–10, 1996.

[22] W. Fokkink and C. Verhoef. A conservative look at operational semantics with variable binding. Information and Computation, 146:24–54, 1998.

[23] V.C. Galpin. Equivalence semantics for concurrency: comparison and application. PhD thesis, Department of Computer Science, University of Edinburgh, 1998. LFCS report ECS-LFCS-98-397.
`http://www.dcs.ed.ac.uk/lfcsreps/`.

[24] V.C. Galpin. Comparison of process algebra equivalences using formats. In J. Wiedermann, P. van Emde Boas, and M. Nielsen, editors, ICALP '99, Lecture Notes in Computer Science 1644, pages 341–350. Springer-Verlag, Berlin, 1999.

[25] V.C. Galpin. Algebraic results for structured operational semantics. South African Computer Journal, 26:13–21, November 2000.

[26] R. Gorrieri and C. Laneve. The limit of split$_n$-bisimulations for CCS agents. In A. Tarlecki, editor, MFCS '91, Lecture Notes in Computer Science 520, pages 170–180. Springer-Verlag, 1991.

[27] J.F. Groote. Transition systems specifications with negative premises. Theoretical Computer Science, 118:263–299, 1993.

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

[29] M. Hennessy. A proof system for weak ST-bisimulation over a finite process algebra. Technical Report 6/91, Computer Science, University of Sussex, 1991.

[30] D.J. Howe. Proving congruence of bisimulation in functional programming languages. Information and Computation, 124:103–112, 1996.

[31] A. Kiehn. Comparing locality and causality based equivalences. Acta Informatica, 31(8):697–718, 1994.

[32] P. Krishnan. Architectural CCS. Formal Aspects of Computing, 162:162–187, 1996.

[33] C.A. Middelburg. Variable binding operators in transition system specifications. Journal of Logic and Algebraic Programming, 47(1):15–45, 2001.

[34] R. Milner. Communication and concurrency. Prentice Hall, Englewood Cliffs, NJ, 1989.

[35] R. Milner. Operational and algebraic semantics of concurrent processes. In J. van Leeuwen, editor, Handbook of Theoretical Computer Science, volume B, pages 1201–1242. MIT Press, Cambridge, MA, 1994.

[36] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes I & II. Information and Computation, 100:1–77, 1992.

[37] P.D. Mosses. Foundations of modular SOS. In M. Kutyłowski, L. Pacholski, and T. Wierzbicki, editors, MFCS '99, Lecture Notes in Computer Science 1672, pages 70–80, Berlin, 1999. Springer-Verlag.

[38] G. Plotkin. A structural approach to operational semantics. Report DAIMI FN-19, Computer Science Department, Aarhus University, 1988.

[39] R.J. van Glabbeek. Full abstraction in structural operational semantics. In M. Nivat, C. Rattray, T. Rus, and G. Scollo, editors, AMAST '93, Workshops in Computing, pages 77–84. Springer-Verlag, Berlin, 1993.

[40] R.J. van Glabbeek. The meaning of negative premises in transition system specifications II. In F. Meyer auf der Heide and B. Monien, editors, ICALP 96, volume 1099 of Lecture Notes in Computer Science, pages 502–513. Springer-Verlag, Berlin, 1996.

[41] C. Verhoef. A general conservative extension theorem in process algebra. IFIP Transactions A, 56:149–168, 1994.

[42] C. Verhoef. A congruence theorem of structured operational semantics with predicates and negative premises. Nordic Journal of Computing, 2:274–302, 1995.