Equivalence semantics for concurrency: comparison and application

Vashti Christina Galpin

Doctor of Philosophy
University of Edinburgh
1998

Abstract

Since the development of CCS and other process algebras, many extensions to these process algebras have been proposed to model different aspects of concurrent computation. It is important both theoretically and practically to understand the relationships between these process algebras and between the semantic equivalences that are defined for them.

In this thesis, I investigate the comparison of semantic equivalences based on bisimulation which are defined for process algebras whose behaviours are described by structured operational semantics, and expressed as labelled transition systems. I first consider a hierarchy of bisimulations for extensions to CCS, using both existing and new results to describe the relationships between their equivalences with respect to pure CCS terms. I then consider a more general approach to comparison by investigating labelled transition systems with structured labels. I define bisimulation homomorphisms between labelled transition systems with different labels, and show how these can be used to compare equivalences.

Next, I work in the meta-theory of process algebras and consider a new format that is an extension of the *tyft/tyxt* format for transition system specifications. This format treats labels syntactically instead of schematically, and hence I use a definition of bisimulation which requires equivalence between labels instead of exact matching. I show that standard results such as congruence and conservative extension hold for the new format.

I then investigate how comparison of equivalences can be approached through the notion of extension to transition system specifications. This leads to the main results of this study which show how in a very general fashion the bisimulations defined for two different process algebras can be compared over a subset of terms of the process algebras.

I also consider what implications the conditions which are required to obtain these results have for modelling process algebras, and show that these conditions do not impose significant limitations. Finally, I show how these results can be applied to existing process algebras. I model a number of process algebras with the extended format and derive new results from the meta-theory developed.

Acknowledgements

I would first like to thank my supervisor, Julian Bradfield, for his guidance and support through this long and at times arduous process. I would also like to thank Jane Hillston for her comments on a draft of this document.

I would like to thank Scott Hazelhurst, Ian Sanders and Conrad Mueller of the Department of Computer Science, University of the Witwatersrand, for encouraging me to pursue a doctoral degree outside of South Africa, and for providing me with working space in which to finish my thesis once I had returned to Johannesburg. I would also like to thank Conrad Mueller for introducing me to Robin Milner's work on CCS.

Finally, I would like to thank András Salamon for his support, proofreading and frequent visits, as well as my family in South Africa and my friends in Edinburgh for their support.

I would also like to thank the following organisations for the funding which made my time at the University of Edinburgh possible.

- The Patrick and Margaret Flanagan Scholarship
- The Overseas Research Students Awards Scheme
- The Foundation for Research and Development
- The Canon Collins Educational Trust for Southern Africa

Declaration

I declare that this thesis was composed by myself and that the work contained therein is my own, except where explicitly stated otherwise in the text.

(*Vashti Christina Galpin*)

Table of Contents

List of Figures

List of Tables

List of Definitions

List of Results

List of Examples

List of Counter-Examples

Notation

states	$\mathcal{S}$	s, t
actions	$\mathcal{A}$	a, b
transitions	$\mathcal{T}$	$s \xrightarrow{a} t$
labelled transition system (LTS)	$\mathcal{L} = (\mathcal{S}, \mathcal{A}, \mathcal{T})$	
strong bisimulation	$\sim$	
weak bisimulation	$\approx$	
strong generalised bisimulation with respect to a relation $\mathcal{B}$	$\sim_{\mathcal{B}}$	
weak generalised bisimulation with respect to a relation $\mathcal{B}$	$\approx_{\mathcal{B}}$	
identity relation	$\mathbf{Id}$	
bisimulation homomorphism	$(h_{\mathcal{S}}, h_{\mathcal{A}}, h_{\mathcal{T}})$	
S-sorted set	$\{A_s\}_{s \in S}$	
S'-sorted subset of S-sorted set	$A_{S'}$	
signature	$\Sigma = (S, F)$	
	$(s_1, s_2, \ldots; f_1, f_2, \ldots; g_1, g_2, \ldots)$	
variables	V, W	x
set of open terms over a subset of variables W	$T(\Sigma, W)$	
set of open terms over a subset of variables W of sort s	$T(\Sigma, W)_s$	
set of open terms over Σ	$\mathbb{T}(\Sigma)$	t
set of closed terms over Σ	$\mathbf{T}(\Sigma)$	
set of open terms over Σ of sort s	$\mathbb{T}(\Sigma)_s$	
set of closed terms over Σ of sort s	$\mathbf{T}(\Sigma)_s$	

variables of sort s that appear in a term t	$\mathrm{Var}_s(t)$	
variables that appear in a term t	$\mathrm{Var}(t)$	
substitution		σ, ρ
Σ-algebra	$\mathcal{A}$	
process sort	P	
S-sorted set $\mathcal{A}$ with $\mathsf{P} \notin S$	$\mathcal{A}_{\overline{\mathsf{P}}}$	
process variables	V_{P}	x, y
non-process variables	$V_S, V_{\overline{\mathsf{P}}}$	z
open process terms	$\mathbb{T}(\Sigma)_{\mathsf{P}}$	p, q
closed process terms	$\mathbf{T}(\Sigma)_{\mathsf{P}}$	u, v
open non-process terms	$\mathbb{T}(\Sigma)_S$, $\mathbb{T}(\Sigma)_{\overline{\mathsf{P}}}$	λ, η
closed non-process terms	$\mathbf{T}(\Sigma)_S$, $\mathbf{T}(\Sigma)_{\overline{\mathsf{P}}}$	α, β, μ, ν
extended transition specification system (eTSS)	$\mathcal{E} = (\Sigma, R)$	
transitions		ϕ, ψ, χ
bisimulation up to an equivalence $\equiv$	$\sim_{\equiv}$	
bisimulation up to an equivalence $\equiv$ on an eTSS $\mathcal{E}$	$\sim_{\equiv}^{\mathcal{E}}$	
sum of two signatures	$\Sigma_0 \oplus \Sigma_1$	
sum of two eTSSs	$\mathcal{E}_0 \oplus \mathcal{E}_1$	
asymmetric sum of two signatures	$\Sigma_0 \oplus\!\!> \Sigma_1$	
asymmetric sum of two eTSSs	$\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$	
sum of two equivalences	$\equiv_0 \oplus \equiv_1$	
sum of two algebras	$\mathcal{A}_0 \oplus \mathcal{A}_1$	
schema variables	$\mathcal{V}$	
process schema variables	$\mathcal{V}_{\mathsf{P}}$	X, Y
non-process schema variables	$\mathcal{V}_S, \mathcal{V}_{\overline{\mathsf{P}}}$	Z

Chapter 1

Introduction

Concurrency is a core issue in computer science, and it is vital to have theoretical foundations from which to understand it. There are many different approaches in the literature—Petri nets, automata and process algebras being just a few. This thesis will focus on process algebras.

Since the development of CCS and other process algebras, many extensions to these process algebras have been proposed to model different aspects of concurrent computation. It is important both theoretically and practically to understand the relationships between these process algebras and between the semantic equivalences that are defined for them. In this thesis, I investigate the comparison of semantic equivalences based on bisimulation which are defined for process algebras whose behaviours are described by structured operational semantics, and expressed as labelled transition systems.

A significant aspect of the different approaches to extensions to process algebras is the introduction of non-atomic labels; by this, I mean labels that have structure or contain information beyond what the action is. In CCS and CSP, basic actions are thought of as atomic, and are drawn from single set A. However in CCS, additional types of actions are required—to effect communication, a barred version of each action is required, drawn from a set $\overline{A}$, and then to represent an internal action, or the result of a communication, a distinguished action τ is required. So, even in CCS, the actions have in some sense more structure or information than simple atomic actions. In some timed transition systems, there are labels which represent actions and those that represent the passing of time. In transition systems obtained from the operational semantics of process algebras that look at the dependency (locational or causal) between actions, tags or markers appear in the labels (and are stored in the process terms) to keep a record of these dependencies.

As this short discussion shows, structured labels are used in many different types of process algebras, and hence their nature is of interest for study. Moreover, as can be seen by the proliferation of process algebras, it is relatively easy to design a new process algebra with features for the specific use one may want to put it to. This indicates that the notion of a process algebra has wide application and is flexible. A negative aspect to the proliferation of process algebras is that it is not immediately obvious how a process algebra and any semantic equivalences defined for its labelled transition system relate to other process algebras and equivalences. The aim of this thesis is to investigate how this comparison can be done. This research draws on the meta-theory of process algebras, specifically formats, to address this question.

1.1 Organisation of the thesis

Chapter 2 This chapter gives an overview of process algebras proposed in the literature, and looks at their differences and similarities. I also look at related work and give motivation for the results in this document.

Chapter 3 This chapter looks at two approaches to comparison of equivalences. In the first, a number of equivalences defined for extensions to CCS are compared in terms of which pure CCS term they equate. The shortcomings of this approach are discussed. Next, I look at using bisimulation homomorphisms as a way to compare equivalences.

Chapter 4 In this chapter, I start with a more syntactic approach to the comparison of equivalences, and develop a new format that explicitly deals with structured labels. I prove that this format gives congruence.

Chapter 5 In this chapter, a number of results are presented for the new format that relate to combining process algebras.

Chapter 6 In this chapter, I look at the implications of some of the conditions on the results for the underlying algebras used to represent structured labels. Next, a notion of rule schemas is developed to allow the expression of infinite rule sets. Finally, a number of process algebras are expressed in the new format and existing and new results are shown for the comparison of equivalences over these process algebras.

Chapter 7 This chapter presents conclusions and issues for further work.

Chapter 2

Background and motivation

2.1 Introduction

In this chapter, I present existing material that provides a background to the research presented in this thesis. I first spend some time looking at the extensions to process algebras based on structured operational semantics, labelled transition systems and process equivalences which are the inspiration and spur for my research. I also describe other approaches to comparing these extensions. I present background material on formats, as I will describe a new format in the body of this work. Finally, I discuss related research and give some motivation, including motivation for the particular subset of process algebras that I will be focusing on, namely those concerned with non-interleaving equivalences.

2.2 Background

Structured operational semantics (SOS) [Plo88] have been used to give labelled transition system semantics to models of concurrency called process algebras or process calculi. CCS [Mil89] is defined using this approach, together with notions of equivalences over processes to equate those processes which have the same behaviour. Other well-known process algebras that can be defined in this manner are CSP [OH86] and ACP [vG87]. These process algebras are based on the notion of atomic actions and communication via simple interaction. The labelled transitions embody the notion that actions take place one at a time, and these process algebras are interleaving, in the sense that nondeterministic sequentiality cannot be distinguished from concurrency. Other early process algebras took a slightly different approach and allowed for a number of actions to happen simultaneously, such as Meije [Bou84] and SCCS [Mil89]. Since then there has been an explosion of process algebras—many of them extensions of these first process algebras—developed to deal with different aspects of concurrency, such as true concurrency/non-interleaving, time, priorities, probabilistic and stochastic behaviour. These extensions have been approached in a number of different ways. In this section of this chapter, I give an overview of how these extensions have been developed without going into detail—my aim is here is to give a flavour of the ways in which extensions have been done, and how information can be added to labelled transition systems. I will also discuss approaches to the comparison of these process algebras and equivalences that have appeared in the literature. To support the descriptions of these process algebras and equivalences, I will also give some diagrammatic examples of processes. Many of these examples will use the 'canonical' non-interleaving examples $a.b.nil + b.a.nil$ and $a \mid b$.

Since labelled transition systems and bisimulation will play a large part in the work I am reviewing here, I present some definitions, and describe some phrases that I will be using to make the presentation more concise. The following two definitions are standard.

Definition 2.2.1 (Labelled transition system)
A *labelled transition system* (LTS) is defined as:

$$\mathcal{L} = (\mathcal{S}, \mathcal{A}, \mathcal{T})$$

where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of (not necessarily atomic) actions, $\mathcal{T} \subseteq (\mathcal{S} \times \mathcal{A} \times \mathcal{S})$. I write $s \xrightarrow{a} s'$ for $(s, a, s') \in \mathcal{T}$. ■

Definition 2.2.2 (Strong bisimulation)
Let $\mathcal{L} = (\mathcal{S}, \mathcal{A}, \mathcal{T})$ be an LTS. A *strong bisimulation* is a binary relation $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S}$ such that $(s, s') \in \mathcal{R}$ only if for all $a \in \mathcal{A}$,

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

Two states, s and s' are said to be *strongly bisimilar*, $s \sim s'$, if there exists a strong bisimulation $\mathcal{R}$ such that $(s, s') \in \mathcal{R}$. ∎

It can be shown that $\sim$ is the largest bisimulation. Note that it is also possible to define a bisimulation between the states of two different labelled transition systems if they have the same set of actions. In some of the extensions I will be discussing below, modified versions of the above definition are used, in that the elements of $\mathcal{A}$ that appear on the transitions do not have to be equal. Often in these extensions, the set $\mathcal{A}$ contains more than atomic actions, and hence it is possible to require equality on some components of the action and not others. Other extensions introduce a number of different sets for the transition labels (sometimes called multi-coloured transition systems [Tof94]) and there may be additional clauses in the bisimulation definition to deal with transitions with labels drawn from another set. I will use the phrase '*matching on ...*' to describe the equality of selected components of an action that are required for clauses (1) and (2) (or any additional clauses) in the above definition. I will also use the phrase '*bisimulation has the expected definition*' when all components are matched on.

I now discuss different extensions under headings which describe their main focus. Note that some equivalences may capture more than one aspect of concurrent behaviour.

2.2.1 True concurrency/non-interleaving equivalences

As discussed above, most process algebras are interleaving and the issue of how to describe true concurrency in the algebraic and compositional framework provided by process algebras is an ongoing one. By interleaving, I refer to process algebras where the semantic equivalence does not distinguish between non-deterministic sequentiality and concurrency; for example when $a.b.nil + b.a.nil$ and $a \mid b$ are equated. Process algebras that are non-interleaving have semantic equivalences which do not equate processes such as the two given above.

One approach to achieving non-interleaving/true concurrency is to map the syntax of a process algebra onto some construct that allows for definition of true concurrency semantics such as event structures or Petri nets. Examples of this approach are [BC94, BKLL95, DM87b, DDNM88a, DDNM88b, DDNM88c, GM84, Gol90, Lu93, Mur93, Tau90, vGV87, Win82]. In many of these papers, semantic equivalences are not defined—the authors are concerned with true concurrency at an operational level only. Since the focus of my research is on labelled transition systems, and semantic equivalences on these systems, I will not discuss these further. There are also attempts to take the type of semantic equivalences defined on labelled transition systems and apply them to other domains for processes. Examples of research that uses event structures are [BMC94, BC88a, GKP92, GL95, DDNM88c, ADNF87, Mur91, vGG90] and Petri nets [Old88, AS92, JM92, Pom85, PRS92]. A discussion of these approaches is beyond the scope of this document.

A second approach is to modify the underlying labelled transition system so that it contains the information that allows for the distinction between concurrency and nondeterministic sequentiality. I now discuss process algebras where this approach has been taken.

2.2.1.1 Causal approaches

In [DD89, DD90], information about the causal structure of processes is used to distinguish nondeterminism from concurrency. In [DD90], causal trees are defined and they can be viewed as a labelled transition system with transitions of the form $\xrightarrow{\langle a,K\rangle}$ where a is the action and $K \subset \mathbb{N}$ is a set that contains information describing which of the previous transitions the current transition is causally dependent on. Causal equivalence requires that the matching transition has the same set of causes. Figure 2.1 gives an examples of two processes that are not equated by causal equivalence. In the case of the sequential process, the second transition is dependent on the first, whereas in the concurrent process,the transitions are independent. Degano and Gorrieri have investigated action refinement in the setting of causal trees [DG93]. In [CdCC92], strong and weak bisimulations are defined in a framework that takes causality into account. Kiehn [Kie94] investigates notions of global and local causality which I will give details of below.

$$\begin{tikzcd}
& a.b.nil + b.a.nil \arrow[dl, "\langle a, \emptyset\rangle"'] \arrow[dr, "\langle b, \emptyset\rangle"] & & & a.nil \mid b.nil \arrow[dl, "\langle a, \emptyset\rangle"'] \arrow[dr, "\langle b, \emptyset\rangle"] & \\
{} \arrow[d, "\langle b, \{1\}\rangle"'] & & {} \arrow[d, "\langle a, \{1\}\rangle"] & {} \arrow[d, "\langle b, \emptyset\rangle"'] & & {} \arrow[d, "\langle a, \emptyset\rangle"] \\
{} & & {} & {} & & {}
\end{tikzcd}$$

Figure 2.1: Examples of processes which are not causally equivalent [DD90]

2.2.1.2 Distributed/location-based equivalences

A number of different approaches to location-based bisimulation have been developed over the last few years. In [Cas88,CH89,Hen88b,Kie,Kie89], the structure of states in the labelled transition system is modified to have the following form: $p \xrightarrow{a} \langle p', p''\rangle$. The first component p' is a local residual of the transition and the second component p'' is the global residual. Hence, there are no structured labels in the sense I have used above. Distributed bisimulation requires that actions are matched exactly and that local residuals are bisimilar and global residuals are bisimilar. Figure 2.2 illustrates how sequential processes have different residuals to concurrent processes. In a recent paper, Corradini and De Nicola [CDN97], give an alternative characterisation of this equivalence, that relies on decomposing processes into their sequential components called grapes. They also present a new equivalence over these decomposed processes called generalized distributed equivalence.

In [BCHK92,BCHK94], a location transition system is defined and transitions have the form $p \xrightarrow[u]{a} p'$ where $u \in Loc^*$ where Loc is a set of atomic locations, and Loc^* is the set of strings over this set of locations. A new operator, location prefixing ($l :: p$) is introduced to the process algebra. Each time an action is performed, a location is introduced. Location bisimulation requires matching on both actions and location strings. Figure 2.3 gives an example of two processes which are not location bisimulation equivalent. Note that the dotted lines indicate that other transitions involving different locations are possible. In the case of the sequential process, an action that occurs at the same location as a previous action,

$$\begin{tikzcd}
 & a.b.nil + b.a.nil \arrow[dl, "a"'] \arrow[dr, "b"] & & & a.nil \mid b.nil \arrow[dl, "a"'] \arrow[dr, "b"] & \\
\langle b.nil, nil \rangle & & \langle a.nil, nil \rangle & \langle nil, nil \mid b.nil \rangle & & \langle nil, a.nil \mid nil \rangle
\end{tikzcd}$$

Figure 2.2: Example of processes which are not distributed bisimulation equivalent

has a location string that consists of a new location as well as the action string of the previous action. In the concurrent process, actions occur at different locations and hence have different location strings. Parameterised location bisimulation [BCHK94] is defined with respect to a location relation $R \subseteq (Loc^* \times Loc^*)$, and does not require that actions are identical, but rather that the two locations are in R. It has been shown that location bisimulation can be expressed in the ordinary interleaving observation equivalence of the π–calculus, a process algebra that expresses mobility of processes [San96]. In [Ace94c], an equivalence defined with static locations is shown to be equivalent to location bisimulation [BCHK92]. This work is further developed in [Cas95]. Corradini and De Nicola [CDN94, CDN97] develop distributed grapes/maximal distribution equivalence which coincides with location bisimulation on CCS terms.

In earlier work, Boudol *et al.* [BC91,BCHK93], investigated a slightly different process algebra involving locations. In this algebra, a new string of locations are introduced for each action. Bisimulation requires matching on actions and strings of locations.

In [Kie94], a labelled transition system that allows for the capturing of information about local causality and global causality is defined. Specific instantiations of the general bisimulation defined gives rise to three equivalences, one of which coincides with location equivalence [BCHK92] on CCS processes and one of which coincides with causal bisimulation [DD89] on CCS processes. The third equivalence combines both local and global causality and is called local/global cause

$$\begin{tikzcd}
 & & a.b.nil + b.a.nil \arrow[dll, dotted] \arrow[dl, "a"', "l"] \arrow[dr, "b", "m"'] \arrow[drr, dotted] & & \\
{} & l :: b.nil \arrow[dl, dotted] \arrow[d, "b"', "ml"] & & m :: a.nil \arrow[d, "lm"', "a"] \arrow[dr, dotted] & {} \\
{} & l :: m :: nil & & m :: l :: nil & {}
\end{tikzcd}$$

$$\begin{tikzcd}
 & & a.nil \mid b.nil \arrow[dll, dotted] \arrow[dl, "a"', "l"] \arrow[dr, "b", "m"'] \arrow[drr, dotted] & & \\
{} & l :: nil \mid b.nil \arrow[dl, dotted] \arrow[dr, "b", "m"'] & & a.nil \mid m :: nil \arrow[dl, "a"', "l"] \arrow[dr, dotted] & {} \\
{} & & l :: nil \mid m :: nil & & {}
\end{tikzcd}$$

Figure 2.3: Example of processes which are not location bisimulation equivalent

$$\begin{tikzcd}
 & & a.b.nil + b.a.nil \arrow[dl, "a", "{\emptyset,\emptyset,l}"'] \arrow[dr, "b"', "{\emptyset,\emptyset,m}"] \arrow[dll, dotted] \arrow[drr, dotted] & & \\
 & l :: b.nil \arrow[d, "b"', "{\{l\},\{l\},m}"] \arrow[dl, dotted] & & m :: a.nil \arrow[d, "a", "{\{m\},\{m\},l}"'] \arrow[dr, dotted] & \\
 & l :: m :: nil & & m :: l :: nil &
\end{tikzcd}$$

$$\begin{tikzcd}
 & & a.nil \mid b.nil \arrow[dl, "a", "{\emptyset,\emptyset,l}"'] \arrow[dr, "b"', "{\emptyset,\emptyset,m}"] \arrow[dll, dotted] \arrow[drr, dotted] & & \\
 & l :: nil \mid b.nil \arrow[dr, "b"', "{\emptyset,\emptyset,m}"] \arrow[dl, dotted] & & a.nil \mid m :: nil \arrow[dl, "a", "{\emptyset,\emptyset,l}"'] \arrow[dr, dotted] & \\
 & & l :: nil \mid m :: nil & &
\end{tikzcd}$$

Figure 2.4: Example of processes which are not local/global cause equivalent

equivalence. Figure 2.4 illustrates how sequential and concurrent processes can differ with respect to local/global bisimulation. Each action has a cause associated with it, and has a local and global cause set to describe previous causes upon which the action may be dependent. This example has similarities to Figure 2.3. Again the dotted lines indicate that other transitions are possible—these transitions involve associating a different cause with the action under consideration.

In [Kri96, Kri91], a notion of location is used which is stricter in that synchronisation can only take place between processes at the same locality, hence it is necessary to use special sending actions to send an action to a specific location. Moreover, actions which happen in the same location must use the same location identifier. The labelled transition system is defined such that each transition is labelled with a location and an action. A bisimulation is defined with respect to this labelled transition system. Figure 2.5 illustrates how sequential and con-

$$\begin{tikzcd}
& & a.b.nil + b.a.nil \arrow[dll, dotted] \arrow[dl, "\langle l, a\rangle"'] \arrow[dr, "\langle m, b\rangle"] \arrow[drr, dotted] & & \\
{} & l :: b.nil \arrow[d, "\langle l, b\rangle"'] & & m :: a.nil \arrow[d, "\langle m, a\rangle"] & {} \\
& l :: nil & & m :: nil &
\end{tikzcd}$$

$$\begin{tikzcd}
& & a.nil \mid b.nil \arrow[dll, dotted] \arrow[dl, "\langle l, a\rangle"'] \arrow[dr, "\langle m, b\rangle"] \arrow[drr, dotted] & & \\
{} & l :: nil \mid b.nil \arrow[dl, dotted] \arrow[dr, "\langle m, b\rangle"'] & & a.nil \mid m :: nil \arrow[dl, "\langle l, a\rangle"] \arrow[dr, dotted] & {} \\
{} & & l :: nil \mid m :: nil & & {}
\end{tikzcd}$$

Figure 2.5: Example of processes which are not (Krishnan) distributed bisimulation equivalent

current processes can be distinguished. The dotted transitions for the sequential process indicate that the action could have occurred at other locations; however this is not the case for successive actions which occur at the same location as the initial action. In the concurrent process, there is a choice of location for each action. Note that this example does not demonstrate the communication aspects of this process algebra.

In [BB93b], a real-space process algebra is defined where actions are associated with three space co-ordinates and which are composed into multi-actions by operators which describe the relationship between the locations of the actions. Bisimulation has the expected definition. Another approach to distribution is given in [Fan92].

2.2.2 Equivalences involving duration or time

In [AH94, AH93], it is assumed that actions have non-zero duration. A modified labelled transition system is defined where the label on the transition can indicate the start or finish of an action, and a new prefixing operator is introduced to indicate actions that have not been completed. It is possible to distinguish concurrency and nondeterminism, since it can be distinguished whether actions can overlap or not. A number of different bisimulation equivalences have been defined for this transition system [AH94, AH93, Hen88a, Hen91, GL91] based on the notion of ST-bisimulation which was originally defined on Petri nets [vGV87]. In [Hen95] a testing equivalence is defined. Figure 2.6 gives an example of two processes that are not ST-bisimulation equivalent. Here, actions consist of a start $s(a)$ and a finish $f(a)$.

In [AM96], Aceto and Murphy define a transition system where transitions come with an action, a duration and the time at which the action occurred. Each non-τ action has a fixed duration associated with it. Operators are defined to indicate waiting and to specify the time at which a process starts. A timed branching bisimulation is defined which requires matching on all components of the transition label. Ill-timed paths (which are sequences of actions and times where the ordering does not reflect the order given by time) can be used to discover independent (concurrent) states. In Figure 2.7, two processes are illustrated. Each process has an associated clock, and each transition is labelled with the action, the time at which it occurred and a duration for the action. As can be seen, the sequential process has two actions which occur one after the other, and take total time $\Delta(a) + \Delta(b)$; whereas each component in the concurrent process has its own clock and hence it is possible for each action to start at time 0.

Gorrieri *et al.* [GRS95] define a performance equivalence. Each action has duration associated with it specified by a function from actions to durations, and the equivalence is parameterised by this function. Each transition is labelled with an action, time of occurrence and locational information; however the locational information is ignored for the purposes of defining the performance equivalence. A clock prefixing operator is introduced and is used to prefix all sequential agents.

$$\begin{tikzcd}
 & & & a.nil \mid b.nil \arrow[dl, "s(a)"'] \arrow[dr, "s(b)"] & & & \\
 & & f(a).nil \mid b.nil \arrow[dl, "f(a)"'] \arrow[dr, "s(b)"] & & a.nil \mid f(b).nil \arrow[dl, "s(a)"'] \arrow[dr, "f(b)"] & & \\
 & nil \mid b.nil \arrow[dr, "s(b)"] & & f(a).nil \mid f(b).nil \arrow[dl, "f(a)"'] \arrow[dr, "f(b)"] & & a.nil \mid nil \arrow[dl, "s(a)"'] & \\
 & & nil \mid f(b).nil \arrow[dr, "f(b)"] & & f(a).nil \mid nil \arrow[dl, "f(a)"'] & & \\
 & & & nil \mid nil & & &
\end{tikzcd}$$

$$\begin{tikzcd}
 & a.b.nil + b.a.nil \arrow[dl, "s(a)"'] \arrow[dr, "s(b)"] & \\
f(a).b.nil \arrow[d, "f(a)"'] & & f(b).a.nil \arrow[d, "f(b)"] \\
b.nil \arrow[d, "s(b)"'] & & a.nil \arrow[d, "s(a)"] \\
f(b).nil \arrow[dr, "f(b)"] & & f(a).nil \arrow[dl, "f(a)"'] \\
 & nil &
\end{tikzcd}$$

Figure 2.6: Example of processes which are not ST-bisimulation equivalent

$$\begin{tikzcd}
 & (a.b.nil + b.a.nil)\ 0 \arrow[dl, "a@0"', "\Delta(a)"] \arrow[dr, "b@0", "\Delta(b)"'] & \\
b.nil\ \Delta(a) \arrow[dr, "b@\Delta(a)", "\Delta(b)"'] & & a.nil\ \Delta(b) \arrow[dl, "a@\Delta(b)"', "\Delta(a)"] \\
 & nil\ (\Delta(a) + \Delta(b)) &
\end{tikzcd}$$

$$\begin{tikzcd}
 & (a.nil \parallel b.nil)\ 0 \arrow[dl, "a@0"', "\Delta(a)"] \arrow[dr, "b@0", "\Delta(b)"'] & \\
(nil\ \Delta(a)) \parallel (b.nil\ 0) \arrow[dr, "b@0", "\Delta(b)"'] & & (a.nil\ 0) \parallel (nil\ \Delta(b)) \arrow[dl, "a@0"', "\Delta(a)"] \\
 & (nil\ \Delta(a)) \parallel (nil\ \Delta(b)) &
\end{tikzcd}$$

Figure 2.7: Example of processes which are not (Aceto and Murphy) timed bisimulation equivalent

A diagram of the processes $a.b.nil + b.a.nil$ and $a.nil \mid b.nil$ (without the locational information) would be similar to Figure 2.7.

Baeten and Bergstra have investigated a number of notions related to time for ACP. In [BB91b], a real-time process algebra is introduced where some transitions have both an action and an element of the reals to represent the time at which the action occurred, and others are unlabelled and represent idling. States consist of pairs of process algebra terms and times. Bisimulation requires matching on steps, idling and termination. A number of time related operators are introduced, and the notion of relative time is investigated. This work is further extended by the introducing nonstandard reals, resulting in a process algebra that can express aspects of other process algebras involving time [BB95]. In [BB91a] a real space-time algebra is defined. Each action has four co-ordinates which can either be interpreted as three space co-ordinates and an independent time co-ordinate as in classical (Newtonian) mechanics, or as four related co-ordinates as in special relativity. Equivalences are defined with respect to these two viewpoints.

Chen [Che93, Che92] defined a timed bisimulation on a timed version of CCS, that distinguishes concurrency from nondeterministic sequentiality. Transitions are labelled with actions and the time at which the action occurs, or alternatively they can just be labelled with time which denotes idling up until that time is reached. Action prefixing has two new parameters indicating the earliest and latest the action can be performed. Bisimulation requires matching on actions and times. As an example, consider the process $a(t) \mid_1^6 .b(s) \mid_0^{10} .nil(0)$ which can perform the a action at any time between 1 and 6 time units, after which it can immediately perform the b action or delay for 0 to 10 ten units before b happens, and then can do no further actions nor let time proceed. The operational version of Timed CSP [Sch95] uses a similar type of transition system, and uses wait and timeout operators in the process algebra. Daniels [Dan91] also uses a similar transition system.

In [Yi90, Yi91, MT90, Jef92, QdFA93], a different transition system is used in which there are two different types of evolution of state—one in which an action can be performed (and the transition is labelled with the action) and one in which time passes (and the transition is labelled with a time value). Consider the following process with respect to Moller and Tofts' semantics [MT90] $a.(2).b.nil$ which can perform an a action, and become $(2).b.nil$ which has the transitions $(2).b.nil \xrightarrow{1} (1).b.nil \xrightarrow{1} b.nil$ or $(2).b.nil \xrightarrow{2} b.nil$ after which it can perform b and then not be capable of performing any further actions or of letting time proceed. Ho-Stuart *et al.* [HSZM93] take a similar approach, but use a monoid of actions as in SCCS and define two notions of bisimulation over this transition system. Aceto and Jeffrey [AJ95] generalise the approach of Wang Yi [Yi90] and introduce a time domain that is a left-cancellative anti-symmetric monoid.

A simpler notion of the passage of time is used in [HR95, NS94, BB96] where a distinguished action is added that is understood as passage of time to the next time slice. In each case, a number of time-related operators are introduced. In the semantics of Baeten and Bergstra, the process $\sigma_{\text{rel}}(\underline{\underline{a}}.\underline{\underline{b}}) + \sigma_{\text{rel}}(\underline{\underline{a}}.\underline{\underline{c}})$ can perform a unit delay to become $\underline{\underline{a}}.\underline{\underline{b}} + \underline{\underline{a}}.\underline{\underline{c}}$. Note that for the sum of two processes to be able to delay to the next time unit, both components must be able to delay. The action $\underline{\underline{a}}$ is one that must occur in the current time slice (the action a is allowed to delay until the next time slice). Hence $\underline{\underline{a}}.\underline{\underline{b}} + \underline{\underline{a}}.\underline{\underline{c}}$ can perform an a action and then b or alternatively an a action and then c, all without further delays. In [Jef91a], a partially ordered time domain is used and operational and denotational semantics are given. Although partially ordered time is used, it is not used to distinguish between concurrency and nondeterminism. An overview of the way in which timed process algebras differ is presented in [NS92].

Stochastic process algebras also assume that actions have duration, and this duration is characterised by a random variable, usually taken to be exponentially distributed. In [Hil96], transitions are labelled with an action and a rate (which defines the exponential distribution used for that action). From the multi-transition system, it is possible to obtain a Markov process. A number of notions of equivalences are proposed including a bisimulation that requires matching of actions as well as the matching of the rates at which each possible action can occur.

2.2.3 Equivalences based on with priorities

There have been a number of different approaches to priority in process algebras. One approach taken is to have prioritised actions, indicated by some annotation of the action as in [CH90, NCCC94]. Operators are introduced to prioritise and de-prioritise actions, and bisimulation requires matching on actions. In the process $\underline{\tau}.nil + b.nil$ the only transition possible is $\underline{\tau}$ since as a prioritised action (indicated by the underline) it has priority over the unprioritised b action. In [CLN96], a process algebra based on the notion of distributed priorities is presented where actions can only pre-empt other actions that appear in the same location.

In [Cam91, CW95] a number of priority-related operators are introduced, including an unless operator, and a *prisum* operator which is weighted in favour of the first operand. Transitions are decorated with actions and readysets of output actions. Bisimulation requires matching on actions and readysets.

In [Gro93], an operational version of the prioritised process algebra of Baeten *et al.* [BBK86] is presented. Here it is assumed that there is a partial ordering over actions representing their relative priorities. Bisimulation requires matching on actions. Assuming that $a < b$ in the partial order, and Θ is the priority operator, then $\Theta(a + b)$ can only perform a b action.

Gerber and Lee [GL94] define a process algebra where priorities are used to determine the interleaving of processes in situations where there are not sufficient resources for them to run simultaneously. Jeffrey [Jef91b] investigates the relationship between time and priority. A calculus of broadcast systems (CBS) [Pra95] which is based on a broadcast notion of communication as opposed to pairwise communication as in CCS, can be extended to include priorities (PCBS). Each broadcast has a priority associated with it, and this affects which broadcasts can occur.

2.2.4 Probabilistic equivalences

Larsen and Skou [LS92b] present a probabilistic process algebra in a SCCS-like style that defines a probabilistic transition system where each transition is labelled with an action and probability. Conditions are imposed on the probabilities that appear on the transitions and bisimulation requires matching on both actions and probabilities. Jou and Smolka [JS90] take a slightly different approach and define process algebras where there are two different types of transitions—those which are labelled with actions, and those that are labelled with probabilities. Bisimulation requires that the total probability of evolving into an equivalent state is the same. Tofts [Tof94] defines a bisimulation that takes into account the notion of relative frequency of actions. Smolka and Steffen [SS96] investigate priority in the framework of probabilistic process algebras. All of these process algebras introduce some sort of probabilistic choice operator.

2.2.5 Equivalences based on proved transitions

In the literature, there are a number of labelled transition systems defined where the set of actions contains information about the inference rules that are used to prove that transitions can happen. In [Cas88], pomset bisimulation is defined on a transition system labelled with actions that indicate the use of specific transition rules from the structured operational semantics. A congruence is defined over the actions, and a bisimulation is defined with respect to these actions—a transition has to be matched by a transition with a congruent action. Figure 2.8 illustrates two processes and demonstrates how the additional transitions are added. In [BC88b, BC88c], proved transition systems are defined where the actions contain information about the proofs used for a CCS-like language.

In [FM90, CFM90], a categorical notion of structured transition systems is defined with algebraic structure on both states and transitions for a language in the style of CCS. The structure of the actions is used to determine which transitions are independent and hence can be permuted. Equivalence is defined in terms of permutations of these actions. In [FM91], observation algebras are defined. Here the actions in the transition form an algebra and contain information about which transition rules were used. Equivalence classes of processes are induced by a concurrency relation. In [FGM91], a similar approach is taken to [FM91]. The actions form an observation algebra for a CCS-like algebra of processes; however the equivalence defined is a bisimulation that is parameterised by the observation algebra.

$$\begin{tikzcd}
 & (a.b.nil + b.a.nil) \arrow[dl, "a"'] \arrow[dr, "b"] \arrow[dd, shift right, "a : b"'] \arrow[dd, shift left, "b : a"] & \\
b.nil \arrow[dr, "b"] & & a.nil \arrow[dl, "a"'] \\
 & nil &
\end{tikzcd}$$

$$\begin{tikzcd}
 & (a.nil \mid b.nil) \arrow[dl, "a"'] \arrow[dr, "b"] \arrow[dd, "a \mid b"'] & \\
nil \mid b.nil \arrow[dr, "b"] & & a.nil \mid nil \arrow[dl, "a"'] \\
 & nil \mid nil &
\end{tikzcd}$$

Figure 2.8: Example of processes which are not pomset bisimulation equivalent

$$\begin{tikzcd}
& a.b.nil + b.a.nil \arrow[dl, "+_0 a"'] \arrow[dr, "+_1 b"] & & & & & a.b.nil + b.a.nil \arrow[dl, "\langle a, \emptyset \rangle"'] \arrow[dr, "\langle b, \emptyset \rangle"] & \\
b.nil \arrow[d, "b"'] & & a.nil \arrow[d, "a"] & {} \arrow[r] & {} & b.nil \arrow[d, "\langle b, \{1\} \rangle"'] & & a.nil \arrow[d, "\langle a, \{1\} \rangle"] \\
nil & & nil & & & nil & & nil
\end{tikzcd}$$

$$\begin{tikzcd}
& a.nil \mid b.nil \arrow[dl, "\|_0 a"'] \arrow[dr, "\|_1 b"] & & & & & a.nil \mid b.nil \arrow[dl, "\langle a, \emptyset \rangle"'] \arrow[dr, "\langle b, \emptyset \rangle"] & \\
nil \mid b.nil \arrow[d, "\|_1 b"'] & & a.nil \mid nil \arrow[d, "\|_0 a"] & {} \arrow[r] & {} & nil \mid b.nil \arrow[d, "\langle b, \emptyset \rangle"'] & & a.nil \mid nil \arrow[d, "\langle a, \emptyset \rangle"] \\
nil \mid nil & & nil \mid nil & & & nil \mid nil & & nil \mid nil
\end{tikzcd}$$

Figure 2.9: Example of processes which are not read-write bisimulation equivalent

In [DDNM92, DDNM93], observation trees are defined, and bisimulation is defined with respect to an observation function on the observation trees. Observation trees form unlabelled transition systems with structured states that store details of the computation up until that state. In [MY92, MY95], observation trees are used to present a parametric approach to localities in CCS.

A slightly different approach is taken in [DP92] where the transition system is labelled with proofs (of the transitions of CCS terms) to form proved trees after which an observation function is used to extract the relevant information from these proofs. In [PY94], an equivalence based on read–write causality is defined using this approach. An example of this is given in Figure 2.9. The proved trees on the left hand side are transformed by an observation function to the trees on the right hand side. Note that this example does not involve communication and hence does not demonstrate the read-write features.

In [IPY93, IPY94], extended transition systems for CCS terms are defined. These incorporate ideas from observation trees [DDNM92, DDNM93], proved transition systems [BC88b] and proved trees [DP92], and are unlabelled transition systems with nodes labelled with summations of regular expressions over the alphabet of proof terms which represent all computations from the root to the current node. Bisimulation is parameterised by an observation function.

2.2.6 Other approaches

Labelled transition systems can come equipped with sets of axioms that determine which transitions are allowed. Examples are concurrent transition systems [Sta89], asynchronous transitions systems [Bed87], elementary transition systems [NRT92] and transition systems with independence [NC94]. In [MN92], a notion of bisimulation that preserves independence is defined and the author conjectures that this equivalence coincides with location equivalence [BCHK92]. Figure 2.10 gives an examples of two processes from which asynchronous transition systems can be extracted. In the sequential process, there are four distinct events; whereas in the concurrent one, there are two distinct independent events.

In [BB93a], a non-interleaving process algebra is defined using multiset actions and step bisimulation semantics. In [DM87a], distributed transition systems are defined where states are sets of processes and transitions specify which processes stay idle. Bisimulation is defined on nondeterministic measurement systems which are unfoldings of computations from the distributed transition system defined by an observation function. This type of transition system is further developed in [DDNM88b, DDNM90].

Krishnan [Kri96, Kri92] investigates a process algebra to model the behaviour of multiprocessors. Transitions are labelled with observations that capture the idea that one can observe at most n actions in one step if there are n processors. Bisimulation requires matching on observations. Figure 2.11 illustrates an example where there are 2 multiprocessors. In both processes, it is possible for any action to be executed on one processor with the other idle (δ indicates idleness); however only the concurrent process has the possibility of executing both actions

$$\begin{tikzcd}
& (a.b.nil + b.a.nil) \arrow[dl, "a"', "{[ab+ba][b]}"] \arrow[dr, "b", "{[ab+ba][a]}"'] & \\
b.nil \arrow[dr, "b", "{[b][nil]}"'] & & a.nil \arrow[dl, "a"', "{[a][nil]}"] \\
& nil &
\end{tikzcd}$$

$$\begin{tikzcd}
& (a.nil \parallel b.nil) \arrow[dl, "a"', "{0[a][nil]}"] \arrow[dr, "b", "{1[b][nil]}"'] & \\
nil \parallel b.nil \arrow[dr, "b", "{1[b][nil]}"'] & & a.nil \parallel nil \arrow[dl, "a"', "{0[a][nil]}"] \\
& nil \parallel nil &
\end{tikzcd}$$

Figure 2.10: Example of processes which are not equated by bisimulation over asynchronous transition systems [MN92]

$$\begin{tikzcd}
& a.b.nil + b.a.nil \arrow[dl, bend right, "\langle \delta, a \rangle"'] \arrow[dl, "\langle a, \delta \rangle"] \arrow[dr, "\langle b, \delta \rangle"'] \arrow[dr, bend left, "\langle \delta, b \rangle"] & \\
b.nil & & a.nil
\end{tikzcd}$$

$$\begin{tikzcd}
& a.nil \mid b.nil \arrow[dl, bend right, "\langle \delta, a \rangle"'] \arrow[dl, "\langle a, \delta \rangle"] \arrow[d, shift right=2, "\langle a, b \rangle"'] \arrow[d, shift left=2, "\langle b, a \rangle"] \arrow[dr, "\langle b, \delta \rangle"'] \arrow[dr, bend left, "\langle \delta, b \rangle"] & \\
nil \mid b.nil & nil \mid nil & a.nil \mid nil
\end{tikzcd}$$

Figure 2.11: Example of processes which are not multiprocessor bisimulation equivalent

simultaneously—one on each processor.

Other papers of interest deal with modified labelled transition systems but do not involve equivalences. Examples are [LRT88] where distributed transition systems are defined with transitions labelled with sets of actions and the notion of a concurrent step is presented, but no equivalences are defined.

Process algebras have also been used to investigate fault-tolerant systems. Krishnan [Kri94] proposes a process algebra for replicated systems with voting and introduces a replication operator. The labelled transition system involves transitions that can be perceived as internal moves which occur before voting has happened, and transitions that represent the external behaviour of the system. Bisimulation is defined over the external actions only. Janowski [Jan94] uses two levels of transitions—the first represents the actions of the system in a fault-free environment and the second the actions in the environment with faults. He defines a number of equivalences which involve matching of actions on transitions from the different levels.

2.2.7 Comparison

Some research has attempted an overview of the different semantics and models. Category theoretical approaches are taken in [KLP90, Mes90, MY89, SNW93] and other approaches appear in [BC94, DN87, Fer93, Sha92, vG90a]. Many of these approaches do not investigate semantic equivalences; for example, [KLP90, Mes90, MY89, SNW93] and [BC94] where three equivalent semantics for CCS are presented—one based on proved transition systems, one on flow event structures and one on flow nets—and it is shown that the three notions coincide by using transition systems of 'trace computations' that record the past.

In [DN87], an early investigation into equivalences on labelled transition systems is given. In [Sha92], the issue of language embedding is investigated for a number of concurrent programming languages, and in [Fer93], the concept of data is introduced into labelled transition systems—five different labelled transition systems are defined, a labelled transition system is derived that can be used to express any one of the five, and bisimulation equivalence is investigated in this setting. Different notions of equivalence have been investigated on event structures [vG90a].

I now discuss the approaches to comparison which are most relevant to my work.

- Interleaving semantics defined on labelled transition systems have been extensively investigated by van Glabbeek [vG90b, vG93], both with respect to linear and branching time, and abstraction from internal actions. In the first of two papers about semantic equivalences [vG90b], he looks at processes defined in labelled transition systems, specifically those that are interleaving (which he refers to as sequential), finitely branching, with uniform concurrency and no abstraction from internal actions, and develops a complete lattice of 11 different semantic notions. The equivalence semantics range from the finest, bisimulation, to the coarsest, trace equivalence. More recent equivalences which could be added to this framework are undo-trace equivalence and undo-failure equivalence [Sch91]. In the second paper [vG93], he investigates the linear time/branching time spectrum for semantics which abstract from internal actions, and presents a hierarchy of 155 different equivalences. A number of these equivalences are compared over probabilistic processes in [JS90].

- Some of the approaches discussed in Section 2.2.5 allow for the comparison of different equivalences over CCS terms. The observation trees of Degano *et al.* [DDNM92, DDNM93] permit different types of observations to be made and hence it is possible to compare the different equivalences generated by these observations. In [MY92, MY95] the equivalences defined by taking combinations of mixed orderings and partial orderings, and localities and causalities are compared. In [IPY94] the authors describe how a number of different equivalences can be defined on extended transition systems.

- In [Kie94], Kiehn defines a new transition system based on local and global causes. Each transition is labelled with an action, a set of local causes and a set of global causes. Local causes are understood to be due to actions that occurred in the component from which the current action came, and global causes are understood to come from an action in any component. Bisimulation equivalence is parameterised by a function which extracts information from the causes on a transition. She shows that causal bisimulation and location equivalence can be characterised by appropriate instantiations of this function. In [KH94], it is shown that ST-equivalence can also be formulated using a variant of this transition system where only local causes are considered, with the start of an action having an empty cause set and the completion of the action having a singleton cause set.

- Gorrieri and Laneve [GL91] compare different split action transition systems. Their approach to split actions differs from that of Aceto and Hennessy [AH94, AH93] in that they assume that τ actions are split as well as non-τ actions. Their approach to comparison is of interest and I describe it here. Given two labelled transition systems $\mathcal{L}_i = (\mathcal{S}_i, \mathcal{A}_i, \mathcal{T}_i)$, generated by two processes algebras, and their respective strong equivalences $\sim_i$ for $i = 1, 2$, it is required to prove $\sim_1 \subseteq \sim_2$, i.e. given s and t that appear

$$\begin{tikzcd} s \arrow[d, Rightarrow] & \sim_1 & t \arrow[d, Rightarrow] \\ s' \arrow[d, "h"'] & \sim_1' & t' \arrow[d, "h"] \\ s & \sim_2 & t \end{tikzcd}$$

Figure 2.12: Proof technique used by Gorrieri and Laneve [GL91]

as states in both transition systems, such that $s \sim_1 t$, show that $s \sim_2 t$. First, transform $\mathcal{L}_1$ to $\mathcal{L}_1' = (\mathcal{S}_1', \mathcal{A}_2, \mathcal{T}_1')$ by a set of transformation rules. Next show that the transformations of s and t, s' and t' respectively, are equated by $\sim_1'$ in $\mathcal{L}_1'$, by exhibiting a suitable bisimulation. Finally, show that each relevant state (s' and t') in $\mathcal{L}_1'$ is equivalent to a state in $\mathcal{L}_2$—this is achieved by finding a suitable transition system/transition preserving homomorphism [AD89,Arn93] that maps from the subtransition system[*] associated with the state in $\mathcal{L}_1'$ onto the subtransition system associated with the state in $\mathcal{L}_2$. The existence of such a homomorphism implies the two subtransition systems are bisimilar, and hence any state u is bisimilar to its image $h(u)$. So to sum up, if two states s and t are equated by $\sim_1$ in $\mathcal{L}_1$, show that their transformations s' and t' are bisimilar in $\mathcal{L}_1'$, then by using the transition preserving homomorphism h, show that s' is bisimilar to $h(s')$ and t' is bisimilar to $h(t')$, but h is chosen so that $s = h(s')$ and $t = h(t')$, hence $s \sim_2 t$ as required. This is illustrated in Figure 2.12, where the double arrows indicate the transformation rules.

A crucial part of this proof is the use of homomorphisms between transition systems. This notion has been introduced in a number of places in the literature but with some confusing terminology. These homomorphisms map states to states and transitions to transitions in the expected manner and have a condition that requires the resultant state of a transition from a state

[*]A subtransition system associated with a state consists of the states and transitions that are reachable from that state.

in the image, to be equal to the image of the resultant state of that transition from the state in the domain. A number of authors call these homomorphisms transition preserving [GL91, FMM91, CFM90, FM90, DDNM88a], they have also been referred to as transition system homomorphisms [AD89] and bisimulation homomorphisms [Arn93]. In [AD89], it is shown how these homomorphisms can be used in proving that two labelled transition systems are strongly bisimilar. These homomorphisms have also been investigated in a categorical setting [BBS88,CFM90,FM90,FMM91]. More recently, Arnold and Castellani [AC96] have considered homomorphisms for weak bisimulation. They define transition system homomorphisms as homomorphisms between transition systems, but without the condition described above. They then define homomorphisms which are saturating for all operators drawn from the action set of the transition system under consideration and show that this saturating condition holds if and only if the homomorphism satisfies a weak variant of the condition described above (which they refer to as the *zig-zag condition*.) They also compare these saturating homomorphisms to Castellani's abstraction homomorphisms. [Cas88].

2.2.8 Formats

In the main part of this research, I will work with formats. A format is a class of rules used to describe models such as process algebras defined by structured operational semantics which generate labelled transition systems. There are a number of different formats and in the discussion below, I will describe their differences. To start, consider the following very general rule

$$\frac{\{t_i \xrightarrow{a_i} y_i \mid i \in I\}\{u_j \overset{b_j}{\not\rightarrow} \mid j \in J\}}{f(x_1, \ldots, x_n) \xrightarrow{a} t}$$

where I and J are index sets, the y_i's and x_k's are distinct variables and the t_i's, u_j's and t are open terms over a set of operators and f is an operator. I refer to the transitions above the line as *premises*, the transition below the line as the *conclusion*, the left-hand side of a transition as the *source* of the transition, and the right-hand side of a transition as the *target*. Also the first set of premises are called *positive premises* and the second set are called *negative premises*. (For a formal treatment of this, please consult the start of Chapter 4.)

The first format proposed was the De Simone format [dS85] which requires that there only be positive premises. Moreover the only terms that can appear in the sources of the premises are variables from the source of the conclusion and each can appear at most once. This means there can be no copying of terms in the premises. There are also restrictions on which variables can appear in t. This format also allows for a condition over the labels that appear in the premises and conclusion, and specifies a finite set of labels.

GSOS format [BIM95] is more general than De Simone format. Variables from the source of the conclusion can appear repeatedly in the sources of premises; however they are the only terms that can appear in sources of premises. The term t must only contain variables that appear in the rules, and there can only be a finite number of premises. Again a finite label set is assumed.

Another format that is more general than the De Simone format is the *tyft/tyxt* format [GV92]. This format does not allow negative premises, but the t_i's and t can be any terms. It allows lookahead since variables which appear as targets of the premises can also appear in the sources of premises. This format has been extended to deal with predicates [BV93].

There is a format that is more general than both GSOS and (pure*) *tyft/tyxt* format, and that is (pure) *ntyft/ntyxt* format [Gro93]. The *ntyft/ntyxt* format is similar to *tyft/tyxt* but allows negative premises. Stratification techniques are used to prevent inconsistent rules. Recently a more general technique has been developed to ensure consistent rules for the *ntyft/ntyxt* format [BG96]. This format has also been extended to deal with predicates resulting in the *panth* format [Ver95].

Groote [Gro93] characterises the relationship between these formats as shown in Figure 2.13 where positive GSOS refers to GSOS without negative premises.

An important issue in the research involving formats is that of congruence of the format with respect to a semantic equivalence defined over the terms of

*This concept will be defined in Chapter 4.

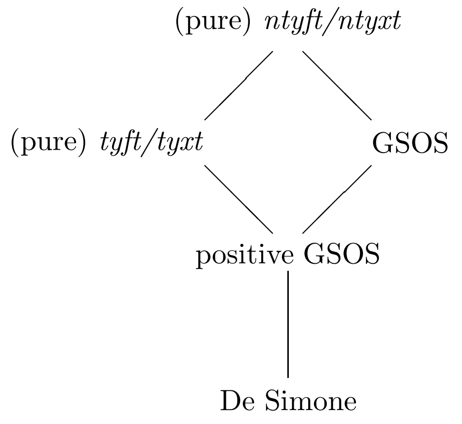

Figure 2.13: Hierarchy of formats [Gro93]

the process algebra, such as testing equivalence or bisimulation. It is of interest because first it is a desirable property for process algebra operators to have, and second because each format induces a trace and completed trace congruence. In the case of De Simone format, it induces a completed trace congruence that is the same as failure equivalence, whereas *ntyft/ntyxt* induces bisimulation equivalence, and the other formats in Figure 2.13 induce completed trace congruences which fall between these two [Gro93].

Other research directions for formats include looking at formats in the light of bisimulation with silent moves—Bloom [Blo95] considers four different bisimulations and identifies what sort of rules preserve congruence in each case, and Ulidowski [Uli92] defines the ISOS format and shows that ISOS trace congruence coincides with copy+refusal equivalence. It has also been shown that any GSOS format can be converted to a finite complete equational axiom system with possibly one infinitary induction principle [ABV94]. Aceto also considers a class of infinitary GSOS formats which have a countable action set, signature and set of rules [Ace94a] and a restricted version of GSOS which generates finite labelled transition systems [Ace94b]. Aceto and Ingólfsdóttir have given denotational semantics for a class of GSOS formats and shown full abstraction [AI96].

There are other results related to formats [Ver94, FvG96] that are of relevance to the research I am presenting here, and I will discuss them in the appropriate places in this document.

Verhoef considers a general conservative extension theorem for process algebras [Ver94]. By considering each transition as a separate relation, he is able to find a wider range of rules that are allowed in extensions of transition systems specified by a certain format, and still obtain a conservative extension. I will describe later in this document how this compares with the approach I have taken.

Fokkink and van Glabbeek [FvG96] have shown that the well-foundedness is not required to show congruence in the *ntyft/ntyxt* format. I will discuss how this relates to my research later in the document.

2.3 Related work

Because of the number of formalisms proposed to model concurrency, as well as the proliferation of process algebras, there are a number of articles that look at comparing these approaches or that present unifying frameworks. In the following, I describe some approaches that have been taken.

Astesiano *et al.* [AG92] define observational structures, where process algebras are modelled as many-sorted algebras with predicates. Equivalence semantics are defined in terms of observations, where processes have observable sorts, and an abstract generalisation of bisimulation is defined by experiments considered similar by a similarity law. Propagation laws allow relations over processes to be propagated to relations over elements of non-observable sorts, such as actions. This approach gives a generalisation for bisimulation that allows for different process algebras to be expressed, including CCS, a process algebra with mobility and distributed CCS. However, the approach is not used to compare different process algebras or equivalences.

In Section 2.2.7, I have discussed related work that deals with the comparison of process algebras and their equivalences. As mentioned, many approaches only consider the comparison of operational semantics, whereas my focus here is equivalence semantics. Some of the research discussed such as Van Glabbeek [vG90b, vG93] focuses on the linear-time/branching-time spectrum and gives a very full account of it. Other work such as Kiehn's [Kie94] and Gorrieri and Laneve's focus on the comparison of specific process algebras without providing a general theory for the comparison of process algebras. Degano and Priami's proved trees [DP92] allow for the comparison of location, causal, interleaving and read-write causal equivalences for CCS terms. The most wide-reaching comparisons in terms of the number of equivalences considered are the observation trees of Degano *et al.* [DDNM92, DDNM93] and the extended transition systems of [IPY93, IPY94]. However as the underlying theory of these relies on the SOS of CCS, they are not a general approach for comparing equivalences of process algebras.

2.4 Motivation

As can be seen from the Background section in this chapter, there are many different process algebras, designed to cover a range of concurrent behaviours. It is important to be able to understand how the different process algebras and equivalences relate to each other, so that it is possible to choose which one to use in a particular set of circumstances, or when designing a new one, to see how it differs from existing process algebras. In the previous section, I have discussed how others have approached comparing and unifying different process algebras. In this document, I look at three different approaches. The first looks at a comparison of extensions to CCS over pure CCS terms, based on ad hoc results and counter-examples. The second approach moves away from syntax, and works with an extended notion of a bisimulation homomorphism. Neither of these two approaches is entirely satisfactory. The third approach is to work with the meta-theory of process algebra. This is based on the notion of format, and I prove results about formats that allow for the comparison of process algebra equivalences. This is a new approach to comparison, and gives a broad theoretical basis upon which further work can be developed.

In this document, I will focus on a subset of process algebras, specifically those based on dependencies of some sort or involving true concurrency or non-interleaving features. This includes process algebras and equivalences which have been investigated in studies of comparison, such as location equivalences [BCHK93, BCHK94], cause-based equivalences [Kie94], ST-equivalences [AH94, AH93, Hen88a, Hen91, GL91], as well as those that have not been considered in studies of comparison such as the various process algebras proposed by Krishnan [Kri91, Kri92, Kri94, Kri96] and the pomset equivalence of Castellani [Cas88]. I have chosen this subset for a number of reasons; first, these are some of process algebras that piqued my interest because of the ways in which they distinguish interleaving versus concurrent behaviour, and drew me to this area of study; and second they cover a range of process algebras that have and have not been used in comparisons before. I wish to look at the application of the results here to other process algebras as further work.

2.5 Conclusion

In this chapter, I presented background to this thesis, including a broad survey of different process algebras, a discussion of existing approaches to comparison and work that has been done in the area of formats. Finally, I presented a motivation for the approaches I will investigate in this thesis.

Chapter 3

Two approaches to comparison

3.1 Introduction

In this chapter I will investigate two different approaches to comparing equivalences. The first approach works with the various *ad hoc* comparison results and counter-examples. By *ad hoc*, I mean that each result is individually obtained using techniques specific to the equivalences being compared. I collect these results and counter-examples to give a hierarchy of equivalences for non-interleaving extensions to CCS. The second approach involves considering the underlying process domain, namely the labelled transition systems, and obtaining results which describe how equivalences can be compared—an important aspect of this approach is the notion of bisimulation homomorphism.

3.2 Comparison over pure CCS terms

In this section, I will look at existing results for the comparison of extensions to CCS, and some of my own. These results will yield a hierarchy of equivalences with respect to pure CCS terms. I will also give examples of process algebras which cannot be included in this hierarchy, and explain why it is not possible to include them.

As mentioned above, a number of extensions have been proposed to CCS which permit non-interleaving semantics. It is possible to compare the equivalences on finite CCS terms, although most of the process algebras used for the definition of these equivalences have additional operators. This comparison is possible because pure CCS terms are still valid terms in these process algebras and they also display

$\approx$	observation equivalence [Mil89]
$\approx_d$	distributed bisimulation [Cas88, CH89, Kie89]
$\approx_{da}$	K-grapes distributed bisimulation [CDN97]
$\approx_g$	generalised distributed bisimulation [CDN97]
$\approx_l$	location bisimulation [BCHK94]
$\approx_{ll}$	loose location bisimulation [BCHK93]
$\approx_l^s$	static location bisimulation [Cas93]
$\approx_{dg}$	distributed grapes equivalence [CDN94] maximal distribution equivalence [CDN97]
$\approx_{loc}$	equivalence with location observations [MY92]
$\approx_c$	weak causal bisimulation [DD89]
$\approx_{lc}$	local cause bisimulation [Kie94]
$\approx_{gc}$	global cause bisimulation [Kie94]
$\approx_{lg}$	local/global cause bisimulation [Kie94]
$\approx_{rw}$	read-write bisimulation [PY94]
$\approx_{ST}$	ST-bisimulation [Hen91]

Table 3.1: Equivalences that form the hierarchy

the specific concurrent behaviour for which the process algebra is designed. I wish to do this comparison over pure CCS terms because

- many authors give some comparison when presenting new process algebras, and I wish to present an overview of this, and
- many of the existing comparisons are done for process algebras for which pure CCS terms are of interest, hence my overview takes this approach, since it allows consideration of a number of process algebras. I will detail below the process algebras that cannot be compared in this manner.

In the spirit of van Glabbeek, I have developed a hierarchy of equivalences that are finer than Milner's observation equivalence, by assembling results from the literature and providing additional counter-examples. Table 3.1 lists the equivalences that appear in the hierarchy and Figure 3.1 displays the relationship between these equivalences. A path from an equivalence to one lower in the diagram means that the higher equivalence is contained in the lower one. This means that the lower equivalence equates the same terms that the higher one does, but

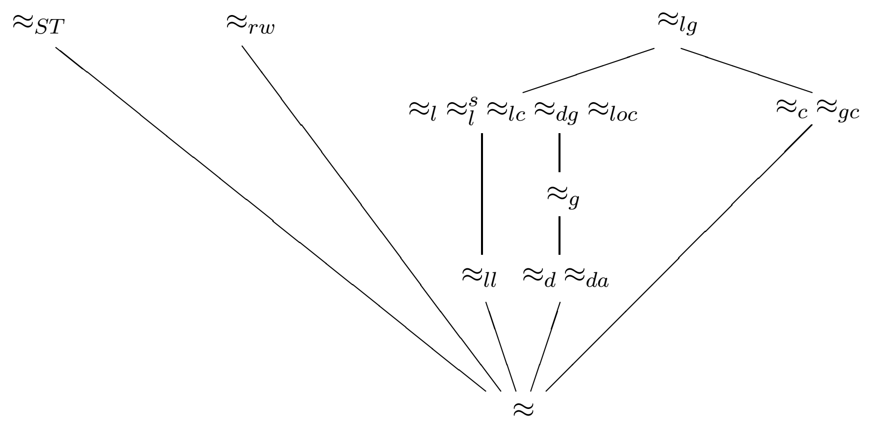

Figure 3.1: A hierarchy of equivalences for finite CCS terms

also equates other terms as well, namely it is coarser*. Note that observation equivalence is the coarsest of all these equivalences as would be expected. The equivalences that are not connected by a downwards path are incomparable; that is, it is possible to find two pairs of terms such that the first pair is equated by the first equivalence but not the second, and such that the second pair are equated by the second equivalence, but not the first. The details of this comparison are presented below. It is also known that on finite restriction and renaming free CCS, $\approx_d$, $\approx_l$ and $\approx_{ll}$ coincide [BCHK94], as do $\approx_d$, $\approx_{da}$, $\approx_g$ and $\approx_{dg}$ [CDN97]; and that on finite restriction and renaming free CCS without communication, $\approx_l$, $\approx_c$ and $\approx_{ST}$ have the same axiomatisation [Kie93]. Aceto and Murphy show that their timed bisimulation also coincides with these equivalences on this subset [AM96].

There are a number of equivalences based on extensions to CCS that are not suitable to add to this hierarchy using this type of comparison. They are as follows:

[Kri96, Kri91] In this extension to CCS, the action set consists of local and send actions, local actions come from a set of actions as in CCS, whereas send actions are more complex objects. Although pure CCS terms do lead to

*An equivalence that makes more equations than another equivalence is said to be *coarser*, and an equivalence that makes fewer equations than another equivalence is said to be *finer*. This usage is consistent with the fact that a finer equivalence results in more equivalence classes than one that is coarser.

some non-interleaving behaviour (see Figure 2.5), since they do not express all the different aspects of the process algebra, I have chosen not to add this to the hierarchy. Moreover, there is only a strong version of the equivalence.

[**Kri96, Kri92**] This extension to CCS presents a multiprocessor model of concurrency. Since there is only a strong version of the equivalence, I have not included it. However, I do obtain results relating to this process algebra and equivalence in Chapter 6.

[**Fan92**] This extension is unsuitable for inclusion since the pure CCS terms do not express non-interleaving behaviour.

[**MN92**] In this paper, CCS with guarded sums is used, hence the set of base CCS terms (those consisting of the CCS operators) are only a subset of the pure CCS terms. Therefore, the comparison cannot be done over the pure terms of CCS, and hence it is not included in the hierarchy. The equivalence defined coincides with the equivalence of Aceto [Ace94c] on nets of automata (see the next point), and the authors conjecture that their equivalence coincides with location bisimulation [BCHK94] on CCS with guarded sums.

[**Ace94c**] Here, a subset of CCS terms is used, where the static operators, namely parallel, renaming and relabelling, can only appear at the highest level of a term. These terms are called *nets of automata*. Hence a comparison cannot be made over the pure terms of CCS. The author has shown that on the set of nets, his equivalence coincides with location bisimulation [BCHK94]. Moreover, this work has been generalised, and the generalisation $\approx_l^s$ [Cas93] is included in the hierarchy.

[**Mur93**] In this paper a subset of CCS is used where terms represent parallel combinations of sequential process. Once again, this is a subset and does not allow for comparison.

[**AH93, AH94**] The ST-equivalences of Aceto and Hennessy which have been developed to investigate action refinement are based on a process algebra consisting of actions (as opposed to prefixed actions) and a termination predicate, and hence the basic terms of the process algebra differ from those of CCS.

[GL91] The split-action bisimulations of Gorrieri and Laneve involve the splitting of non-τ actions and τ actions, and is defined only for a strong version of bisimulation. It is not clear how this approach can be extended to weak bisimulation.

As seen in the previous chapter, there are many other process algebras; however, I have chosen here to focus on a certain subset, and this is sufficient to demonstrate this approach. Also note that for many other process algebras that are extensions of CCS, especially for those based on time, priorities and probabilities, when considering the equivalence on pure CCS terms, then it is no different from weak bisimulation over CCS terms. Hence this method of comparison does not help, since the comparison method is not powerful enough to distinguish meaningful differences.

I now give the details of the comparisons which are summarised in Figure 3.1. These are presented as follows:

- Table 3.2 contains pairs of processes that are used to show when two equivalences are incomparable.

- Tables 3.3, 3.4 and 3.5 describe the relationship between each pair of equivalences. For each pairwise comparison, there is a block in the table. In the centre of each block, there is a symbol giving the relationship—# indicates when the two equivalences in question are incomparable. As an example, the block at the fourth column and third row of Table 3.3 is to be read as $\approx_g \subset \approx_{da}$. If the result of the comparison comes from the literature, there will be a citation at the bottom of the block. Finally, if the two equivalences are incomparable, there may be two letters at the top of the block, each of which refer to a pair of processes in Table 3.2. The first pair are equated by the equivalence that appears in the column, and not equated by the

A	$(ca \mid b(\overline{c} + a))\backslash c$	$(bc \mid \overline{c}a)\backslash c$
B	$(e(c\Sigma_a + d\Pi_b + \overline{c}\Sigma_a + \overline{d}\Pi_b \mid$ $(\overline{d}\Sigma_b + \overline{c}\Pi_b + c\Sigma_a + d\Pi_a))\backslash\{c, d\}$	$(e(c\Sigma_a + d\Pi_b + \overline{c}\Sigma_a + \overline{d}\Pi_b \mid$ $(\overline{c}\Sigma_b + \overline{d}\Pi_b + d\Sigma_a + c\Pi_a))\backslash\{c, d\}$
C	$(aec \mid b\overline{e}d)\backslash e$	$(aed \mid b\overline{e}c)\backslash e$
D	$(ab \mid cd)$	$(a(eb + b) \mid c(\overline{e}d + d)\backslash e$
E	$(ac + bd \mid \overline{c}b + \overline{d}a)$	$(ac + bd \mid \overline{c}b + \overline{d}a) + ab$
F	$(ac + bd \mid \overline{c}b + \overline{d}a)$	$(ac + bd \mid \overline{c}b + \overline{d}a) + (a \mid b)$
G	$(a\overline{b} \mid bc)\backslash b$	ac
H	$(aec \mid b\overline{e}d)\backslash e$	$(a\overline{e}c \mid bed)\backslash e$
I	$(a\overline{b}d \mid bc)\backslash b + (cb \mid \overline{b}ad)\backslash b$	$(ad \mid c)$
J	$(a \mid b) + (ac \mid \overline{c}b)\backslash c$	$(a \mid b)$
K	$(ab \mid \overline{b}c) + ac$	$(ab \mid \overline{b}c)$
L	$a(b + \tau.c) + ac$	$a(b + \tau.c)$
M	$(e.l.(m.a.p \mid \overline{n}.c) \mid (\overline{l}.f.s.(m.a.b \mid \overline{p}.c) \mid$ $((m.a.n \mid \overline{b}.c) \mid \overline{s}.\overline{m})))\backslash\{b, l, m, n, p, s\}$	$(e.l.(m.a.p \mid \overline{n}.c) \mid (\overline{l}.f.s.(m.a.b \mid \overline{b}.c) \mid$ $((m.a.n \mid \overline{p}.c) \mid \overline{s}.\overline{m})))\backslash\{b, l, m, n, p, s\}$

where $\Sigma_a = a_1a_2 + a_2a_1$ and $\Pi_a = a_1 \mid a_2$.

Table 3.2: CCS processes used for comparison

equivalence that appears in the row; and the second pair are not equated by the equivalence that appears in column, and are equated by the equivalence that appears in the row. These pairs of letters do not appear for all incomparable equivalences—since there are a number of equal equivalences in the table, I have only given letters pairs to one (usually the first one) in a group of equal equivalences.

As can be seen, this *ad hoc* approach is somewhat limited since it is based upon pure CCS terms. In the next section and the following three chapters, I look at more general approaches to the comparison of process algebras and equivalences.

	$\approx$	$\approx_d$	$\approx_{da}$	$\approx_g$	$\approx_{ll}$
$\approx$	—	$\subset$ [Cas88]	$\subset$	$\subset$	$\subset$ [BCHK93]
$\approx_d$		—	$=$ [CDN97]	$\subset$ [CDN97]	A, B # [BCHK91a]
$\approx_{da}$			—	$\subset$	A, B #
$\approx_g$				—	A, M # [CDN97]
$\approx_{ll}$					—
$\approx_l$					
$\approx_l^s$					
$\approx_{lc}$					
$\approx_{dg}$					
$\approx_{loc}$					
$\approx_c$					
$\approx_{gc}$					
$\approx_{lg}$					
$\approx_{rw}$					
$\approx_{ST}$					

Table 3.3: Summary of relationship between equivalences

	$\approx_l$	$\approx_l^s$	$\approx_{lc}$	$\approx_{dg}$	$\approx_{loc}$
$\approx$	$\subset$	$\subset$	$\subset$	$\subset$	$\subset$
$\approx_d$	$\subset$ [BCHK94]	$\subset$	$\subset$	$\subset$	$\subset$
$\approx_{da}$	$\subset$	$\subset$	$\subset$	$\subset$	$\subset$
$\approx_g$	$\subset$	$\subset$	$\subset$	$\subset$ [CDN97]	$\subset$
$\approx_{ll}$	$\subset$ [BCHK94]	$\subset$	$\subset$	$\subset$	$\subset$
$\approx_l$	—	$=$ [Cas93]	$=$ [Kie94]	$=$ [CDN94]	$=$ [MY92]
$\approx_l^s$		—	$=$	$=$	$=$
$\approx_{lc}$			—	$=$	$=$
$\approx_{dg}$				—	$=$
$\approx_{loc}$					—
$\approx_c$					
$\approx_{gc}$					
$\approx_{lg}$					
$\approx_{rw}$					
$\approx_{ST}$					

Table 3.4: Summary of relationship between equivalences continued

	$\approx_c$	$\approx_{gc}$	$\approx_{lg}$	$\approx_{rw}$	$\approx_{ST}$
$\approx$	$\subset$	$\subset$ [Kie94]	$\subset$	$\subset$ [PY94]	$\subset$ [Hen91]
$\approx_d$	E, F $\#$	$\#$	$\subset$	A, H $\#$	A, F $\#$
$\approx_{da}$	E, F $\#$	$\#$	$\subset$	A, H $\#$	A, F $\#$
$\approx_g$	E, F $\#$	$\#$	$\subset$	A, H $\#$	A, J $\#$
$\approx_{ll}$	C, D $\#$ [BCHK93]	$\#$	$\subset$	G, H $\#$	G, J $\#$
$\approx_l$	E, F $\#$ [BCHK91b]	$\#$	$\subset$	G, H $\#$ [PY94]	G, J $\#$
$\approx_l^s$	$\#$	$\#$	$\subset$	$\#$	$\#$
$\approx_{lc}$	$\#$	$\#$	$\subset$ [Kie94]	$\#$	$\#$
$\approx_{dg}$	$\#$	$\#$	$\subset$	$\#$	$\#$
$\approx_{loc}$	$\#$	$\#$	$\subset$	$\#$	$\#$
$\approx_c$	—	$=$ [Kie94]	$\subset$	$\#$ [PY94]	$\#$
$\approx_{gc}$		—	$\subset$ [Kie94]	G, H $\#$	G, L $\#$
$\approx_{lg}$			—	I, H $\#$	D, K $\#$
$\approx_{rw}$				—	H, I $\#$
$\approx_{ST}$					—

Table 3.5: Summary of relationship between equivalences continued

3.3 A syntax-free approach to comparison

As seen in the previous section, it is possible to consider a number of *ad hoc* results and use them to compare equivalences. However, it would be preferable to approach the comparison in a more systematic manner. In this section, I look at the underlying process domain, namely the labelled transition systems that describe the behaviour of process algebra terms and develop an approach to comparison.

As discussed in Chapter 2, the notion of transition system/transition preserving/saturating/bisimulation homomorphism has been defined and can be used to show that two labelled transition systems or states within a labelled transition system are bisimilar. As there are a number of different names used in the literature, in sometimes conflicting ways, I will use the term *bisimulation homomorphism* to describe these functions.

In the existing work that relates to bisimulation homomorphisms, it is assumed that the transition systems under consideration have labels that come from the same set. In the material that follows I will relax that assumption and broaden the definition of bisimulation homomorphism.

3.3.1 A general model of transition systems

I propose a model which will formalise the notions encapsulated in a number of the labelled transition systems that have been presented in the previous chapter. These labelled transition systems are characterised by transitions which are labelled with actions that are not necessarily atomic, i.e. they may have some structure. I will refer to them as extended labelled transition systems—'extended' because of the structured action. This is essentially the same as Definition 2.2.1. I am restating it here to emphasise the nature of the label set, and to add the definition of weak transitions.

Definition 3.3.1 (Extended labelled transition system)
An *extended labelled transition system* (LTS) is defined as

$$\mathcal{L} = (\mathcal{S}, \mathcal{A}, \mathcal{T})$$

where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of (not necessarily atomic) actions, and

$$\mathcal{T} \subseteq (\mathcal{S} \times \mathcal{A} \times \mathcal{S}).$$

I will usually write $s \xrightarrow{a} s'$ for $(s, a, s') \in \mathcal{T}$, and if it is required to prevent confusion, I will use $s \xrightarrow{a}_{\mathcal{T}} s'$. Additionally, I am interested in transition systems that have a distinguished action τ and I have the following definitions which define a new labelled transition system over the states of $\mathcal{S}$, using standard notation.

$$\Longrightarrow = (\xrightarrow{\tau})^* \quad \text{and} \quad \overset{a}{\Longrightarrow} = \Longrightarrow\xrightarrow{a}\Longrightarrow \quad \text{and} \quad \overset{\tau}{\Longrightarrow} = \Longrightarrow\xrightarrow{\tau}\Longrightarrow$$ ■

I want to work with a general definition of bisimulation and I will make this more specific as required.

Definition 3.3.2 (Strong generalised bisimulation with respect to a relation)
Let $\mathcal{L}_i = (\mathcal{S}_i, \mathcal{A}_i, \mathcal{T}_i)$ for $i = 0, 1$ be two LTSs, and let $\mathcal{B}$ be a relation on $\mathcal{A}_0 \times \mathcal{A}_1$. A *strong generalised bisimulation with respect to a relation* $\mathcal{B}$ is a binary relation $\mathcal{R} \subseteq \mathcal{S}_0 \times \mathcal{S}_1$ such that $(s_0, s_1) \in \mathcal{R}$ only if

1. for all $a_0 \in \mathcal{A}_0$, whenever $s_0 \xrightarrow{a_0}_{\mathcal{T}_0} t_0$, then there exists $t_1 \in \mathcal{S}_1$ and $a_1 \in \mathcal{A}_1$ such that $s_1 \xrightarrow{a_1}_{\mathcal{T}_1} t_1$, $(a_0, a_1) \in \mathcal{B}$ and $(t_0, t_1) \in \mathcal{R}$,

2. for all $a_1 \in \mathcal{A}_1$, whenever $s_1 \xrightarrow{a_1}_{\mathcal{T}_1} t_1$, then there exists $t_0 \in \mathcal{S}_0$ and $a_0 \in \mathcal{A}_0$ such that $s_0 \xrightarrow{a_0}_{\mathcal{T}_0} t_0$, $(a_0, a_1) \in \mathcal{B}$ and $(t_0, t_1) \in \mathcal{R}$.

Two states, s_0 and s_1 are said to be *strongly generalised bisimilar with respect to* $\mathcal{B}$, $s_0 \sim_{\mathcal{B}} s_1$, if there exists a strong generalised bisimulation $\mathcal{R}$ such that $(s_0, s_1) \in \mathcal{R}$. Let $\sim_{\mathcal{B}} = \bigcup\{\mathcal{R} \mid \mathcal{R}$ is a strong generalised bisimulation with respect to $\mathcal{B}$ $\}$. ■

Proposition 3.3.1 (Properties of strong generalised bisimulation with respect to a relation)
Let $\mathcal{L}_i = (\mathcal{S}_i, \mathcal{A}_i, \mathcal{T}_i)$ for $i = 0, 1$ be two LTSs, and let $\mathcal{B}$ be a relation on $\mathcal{A}_0 \times \mathcal{A}_1$.

1. $\sim_{\mathcal{B}}$ is the largest strong generalised bisimulation with respect to $\mathcal{B}$.

2. If $\mathcal{L}_0 = \mathcal{L}_1$ and $\mathcal{B}$ is an equivalence relation then $\sim_{\mathcal{B}}$ is an equivalence relation.

Proof: Straightforward. ■

Note that the use of a symmetrical relation symbol, namely $\sim$, is not intended to indicate that the relation is necessarily symmetric or an equivalence*. If I take $\mathcal{B}$ to be the identity relation then I obtain the standard definition of strong bisimulation, and I will use $\sim$ for $\sim_{\mathbf{Id}}$.

I am interested in the definition of weak bisimulation and also make a general definition. Here I assume that I have an element τ that is distinct from all other actions, and use the transitions defined above. Transitions labelled with τ play an important rôle in process algebras, as they indicate internal actions which result from communication. One often wishes to abstract away from these actions when observing the external behaviour of a process, and hence the notion of weak bisimulation is required. So although I wish to work in a syntax-free manner, the τ action is an important facet of the notion of equivalence.

Definition 3.3.3 (Weak generalised bisimulation with respect to a relation)
Let $\mathcal{L}_i = (\mathcal{S}_i, \mathcal{A}_i \cup \{\tau\}, \mathcal{T}_i)$ for $i = 0, 1$ be two LTSs, and let $\mathcal{B}$ be a relation on $\mathcal{A}_0 \times \mathcal{A}_1$. A *weak generalised bisimulation with respect to a relation* $\mathcal{B}$ is a binary relation $\mathcal{R} \subseteq \mathcal{S}_0 \times \mathcal{S}_1$ such that $(s_0, s_1) \in \mathcal{R}$ only if

*The word *equivalence* is used both to mean *equivalence relation* and *semantic equivalence* (which may or may not be an equivalence relation) in this document. When there may be confusion, I will use the longer phrases.

1. for all $a_0 \in \mathcal{A}_0$, whenever $s_0 \xrightarrow{a_0}_{\mathcal{T}_1} t_0$, then there exists $t_1 \in \mathcal{S}_1$ and $a_1 \in \mathcal{A}_1$ such that $s_1 \overset{a_1}{\Longrightarrow}_{\mathcal{T}_0} t_1$, $(a_0, a_1) \in \mathcal{B}$ and $(t_0, t_1) \in \mathcal{R}$

2. for all $a_1 \in \mathcal{A}_1$, whenever $s_1 \xrightarrow{a_1}_{\mathcal{T}_1} t_1$, then there exists $t_0 \in \mathcal{S}_0$ and $a_0 \in \mathcal{A}_0$ such that $s_0 \overset{a_0}{\Longrightarrow}_{\mathcal{T}_0} t_0$, $(a_0, a_1) \in \mathcal{B}$ and $(t_0, t_1) \in \mathcal{R}$.

3. whenever $s_0 \xrightarrow{\tau}_{\mathcal{T}_0} t_0$, then there exists $t_1 \in \mathcal{S}_1$ such that $s_1 \Longrightarrow_{\mathcal{T}_1} t_1$ and $(t_0, t_1) \in \mathcal{R}$,

4. whenever $s_1 \xrightarrow{\tau}_{\mathcal{T}_0} t_1$, then there exists $t_0 \in \mathcal{S}_0$ such that $s_0 \Longrightarrow_{\mathcal{T}_1} t_0$ and $(t_0, t_1) \in \mathcal{R}$.

Two states, s_0 and s_1 are said to be *weakly generalised bisimilar with respect to* $\mathcal{B}$, $s_0 \approx_{\mathcal{B}} s_1$, if there exists a weak generalised bisimulation $\mathcal{R}$ such that $(s_0, s_1) \in \mathcal{R}$. Let $\approx_{\mathcal{B}} = \bigcup\{\mathcal{R} \mid \mathcal{R}$ is a weak generalised bisimulation with respect to $\mathcal{B}$ $\}$. ■

Proposition 3.3.2 (Properties of weak generalised bisimulation with respect to a relation)
Let $\mathcal{L}_i = (\mathcal{S}_i, \mathcal{A}_i, \mathcal{T}_i)$ for $i = 0, 1$ be two LTSs, and let $\mathcal{B}$ be a relation on $\mathcal{A}_0 \times \mathcal{A}_1$.

1. $\approx_{\mathcal{B}}$ is the largest weak generalised bisimulation with respect to $\mathcal{B}$.

2. If $\mathcal{L}_0 = \mathcal{L}_1$ and $\mathcal{B}$ is an equivalence relation then $\approx_{\mathcal{B}}$ is an equivalence relation.

Proof: Straightforward. ■

Again if I take $\mathcal{B}$ to be the identity relation then I obtain the standard definition of weak bisimulation, and I will use $\approx$ for $\approx_{\mathbf{Id}}$. I choose in this chapter to work with weak bisimulation for two reasons: first, most of the examples presented in the literature use a weak form of bisimulation, and second, in this setting, working with τ transitions and weak bisimulation does not add significantly to the complexity of the results. I will use the term *bisimulation* when referring to the notion of weak bisimulation.

3.3.2 Bisimulation homomorphisms

I first present a general result that relates the bisimulations defined on different extended LTSs. I define homomorphisms on extended LTSs.

Definition 3.3.4 (Bisimulation homomorphism)
A *bisimulation homomorphism* is a mapping

$$(h_{\mathcal{S}}, h_{\mathcal{A}}, h_{\mathcal{T}}) : (\mathcal{S}_0, \mathcal{A}_0, \mathcal{T}_0) \to (\mathcal{S}_1, \mathcal{A}_1, \mathcal{T}_1)$$

with $h_{\mathcal{S}} : \mathcal{S}_0 \to \mathcal{S}_1$, $h_{\mathcal{A}} : \mathcal{A}_0 \to \mathcal{A}_1$ and $h_{\mathcal{T}} : \mathcal{T}_0 \to \mathcal{T}_1$ such that

$$h_{\mathcal{T}}(s \xrightarrow{a}_{\mathcal{T}_0} s') = h_{\mathcal{S}}(s) \xrightarrow{h_{\mathcal{A}}(a)}_{\mathcal{T}_1} h_{\mathcal{S}}(s') \quad \text{and}$$

$$h_{\mathcal{T}}(s \xrightarrow{\tau}_{\mathcal{T}_0} s') = h_{\mathcal{S}}(s) \xrightarrow{\tau}_{\mathcal{T}_1} h_{\mathcal{S}}(s') \ \text{ if } \quad h_{\mathcal{S}}(s) \neq h_{\mathcal{S}}(s')$$

(hence $h_{\mathcal{T}}$ is determined by $h_{\mathcal{S}}$ and $h_{\mathcal{A}}$) satisfying the following conditions

1. for each $t \xrightarrow{a_1}_{\mathcal{T}_1} t' \in h_{\mathcal{T}}(\mathcal{T}_0)$ and each s such that $h_{\mathcal{S}}(s) = t$, there exists $s \xrightarrow{a_0}_{\mathcal{T}_0} s' \in \mathcal{T}_0$ such that $h_{\mathcal{S}}(s') = t'$ and $h_{\mathcal{A}}(a_0) = a_1$,
2. for each $t \xrightarrow{\tau}_{\mathcal{T}_1} t' \in h_{\mathcal{T}}(\mathcal{T}_0)$ with $t \neq t'$ and each s such that $h_{\mathcal{S}}(s) = t$, there exists $s \xrightarrow{\tau}_{\mathcal{T}_0} s' \in \mathcal{T}_0$ such that $h_{\mathcal{S}}(s') = t'$. ■

Notation For convenience and where there is no confusion, I will refer to bisimulation homomorphisms as single functions, such as h.

This differs from Arnold and Castellani's definition of a transition system homomorphism with zig-zag condition (given as conditions 1 and 2 in the definition above) [AC96], in that here I do not assume that the two transition systems have the same set of labels, and introduce a map between the two different label sets.

Note that this definition ignores τ loops in the image by only requiring a τ transition in the image when the end points are different—this also means that the image does not need to have a τ transition when the endpoints are the same. The intuition behind this is that this function is to have similar properties to bisimulation and hence certain (but not all) τ actions can be ignored. The second condition ensures that only the correct τ actions can be ignored. Moreover, the condition that $t \neq t'$ in the second point of the definition ensures that τ loops in the image do not have to be matched by τ transitions in the domain.

I am now interested in relating the above definition to weak bisimulation, with respect to both the identity relation and to other relations over the set of labels.

As I will be working with a definition of bisimulation which requires that labels be related by a relation rather than matching, I first need to consider how a bisimulation homomorphism will interact with this relation.

Definition 3.3.5 (Consistency)
Given two sets $\mathcal{A}_0$ and $\mathcal{A}_1$, a function $f : \mathcal{A}_0 \to \mathcal{A}_1$ and relations $\mathcal{B}_i \subseteq \mathcal{A}_i \times \mathcal{A}_i$ for $i = 0, 1$. Then f is *consistent with* $\mathcal{B}_0$ and $\mathcal{B}_1$ if for $a, a' \in \mathcal{A}_0$,

$$a\, \mathcal{B}_0\, a' \Rightarrow f(a)\, \mathcal{B}_1\, f(a') \qquad \blacksquare$$

Proposition 3.3.3 (Bisimulation homomorphisms and generalised bisimulation)
Let $\mathcal{L}_i = (\mathcal{S}_i, \mathcal{A}_i, \mathcal{T}_i)$ for $i = 0, 1$ be two LTSs with $\mathcal{B}_i$ a binary relation over $\mathcal{A}_i$ for $i = 0, 1$. If $s \approx_{\mathcal{B}_0} s'$ in the LTS $(\mathcal{S}_0, \mathcal{A}_0, \mathcal{T}_0)$, and if there exists a bisimulation homomorphism $h : (\mathcal{S}_0, \mathcal{A}_0, \mathcal{T}_0) \to (\mathcal{S}_1, \mathcal{A}_1, \mathcal{T}_1)$ such that $h : \mathcal{A}_0 \to \mathcal{A}_1$ is consistent with $\mathcal{B}_0$ and $\mathcal{B}_1$, then $h(s) \approx_{\mathcal{B}_1} h(s')$ in $(h(\mathcal{S}_0), \mathcal{A}_1, h(\mathcal{T}_0))$.

Proof: Since $s \approx_{\mathcal{B}_0} s'$, there exists a generalised bisimulation $\mathcal{R}_0 \subseteq \mathcal{S}_0 \times \mathcal{S}_0$ such that $(s, s') \in \mathcal{R}_0$. Define $\mathcal{R}_1 = \{(s_1, s'_1) | \exists (s_0, s'_0) \in \mathcal{R}_0$ such that $h(s_0) = s_1$ and $h(s'_0) = s'_1\}$. I wish to show that $\mathcal{R}_1$ is a bisimulation. Consider $(s_1, s'_1) \in \mathcal{R}_1$. From the definition of $\mathcal{R}_1$, there exists $(s_0, s'_0) \in \mathcal{R}_0$ such that $h(s_0) = s_1$ and $h(s'_0) = s'_1$. There are two cases.

1. Suppose that $s_1 \xrightarrow{a_1} t_1$. Now since $s_1 = h(s_0)$ for some $s_0 \in \mathcal{S}_0$, and from condition 1 in Definition 3.3.4, there exists $s_0 \overset{a_0}{\Longrightarrow} t_0 \in \mathcal{T}_0$ such that $h(t_0) = t_1$ and $h(a_0) = a_1$. Now since $\mathcal{R}_0$ is a bisimulation, it can easily be shown that $s'_0 \overset{a'_0}{\Longrightarrow} t'_0$ for some $a'_0 \in \mathcal{A}_0$ and $t'_0 \in \mathcal{S}_0$ with $(a_0, a'_0) \in \mathcal{B}_0$ and $(t_0, t'_0) \in \mathcal{R}_0$. Furthermore, $h(s'_0) \overset{h(a'_0)}{\Longrightarrow} h(t'_0)$ and since h is consistent with $\mathcal{B}_0$ and $\mathcal{B}_1$, it is clear that $h(a_0)\, \mathcal{B}_1\, h(a'_0)$. But since $h(s'_0) = s'_1$, $s'_1 \overset{h(a'_0)}{\Longrightarrow} h(t'_0)$ and $(t_1, h(t'_0)) \in \mathcal{R}_1$ as required.

2. Suppose that $s_1 \xrightarrow{\tau} t_1$. First consider when $s_1 \neq t_1$. Now since $s_1 = h(s_0)$ for some $s_0 \in \mathcal{S}_0$, and from condition 2 in Definition 3.3.4, there exists $s_0 \overset{\tau}{\Longrightarrow} t_0 \in \mathcal{T}_0$ such that $h(t_0) = t_1$. Now since $\mathcal{R}_0$ is a bisimulation, it can easily be shown that $s'_0 \Longrightarrow t'_0$ for some t'_0 and $(t_0, t'_0) \in \mathcal{R}_1$. Furthermore, it can also be shown that $h(s'_0) \Longrightarrow h(t'_0)$. Hence, $h(s'_0) = s'_1$, therefore $s'_1 \Longrightarrow h(t'_0)$ and $(t_1, h(t'_0)) \in \mathcal{R}_1$ as required. Next consider when $s_1 = t_1$, hence $s_1 \xrightarrow{\tau} s_1$. Also $s'_1 \Longrightarrow s'_1$ and $(s_1, s'_1) \in \mathcal{R}_1$ as required.

The symmetric conditions can be proved by similar arguments. ■

Because of the fact that bisimulation homomorphisms are functions, the converse (given two pairs of equivalent states in different transition systems and a function between the labels) does not hold. For example, if the two states in the first transition system are s_1 and s_2 with the transitions $s_1 \xrightarrow{a} s'_1$ and $s_2 \xrightarrow{a} s'_2$, then clearly s_1 and s_2 are equivalent. Moreover if the states in the second transition system are t_1 and t_2 with the transitions $t_1 \xrightarrow{b} t'_1$, $t_2 \xrightarrow{b} t'_2$ and $t_2 \xrightarrow{b} t''_2$, then clearly t_1 and t_2 are equivalent. Suppose the function $f(a) = b$ is given, then I have the conditions for the converse of the theorem. But because I am trying to find a bisimulation homomorphism (which is a function), I cannot find a satisfactory way to map from the transition $s_2 \xrightarrow{a} s'_2$ to the transitions $t_2 \xrightarrow{b} t'_2$ and $t_2 \xrightarrow{b} t''_2$.

An obvious candidate for the relation on $\mathcal{A}_1$ in the above is the one induced by h, namely for $a_1, a'_1 \in \mathcal{A}_1$, $a_1 \, \mathcal{B}_1 \, a'_1$ if

$$\exists a_0, a'_0 \in \mathcal{A}_0, h(a_0) = a_1, h(a'_0) = a'_1 \text{ and } a_0 \, \mathcal{B}_0 \, a'_0.$$

Proposition 3.3.3 describes the relationship between generalised bisimulations between states from the same LTS. The next result shows how to obtain a generalised bisimulation between states in different LTSs.

Proposition 3.3.4 (Bisimulation homomorphism defines a generalised bisimulation)

Let $\mathcal{L}_i = (\mathcal{S}_i, \mathcal{A}_i, \mathcal{T}_i)$ for $i = 0, 1$ be two LTSs. If there exists a bisimulation homomorphism $h : (\mathcal{S}_0, \mathcal{A}_0, \mathcal{T}_0) \to (\mathcal{S}_1, \mathcal{A}_1, \mathcal{T}_1)$, then for all $s \in \mathcal{S}_0$,

$$s \approx_h h(s).$$

Proof: Define a relation $\mathcal{R}$ over $\mathcal{S}_0 \times \mathcal{S}_1$ as $\mathcal{R} = \{(s, h(s)) \mid s \in \mathcal{S}_0\}$. I wish to show that this is a generalised bisimulation with respect to the function $h : \mathcal{A}_0 \to \mathcal{A}_1$ (viewed as a binary relation). Consider $(s_0, h(s_0)) \in \mathcal{R}$; there are four cases.

- If $s_0 \xrightarrow{a_0} t_0$ for $a_0 \in \mathcal{A}_0$ and $t_0 \in \mathcal{S}_0$, then $h(s_0) \xrightarrow{h(a_0)} h(t_0)$ and clearly $(a_0, h(a_0)) \in h$ and $(t_0, h(t_0)) \in \mathcal{R}$ as required.

- If $s_0 \xrightarrow{\tau} t_0$ for $t_0 \in \mathcal{S}_0$, then either $h(s_0) = h(t_0)$ and hence $h(s_0) \Longrightarrow h(t_0)$ or $h(s_0) \xrightarrow{\tau} h(t_0)$ and also $h(s_0) \Longrightarrow h(t_0)$. Moreover, in both cases $(t_0, h(t_0)) \in \mathcal{R}$ as required.

- If $h(s_0) \xrightarrow{a_1} t_1$ with $h(s_0) \neq t_1$ for $a_1 \in \mathcal{A}_1$ and $t_1 \in \mathcal{S}_1$ then since h is a bisimulation homomorphism, there exists $a_0 \in A_0$ and $t_0 \in \mathcal{S}_0$ such that $h(a_0) = a_1$, $h(t_0) = t_1$ and $s_0 \xrightarrow{a_0} t_0$. Clearly $(a_0, a_1) \in h$ and $(t_0, t_1) \in \mathcal{R}$ as required.

- If $h(s_0) \xrightarrow{\tau} t_1$ for $t_1 \in \mathcal{S}_1$ and $h(s_0) \neq t_1$ then since h is a bisimulation homomorphism, there exists $t_0 \in \mathcal{S}_0$ such that $h(t_0) = t_1$ and $s_0 \xrightarrow{\tau} t_0$, and clearly $(t_0, t_1) \in \mathcal{R}$ as required. If $h(s_0) = t_1$, then $h(s_0) \xrightarrow{\tau} h(s_0)$ and also $s_0 \Longrightarrow s_0$ and $(s_0, h(s_0)) \in \mathcal{R}$ as required. ■

Note that the surjectivity of $h_{\mathcal{T}}$ is not needed for this result, since these results hold for the transition system that is reachable from $h(\mathcal{S})$ and because I only work with the transition system that is the image of the homomorphism. A different approach can be taken by requiring that $h(\mathcal{T}_0) = \mathcal{T}_1$.

Arnold and Dicky [AD89] give two results that relate bisimulation and homomorphisms

- transition systems are (strongly) bisimilar if and only if they are bisimulation homomorphic images of a common transition system,

- states of a transition system are (strongly) bisimilar if and only if they have a common image by a bisimulation homomorphism.

I now investigate whether it is possible to obtain general results for weak generalised bisimulation along similar lines.

Proposition 3.3.5 (States that are images of a common state are bisimilar)
Let $\mathcal{L}_i = (\mathcal{S}_i, \mathcal{A}_i, \mathcal{T}_i)$ for $i = 0, 1$ be two LTSs with $\mathcal{B}$ a binary relation over $\mathcal{A}_0 \times \mathcal{A}_1$. Given two states $s_i \in S_i$ for $i = 0, 1$, then $s_0 \approx_{\mathcal{B}} s_1$ if there exists a transition system $\mathcal{L} = (\mathcal{S}, \mathcal{A}, \mathcal{T})$ and two bisimulation homomorphisms $h_i : \mathcal{L} \to \mathcal{L}_i$ for $i = 0, 1$ such that s_0 and s_1 are the images under h_0 and h_1 respectively of a state $s \in \mathcal{S}$, and for any $a \in \mathcal{A}$, $h_0(a) \, \mathcal{B} \, h_1(a)$.

Proof:

Let $\mathcal{R} = \{(s_0', s_1') \mid \exists s' \in S, h_0(s') = s_0' \text{ and } h_1(s') = s_1'\}$. I need to show that this is a generalised weak bisimulation with respect to $\mathcal{B}$. Let $(s_0', s_1') \in \mathcal{R}$ and consider $s_0' \xrightarrow{a_0} s_0''$. There exists $s' \in S$ such that $h_0(s') = s_0'$ (and also $h_1(s') = s_1'$), therefore there exists s'' and a such that $s' \xrightarrow{a} s''$, $h_0(s'') = s_0''$ and $h_0(a) = a_0$. Consider $h_1(s') \xrightarrow{h_1(a)} h_1(s'')$, namely $s_1' \xrightarrow{h_1(a)} h_1(s'')$. Hence $a_0 \, \mathcal{B} \, h_1(a)$ and $(s_0'', h_1(s'')) \in \mathcal{R}$. Next, consider $s_0' \xrightarrow{\tau} s_0''$ with $s_0' \neq s_0''$. By a similar argument, there exists s'' such that $s' \xrightarrow{\tau} s''$ and $h_0(s'') = s_0''$, and hence $s_1' \xrightarrow{\tau} h_1(s'')$, with $(s_0'', h_0(s'')) \in \mathcal{R}$ as required. If $s_0' = s_0''$ then $s_0' \xrightarrow{\tau} s_0'$, and also $s_1' \Longrightarrow s_1'$ and $(s_0', s_1') \in \mathcal{R}$ as required. The other conditions for the bisimulation is shown in a similar way. ■

Note that the condition on the elements of $\mathcal{A}_0$ and $\mathcal{A}_1$ could have been made more specific to include only those that occur in the sub-transition system; however it does not seem necessary here. Arnold and Dicky's original result related to whole transition systems; here I prefer to deal with states, but the proposition can be generalised by considering a collection of states and requiring that all states in $\mathcal{L}_0$ and $\mathcal{L}_1$ fall into the image of $\mathcal{L}$.

The question arises as to whether the converse of this theorem holds. An obvious approach is to use a product construction and projection functions. Note, however, that for the construction and projection functions to fit the definitions of eLTS and bisimulation homomorphism, they must treat τ actions in a particular manner, and hence the standard construction cannot be used. The following example illustrates the difficulties with this approach.

$$\begin{tikzcd}
 & & s_0 \arrow[dl, "a_0"'] \arrow[dr, "a_0"] & & & & t_0 \arrow[d, "b_0"'] & \\
 & s_1 \arrow[dl, "a_1"'] \arrow[dr, "\tau"] & & s'_1 & & & t_1 \arrow[dl, "b_1"'] \arrow[dr, "\tau"] & \\
s_2 & & s_3 & & & t_2 & & t_3
\end{tikzcd}$$

Figure 3.2: Examples of transition systems

Consider the two transition systems in Figure 3.2 and assume that $a_0 \mathcal{B} b_0$, $a_1 \mathcal{B} b_1$ and there are no other elements of $\mathcal{B}$. Then $s_0 \approx_{\mathcal{B}} t_0$—consider the bisimulation $\{(s_0, t_0), (s_1, t_1), (s_2, t_2), (s_3, t_3), (s'_1, t_3)\}$. It appears reasonable to use this bisimulation to define the states of the product transition system. However the state (s'_1, t_3) is problematic and it cannot be omitted. If there is a transition $(s_0, t_0) \xrightarrow{(a_0,b_0)} (s'_1, t_3)$ then the projection function from the product to the second transition system maps this transition to $t_0 \xrightarrow{b_0} t_3$ which does not exist. However, to omit the transition in the product means there is no transition to map onto $s_0 \xrightarrow{a_0} s'_1$. The definition of bisimulation homomorphism does not permit the removal of transitions from the range, and it is clear that the transition in the product cannot not be mapped to $t_0 \xrightarrow{b_0} t_1$. This situation occurs because of the manner in which weak bisimulation is defined. Since $s_0 \approx_{\mathcal{B}} t_0$, $s_0 \xrightarrow{a_0} s'_1$ is matched by $t_0 \xrightarrow{b_0} t_1 \xrightarrow{\tau} t_3$ with $s'_1 \approx_{\mathcal{B}} t_3$. However, it is not the case that $s'_1 \approx_{\mathcal{B}} t_1$. It is not clear whether this can be resolved by looking for a different LTS and bisimulation homomorphisms. I now look at a generalisation of the second result.

Proposition 3.3.6 (States that have a common image are bisimilar)
Let $\mathcal{L}_i = (\mathcal{S}_i, \mathcal{A}_i, \mathcal{T}_i)$ for $i = 0, 1$ be two LTSs with $\mathcal{B}$ a binary relation over $\mathcal{A}_0 \times \mathcal{A}_1$. Given two states $s_i \in S_i$ for $i = 0, 1$, then $s_0 \approx_{\mathcal{B}} s_1$ if there exist a transition system $\mathcal{L} = (\mathcal{S}, \mathcal{A}, \mathcal{T})$ and two bisimulation homomorphisms $h_i : \mathcal{L}_i \to \mathcal{L}$ for $i = 0, 1$ such that $h_0(s_0) = h_1(s_1)$, and for $a_0 \in \mathcal{A}_0$ and $a_1 \in \mathcal{A}_1$, $h_0(a_0) = h_1(a_1)$ implies $a_0 \, \mathcal{B} \, a_1$.

Proof: Define $\mathcal{R} = \{(s_0', s_1') \mid h_0(s_0') = h_1(s_1')\}$. I need to show that this is a generalised weak bisimulation with respect to $\mathcal{B}$. Consider $s_0' \xrightarrow{a_0} s_0''$, then $h_0(s_0') \xrightarrow{h_0(a_0)} h_0(s_0'')$, namely $h_1(s_1') \xrightarrow{h_0(a_0)} h_0(s_0'')$. By the definition of bisimulation homomorphism, there exist a_1 and s_1'' such that $s_1' \xrightarrow{a_1} s_1''$, $h_1(a_1) = h_0(a_0)$ and $h_1(s_1'') = h_0(s_0'')$. Hence $a_0 \, \mathcal{B} \, a_1$ and $(s_0'', s_1'') \in \mathcal{R}$. Next consider $s_0' \xrightarrow{\tau} s_0''$. There are two cases. Firstly if $h_0(s_0') = h_0(s_0'')$, then $s_1' \Longrightarrow s_1'$, and $h_0(s_0'') = h_1(s_1')$ as required. Next, if $h_0(s_0') \xrightarrow{\tau} h_0(s_0'')$ then by a similar argument to above, I can show that $s_1' \xrightarrow{\tau} s_1''$ with $(s_0'', s_1'') \in \mathcal{R}$. The other conditions for bisimulation can be shown by a symmetric argument. ∎

Since $\mathcal{B}$ is an arbitrary relation in the above proposition, I need to the additional condition on h_0 and h_1 that $h_0(a_0) = h_1(a_1)$ implies that $a_0 \, \mathcal{B} \, a_1$ to obtain the fact that it is a bisimulation with respect to $\mathcal{B}$. If I wish to prove the converse of this above proposition, including the relationship between $\mathcal{B}$ and the bisimulation homomorphisms, I require an additional condition on $\mathcal{B}$.

Definition 3.3.6 (Separation property)
Let $\mathcal{B}$ be a binary relation over $\mathcal{A}_0 \times \mathcal{A}_1$. Let $\overline{\mathcal{B}}$ be the reflexive, symmetric and transitive closure of $\mathcal{B}$ over the disjoint union of $\mathcal{A}_0$ and $\mathcal{A}_1$ ($\mathcal{A}_0 \uplus \mathcal{A}_1$). Then $\mathcal{B}$ has the *separation property* if for $a_0 \in \mathcal{A}_0$ and $a_1 \in \mathcal{A}_1$,

$$a_0 \, \overline{\mathcal{B}} \, a_1 \Rightarrow a_0 \, \mathcal{B} \, a_1$$ ∎

Examples of relations with this property are

- $\mathcal{B} \subseteq \mathcal{A} \times \mathcal{A}$ where $\mathcal{B}$ is an equivalence relation,
- $\mathcal{B} \subseteq \mathcal{A}_0 \times \mathcal{A}_1$ where $\mathcal{B}$ is an injective function.

Proposition 3.3.7 (States that are bisimilar have a common image) Let $\mathcal{L}_i = (\mathcal{S}_i, \mathcal{A}_i, \mathcal{T}_i)$ for $i = 0, 1$ be two LTSs with $\mathcal{B}$ a binary relation over $\mathcal{A}_0 \times \mathcal{A}_1$ having the separation property. Given two states $s_i \in S_i$ for $i = 0, 1$, then if $s_0 \approx_{\mathcal{B}} s_1$ there exist a transition system $\mathcal{L} = (\mathcal{S}, \mathcal{A}, \mathcal{T})$ and two bisimulation homomorphisms $h_i : \mathcal{L}_i \to \mathcal{L}$ for $i = 0, 1$ such that $h_0(s_0) = h_1(s_1)$, and for $a_0 \in \mathcal{A}_0$ and $a_1 \in \mathcal{A}_1$, $h_0(a_0) = h_1(a_1)$ implies $a_0 \, \mathcal{B} \, a_1$.

Proof: I wish to construct a transition system and two bisimulation homomorphisms. I work with the disjoint unions $\mathcal{S}_0 \uplus \mathcal{S}_1$ and $\mathcal{A}_0 \uplus \mathcal{A}_1$. First, let $\overline{\mathcal{B}}$ be the reflexive, symmetric and transitive closure of $\mathcal{B}$ over $\mathcal{A}_0 \uplus \mathcal{A}_1$. Since $\overline{\mathcal{B}}$ is an equivalence relation, $\approx_{\overline{\mathcal{B}}}$ is an equivalence over $\mathcal{S}_0 \uplus \mathcal{S}_1$ with respect to the transitions $\mathcal{T}' = \{s \xrightarrow{a} s' \mid s \xrightarrow{a} s' \in \mathcal{T}_0 \text{ or } s \xrightarrow{a} s' \in \mathcal{T}_1\} \cup \{s \xrightarrow{\tau} s' \mid s \xrightarrow{\tau} s' \in \mathcal{T}_0 \text{ or } s \xrightarrow{\tau} s' \in \mathcal{T}_1\}$ by Proposition 3.3.2. Consider the following transition system $\mathcal{L}$

$$\begin{aligned}
\mathcal{S} &= (\mathcal{S}_0 \uplus \mathcal{S}_1)/{\approx_{\overline{\mathcal{B}}}} \\
\mathcal{A} &= (\mathcal{A}_0 \uplus \mathcal{A}_1)/\overline{\mathcal{B}} \\
\mathcal{T} &= \{S \xrightarrow{A} S' \mid S, S' \in \mathcal{S}, A \in \mathcal{A}, \exists s \in S, s' \in S', a \in A \text{ such that} \\
&\quad\ s \xrightarrow{a} s' \in \mathcal{T}'\} \cup \{S \xrightarrow{\tau} S' \mid S, S' \in \mathcal{S}, \exists s \in S, s' \in S', \text{ such that} \\
&\quad\ s \xrightarrow{\tau} s' \in \mathcal{T}'\}.
\end{aligned}$$

For $i = 0, 1$ and $s, s' \in \mathcal{S}_i$, $a \in \mathcal{A}_i$, define the following homomorphisms

$$\begin{aligned}
h_i(s) &= [s]_{\approx_{\overline{\mathcal{B}}}} \\
h_i(a) &= [a]_{\overline{\mathcal{B}}} \\
h_i(s \xrightarrow{a} s') &= h_i(s) \xrightarrow{h_i(a)} h_i(s') \\
h_i(s \xrightarrow{\tau} s') &= h_i(s) \xrightarrow{\tau} h_i(s') \text{ if } h_i(s) \neq h_i(s').
\end{aligned}$$

I now need to show that they are bisimulation homomorphisms. Consider $S \xrightarrow{A} S'$ with $S = [s_0]_{\approx_{\overline{\mathcal{B}}}}$ for some $s_0 \in \mathcal{S}_0$. I need to find $s_0' \in \mathcal{S}_0$ and $a_0 \in \mathcal{A}_0$ such that $[s_0']_{\approx_{\overline{\mathcal{B}}}} = S'$ and $[a_0]_{\overline{\mathcal{B}}} = A$. By definition of $\mathcal{T}$, there exists $a \in A$ and $s \in S'$ such that $s_0 \xrightarrow{a} s \in \mathcal{T}'$, but by definition of $\mathcal{T}'$, $a \in \mathcal{A}_0$ and $s \in \mathcal{S}_0$, as required. The condition for τ transitions can be shown in a similar way, and a similar proof can be used to show that h_1 is a bisimulation homomorphism. Finally I need to show that given $a_0 \in \mathcal{A}_0$ and $a_1 \in \mathcal{A}_1$, $h_0(a_0) = h_1(a_1)$ implies $a_0 \, \mathcal{B} \, a_1$, Clearly, if $h_0(a_0) = h_1(a_1)$ then $a_0 \, \overline{\mathcal{B}} \, a_1$. However, since $\overline{\mathcal{B}}$ has the separation property, $a_0 \, \mathcal{B} \, a_1$. ■

This result is more general than Arnold and Dicky's, both because it deals with different label sets, and because it considers two transition systems instead of a single one. Note that Proposition 3.3.4 is a special case of this proposition, with $\mathcal{L} = \mathcal{L}_0$, $h_0 = id$, $h_1 = h$, and $\mathcal{B} = h$.

3.3.3 Discussion

My general aim in this thesis is to compare semantic equivalences. In this section, I will discuss the difficulties that occur when trying to use the results of Section 3.3 to effect a comparison of semantic equivalences. These results relate to labelled transition systems, which are the objects upon which semantic equivalences are defined in process algebras (within the context of this document). The results do not involve any notion of process algebra syntax, hence the term syntax-free. I first look at two possible approaches, and then will discuss the second approach in more detail.

- In the comparison over extensions to CCS given in the first part of this chapter, I am comparing the equivalences over the same states, i.e. over states with the same syntactic form, so given two equivalences, $\approx_1$ and $\approx_2$, $\approx_1 \subseteq \approx_2$ is equivalent to the statement that for all p and q, $p \approx_1 q$ implies $p \approx_2 q$, or that $\approx_1$ equates fewer states than $\approx_2$. It is not clear here how to chose the states that one wants to compare the equivalence over. A possible approach when comparing two equivalences defined by relations $\mathcal{B}_0$ over $\mathcal{A}_0$ and $\mathcal{B}_1$ over $\mathcal{A}_1$ on two LTSs $\mathcal{L}_i = (\mathcal{S}_i, \mathcal{A}_i, \mathcal{T}_i)$ for $i = 0, 1$, is to define an relation over $\mathcal{S}_0 \times \mathcal{S}_1$ which equates the states are to be considered equivalent. Then I can use the following definition:

  Let Φ be a relation over $\mathcal{S}_0 \times \mathcal{S}_1$. Then $\approx_{\mathcal{B}_0} \subseteq_\Phi \approx_{\mathcal{B}_1}$ if for all $(s_0, s_1), (s_0', s_1') \in \Phi$, $s_0 \approx_{\mathcal{B}_0} s_0' \Rightarrow s_1 \approx_{\mathcal{B}_1} s_1'$

This definition can be written informally as follows: if two states are equated in the first transition system by the first equivalence, then any Φ-related pair of states must be equated in the second transition system by the second equivalence. Note that this involves LTSs where the equivalence is defined within the LTS, and not between different LTSs. The definition could be extended to allow this, although this would add complexity.

- The approach taken by van Glabbeek to comparing equivalence [vG90b] involves using LTSs with the standard atomic label set. He doesn't concern himself with process algebra syntax—counter-examples are given by means of process graphs. This an approach that I will discuss in more detail.

For the rest of this section, I will assume that the semantic equivalences under consideration are based on bisimulation within an LTS that involves the identity of labels on transitions.

First assume that there is a notion of canonical LTS—by this I refer to a manner in constructing an LTS for a given label set that is in some sense as complete as possible, namely such that the equivalence classes induced by the expected definition of bisimulation for the label set cover as many different processes as is possible. As I have not pursued this research at a detailed level, I will leave this concept somewhat vague. Hence, although it is not possible to compare equivalences over LTSs based on the same label set as in van Glabbeek's work, it may be possible to compare between canonical LTSs. An obvious tool to consider using is a bisimulation homomorphism. If the two LTSs are formed in a similar way which one would expect as they are both canonical LTSs, and there is suitable function between label sets, then it seems it should be possible to define a surjective bisimulation homomorphism between the LTSs. The crucial question then relates to what function should be used to map between the labels—ideally it should lose as little information as possible—for example, in the LTS for locations [BCHK93, BCHK94], each transition is labelled with a string of locations, and hence the ordering given by the string can be viewed as information that should be retained.

As an example of some of the difficulties involved, consider canonical LTSs for CCS with locations [BCHK94] and CCS with local/global causes [Kie94]. In the first case, each transition of the LTS will be labelled with an action and a string of location; and in the second, each transition will be labelled with an action and two sets of causes and a cause (See Section 2.2.1.2 for more details). I will ignore τ transitions for the purposes of this discussion. Assume that the set of locations and causes are the same (or that there is some bijective map between them). It is not clear how a map can be constructed from the labels of one transition system to the labels of the other. Taking an approach whereby the action is mapped to the action and any string of locations is mapped to the empty set, the empty set, and the first (or last) location in the string, does not fit well with the principle of retaining as much information as possible. A better approach may be to map the string of locations to the set of of the elements of the string, the set of the elements of the string and the last location in the string.

However, neither of these mappings take into account the implied semantics of the local/global cause labels, namely that the first set is a set of global cause and the second is a set of local causes. To successfully map from the two label sets, one requires an algebra to describe how these labels can be built up, and one must ensure that the map is a homomorphism with respect to the operators. I investigated some work in this direction, but it has not been particularly fruitful.

Another complication occurs when considering the LTS created by a particular SOS. Looking at the same example, when considering a comparison using algebras to reflect the way labels are constructed, it seems that the two label sets are not comparable—in the location labels, there is no constructor for anything similar to global causes, so it is not clear how to map from the string of locations to the global cause set; and for the local/global cause labels, there is no notion of order because only sets are used and hence there is no obvious way to map from the local cause set to the string of locations. However, looking at the way the labels are generated by the SOS, I end with a situation (when working with the pure CCS terms) that $\approx_{lg} \subset \approx_{l}$, since the way the labels appear in the LTS are constrained by the SOS, and the local/global cause labels actually contain more information than the location labels. This can be explained informally by the fact that in CCS with locations, any location can be used to label an action and hence the ordering of locations is not used differentiate between processes.

The above example illustrates that comparing over canonical LTSs may give different results to comparing over the LTSs generated by SOS. Hence it is not clear how to interpret results that could come out of this approach. Is it acceptable that the results from the comparison over the canonical LTSs don't coincide with those for comparison over pure CCS terms (or other process algebra based comparisons), or more generally over LTSs with different characteristics to the canonical LTSs, and it is possible to come up with formal explanations for why this is this case?

Another approach then is to perform the comparison over the LTSs generated by SOS instead of the canonical LTSs. This approach still uses the canonical LTSs as a basis for the comparison, so the problems described above still need to be dealt with. A suitable bisimulation homomorphism is chosen to map between the two canonical LTSs with a suitable map between the label sets. Assuming the semantic equivalence under consideration is an equivalence relation, then the states of each canonical LTS can be partitioned by the equivalence. Moreover, it should be possible to show that there is an function on the equivalence classes induced by the bisimulation homomorphism that is surjective. Next, consider the equivalence classes induced by the semantic equivalences over the specific LTSs. Each equivalence class from the set of states of a specific LTS can be mapped into the equivalence class of the relevant canonical LTSs such that the elements of the equivalence classes are bisimilar (this should be possible if the definition of canonical LTS is correct). This map should be injective since in each case the same equivalence is being used. It may then be possible to induce a function from the equivalence classes of one specific LTS to the equivalence classes of the other, and hence then possible to do a comparison of the semantic equivalences by investigating which equivalence classes from the specific LTSs are mapped to each other by this construction. The interpretation of these results is still unclear, however. If there is a relation Φ over the states of the two specified

LTSs, it is possible to check whether Φ is satisfied. However, this is dependent on the function used to map between the label sets and it is unclear how to ensure that this is satisfactory.

As an example, consider the two CCS extensions with locations by Boudol *et al.* [BCHK93, BCHK94]. They have slightly different approaches for how strings of locations are added to transitions—in the one case, only one location can be added per action, whereas in the other case, a string of locations can be added (including the empty string). However in both case, the bisimulation definition is the same (if in the case of parameterised bisimulation, the assumption is made that the relation is the identity relation) in that they require matching on actions and strings of locations. For this example, I will assume that these notions of bisimulation are identical (this assumption is open to question) and hence it is necessary only to work with one canonical LTS which has transitions labelled with actions and strings of locations (I will ignore τ transitions for the purposes of this discussion). Then I need to consider the two specific transition systems, one which is generated by the SOS of the 'strict' locations and the one generated by the 'loose' locations. In this case, I will assume that Φ is the relation that relates states with the same process names. Considering the processes $P = (a.c \mid \overline{c}.b)\backslash c$ and $Q = (a.(c+b) \mid \overline{c}.b)\backslash c$ [BCHK94]. Since $P \not\approx_l Q$ and $P \approx_{ll} Q$, one would expect that P and Q to be in different equivalence classes in the strict location LTS and to be in the same class in the loose location LTS, and that the map on equivalence classes will map both $[P]_{\approx_l}$ and $[Q]_{\approx_l}$ to $[P]_{\approx_{ll}}$ $(= [Q]_{\approx_{ll}})$. It is not clear how to interpret the results if this does not hold—a possible consideration is to review the assumption that it is possible to use the same canonical LTS.

As can be seen from the above, the details of this approach have not been formalised and the benefits of this approach are not clear. Additional complexity is introduced if the equivalences to be compared do not require exact matches of labels. Hence, I have chosen to look at a syntactic approach to comparison. However, in the course of this exploration, I have developed extensions of results relating to bisimulation homomorphism that cater for labelled transitions system with differing sets of labels.

3.4 Conclusion

In this chapter, I have considered two approaches to the comparison of process algebra equivalences. Neither are entirely satisfactory; the first relies on an *ad hoc* approach to comparison; and the second abstracts away from the syntax of the process algebra in a manner that makes it difficult to determine how to do the comparison. I have, however, in second section of the chapter, introduced some results about LTSs that involve a more general notion of bisimulation. In the next two chapters, I look at a method of comparison that is based on the operational semantics of process algebras.

Chapter 4

A new format

4.1 Introduction

In this chapter, I start to look at how an understanding of the syntax of process algebras can be used to compare equivalences over process algebras. As was seen in the previous chapter, there is a limit to what can be achieved without considering the syntax. In this chapter, I take existing work on formats and extend it by considering structured labels. I then show under which conditions congruence with respect to strong bisimulation can be obtained.

I will first start with a number of standard definitions for many-sorted signatures and algebras, so that I can fix the notation which will be used in the rest of the document. Then in Section 4.3, I will present justification for why a new format is required and then present the new format—this involves working with a specific kind of many-sorted signature. I then extend a number of results to the new format. As the new format is somewhat more complex due to the fact that labels of transitions are dealt with syntactically instead of schematically, I require some additional conditions to ensure that congruence holds. I end the chapter with counter-examples to show that most of the conditions cannot be relaxed without losing congruence, and discuss the situation with the conditions for which I have no counter-examples.

4.2 Definitions

4.2.1 Many-sorted signatures and algebras

Definition 4.2.1 (Sorted set)

For any set S, an *S-sorted set* A is a family $\{A_s\}_{s \in S}$ of sets indexed by S. ■

Consider two S-sorted sets A and B. Intersection, union, difference and subset are defined component-wise. For example, $A \cap B = \{A_s \cap B_s\}_{s \in S}$. The union of sorted sets over different index sets is given in Definition 6.2.1 in Chapter 6.

Definition 4.2.2 (Signature)

A *signature* Σ is a pair (S, F) where S is a set of *sorts* and F is a set of *function symbols* such that F is equipped with a mapping $type : F \rightarrow S^* \times S$. If $type(f) = (\epsilon, s)$ for some $s \in S$, then f is called a *constant symbol.* I write $f : w \rightarrow s$ for $f \in F$ with $type(f) = (w, s)$, and $f : s_1 \ldots s_n \rightarrow s$ if $w = s_1 \ldots s_n$. If $w = \epsilon$ is the empty string, then I write $f :\rightarrow s$. ■

In some texts, F is viewed as an $S^* \times S$-sorted set, which allows 'overloading' of function symbols. I will not go into this in detail here.

Let V be an S-sorted set of variables disjoint from F. The set of terms over V can be formed.

Definition 4.2.3 (Open and closed terms)

Let $\Sigma = (S, F)$ be a signature, and 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 $x \in W_s$ of sort s and every constant symbol $f \rightarrow s \in F$,
- every $f(t_1, \ldots, t_n)$ where $f : s_1 \ldots s_n \rightarrow s$ 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 $T(\Sigma, W)_{s_i}$.

I use $T(\Sigma, W)$ to denote the S-sorted set $\{T(\Sigma, W)_s\}_{s \in S}$ and call this the *set of Σ-terms over W*.

$T(\Sigma, \emptyset)$ is called the set of *closed* or *ground terms*, and is abbreviated $\mathbf{T}(\Sigma)$. $T(\Sigma, \emptyset)_s$ (abbreviated $\mathbf{T}(\Sigma)_s$) is called the set of *closed terms of sort s* or *ground terms of sort s*.

$T(\Sigma, V)$ (abbreviated $\mathbb{T}(\Sigma)$) is called the set of *open terms*, and $T(\Sigma, V)_s$ (abbreviated $\mathbb{T}(\Sigma)_s$) is called the set of *open terms of sort s*. ■

Notation Let $S' \subseteq S$, then $\mathbf{T}(\Sigma)_{S'}$ is the S'-sorted set of closed terms with a sort in S' and $\mathbb{T}(\Sigma)_{S'}$ is the S'-sorted set of open terms with a sort in S'. ■

Definition 4.2.4 (Sensible signature)
Let $\Sigma = (S, F)$ be a signature. Σ is called *sensible* if it admits one ground term for each sort, i.e. for all $s \in S$, $\mathbf{T}(\Sigma)_s \neq \emptyset$. ■

Definition 4.2.5 (Variables contained within a term)
Let $\Sigma = (S, F)$ be a signature, and let $t \in \mathbb{T}(\Sigma)$. I define the variables of sort s in t, $\mathrm{Var}_s(t)$, as follows

$$\mathrm{Var}_s(t) \quad = \quad \begin{cases} \{x\} & \text{if } t = x \text{ and } x \in V_s \\ \mathrm{Var}_s(t_1) \cup \ldots \cup \mathrm{Var}_s(t_n) & \text{if } t = f(t_1, \ldots, t_n), f \in F \\ \emptyset & \textit{otherwise} \end{cases}$$

$\mathrm{Var}(t)$ denotes the S-sorted set $\{\mathrm{Var}_s(t)\}_{s \in S}$. ■

Definition 4.2.6 (Substitution)
Let $\Sigma = (S, F)$ be a signature. 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 by the following definition for $f \in F$ with $f : s_1 \ldots s_n \to s$

$$\sigma(f(t_1, \ldots, t_n)) = f(\sigma(t_1), \ldots, \sigma(t_n)) \text{ for } f \in F,\ t_i \in \mathbb{T}(\Sigma)_{s_i},\ 1 \leqslant i \leqslant n.$$

If σ and ρ are substitutions, then the substitution $\sigma \circ \rho$ is defined by $(\sigma \circ \rho)(x) = \sigma(\rho(x))$. ■

Definition 4.2.7 (Σ-algebra)
Let $\Sigma = (S, F)$ be a signature. 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 4.2.8 (Σ-homomorphism)

Let $\Sigma = (S, F)$ be a signature and 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 each $f \in F$ where $f : s_1 \dots s_n \to s$, and $a_1 \in \mathcal{A}_{s_1}, \dots, a_n \in \mathcal{A}_{s_n}$,

$$h_s(f^{\mathcal{A}}(a_1, \dots, a_n)) = f^{\mathcal{B}}(h_{s_1}(a_1), \dots, h_{s_n}(a_n)). \qquad \blacksquare$$

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 $\mathcal{A}$.

Definition 4.2.9 (Σ-congruence)

Let $\Sigma = (S, F)$ be a signature and 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 $f \in F$ such that $f : s_1 \dots s_n \to s$ and for all $a_i, b_i \in \mathcal{A}_{s_i}$ for $1 \leqslant i \leqslant n$

$$a_i \equiv_{s_i} b_i \ (1 \leqslant i \leqslant n) \Rightarrow f^{\mathcal{A}}(a_1, \dots, a_n) \equiv_s f^{\mathcal{A}}(b_1, \dots, b_n). \qquad \blacksquare$$

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 $t, t' \in \mathbf{T}(\Sigma)$. I will sometimes use the following notation for a signature Σ

$$(\ s_1, s_2, \dots;\ \ f_1, f_2, \dots;\ \ g_1, g_2, \dots\ )$$

where $s_1, s_2, \dots$ is a list of the sorts of S; $f_1, f_2, \dots$ is a list of the functions from F with type $\to s$ for $s \in S$; and $g_1, g_2, \dots$ is a list of the remaining functions from F. A Σ-algebra $\mathcal{A}$ will also be written in a similar fashion

$$(\ \mathcal{A}_{s_1}, \mathcal{A}_{s_2}, \dots;\ \ f_1^{\mathcal{A}}, f_2^{\mathcal{A}}, \dots;\ \ g_1^{\mathcal{A}}, g_2^{\mathcal{A}}, \dots\ ).$$

4.2.2 Labelled transition systems

In the previous chapter, I defined labelled transition systems. I am now interested in labelled transition systems which have a sorted set of labels. The definition can be extended in the obvious way.

Definition 4.2.10 (Sorted labelled transition system)

Let S be a set. An *S-sorted labelled transition system (LTS)* is defined as $\mathcal{L} = (\mathcal{S}, \mathcal{A}, \mathcal{T})$ where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is an S-sorted set of transition labels, and the relation $\mathcal{T} \subseteq \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ describes which transitions occur between states. Generally, I write $s \xrightarrow{a} s'$ for $(s, a, s') \in \mathcal{T}$. ■

4.3 Extended transition system specifications

I wish to use many-sorted signatures and algebras as a way of extending the notion of format. Prior work in formats relies on using a single-sorted signature and the corresponding term algebra to represent the processes, and assumes an atomic set of actions. Moreover, the actions are treated in a different manner to the process terms, since they are treated in a schematic way, namely a rule is understood to represent a number of rules, each with a different label appearing on each transition. This approach is satisfactory for dealing with an atomic action set, but quite soon becomes unsatisfactory when dealing with more complex action sets. The general idea behind this new format is that all components will be dealt with syntactically, both processes and transition labels (actions), and this will be done by using the term algebra of a given signature, Σ. Then the actual labels of the process algebra will be represented as terms in a Σ-algebra. Since there is a unique homomorphism from the term algebra to any Σ-algebra which induces an equivalence on the elements of the term algebra, this equivalence can then be used to match labels in the definition of bisimulation.

Since the aim of formats is to be able to prove theorems about process algebras based on SOS in a syntactic manner, taking the approach I have taken here is a logical extension to the existing notion of format. It introduces some additional complexity, since in essence it requires that there is also a semantic equivalence over the labels (by this, I refer to the equivalence which equates syntactic forms which I wish to view as the same) as well as one over the processes (bisimulation). The fact that the semantic equivalence over the labels is induced by another Σ-algebra, means that it is possible to work with an equivalence over the labels without considering the specific Σ-algebra, and this is how I will proceed for the next two chapters. Then in Chapter 6, I will investigate what effect the conditions required in this chapter for congruence have on the Σ-algebras that can be used to represent process algebra labels.

Another reason for dealing with labels in a syntactic manner is the fact that in some of the newer process algebras, the equivalence on processes does not require an exact match between labels. It is not clear that the existing formats can fit with a more general definition of bisimulation, because of the schematic use of variables.

Finally, an important aspect of some of the extensions to CCS is the fact that information about the computation is stored in the process terms. Examples of process algebras that use this technique are CCS with locations [BCHK93, BCHK94], CCS with local and global causes [Kie94] and CCS with causalities [DD89, DD90]. Using labels syntactically gives a full account of how this passing of information is performed.

The issue of schemas will be returned to in Chapter 6 where I make use of schematic representation as a means to express large rule sets, and I will explain why these schemas differ from the variables used in the format.

I have focussed on extending the *tyft/tyxt* format, and hence will not concern myself in this document with negative premises or predicates, but leave these as an issue for further work. I will discuss this further in Chapter 7.

As I am interested in process algebras which have more complex sets of actions, I will now take this into account in my model of these process algebras and define a new format. As will be seen in Chapter 5, taking this syntactic approach and permitting different sorts in the labels is a powerful tool when comparing process algebras and their equivalences.

I will start by defining a specific type of sorted set and signature which I will use to represent the terms that appear in the rules of the format.

I will assume that the S-sorted sets under consideration do not contain a distinguished sort P (named for processes), and will insist that the functions of the signatures of interest have the following constraint for any function symbol $f : s_1 \dots s_n \to s$, if $s \neq \mathsf{P}$ then for all $1 \leqslant i \leqslant n$, $s_i \neq \mathsf{P}$.

This means that only functions with range of sort P can take arguments of sort P, and this will mean that only process terms can contain process terms. This is reasonable because it is the way process algebras are specified; moreover if there is a need for a label term to contain a process term, it would be possible to define a label term that would be understood to represent the process term.

Definition 4.3.1 (Suitable signature)
A signature $\Sigma = (S \cup \{\mathsf{P}\}, F)$ is called *suitable* if and only if

- S does not contain the distinguished element P and is non-empty,
- for any function symbol $f \in F$ such that $f : s_1 \dots s_n \to s$, whenever $s \neq \mathsf{P}$ then for all $1 \leqslant i \leqslant n$, $s_i \neq \mathsf{P}$. ■

At this stage of the work, I wish to work with signatures that are both sensible and suitable. However, I will not insist that suitable signatures are sensible because in the next chapter when I consider extensions, I wish to use the two concepts separately.

For convenience, I will assume that functions that have a range of sort P, will be written with the non-P arguments first and then the arguments of sort P; for example, $f : s_1 \dots s_m \mathsf{P} \dots \mathsf{P} \to \mathsf{P}$; and will also assume that there are $n \geqslant 0$ arguments with sort P, making the total number of arguments that f takes to be $m + n$.

Notation Let $\Sigma = (S \cup \{\mathsf{P}\}, F)$ be a suitable signature and let V be an S-sorted set of variables. I will use both V_S and $V_{\overline{\mathsf{P}}}$ for $V - V_{\mathsf{P}}$, both $\mathbb{T}(\Sigma)_S$ and $\mathbb{T}(\Sigma)_{\overline{\mathsf{P}}}$ for $\mathbb{T}(\Sigma) - \mathbb{T}(\Sigma)_{\mathsf{P}}$, and both $\mathbf{T}(\Sigma)_S$ and $\mathbf{T}(\Sigma)_{\overline{\mathsf{P}}}$ for $\mathbf{T}(\Sigma) - \mathbf{T}(\Sigma)_{\mathsf{P}}$. For variables, I will use $x, x', y, y', \dots$ to range over V_{P}, and $z, z', \dots$ to range over variables from $V_{\overline{\mathsf{P}}}$. However, I will also occasionally use $x, x', \dots$ to range over V. I will use $p, p', q, q', \dots$ to range over $\mathbb{T}(\Sigma)_{\mathsf{P}}$ and $u, u', v, v', \dots$ to range over $\mathbf{T}(\Sigma)_{\mathsf{P}}$. I will use $t, t', \dots$ to range over all terms from $\mathbb{T}(\Sigma)$ and $\lambda, \lambda', \eta, \eta', \dots$ to range over terms from $\mathbb{T}(\Sigma)_{\overline{\mathsf{P}}}$. Furthermore, I will use $\alpha, \alpha', \beta, \beta', \mu, \mu', \nu, \nu', \ldots$ to range over terms from $\mathbf{T}(\Sigma)_{\overline{\mathsf{P}}}$. ■

Note that the requirement on functions with range P implies that terms from $\mathbb{T}(\Sigma)_{\overline{\mathsf{P}}}$ contain no variables from V_{P} or functions with range P.

Proposition 4.3.1 (Terms over a suitable signature)
Let $\Sigma = (S \cup \{\mathsf{P}\}, F)$ be a suitable signature. Then $t \in \mathbb{T}(\Sigma)_{\overline{\mathsf{P}}}$ contains no variables from V_{P} or functions with range P.

Proof: Let $t \in \mathbb{T}(\Sigma)_{\overline{\mathsf{P}}}$. If $t = x$, then t has sort s for some $s \neq \mathsf{P}$. If $t = f(t_1, \ldots, t_n)$, then $f : s_1 \ldots s_n \to s$ and $s_i \neq \mathsf{P}$ for all $1 \leqslant i \leqslant n$ since $s \neq \mathsf{P}$. So by an inductive argument based on the structure of the term, it is clear that no t_i contains a variable from V_{P} or function with range P, hence t does not. ■

Note that the definitions that follow could have been given in a more general fashion, by not specifying the sorts of the labels, targets and sources; and this distinction could have been made when the extended *tyft/tyxt* format was defined. However, since there appears to be an inherent lack of symmetry in the way processes and actions are treated, I have chosen to be more specific in the following definitions.

I now define an extended transition system specification. This definition extends the earlier definitions by allowing a richer structure for the labels.

Definition 4.3.2 (Extended transition system specification)
An *extended transition system specification (eTSS)* is a pair (Σ, R) with Σ a suitable signature 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 $\lambda_i, \lambda \in \mathbb{T}(\Sigma)_{\overline{\mathsf{P}}}$ 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 called the *premises* or *hypotheses* of r, and $p \xrightarrow{\lambda} p'$ is called the *conclusion* of r. A rule with $I = \emptyset$ is called an *axiom* and is written $p \xrightarrow{\lambda} p'$. An expression of the form $p \xrightarrow{\lambda} p'$ with $\lambda \in \mathbb{T}(\Sigma)_{\overline{\mathsf{P}}}$ and $p, p' \in \mathbb{T}(\Sigma)$ is called a *transition (labelled with*

$\lambda)$; p is called the *source*, and p' is called the *target* of the transition. $\phi, \psi, \chi, \ldots$ are used to range over transitions. The notions of closed, substitution and Var can be extended to transitions in the obvious way. ■

Definition 4.3.3 (Proof)
Let $\mathcal{E} = (\Sigma, R)$ be an eTSS with Σ a sensible signature. A *proof* of a transition ψ from $\mathcal{E}$ is a well-founded, upwardly branching tree of which the nodes are labelled by transitions $p \xrightarrow{\lambda} p'$ with $\lambda \in \mathbb{T}(\Sigma)_{\overline{\mathsf{P}}}$ 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, I say that ψ is *provable* from $\mathcal{E}$, notation $\mathcal{E} \vdash \psi$. A proof is *closed* if it only contains closed transitions. ■

I now present a running example which I will use in this chapter and the next. It is based on a subset of CCS. Note that instead of a schematic prefix operator $a.x$ as used in *tyft/tyxt* format, I now use a prefix operator with two arguments, $\mathsf{pref}(z, x)$—the first argument is for the label and the second for the process. Hence instead of an infinite number of prefix operators, I now have only one. More detailed examples will be given in Chapter 6.

Example 4.3.1 (Example of an eTSS)
Let A be a disjoint set, disjoint from the variables and any other function symbols. I will also use A as a sort name—this does not cause problems. Consider the signature Σ_{CCSSub}, (A, P; $\{\mathsf{a}\}_{\mathsf{a} \in \mathsf{A}}, \mathsf{nil}$; $\mathsf{pref}, \mathsf{plus}$) with the types $\mathsf{a} :\to \mathsf{A} \quad \forall \mathsf{a} \in \mathsf{A}$, $\mathsf{nil} :\to \mathsf{P}$, $\mathsf{plus} : \mathsf{P}, \mathsf{P} \to \mathsf{P}$, $\mathsf{pref} : \mathsf{A}, \mathsf{P} \to \mathsf{P}$. Clearly this is a sensible and suitable signature. Consider the rule set R_{CCSSub}

$$\frac{}{\mathsf{pref}(z, x) \xrightarrow{z} x} \qquad \frac{x \xrightarrow{z} y}{\mathsf{plus}(x, x') \xrightarrow{z} y} \qquad \frac{x \xrightarrow{z} y}{\mathsf{plus}(x', x) \xrightarrow{z} y}$$

and define $\mathcal{E}_{\text{CCSSub}} = (\Sigma_{\text{CCSSub}}, R_{\text{CCSSub}})$. Then $\mathcal{E}_{\text{CCSSub}}$ is an eTSS. I can prove the transition $\mathsf{plus}(\mathsf{pref}(\mathsf{a}, \mathsf{nil}), \mathsf{pref}(\mathsf{b}, \mathsf{nil})) \xrightarrow{\mathsf{b}} \mathsf{nil}$ in the following manner. Consider the proof tree

$$\frac{\mathsf{pref}(\mathsf{b}, \mathsf{nil}) \xrightarrow{\mathsf{b}} \mathsf{nil}}{\mathsf{plus}(\mathsf{pref}(\mathsf{a}, \mathsf{nil}), \mathsf{pref}(\mathsf{b}, \mathsf{nil})) \xrightarrow{\mathsf{b}} \mathsf{nil}}.$$

It is possible to find suitable rules and substitutions to construct this proof tree. For example, taking the last rule, and the substitution σ with σ the identity except that $\sigma(x) = \mathsf{pref}(\mathsf{b}, \mathsf{nil})$, $\sigma(x') = \mathsf{pref}(\mathsf{a}, \mathsf{nil})$, $\sigma(y) = \mathsf{nil}$ and $\sigma(z) = \mathsf{b}$, the conditions are satisfied for the bottom node and the node above it. For the top node, the rule to use is the axiom together with the substitution σ' with σ' the identity everywhere except $\sigma'(z) = \mathsf{b}$ and $\sigma'(x) = \mathsf{nil}$. ■

Lemma 4.3.1 (Closed transitions are provable by closed proofs)
Let $\mathcal{E} = (\Sigma, R)$ be an eTSS with Σ a sensible signature, let $\lambda \in \mathbf{T}(\Sigma)_{\overline{\mathsf{P}}}$, $u, u' \in \mathbf{T}(\Sigma)_{\mathsf{P}}$ such that $\mathcal{E} \vdash u \xrightarrow{\lambda} u'$. Then $u \xrightarrow{\lambda} u'$ is provable by a closed proof.

Proof: Since $\mathcal{E} \vdash u \xrightarrow{\lambda} u'$ there is a proof tree T for $u \xrightarrow{\lambda} u'$. Define the substitution $\sigma : V \to \mathbf{T}(\Sigma)$ by $\sigma(x_s) = t$ for $x_s \in V_s$ where t is a closed term from $\mathbf{T}(\Sigma)$. Such a t exists for each sort since Σ is sensible. Applying σ to all transitions in the proof T yields a closed proof tree T' which is also a proof of $u \xrightarrow{\lambda} u'$. ■

I can define the labelled transition system generated by an eTSS.

Definition 4.3.4 (LTS specified by an eTSS)
Let $\mathcal{E} = (\Sigma, R)$ be an eTSS with Σ, a sensible signature. The *LTS* $TS(\mathcal{E})$ specified by $\mathcal{E}$ is given by

$$TS(\mathcal{E}) = (\mathbf{T}(\Sigma)_{\mathsf{P}}, \mathbf{T}(\Sigma)_{\overline{\mathsf{P}}}, \longrightarrow)$$

where $\longrightarrow \subseteq \mathbf{T}(\Sigma)_{\mathsf{P}} \times \mathbf{T}(\Sigma)_{\overline{\mathsf{P}}} \times \mathbf{T}(\Sigma)_{\mathsf{P}}$ is defined by $u \xrightarrow{\alpha} u' \iff \mathcal{E} \vdash u \xrightarrow{\alpha} u'$. ■

Part of the LTS described by the eTSS $\mathcal{E}_{\text{CCSSub}}$ is given in Figure 4.1. In the figure, I give the transitions from the term $\mathsf{plus}(\mathsf{pref}(\mathsf{a}, \mathsf{nil}), \mathsf{pref}(\mathsf{b}, \mathsf{nil}))$.

$$\begin{tikzcd}
\mathsf{plus(pref(a, nil), pref(b, nil))} \arrow[d, bend right, "\mathsf{a}"'] \arrow[d, bend left, "\mathsf{b}"] \\
\mathsf{nil}
\end{tikzcd}$$

Figure 4.1: LTS given by term from Example 4.3.1

Definition 4.3.5 (Transition equivalence)

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

Example 4.3.2 (Transition equivalence)

Consider the eTSS from Example 4.3.1. If I define a new eTSS that is the same as $\mathcal{E}_{\text{CCSSub}}$ but with the additional rule

$$\frac{x \xrightarrow{z} y}{x \xrightarrow{z} y}$$

then the two eTSSs are transition equivalent. ■

I could work with the standard definition of bisimulation, but that would not be of much interest since I would only be comparing labels which are syntactically equal. I will assume that I have an S-sorted equivalence that equates terms I wish to view as the same, hence I have the following definition.

Definition 4.3.6 (Strong bisimulation with respect to an equivalence over a sorted set)

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

1. whenever $s \xrightarrow{a} s'$, then there exists $t' \in \mathcal{S}$ and $b \in \mathcal{A}$ such that $t \xrightarrow{b} t'$, $a \equiv b$ and $(s', t') \in \mathcal{R}$
2. whenever $t \xrightarrow{a} t'$, then there exists $s' \in \mathcal{S}$ and $b \in \mathcal{A}$ 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 $\mathcal{R}$ 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 I am only interesting in comparing transitions with labels of the same sort and this appears to be a reasonable requirement, and as will be shown in the rest of the document, a powerful mechanism for comparing process algebras. I have chosen an equivalence in the definition above, as I wish the bisimulation to be an equivalence.

As mentioned earlier, when expressing process algebras, a Σ-algebra will be used to define the semantics of the actual labels and this will induce an equivalence over the terms of the term algebra. This can expressed as a requirement that the transition system of the process algebra is isomorphic to the transition system

$$\big(\mathbf{T}(\Sigma)_{\mathsf{P}}/\equiv, \mathbf{T}(\Sigma)_{\overline{\mathsf{P}}}/\equiv, \mathcal{T}\big),$$

$$\mathcal{T} = \{S \xrightarrow{A} S' \mid S, S' \in \mathbf{T}(\Sigma)_{\mathsf{P}}/\equiv, A \in \mathbf{T}(\Sigma)_{\overline{\mathsf{P}}}/\equiv, \forall s \in S, s' \in S', a \in A, s \xrightarrow{a} s'\}.$$

Note I am working with strong bisimulation as a starting point, because weak bisimulation would introduce additional complications. I wish to investigate weak bisimulation as further work, and will discuss it further in the final chapter.

4.3.1 Extended *tyft/tyxt* format

I now have a general definition of an eTSS, but I would like to find a more specific definition that has desirable properties, such as being a congruence with respect to bisimulation equivalence. In the rest of this chapter, I follow much the same path as Groote and Vaandrager [GV92] in showing congruence, although my definitions and results require more care because of the new way of dealing with labels, and I also need a new condition on how the equivalence and labels interact. I propose the following definition.

Definition 4.3.7 (Extended *tyft/tyxt* format)

Let $\Sigma = (S \cup \{\mathsf{P}\}, F)$ be a suitable signature and let $\mathcal{E} = (\Sigma, R)$ be an eTSS. 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,
- $f \in F$ such that $f : s_1 \ldots s_m \mathsf{P} \ldots \mathsf{P} \to \mathsf{P}$ with $s_k \neq \mathsf{P}$ for all $1 \leqslant k \leqslant m$,
- x_j $(1 \leqslant j \leqslant n)$ and y_i $(i \in I)$ all different variables from V_{P},
- $p \in \mathbb{T}(\Sigma)_{\mathsf{P}}$, $\lambda \in \mathbb{T}(\Sigma)_{\overline{\mathsf{P}}}$,
- $\eta_k \in \mathbb{T}(\Sigma)_{s_k}$ such that $\mathrm{Var}_{\overline{\mathsf{P}}}(\eta_k) \subset V_{\overline{\mathsf{P}}} - \bigcup_{\substack{1 \leqslant l \leqslant m \\ l \neq k}} \mathrm{Var}_{\overline{\mathsf{P}}}(\eta_l)$ for $1 \leqslant k \leqslant m$,
- $\lambda_i \in \mathbb{T}(\Sigma)_{\overline{\mathsf{P}}}$ for $i \in I$ such that
  $\mathrm{Var}_{\overline{\mathsf{P}}}(\lambda_i) \subset V_{\overline{\mathsf{P}}} - (\bigcup_{l \in I, l \neq i} \mathrm{Var}_{\overline{\mathsf{P}}}(\lambda_l) \cup \bigcup_{1 \leqslant k \leqslant m} \mathrm{Var}_{\overline{\mathsf{P}}}(\eta_k))$ for all $i \in I$.
- $p_i \in \mathbb{T}(\Sigma)_{\mathsf{P}}$ such that $\mathrm{Var}_{\overline{\mathsf{P}}}(p_i) \subset V_{\overline{\mathsf{P}}} - \bigcup_{l \in I} \mathrm{Var}_{\overline{\mathsf{P}}}(\lambda_l)$ 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},
- $p \in \mathbb{T}(\Sigma)_{\mathsf{P}}$, $\lambda \in \mathbb{T}(\Sigma)_{\overline{\mathsf{P}}}$,
- $\lambda_i \in \mathbb{T}(\Sigma)_{\overline{\mathsf{P}}}$ such that $\mathrm{Var}_{\overline{\mathsf{P}}}(\lambda_i) \subset V_{\overline{\mathsf{P}}} - \bigcup_{l \in I, l \neq i} \mathrm{Var}_{\overline{\mathsf{P}}}(\lambda_l)$ for all $i \in I$.
- $p_i \in \mathbb{T}(\Sigma)_{\mathsf{P}}$ such that $\mathrm{Var}_{\overline{\mathsf{P}}}(p_i) \subset V_{\overline{\mathsf{P}}} - \bigcup_{l \in I} \mathrm{Var}_{\overline{\mathsf{P}}}(\lambda_l)$ for $i \in I$.

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

To summarise, I require for this definition that 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 4.3.1 is in *tyft/tyxt* format.

I will proceed to show that for certain equivalences, bisimulation with respect to those equivalences is a congruence. This is similar to the standard requirement for formats. Congruence is an important property of process algebra equivalences, since it can be used to show that systems composed of bisimilar components are bisimilar. Hence, because of this importance, it is reasonable to first evaluate a format's effectiveness in terms of whether congruence can be shown for any process algebra in that format.

I will also give counter-examples to show that the requirements cannot be further relaxed without losing congruence. However, it is not yet known whether the requirement for the p_i's and λ_i's to have distinct variables is necessary for the congruence result. To achieve the congruence result, I require some additional definitions. Firstly, I need some conditions for the type of equivalence that will allow congruence to work. If I were to go for a simpler format, where only variables are allowed to appear in the positions of the p_i's, λ_i's and η_k's then any congruence could be used, but then the format would not have the features that would allow it to be general enough to capture many process algebras of interest.

Definition 4.3.8 (Compatibility)

Let $\Sigma = (S \cup \{\mathsf{P}\}, F)$ be a suitable signature and let $\mathcal{E} = (\Sigma, R)$ be an eTSS in extended *tyft/tyxt* format. Let $\equiv$ be a congruence on $\mathbf{T}(\Sigma)$. $\equiv$ is *compatible* with $r \in R$ if for any $\eta \in \mathbb{T}(\Sigma)_{\overline{\mathsf{P}}}$ that appears on a transition in a premise of r or as an argument to the function in the source of the conclusion of r, then whenever $\sigma(\eta) \equiv \mu$ for $\mu \in \mathbf{T}(\Sigma)_{\overline{\mathsf{P}}}$, there exists a substitution σ' such that $\mu = \sigma'(\eta)$ and $\sigma(z) \equiv \sigma'(z)$ for all $z \in \mathrm{Var}_{\overline{\mathsf{P}}}(\eta)$.

I say that $\equiv$ 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 since one can define $\sigma'(z) = \mu$, and obviously $\sigma(z) \equiv \sigma'(z)$. Furthermore, note that the occurrence of repeated variables in η results in the only compatible congruence being syntactic equivalence. Consider a function symbol g with arity 2, and let $\eta = g(z, z)$. Moreover, assume there exist $\mu_1 \equiv \mu_2$ with μ_1 and μ_2 syntactically different and consider the substitution σ such that $\sigma(z) = \mu_1$. It is clear that $\sigma(g(z, z)) \equiv g(\mu_1, \mu_2)$, however it is not possible to find a substitution σ' such that $\sigma'(g(z, z)) = g(\mu_1, \mu_2)$. So although the extended *tyft/tyxt* format does not rule out repeated variables in labels in premises or in the arguments to the function in the source of the conclusion, generally I will not use these in practice because they are not of interest. Examples of rules that do not exhibit compatibility will be given in the counter-examples that appear after Theorem 4.3.1. This issue will be raised again in Chapter 6 when I consider the implications of compatibility for Σ-algebras.

The following definition is required to ensure that there are no cycles of variable references appearing in the premises. I will discuss recent work relating to well-foundedness at the end of this chapter.

Definition 4.3.9 (Well-foundedness)
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: $\bigcup_{i \in I} \mathrm{Var}_{\mathsf{P}}(p_i \xrightarrow{\lambda_i} p'_i)$,
- Edges: $\{\langle x, y\rangle \mid x \in \mathrm{Var}_{\mathsf{P}}(p_i), y \in \mathrm{Var}_{\mathsf{P}}(p'_i)$ for some $i \in I\}$.

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

To prove the congruence result, this condition is required. Recently it has been shown that for the original *tyft/tyxt* format that any rule can be written in a well-founded form [FvG96]. As yet I do not know if it is possible to show a similar result for the extended *tyft/tyxt* format. However, it is not a condition that in any way affects the process algebras which I will consider later in this thesis, and hence I will not concern myself with it, except for a short discussion in the counter-examples sections in this chapter and as an issue for further work.

Example 4.3.3 (Well-foundedness)
$\mathcal{E}_{\mathrm{CCSSub}}$ is well-founded. Note also that for any congruence over the set of terms, it is compatible, since there are only variables that appear on transitions in premises, or in the source of the conclusion. ■

I require some additional definitions that allow us to work with eTSSs conveniently, and I need to ensure that I can preserve compatibility and well-foundedness when I use the results.

Lemma 4.3.2 (Well-founded extended *tyft/tyxt* can be transformed to well-founded extended *tyft*)
Let $\Sigma = (S \cup \{\mathsf{P}\}, F)$ be a suitable, sensible signature and let $\mathcal{E} = (\Sigma, R)$ be a well-founded eTSS in extended *tyft/tyxt* format. Let $\equiv$ be an congruence on $\mathbf{T}(\Sigma)$ such that $\equiv$ is compatible with R. Then there is a transition equivalent well-founded eTSS $\mathcal{E}' = (\Sigma, R')$ in extended *tyft* format such that $\equiv$ is compatible with R'.

Proof: Define R' by

- every extended *tyft* rule of R is in R',
- for every extended *tyxt* rule $r \in R$ and for every function symbol $f \in F$ such that $f : s_1 \dots s_m\mathsf{P} \dots \mathsf{P} \to \mathsf{P}$, r_f is in R' where r_f is obtained by substituting $f(z_1, \dots, z_m, x_1, \dots, x_n)$ for x in r with $z_k \in V_{s_k} - \mathrm{Var}_{s_k}(r)$ $(1 \leqslant k \leqslant m)$ and $\{x_1, \dots, x_n\} \subseteq V_{\mathsf{P}} - \mathrm{Var}_{\mathsf{P}}(r)$.

If the rules in R are well-founded, then it is clear that the rules in R' are well-founded, since none of the premises of rules have been changed. Note that $\equiv$ is compatible with R' since only function symbols of form $f(z_1, \ldots, z_m, x_1, \ldots, x_n)$ have been added.

Suppose that $u \xrightarrow{\alpha} u'$ is a transition in $TS(\mathcal{E})$. Then there is a closed proof from $\mathcal{E}$ of this transition. One can easily see that this is also a proof of $u \xrightarrow{\alpha} u'$ from $\mathcal{E}'$. A similar argument shows that every transition of $TS(\mathcal{E}')$ is a transition $TS(\mathcal{E})$. ■

Definition 4.3.10 (Freeness of variables)

Let $\mathcal{E} = (\Sigma, R)$ be an eTSS, and let r be a rule in R. A variable in $\mathrm{Var}_{\mathsf{P}}(r)$ is called *free* if it does not occur in the left hand side of the conclusion or in the right hand side of a premise. ■

Definition 4.3.11 (Pureness of rules)

Let $\mathcal{E} = (\Sigma, R)$ be an eTSS. A rule $r \in R$ is called *pure* if it is well-founded and contains no free variables from V_{P}. The eTSS $\mathcal{E}$ is called *pure* if all its rules are pure. ■

Lemma 4.3.3 (Well-founded extended *tyft/tyxt* can be transformed to pure extended *tyft*)

Let $\mathcal{E} = (\Sigma, R)$ be a well-founded eTSS in extended *tyft/tyxt* format with Σ a sensible signature. Let $\equiv$ be a congruence on $\mathbf{T}(\Sigma)$ such that $\equiv$ is compatible with R. Then there is a transition equivalent pure eTSS $\mathcal{E}' = (\Sigma, R')$ in extended *tyft* format such that $\equiv$ is compatible with R'.

Proof: From Lemma 4.3.2, it can be assumed that $\mathcal{E}$ is in extended *tyft* format. Define R' by

- every pure rule of R is in R',
- every rule r in R that is not pure is replaced by a set of new rules where every possible substitution of terms from $\mathbf{T}(\Sigma)_{\mathsf{P}}$ is applied to the free variables.

The rules in R' are well-founded, since the rules in R are well-founded and for each rule, in effect only edges have been removed from its dependency graph. $\equiv$ is compatible with R' since neither the labels of the premises nor the arguments to the function in the source of the conclusion have been modified, and also only closed terms have been used in the substitution, hence the variables in the sources of the premises are still distinct from the variables in the labels of the premises and the variables in the arguments to the function in the source of the conclusion. R' is in extended *tyft* format since no left hand side of any conclusion has been modified. Clearly every closed proof for a transition $u \xrightarrow{\alpha} u'$ from $\mathcal{E}$ is also a proof for $u \xrightarrow{\alpha} u'$ from $\mathcal{E}$ and vice versa. ■

4.3.2 Congruence theorem

I now prove a general result which shows the congruence of bisimulation under certain conditions. I work with a very general definition of congruence, and assume that I have an S-sorted congruence over the terms of $\mathbf{T}(\Sigma)_{\overline{\mathsf{P}}}$. Note that this is slightly different to the use of the congruence in earlier definitions and results where I assumed a congruence over all of $\mathbf{T}(\Sigma)$. However, note that since the definition of bisimulation is only concerned with the congruence over $\mathbf{T}(\Sigma)_{\overline{\mathsf{P}}}$, it is not necessary for the congruence to be defined on $\mathbf{T}(\Sigma)_{\mathsf{P}}$. In any case, since the congruence respects sorts, when given a congruence on $\mathbf{T}(\Sigma)_{\overline{\mathsf{P}}}$, one can extend it to the identity on $\mathbf{T}(\Sigma)_{\mathsf{P}}$ and this results in a congruence over the whole set of closed terms.

The result states that given any terms that are related by this congruence or related by bisimulation up to that congruence if they have sort P, then I know that terms of sort P with subterms that are related are bisimilar up to the congruence. Hence it can be concluded that the bisimulation up to the congruence is a congruence itself.

Theorem 4.3.1 (Congruence)
Let $\Sigma = (S \cup \{\mathsf{P}\}, F)$ be a suitable, sensible signature, let $\mathcal{E} = (\Sigma, R)$ be a well-founded eTSS in extended *tyft/tyxt* format and let $\equiv$ be a congruence on $\mathbf{T}(\Sigma)_{\overline{\mathsf{P}}}$ compatible with $\mathcal{E}$. Then for all $f \in F$ such that $f : s_1 \ldots s_m \mathsf{P} \ldots \mathsf{P} \to \mathsf{P}$, for all terms $\mu_k, \nu_k \in \mathbf{T}(\Sigma)_{\overline{\mathsf{P}}}$ $(1 \leqslant k \leqslant m)$, and for all terms $u_i, v_i \in \mathbf{T}(\Sigma)_{\mathsf{P}}$ $(1 \leqslant i \leqslant n)$,

$$\mu_i \equiv \nu_i \ \ (1 \leqslant k \leqslant m) \text{ and } u_i \sim_{\equiv} v_i \ \ (1 \leqslant i \leqslant n) \Rightarrow$$
$$f(\mu_1, \ldots, \mu_m, u_1, \ldots, u_n) \sim_{\equiv} f(\nu_1, \ldots, \nu_m, v_1, \ldots v_n).$$

Proof: Let $\mathcal{R} \subseteq \mathbf{T}(\Sigma)_{\mathsf{P}} \times \mathbf{T}(\Sigma)_{\mathsf{P}}$ 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 $\mu_k, \nu_k \in \mathbf{T}(\Sigma)_{\overline{\mathsf{P}}}$ $(1 \leqslant k \leqslant m)$, and for all terms $u_i, v_i \in \mathbf{T}(\Sigma)_{\mathsf{P}}$ $(1 \leqslant i \leqslant n)$,

$$\mu_i \equiv \nu_i \ (1 \leqslant k \leqslant m) \text{ and } u_i \, \mathcal{R} \, v_i \ (1 \leqslant i \leqslant n) \Rightarrow$$
$$f(\mu_1, \dots, \mu_m, u_1, \dots, u_n) \, \mathcal{R} \, f(\nu_1, \dots, \nu_m, v_1, \dots v_n).$$

It is enough to show $\mathcal{R} \subseteq \sim_{\equiv}$ since $\sim_{\equiv} \subseteq \mathcal{R}$; hence I need to show that $\mathcal{R}$ is a bisimulation. Assume $u \, \mathcal{R} \, v$. I have two cases—the first is simple since $u \sim_{\equiv} v$. The second requires us to show 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 $1 \leqslant k \leqslant m$ and $u_i \, \mathcal{R} \, v_i$ for $1 \leqslant i \leqslant n$ then there is a $v' \in \mathbf{T}(\Sigma)_{\mathsf{P}}$ and $\alpha' \in \mathbf{T}(\Sigma)_{\overline{\mathsf{P}}}$ 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 I can assume that the rules in R_0 are pure and in extended *tyft* format. I will proceed by induction on the length of the proof. Let r be the last rule used in proof T, in combination with a substitution σ. Assume r is equal to

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

Then I know that $\sigma(\eta_k) = \mu_k$ for $1 \leqslant k \leqslant m$, $\sigma(x_i) = u_i$ for $1 \leqslant i \leqslant n$, $\sigma(p) = u'$ and $\sigma(\lambda) = \alpha$. I want to find a substitution σ' that I can use to show that $f(\nu_1, \dots, \nu_m, v_1, \dots, v_n) \xrightarrow{\alpha'} v'$ with $\alpha \equiv \alpha'$ and $u' \, \mathcal{R} \, v'$.

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 t_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.

Define

- $X = \{x_i \mid 1 \leqslant i \leqslant n\}$
- $Y = \{y_i \mid i \in I\}$
- $Y_d = \{y \in Y \mid \text{depth}(y) = d\}$ for $d \geqslant 0$.

Observe that for any variable $x \in X$, $\text{depth}(x) = 0$, and the sets Y_d form a partition of Y. Since $\mathcal{E}$ is pure, this covers all the variables from V_{P} in r. Next I consider variables from $V_{\overline{\mathsf{P}}}$. These can be partitioned in four sets.

- $Z = \bigcup_{1 \leqslant k \leqslant m} \text{Var}_{\overline{\mathsf{P}}}(\eta_k)$
- $Z' = \bigcup_{i \in I} \text{Var}_{\overline{\mathsf{P}}}(\lambda_i)$
- $Z'' = \bigcup_{i \in I} \text{Var}_{\overline{\mathsf{P}}}(p_i) - Z$
- $Z''' = \text{Var}_{\overline{\mathsf{P}}}(\lambda) \cup \text{Var}_{\overline{\mathsf{P}}}(p) - (Z \cup Z' \cup Z'')$.

I can partition Z' into $Z'_0, Z'_1, \ldots$ by defining $Z'_d = \bigcup_{y_i \in Y_d} \text{Var}_{\overline{\mathsf{P}}}(\lambda_i)$ for each $d \geqslant 0$. I will define a substitution σ' that satisfies the following properties on V_{P} and $V_{\overline{\mathsf{P}}}$

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

Substitution σ' will be constructed in stepwise fashion. To begin, let

- $\sigma'(x_i) = v_i$ for $1 \leqslant i \leqslant n$
- $\sigma'(y) = \sigma(y)$ for $y \in V_{\mathsf{P}} - (X \cup \bigcup_{d \geqslant 0} Y_d)$
- $\sigma'(z) = \sigma(z)$ for $z \in V_{\overline{\mathsf{P}}} - (Z \cup \bigcup_{d \geqslant 0} Z'_d \cup Z'')$.

Note therefore that for all variables $z \in V_{\overline{\mathsf{P}}} - (Z \cup \bigcup_{d \geqslant 0} Z'_d \cup Z'')$, $\sigma(z) \equiv \sigma'(z)$. I still have to define σ' on $\bigcup_{d \geqslant 0} Y_d$, $\bigcup_{d \geqslant 0} Z'_d$, Z and Z''.

First consider Z. Since for a given k such that $1 \leqslant k \leqslant m$, $\sigma(\eta_k) = \mu_k \equiv \nu_k$ and $\equiv$ is compatible with R_0, I know that there is a substitution σ'' such that $\sigma''(\eta_k) = \nu_k$ and for all $z \in \mathrm{Var}_{\overline{\mathsf{P}}}(\eta_k)$, $\sigma(z) \equiv \sigma''(z)$. So let $\sigma'(z) = \sigma''(z)$. I can do this for all $1 \leqslant k \leqslant m$ since there are no variables shared between the terms.

When σ' is defined for $y \in X \cup Y_0 \cup \ldots \cup Y_d$ and $z \in Z \cup Z'_0 \cup \ldots \cup Z'_d$ ($d \geqslant 0$), I will show that $\beta(d)$, $\gamma(d)$ and $\delta(d)$ 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 $z \in Z'_0 \cup \ldots \cup Z'_d$.

So if I can show that $\beta(d)$, $\gamma(d)$ and $\delta(d)$ hold for all $d \geqslant 0$, then I know that the second and third properties hold, and that the fourth property will hold for $Z \cup Z' \cup Z'''$. For $z \in Z''$, when dealing with a particular transition $p_i \xrightarrow{\lambda_i} y_i$, I will simply define $\sigma'(z) = \sigma(z)$ for any $z \in \mathrm{Var}_{\overline{\mathsf{P}}}(p_i)$ that has not yet been defined. Hence once I have shown $\beta(d)$, $\gamma(d)$ and $\delta(d)$ hold for all $d \geqslant 0$, I will have defined all $z \in Z''$ and moreover $\sigma(z) \equiv \sigma'(z)$ for all $z \in Z''$, so the fourth property will be satisfied. I know that the first property holds by definition.

I first need to show that $\beta(0)$, $\delta(0)$ and $\gamma(0)$ hold. First note that for any $x \in X$ then $x = x_i$ for some $1 \leqslant i \leqslant n$ and since $\sigma(x_i) = u_i$ and $\sigma'(x_i) = v_i$, and $u_i \, \mathcal{R} \, v_i$, I have $\sigma(x_i) \, \mathcal{R} \, \sigma'(x_i)$.

Next consider $y^* \in Y_0$. There exists $i \in I$ such that $y^* = y_i$, so I can consider the transition $p_i \xrightarrow{\lambda_i} y_i$. If $\mathrm{Var}_{\overline{\mathsf{P}}}(p_i) \cap Z = \emptyset$, then the situation is straightforward. Let $\sigma'(z) = \sigma(z)$ for $z \in \mathrm{Var}_{\overline{\mathsf{P}}}(p_i) \cup \mathrm{Var}_{\overline{\mathsf{P}}}(\lambda_i)$ and let $\sigma'(y_i) = \sigma(y_i)$. Then $\sigma(y_i) \sim_{\equiv} \sigma'(y_i)$ and hence $\sigma(y_i) \, \mathcal{R} \, \sigma'(y_i)$, and for all $z \in \mathrm{Var}_{\overline{\mathsf{P}}}(p_i) \cup \mathrm{Var}_{\overline{\mathsf{P}}}(\lambda_i)$, $\sigma(z) \equiv \sigma'(z)$ since $\sigma'(z) = \sigma(z)$. Moreover, $\sigma'(p_i \xrightarrow{\lambda_i} y_i) = \sigma'(p_i) \xrightarrow{\sigma'(\lambda_i)} \sigma'(y_i) = \sigma(p_i) \xrightarrow{\sigma(\lambda_i)} \sigma(y_i) = \sigma(p_i \xrightarrow{\lambda_i} y_i)$ Therefore $\mathcal{E} \vdash \sigma'(p_i \xrightarrow{\lambda_i} y_i)$ since $\mathcal{E} \vdash \sigma(p_i \xrightarrow{\lambda_i} y_i)$.

However, if $\mathrm{Var}_{\overline{\mathsf{P}}}(p_i) \cap Z \neq \emptyset$ then I need to take more care. σ' is already defined on $z \in \mathrm{Var}_{\overline{\mathsf{P}}}(p_i) \cap Z$, and I can define $\sigma'(z) = \sigma(z)$ for $z \in \mathrm{Var}_{\overline{\mathsf{P}}}(p_i) - Z$

for which σ' is not defined. Hence I know that $\sigma(z) \equiv \sigma'(z)$ for $z \in \mathrm{Var}_{\overline{\mathsf{P}}}(p_i)$. I need the following fact to proceed.

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

Proof: I will proceed by structural induction. If $p = x$ then I have the result. If $p = f(\eta_1, \ldots, \eta_m, p_1, \ldots, p_n)$, then I know by the induction hypothesis that $\rho(p_i) \mathrel{\mathcal{R}} \rho'(p_i)$ for $1 \leqslant i \leqslant n$, and also that $\rho(\eta_k) \equiv \rho'(\eta_k)$ for $1 \leqslant k \leqslant m$ (since $\equiv$ is a congruence), hence $\rho(t) \mathrel{\mathcal{R}} \rho'(t)$ by the definition of $\mathcal{R}$. ■

Since $\mathrm{Var}_{\mathsf{P}}(p_i) = \emptyset$, I know from the fact above that $\sigma(p_i) \mathrel{\mathcal{R}} \sigma'(p_i)$. I have two cases to consider

- $\sigma(p_i) \sim_{\equiv} \sigma'(p_i)$. Since $\mathcal{E} \vdash \sigma(p_i) \xrightarrow{\sigma(\lambda_i)} \sigma(y_i)$, I can find $w \in T(\Sigma)_{\mathsf{P}}$ and $\alpha_i \in T(\Sigma)_{\overline{\mathsf{P}}}$ such that $\mathcal{E} \vdash \sigma'(p_i) \xrightarrow{\alpha_i} w$, $\sigma(\lambda_i) \equiv \alpha_i$ and $\sigma(y_i) \mathrel{\mathcal{R}} w$. So I can define $\sigma'(y^*) = \sigma'(y_i) = w$.

  Moreover, since $\equiv$ is compatible with R_0, I know there exists a substitution σ'' such that $\alpha_i = \sigma''(\lambda_i)$ and $\sigma(z) \equiv \sigma''(z)$ for all $z \in \mathrm{Var}_{\overline{\mathsf{P}}}(\lambda_i)$. Let $\sigma'(z) = \sigma''(z)$ whence $\alpha_i = \sigma'(\lambda_i)$. (Since $\mathrm{Var}_{\overline{\mathsf{P}}}(p_i) \cap \mathrm{Var}_{\overline{\mathsf{P}}}(\lambda_i) = \emptyset$, it is clear that the use of σ'' does not affect values assigned to $\mathrm{Var}_{\overline{\mathsf{P}}}(p_i)$.)

- there is a function symbol $h \in F$ such that $h : s'_1 \ldots s'_m \mathsf{P} \ldots \mathsf{P} \to \mathsf{P}$, and there are terms $\mu'_{k'}, \nu'_{k'} \in \mathbf{T}(\Sigma)_{\overline{\mathsf{P}}}$ for $1 \leqslant k' \leqslant m'$, $w_{i'}, w'_{i'} \in \mathbf{T}(\Sigma)_{\mathsf{P}}$ for $1 \leqslant i' \leqslant n'$ such that

  $$\sigma(p_i) = h(\mu'_1, \ldots, \mu'_{m'}, w_1, \ldots, w_{n'})$$

  and

  $$\sigma'(p_i) = h(\nu'_1, \ldots, \nu'_{m'}, w'_1, \ldots, w'_{n'})$$

  with $\mu'_{k'} \equiv \nu'_{k'}$ $(1 \leqslant k' \leqslant m')$ and $w_{i'} \mathrel{\mathcal{R}} w'_{i'}$ $(1 \leqslant i' \leqslant n')$. Now I can apply the induction hypothesis. Since $\mathcal{E} \vdash h(\mu'_1, \ldots, \mu'_{m'}, w_1, \ldots, w_{n'}) \xrightarrow{\sigma(\lambda_i)} \sigma(y_i)$, I can find 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 $\sigma(y_i) \mathrel{\mathcal{R}} w$. Again I can define $\sigma'(y_i) = w$.

Since $\equiv$ is compatible with R_0, I know there exists a substitution σ'' such that $\alpha_i = \sigma''(\lambda_i)$ and $\sigma(z) \equiv \sigma''(z)$. Let $\sigma'(z) = \sigma''(z)$ for all $z \in \mathrm{Var}_{\overline{\mathsf{P}}}(\lambda_i)$ whence $\alpha_i = \sigma'(\lambda_i)$.

Hence I know for $y^* = y_i$, that $\sigma(y_i)\,\mathcal{R}\,\sigma'(y_i)$, $\mathcal{E} \vdash \sigma'(p_i \xrightarrow{\lambda_i} y_i)$ and $\sigma(z) \equiv \sigma'(z)$ for all $z \in \mathrm{Var}_{\overline{\mathsf{P}}}(p_i) \cup \mathrm{Var}_{\overline{\mathsf{P}}}(\lambda_i)$. I can do this for all $y \in Y_0$ and thereby show that $\beta(0)$, $\delta(0)$ and $\gamma(0)$ hold.

Let $d > 0$, and suppose that σ' has been defined for all variables in $X \cup Y_0 \cup \ldots Y_{d-1}$ and $Z'_0 \cup \ldots \cup Z'_{d-1}$ such that $\beta(d-1)$, $\gamma(d-1)$ and $\delta(d-1)$ hold.

I now define σ' on Y_d and Z'_d such that $\beta(d)$, $\gamma(d)$ and $\delta(d)$ hold. Consider $y^* \in Y_n$. Then there exists $i \in I$ such that $y^* = y_i$, and so I can consider the transition $p_i \xrightarrow{\lambda_i} y_i$. Since $y_i \in Y_n$, then $\mathrm{Var}_{\mathsf{P}}(p_i) \subseteq X \cup Y_0 \cup \ldots \cup Y_{d-1}$ so $\sigma(y)\mathcal{R}\sigma'(y)$ for $y \in \mathrm{Var}_{\mathsf{P}}(p_i)$. For $z \in \mathrm{Var}_{\overline{\mathsf{P}}}(p_i)$ such that $\sigma'(z)$ is as yet undefined, let $\sigma'(z) = \sigma(z)$. Hence I know that by the fact above $\sigma'(p_i)\,\mathcal{R}\,\sigma(p_i)$. I have two cases as before and the treatment is identical. From this it is easy to see that $\beta(d)$, $\gamma(d)$ and $\delta(d)$ hold for all $d \geqslant 0$, and hence I know that the four properties hold.

So I know that for all $i \in I$, $\mathcal{E} \vdash \sigma'(p_i) \xrightarrow{\sigma'(\lambda_i)} \sigma'(y_i)$ where $\sigma'(z) \equiv \sigma(z)$ for all $z \in \bigcup_{i \in I} \mathrm{Var}_{\overline{\mathsf{P}}}(\lambda_i)$. Hence I can conclude that

$$\mathcal{E} \vdash \sigma'(f(\eta_1, \ldots, \eta_m, x_1, \ldots, x_n) \xrightarrow{\lambda} p)$$

namely

$$\mathcal{E} \vdash f(\nu_1, \ldots, \nu_m, v_1, \ldots, v_n) \xrightarrow{\sigma'(\lambda)} \sigma'(p)$$

To see that $\sigma'(\lambda) \equiv \alpha$, recall that $\alpha = \sigma(\lambda)$, and I know that for all $z \in \mathrm{Var}_{\overline{\mathsf{P}}}(\lambda)$, $\sigma(z) \equiv \sigma'(z)$ and since $\equiv$ is a congruence, I have the required result. To see that $u\,\mathcal{R}\,\sigma'(p)$, recall that $u = \sigma(p)$, and for all $x \in \mathrm{Var}_{\mathsf{P}}(p)$, $\sigma(x)\,\mathcal{R}\,\sigma'(x)$ and for all $z \in \mathrm{Var}_{\overline{\mathsf{P}}}(p)$, $\sigma(z) \equiv \sigma'(z)$, hence by the fact above, $\sigma(p)\,\mathcal{R}\,\sigma'(p)$. ■

From this theorem I can conclude 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 for a given $f \in F$ such that $f : s_1 \ldots s_m \mathsf{P} \ldots \mathsf{P} \to \mathsf{P}$, there are arguments that are not related by $\sim_{\equiv}$, but are related by $\equiv$. But since the arguments of sort P are those that represent processes, whereas the other sorts represent actions or information that is stored in process terms, this does not seem unreasonable. Consider the location set prefix $X :: p$ in Kiehn's local/global cause bisimulation. If there are two equivalent processes p and q, then it seems reasonable to want $X :: p$ and $X :: q$ to be equivalent, no matter how X has been constructed. So in fact, it is desirable for $X_1 :: p$ and $X_2 :: p$ to be equivalent if $X_1 = X_2$.

Example 4.3.4 (Congruence)
Since $\mathcal{E}_{\text{CCSSub}}$ is in *tyft/tyxt* format, and is compatible with any congruence on the set of label terms, bisimulation up to that congruence is a congruence with respect to all function symbols with range P. Let the congruence under consideration be the identity. Since

$$\mathsf{plus}(\mathsf{pref}(\mathsf{a}, \mathsf{nil}), \mathsf{pref}(\mathsf{a}, \mathsf{nil})) \sim_{\mathbf{Id}} \mathsf{pref}(\mathsf{a}, \mathsf{nil})$$

by the above theorem, it is known that

$$\mathsf{pref}(\mathsf{b}, \mathsf{plus}(\mathsf{pref}(\mathsf{a}, \mathsf{nil}), \mathsf{pref}(\mathsf{a}, \mathsf{nil})) \sim_{\mathbf{Id}} \mathsf{pref}(\mathsf{b}, \mathsf{pref}(\mathsf{a}, \mathsf{nil})) \qquad \blacksquare$$

This format has been taken almost as far as it can go in its current form. For most of the conditions that constrain the format, removal of a condition would result in the falsity of the above theorem. In the next section, I will give counter-examples to demonstrate the necessity of the constraints. There is, however, one condition that I have not yet shown to be necessary and that is the requirement that the variables in the p_i's be distinct from those in the λ_i's. I will discuss this further in the next section.

4.3.3 Counter-examples

In this section, I will give counter-examples to show that the constraints on the extended *tyft/tyxt* format cannot be relaxed for the above theorem. First, 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 [GV92] where counter-examples are given to show the constraints on these variables are necessary. So here I will focus on the relationship between the variables from $V_{\mathsf{\overline{P}}}$. All but the first counter-example have accompanying diagrams that demonstrate some of the transitions possible from the terms under consideration. I have chosen to highlight why the terms are not bisimilar as opposed to giving the LTS defined by the rules.

Counter-example 4.3.1 (Signature and rules for counter-examples) For these counter-examples, I will start with the following basic signature and add rules not in extended *tyft/tyxt* format for each counter-example

$$\Sigma = (\ \mathsf{P}, s, s';\ \ nil, ok, \{g_a \mid a \in A\};\ \ k, f, +, \|, h, h'\ )$$

with

$$\begin{array}{lll} ok :\to s' & & + : \mathsf{P}, \mathsf{P} \to \mathsf{P} \\ g_a :\to s & \forall\, a \in A & \|: \mathsf{P}, \mathsf{P} \to \mathsf{P} \\ nil :\to \mathsf{P} & & h : s, \mathsf{P} \to \mathsf{P} \\ k : \mathsf{P} \to \mathsf{P} & & h' : s, s, \mathsf{P} \to \mathsf{P} \\ f : s \to \mathsf{P} & & \end{array}$$

and the axiom

$$\frac{}{f(z_s) \xrightarrow{z_s} nil}$$

where the variable is subscripted to indicate its sort. This axiom can be seen as the BPA axiom $a \xrightarrow{a} nil$. I will write $+$ and $\|$ using infix notation.

I will assume that the congruence $\equiv$ equates no terms except g_a and g_b. ■

Counter-example 4.3.2 (Distinctness of label variables in premise labels) To see why the variables that appear in the labels of premises need to be distinct, consider adding the following rule

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

So it is clear that $f(g_a) \sim_{\equiv} f(g_b)$. However, $f(g_a) \parallel f(g_a) \not\sim_{\equiv} f(g_a) \parallel f(g_b)$ since $f(g_a) \parallel f(g_a) \xrightarrow{ok} nil \parallel nil$, but $f(g_a) \parallel f(g_b)$ cannot perform an ok action. This example is illustrated in Figure 4.2. ■

$$\begin{tikzcd}
f(g_a) \arrow[r, phantom, "\sim_\equiv"] \arrow[d, "g_a"'] & f(g_b) \arrow[d, "g_b"] & f(g_a) \parallel f(g_a) \arrow[r, phantom, "\not\sim_\equiv"] \arrow[d, "ok"'] & f(g_a) \parallel f(g_b) \\
nil \arrow[r, phantom, "\sim_\equiv"] \arrow[ur, phantom, "\equiv"] & nil & nil \parallel nil &
\end{tikzcd}$$

Figure 4.2: Counter-example 4.3.2

$$\begin{tikzcd}
nil \arrow[r, phantom, "\sim_\equiv"] & nil & h'(g_a, g_a, nil) \arrow[r, phantom, "\not\sim_\equiv"] \arrow[d, "ok"'] & h'(g_a, g_b, nil) \\
 & & nil &
\end{tikzcd}$$

Figure 4.3: Counter-example 4.3.3

Counter-example 4.3.3 (Distinctness of label variables in source of conclusion)

To see why the variables that appear in the arguments to the function in the source of the conclusion need to be distinct, consider adding the following axiom

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

It is clear that $nil \sim_\equiv nil$. However, it is not possible to show $h'(g_a, g_a, nil) \sim_\equiv h'(g_a, g_b, nil)$ since $h'(g_a, g_a, nil) \xrightarrow{ok} nil$, but $h'(g_a, g_b, nil)$ cannot perform an *ok* action. Figure 4.3 illustrates this counter-example. ■

Counter-example 4.3.4 (Distinctness of label variables in source of conclusion and labels of premises)

To see why it is not desirable to have the same variable appearing in an argument to the function in the source of the conclusion and in a label of a premise, consider

$$\begin{tikzcd}
f(g_a) \arrow[d, swap, "g_a"] & \sim_\equiv & f(g_b) \arrow[d, "g_b"] & h(g_a, f(g_a)) \arrow[d, swap, "ok"] & \not\sim_\equiv & h(g_a, f(g_b)) \\
|[alias=A]| & \equiv & & & & \\
nil & \sim_\equiv & nil & nil & &
\end{tikzcd}$$

Figure 4.4: Counter-example 4.3.4

adding the following rule

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

Hence $f(g_a) \sim_\equiv f(g_b)$; however $h(g_a, f(g_a)) \not\sim_\equiv h(g_a, f(g_b))$ since $h(g_a, f(g_a)) \xrightarrow{ok} nil$, but $h(g_a, f(g_b))$ cannot perform an *ok* action. This example is illustrated in Figure 4.4. ■

Next I have examples to show why compatibility is required.

Counter-example 4.3.5 (Compatibility of form)

First of all to see that why it is necessary for labels in premises to have compatibility of form, consider adding the rule

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

The rule is not compatible with $\equiv$, since $g_a \equiv g_b$ and for any substitution σ, there does not exist a substitution σ' such that $\sigma'(g_b) = \sigma(g_a)$. It is clear that $f(g_a) \sim_\equiv f(g_b)$; however $k(f(g_a)) \not\sim_\equiv k(f(g_b))$ since $k(f(g_a)) \xrightarrow{ok} nil$ but $k(f(g_b))$ cannot perform an *ok* action. Figure 4.5 illustrates this counter-example. ■

Counter-example 4.3.6 (Compatibility of variables)

Next to see why it is necessary for the values assigned to the variables to be equivalent, consider the following example. Add a new symbol $g : s \to s$ to the signature Σ and assume that $g(g_c) \equiv g(g_d)$. By the axiom $f(z_s) \xrightarrow{z_s} nil$,

$$\begin{tikzcd}
f(g_a) \arrow[d, "g_a"'] & \sim_\equiv \arrow[d, phantom, "\equiv"] & f(g_b) \arrow[d, "g_b"] & & k(f(g_a)) \arrow[d, "ok"'] & \not\sim_\equiv & k(f(g_b)) \\
nil & \sim_\equiv & nil & & nil & &
\end{tikzcd}$$

Figure 4.5: Counter-example 4.3.5

$f(g(g_c)) \sim_\equiv f(g(g_d))$. Then consider adding the rule

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

This rule is not compatible with $\equiv$ since $g(g_c) \equiv g(g_d)$ and although it is clear that given σ such that $\sigma(g(z_s)) = g(g_c)$, any σ' such that $\sigma'(z_s) = g_d$ is a suitable substitution to ensure that $\sigma(g(z_s)) \equiv \sigma'(g(z_s))$, it is not the case that $\sigma(z_s) \equiv \sigma'(z_s)$ (since $g_c \not\equiv g_d$). So $f(g(g_c)) \sim_\equiv f(g(g_d))$, but $k(f(g(g_c))) \not\sim_\equiv k(f(g(g_d)))$, since $k(f(g(g_c))) \xrightarrow{g_c} nil$, but $k(f(g(g_d)))$ can only perform a g_d action, and $g_c \not\equiv g_d$.

Similarly it can be shown that it is necessary for the values of the variables which appear both in labels of premises and in the target of the conclusion to be equivalent.

$$\frac{x \xrightarrow{g(z_s)} y}{k(x) \xrightarrow{ok} f(z_s)}.$$

By a similar argument to above, it is clear that this rule is not compatible with $\equiv$. $k(f(g(g_c))) \not\sim_\equiv k(f(g(g_d)))$, since $k(f(g(g_c))) \xrightarrow{ok} f(g_c)$, but $k(f(g(g_d))) \xrightarrow{ok} f(g_d)$, and $f(g_c) \not\sim_\equiv f(g_d)$. Figure 4.6 illustrates these counter-examples. ■

Counter-example 4.3.7 (More compatibility)

In a similar way, I can show why it is important to consider the variables in the arguments to the function in the source of the conclusion when defining compatibility. Consider adding the axiom

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

$$\begin{tikzcd}
f(g(g_c)) \arrow[dd, "g(g_c)"'] & \sim_\equiv & f(g(g_d)) \arrow[dd, "g(g_d)"] \\
 & \equiv & \\
nil & \sim_\equiv & nil
\end{tikzcd}
\qquad
\begin{tikzcd}
k(f(g(g_c))) \arrow[dd, "g_c"'] & \not\sim_\equiv & k(f(g(g_d))) \arrow[dd, "g_d"] \\
 & \not\equiv & \\
nil & & nil
\end{tikzcd}$$

$$\begin{tikzcd}
k(f(g(g_c))) \arrow[dd, "ok"'] & \not\sim_\equiv & k(f(g(g_d))) \arrow[dd, "ok"] \\
 & & \\
f(g_c) \arrow[dd, "g_c"'] & \not\sim_\equiv & f(g_d) \arrow[dd, "g_d"] \\
 & \not\equiv & \\
nil & & nil
\end{tikzcd}$$

Figure 4.6: Counter-example 4.3.6

The rule is not compatible with $\equiv$ since $g_a \equiv g_b$ and for any substitution σ, there does not exist a substitution σ' such that $\sigma'(g_b) = \sigma(g_a)$. It is clear that $nil \sim_\equiv nil$; however $h(g_a, nil) \not\sim_\equiv h(g_b, nil)$ (and I want them to be strongly equivalent because $g_a \equiv g_b$), since $h(g_a, nil) \xrightarrow{ok} nil$ but $h(g_b, nil)$ cannot perform an ok action.

(Note that this means that I cannot have an axiom of the form $f(g_a) \xrightarrow{g_a} x$, if I wish to have elements of $\{g_a \mid a \in A\}$ equivalent to each other. However, if I wish them all to be distinct then it is not a problem. But if I do wish to equate some of the elements, it requires having a more general rules such as $f(z_s) \xrightarrow{z_s} x$, which also allows us to prove that $f(g(g_c)) \sim_\equiv f(g(g_d))$ which may not be desirable. But since I am working in a many-sorted framework, it is possible to have a new sort s'' and let $g : s \to s''$, thereby preventing the equation of these two terms.)

I also need to show why the values of the variables that appear in the arguments to the function in the source of the conclusion and in the label on the conclusion need to be equivalent. Consider adding the axiom

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

This rule is not compatible with $\equiv$ (see Counter-example 4.3.6 for reasons). $nil \sim_\equiv nil$, but $h(g(g_c), nil) \not\sim_\equiv h(g(g_d), nil)$ since $h(g(g_c), nil) \xrightarrow{g_c} nil$, but $h(g(g_d), nil)$ can only perform a g_d action and $g_c \not\equiv g_d$.

Finally, I need to show why the values of the variables that appear in the arguments to the function in the source of the conclusion and in the target on the conclusion need to be equivalent. Consider adding the axiom

$$\frac{}{h(g(z_s), x) \xrightarrow{ok} f(z_s)}.$$

By a similar argument to above, it is clear that this rule is not compatible with $\equiv$. Clearly, $nil \sim_\equiv nil$, but $h(g(g_c), nil) \not\sim_\equiv h(g(g_d), nil)$ since $h(g(g_c), nil) \xrightarrow{ok} f(g_c)$, but $h(g(g_d), nil) \xrightarrow{ok} f(g_d)$ and $f(g_c) \not\sim_\equiv f(g_d)$. These counter-examples are illustrated in Figure 4.7. ■

Finally in this section, I will look at 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. This requirement can be considered as follows

- First note that in the case of a variable from $V_{\overline{\mathsf{P}}}$ appearing in the source of the conclusion as well as in the source of the premise, then this variable cannot appear in the label of a premise, since the variables in the source of the conclusion and in labels of premises must be distinct.

- Next consider the case when a variable that appears in the source of one premise also appears in the label of a different premise. Consider the eTSS used in Counter-Example 4.3.1 as well as the following rules

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

  I know $f(g_a) \sim_\equiv f(g_b)$ and can show that $k(f(g_a)) \sim_\equiv k(f(g_b))$ by the proofs

$$\frac{h(g_a, f(g_a)) \xrightarrow{g_a} f(g_a) \quad f(g_a) \xrightarrow{g_a} nil}{k(f(g_a)) \xrightarrow{ok} nil}$$

$$\begin{tikzcd}
\mathit{nil} & \sim_{\equiv} & \mathit{nil} & h(g_a, \mathit{nil}) \arrow[dd, "ok"'] & \not\sim_{\equiv} & h(g_b, \mathit{nil}) \\
 & & & & & \\
 & & & \mathit{nil} & &
\end{tikzcd}$$

$$\begin{tikzcd}
h(g(g_c), \mathit{nil}) \arrow[dd, "g_c"'] & \not\sim_{\equiv} & h(g(g_d), \mathit{nil}) \arrow[dd, "g_d"] \\
 & \not\equiv & \\
\mathit{nil} & & \mathit{nil}
\end{tikzcd}$$

$$\begin{tikzcd}
h(g(g_c), \mathit{nil}) \arrow[dd, "ok"'] & \not\sim_{\equiv} & h(g(g_d), \mathit{nil}) \arrow[dd, "ok"] \\
 & & \\
f(g_c) \arrow[dd, "g_c"'] & \not\sim_{\equiv} & f(g_d) \arrow[dd, "g_d"] \\
 & \not\equiv & \\
\mathit{nil} & & \mathit{nil}
\end{tikzcd}$$

Figure 4.7: Counter-example 4.3.7

and

$$\frac{h(g_b, f(g_b)) \xrightarrow{g_b} f(g_b) \quad f(g_b) \xrightarrow{g_b} nil}{k(f(g_b)) \xrightarrow{ok} nil}.$$

However, because the well-foundedness is used crucially in the proof of the theorem, the first premise of the second rule will be dealt with first since it has depth 1, whereas as the second has depth 2. Therefore z_s will be assigned the value g_a, since it does not appear in the source of the conclusion and hence it will be assigned the same value in the substitution being constructed as it had in the original substitution, and hence the following proof would be obtained.

$$\frac{h(g_a, f(g_b)) \xrightarrow{g_a} f(g_b) \quad f(g_b) \xrightarrow{g_a} nil}{k(f(g_b)) \xrightarrow{ok} nil}.$$

However, $f(g_b)$ cannot perform a g_a transition and hence this is not a valid proof of the final transition. Moreover, it is not clear how to modify the proof to obtain the correct substitution.

- Next consider the case when a variable that appears in the source of a premise also appears in the label of that premise. Then using the proof technique of the congruence theorem, the variable in the premise (which can not appear in the source of the conclusion) will be assigned the same value as under the original substitution. The question now arises as to whether this value is different to the one that may be assigned to the occurrence of the variable on the label by the use of compatibility. I have neither proof nor counter-example to answer this question.

As can be seen from the above discussion, the well-foundedness requirement as well as the order in which the new substitutions are created may make it impossible to find the correct substitution. Hence I retain the requirement that variables in sources of premises should be distinct from those in labels of premises. It may be possible to find a unification style technique that will work, however.

Fokkink and van Glabbeek [FvG96] have shown that any TSS in *tyft/tyxt* format can be expressed as a TSS in tree format. A TSS is in tree format when it consists of pure well-founded *xyft* rules, namely rules where the sources of premises can only be variables which appear either in the target of premises or in the source of the conclusion. 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 results from unification theory. It is not clear whether this results extend to the new format I have proposed, and this is an area for further work.

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.

4.3.4 Comparison with other formats

As mentioned earlier in the chapter, 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. How does this compare with formats that do not use negative premises or predicates?

As can be seen in Figure 2.12, there is a hierarchy of formats, and the most general format without negative premises and predicates is the *tyft/tyxt* format. In this section, I will compare this with the extended *tyft/tyxt* format (which I will also refer to as the new format for convenience).

The definition of *tyft/tyxt* format involves a single-sorted algebra for the operators of the process algebra and a infinite set of labels which are used in the rules in a schematic manner and are essentially atomic. *Tyft/tyxt* rules have the following forms:

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

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 from the set of labels. The labels are understood to be any labels from the label set, as long as multiple occurrences of a label within a rule are preserved. Proofs are constructed by finding substitutions for the variables, as well as choosing suitable labels. 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 new format, the first step is to start with a many-sorted algebra consisting of sorts, P and a label sort A; as well as operators which are not created from elements of the label set. The next step is to make all labels constants of sort A and then to take all operators created from elements of the label set, and modify them to take an argument of the label set type.

Finally, each rule needs to be modified. This is where schematic treatment of actions has some advantages, since matching labels across transitions in the premises, such as in the CCS parallel communication rule or in the CSP parallel merge rule can be easily done. First, consider rules where no matching occurs in contravention of the rules for label variables in the new format. Here, it is simple to replace each different label with a different variable of the sort A, so for example a rule such as

$$\frac{}{\mathsf{a}.x \xrightarrow{\mathsf{a}} x}$$

will become

$$\frac{}{\mathsf{pref}(z_{\mathsf{A}}, x) \xrightarrow{z_{\mathsf{A}}} x}.$$

In the case of matching of actions in a way which contravenes the conditions of label variables, there are two approaches. One involves the use of rule schemata which will be defined in Chapter 6, and the other involves the use of a specific type of algebra with constants to indicate undefined transitions, to represent the actual process algebra labels. I will discuss both of these in more detail; however they both deal with the issue successfully.

To see that the new format has more power than *tyft/tyxt* format consider the following example. In CCS with locations [BCHK94], the rule for communication involves only action transitions. As mentioned earlier in Section 2.2.1.2, the definition of bisimulation can be parameterised by a relation over the set of location strings (Loc^*), hence the strings that are matched in the transitions need only match up to the relation. I will assume for the purposes of this example, that the relation is the identity relation plus the pairs (l_1, l_2) and (l_2, l_1) for two distinguished elements l_1 and l_2. (Note that this is an equivalence relation.) Also assume that a new rule is introduced for the parallel operator involving communication (and the old rule removed), where communication can only take place if the actions have the same string of locations. This rule can be expressed as

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

On the surface, it is tempting to view this as a *tyft/tyxt* rule by treating the string of locations in a schematic manner. However doing this leads to an operator that does not preserve congruence, because of the fact that different strings may be equated by the relation over strings. To see this, consider the following rules (taken from the SOS for CCS with locations)

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

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.

So it can be seen from this example, that the introduction of a relation over elements of a label set can lead to a situation where congruence is lost. Relations over a label set are also introduced whenever the labels become more complex and hence it is necessary to equate labels which have been constructed in different ways, but which are to be viewed as semantically identical.

Hence it is possible to translate a transition system specification in *tyft/tyxt* format into one in extended *tyft/tyxt*, and moreover, the extended *tyft/tyxt* format can deal with more complex definitions of bisimulation, particularly those that equate different elements of the label set. The introduction of labels with different sorts will also have a rôle to play in the next chapter, where results are presented relating to the summing of eTSSs.

4.4 Conclusion

In this chapter, I have introduced a number of concepts that allow me to define extended transition system specifications, and moreover I have shown how a specific class of eTSSs give operators which are congruent with respect to bisimulation.

Chapter 5

Comparison results for the new format

5.1 Introduction

In this chapter, I will look at extensions to eTSSs. I prove a standard extension result, and then look at new definitions of extensions, and results which can be used to compare equivalences. These results look at combining two eTSSs and give conditions under which new transitions may be added or not. This allows one to compare equivalences on the original eTSSs and the new combined eTSS, and hence is a starting place for a comparison of equivalences based on a syntactic approach.

I first look at extension of an existing result to the new format; that of conservative extension where conditions are given for when two eTSSs can be combined without adding new transitions, and I compare this with a result by Verhoef [Ver94]. Next, I look at extensions up to bisimulation, and investigate under which conditions it is possible to achieve various relationships between the original equivalences and the equivalences of the combined eTSSs. There are two approaches to this and I present both of them.

The conservative extension result is an extension of the work of Groote and Vaandrager [GV92] to the new format where I use a different definition of the sum of two eTSSs, hence in the counter-examples section, I only deal with counter-examples that relate to label variables. The extension up to bisimulation results are new results which are not extensions of existing results, and hence I give full counter-examples for these results.

5.2 Sums of eTSSs and conservative extensions

I will now define the notion of the sum of two eTSSs which I require to define the notion of an extension.

Definition 5.2.1 (Sum of two signatures)
Let $\Sigma_i = (S_i \cup \{\mathsf{P}\}, F_i)$ for $i = 0, 1$ be two signatures such that $f \in F_0 \cap F_1$ implies that $\mathit{type}_0(f) = \mathit{type}_1(f)$. The *sum* of Σ_0 and Σ_1, $\Sigma_0 \oplus \Sigma_1$ is the signature

$$\Sigma_0 \oplus \Sigma_1 = (S_0 \cup S_1 \cup \{\mathsf{P}\}, F_0 \cup F_1). \qquad \blacksquare$$

It is clear that the sum of two suitable signatures is also suitable.

Definition 5.2.2 (Sum of two eTSSs)
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

$$\mathcal{E}_0 \oplus \mathcal{E}_1 = (\Sigma_0 \oplus \Sigma_1, R_0 \cup R_1). \qquad \blacksquare$$

The two preceding definitions are essentially the same as Groote and Vaandrager [GV92]. However, for reasons which will be become clearer later, I wish to work in the situation where the second component does not necessarily involve a sensible signature, and so I will extend these definitions to asymmetric definitions. This is reasonable in light of the fact that I often will be considering $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ as an extension of $\mathcal{E}_0$ and hence there is an inherent lack of symmetry in the view. I make the following definitions.

Definition 5.2.3 (Asymmetric sum of two signatures)
Let $\Sigma_i = (S_i \cup \{\mathsf{P}\}, F_i)$ for $i = 0, 1$ be two suitable signatures such that $\Sigma_0 \oplus \Sigma_1$ is defined. If Σ_0 is sensible, and $\Sigma_0 \oplus \Sigma_1$ is sensible, then I say that $\Sigma_0 \oplus \Sigma_1$ is *asymmetric* and denote it $\Sigma_0 \oplus\!\!> \Sigma_1$. $\blacksquare$

Note that in the case of an asymmetric sum, Σ_1 can be either sensible or not sensible.

Definition 5.2.4 (Asymmetric sum of two eTSSs)

Let $\mathcal{E}_i = (\Sigma_i, R_i)$ for $i = 0, 1$ be two eTSSs with $\mathcal{E}_0 \oplus \mathcal{E}_1$ defined. If $\Sigma_0 \oplus \Sigma_1$ is asymmetric, then I say that $\mathcal{E}_0 \oplus \mathcal{E}_1$ is *asymmetric* and denote it $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$. ■

Before I continue with results, I present an example that is based on the eTSS defined in Example 4.3.1 on page 69.

Example 5.2.1 (Example of the asymmetric sum of two eTSSs)

Consider the asymmetric sum of $\mathcal{E}_{\text{CCSSub}}$ and the eTSS consisting of Σ_{CCSSub} and the rules

$$\frac{x_1 \xrightarrow{z} y_1 \quad x_2 \xrightarrow{z} y_2}{\mathsf{plus}(x_1, x_2) \xrightarrow{z} \mathsf{plus}(y_1, y_2)} \qquad \frac{x \xrightarrow{z_1} y_1 \quad y_1 \xrightarrow{z_2} y_2}{x \xrightarrow{z_1} \mathsf{pref}(z_2, y_2)}.$$

Call this new eTSS $\mathcal{E}$. By adding these rules, it is possible to lose the branching time characteristic of CCS behaviour. Consider that in $\mathcal{E}_{\text{CCSSub}}$

$$\begin{array}{l}\mathsf{pref}(\mathsf{a}, \mathsf{plus}(\mathsf{pref}(\mathsf{b}, \mathsf{nil}), \mathsf{pref}(\mathsf{c}, \mathsf{nil}))) \\ \quad \not\sim^{\mathcal{E}_{\text{CCSSub}}}_{\mathbf{Id}} \quad \mathsf{plus}(\mathsf{pref}(\mathsf{a}, \mathsf{pref}(\mathsf{b}, \mathsf{nil})), \mathsf{pref}(\mathsf{a}, \mathsf{pref}(\mathsf{c}, \mathsf{nil})))\end{array}$$

but

$$\begin{array}{l}\mathsf{pref}(\mathsf{a}, \mathsf{plus}(\mathsf{pref}(\mathsf{b}, \mathsf{nil}), \mathsf{pref}(\mathsf{c}, \mathsf{nil}))) \\ \quad \sim^{\mathcal{E}}_{\mathbf{Id}} \quad \mathsf{plus}(\mathsf{pref}(\mathsf{a}, \mathsf{pref}(\mathsf{b}, \mathsf{nil})), \mathsf{pref}(\mathsf{a}, \mathsf{pref}(\mathsf{c}, \mathsf{nil})))\end{array}$$

The two new rules have added transitions to each process in such a way to make them bisimilar. The second new rule uses lookahead which is a feature of *tyft/tyxt*-style formats, and reduces branching behaviour to traces. Figure 5.1 illustrates the two terms and their transitions. The transitions that are created by the formation of the sum are indicated by dashed lines. ■

Now I look at some results relating to sums of eTSSs.

Definition 5.2.5 (Conservative extension)

Let $\mathcal{E}_i = (\Sigma_i, R_i)$ for $i = 0, 1$ be two eTSSs with $\mathcal{E} = \mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ defined and let $\mathcal{E} = (\Sigma, R)$. I say that $\mathcal{E}$ is a *conservative extension of* $\mathcal{E}_0$ and that $\mathcal{E}_1$ *can be added conservatively* to $\mathcal{E}_0$ if for all $t_0 \in \mathbf{T}(\Sigma_0)_{\mathsf{P}}$, $\alpha \in \mathbf{T}(\Sigma)_{\overline{\mathsf{P}}}$, and $t \in \mathbf{T}(\Sigma)_{\mathsf{P}}$,

$$\mathcal{E} \vdash t_0 \xrightarrow{\alpha} t \iff \mathcal{E}_0 \vdash t_0 \xrightarrow{\alpha} t. \qquad ■$$

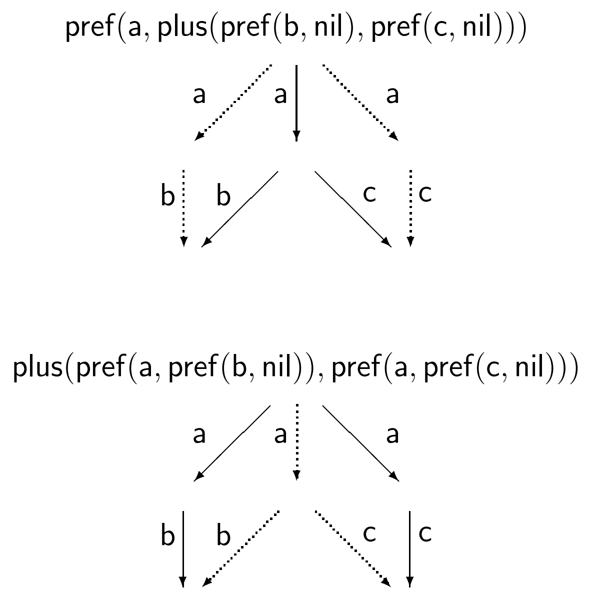
pref(a, plus(pref(b, nil), pref(c, nil)))
a
a
a
b
b
c
c
plus(pref(a, pref(b, nil)), pref(a, pref(c, nil)))
a
a
a
b
b
c
c

Figure 5.1: Example 5.2.1

Note that the implication $\mathcal{E} \vdash t_0 \xrightarrow{\alpha} t \Leftarrow \mathcal{E}_0 \vdash t_0 \xrightarrow{\alpha} t$ holds trivially since all transitions generated by $\mathcal{E}_0$ are also generated by $\mathcal{E}$. The eTSS $\mathcal{E}$ in Example 5.2.1 is not a conservative extension. I will now give a result which gives conditions under which eTSSs can be conservatively extended. I first require some definitions about the label variables which can appear in rules.

Definition 5.2.6 (Label-free variable)
Let $\mathcal{E} = (\Sigma, R)$ be an eTSS, and let r be a rule in R. A variable in $\mathrm{Var}_{\overline{\mathsf{P}}}(r)$ is called *label-free* if it does not occur in the label of a premise or in the left hand side of the conclusion. ■

Definition 5.2.7 (Label-pure rule)
Let $\mathcal{E} = (\Sigma, R)$ be an eTSS. A rule $r \in R$ is called *label-pure* if it contains no label-free variables from $V_{\overline{\mathsf{P}}}$. The eTSS $\mathcal{E}$ is called *label-pure* if all its rules are label-pure. ■

The following theorem works by preventing the addition of rules which may cause new transitions to occur from terms in the first signature.

Theorem 5.2.1 (Conservative extension)

Let $\mathcal{E}_0 = (\Sigma_0, R_0)$ be an eTSS in pure, label-pure extended *tyft/tyxt* format and let $\mathcal{E}_1 = (\Sigma_1, R_1)$ be an eTSS in extended *tyft* format such that there is no rule in R_1 that contains a function symbol from Σ_0 in the source of the conclusion. Let $\mathcal{E} = \mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ be defined. Then $\mathcal{E}_1$ can be added conservatively to $\mathcal{E}_0$.

Proof: I will use a similar strategy to that used in Theorem 4.3.1. Let $\mathcal{E} = (\Sigma, R)$. Let $t_0 \in \mathbf{T}(\Sigma_0)_{\mathsf{P}}$, $\alpha \in \mathbf{T}(\Sigma)_{\overline{\mathsf{P}}}$, $t \in \mathbf{T}(\Sigma)_{\mathsf{P}}$. Let T be a proof of $t_0 \xrightarrow{\alpha} t$ from $\mathcal{E}$. With induction on the length of T, I will show that T is also a proof of $t_0 \xrightarrow{\alpha} t$ from $\mathcal{E}_0$, namely that $\alpha \in \mathbf{T}(\Sigma_0)_{\overline{\mathsf{P}}}$ and $t \in \mathbf{T}(\Sigma_0)_{\mathsf{P}}$.

Let r be the last rule used in T. Since $t_0 \in \mathbf{T}(\Sigma_0)_{\mathsf{P}}$ and all rules in R_1 are extended *tyft* and contain no functions from Σ_0 in the source of their conclusions, r must be in R_0. Suppose r is pure, label-pure extended *tyft* (the case that r is pure, label-pure extended *tyxt* can be proved in a similar fashion). I proceed by induction on the length of T. First suppose that r is an axiom, i.e. r is

$$\frac{}{f(\eta_1, \ldots, \eta_m, x_1, \ldots, x_n) \xrightarrow{\lambda} p}$$

and let σ be the substitution used in the last step of the proof T. Hence $\sigma(f(\eta_1, \ldots, \eta_m, x_1, \ldots, x_n)) = t_0$, $\sigma(\lambda) = \alpha$ and $\sigma(p) = t$, so $\sigma(x) \in \mathbf{T}(\Sigma_0)$ for all $x \in \bigcup_{1 \leqslant i \leqslant n} x_i \cup \bigcup_{1 \leqslant k \leqslant m} \mathrm{Var}_{\overline{\mathsf{P}}}(\eta_k)$. Since r is pure, $\mathrm{Var}_{\mathsf{P}}(p) \subseteq \bigcup_{1 \leqslant i \leqslant n} x_i$, and since r is label-pure, $\mathrm{Var}_{\overline{\mathsf{P}}}(p) \cup \mathrm{Var}_{\overline{\mathsf{P}}}(\lambda) \subseteq \bigcup_{1 \leqslant k \leqslant m} \mathrm{Var}_{\overline{\mathsf{P}}}(\eta_k)$. Therefore, it is immediate that $\alpha \in \mathbf{T}(\Sigma_0)_{\overline{\mathsf{P}}}$, $t \in \mathbf{T}(\Sigma_0)_{\mathsf{P}}$ and $\mathcal{E}_0 \vdash t_0 \xrightarrow{\alpha} t$.

Next suppose that r is

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

and let σ be the substitution used in the last step of the proof T. Hence $\sigma(f(\eta_1, \ldots, \eta_m, x_1, \ldots, x_n)) = t_0$, $\sigma(\lambda) = \alpha$ and $\sigma(p) = t$. Again I need to consider the variables in r.

By considering the dependency graph G of the premises of r, $\mathrm{depth}(x) \in \mathbb{N}$ can be defined for all $x \in \mathrm{Var}_{\mathsf{P}}(r)$ in a similar fashion to the proof of Theorem 4.3.1. Define

- $X = \{x_i \mid 1 \leqslant i \leqslant n\}$

- $Y = \{y_i \mid i \in I\}$
- $Y_d = \{y \in Y \mid \text{depth}(y) = d\}$ for $n \geqslant 0$.

Observe that for any variable $x \in X$, $\text{depth}(x) = 0$, and the sets Y_d form a partition of Y. Next I need to consider the labels from $V_{\overline{\mathsf{P}}}$ that appear in r. Define

- $Z = \bigcup_{1 \leqslant k \leqslant m} \text{Var}_{\overline{\mathsf{P}}}(\eta_k)$
- $Z' = \bigcup_{i \in I} \text{Var}_{\overline{\mathsf{P}}}(\lambda_i)$
- $Z'_d = \bigcup_{y_i \in Y_d} \text{Var}_{\overline{\mathsf{P}}}(\lambda_i)$.

Note that since r is label-pure, $\text{Var}_{\overline{\mathsf{P}}}(r) = Z \cup Z'$. Also the sets Z'_d form a partition of Z' since variables cannot be shared between labels of premises.

With induction on d, I prove that $\sigma(x) \in \mathbf{T}(\Sigma_0)_{\mathsf{P}}$ for all $x \in X \cup Y$, and that $\sigma(z) \in \mathbf{T}(\Sigma_0)_{\overline{\mathsf{P}}}$ for all $z \in Z \cup Z'$.

Because $t_0 \in \mathbf{T}(\Sigma_0)_{\mathsf{P}}$ and $\sigma(f(\eta_1, \ldots, \eta_m, x_1, \ldots, x_n)) = t_0$, $\sigma(x) \in \mathbf{T}(\Sigma_0)_{\mathsf{P}}$ for all $x \in X$, and $\sigma(z) \in \mathbf{T}(\Sigma_0)_{\overline{\mathsf{P}}}$ for all $z \in Z$.

Let $d \in \mathbb{N}$ and suppose that $\sigma(x) \in \mathbf{T}(\Sigma_0)_{\mathsf{P}}$ for all $x \in X \cup Y_0 \cup \ldots \cup Y_{d-1}$ and $\sigma(z) \in \mathbf{T}(\Sigma_0)_{\overline{\mathsf{P}}}$ for all $z \in Z \cup Z'_0 \cup \ldots \cup Z'_{d-1}$. Let $y^* \in Y_d$, then there is a unique $i \in I$ such that $y^* = y_i$. Since r is pure, $\text{Var}_{\mathsf{P}}(p_i) \subseteq X \cup Y_0 \cup \ldots \cup Y_{d-1}$, and since r is label-pure, $\text{Var}_{\overline{\mathsf{P}}}(p_i) \subseteq Z$, therefore $\sigma(p_i) \in \mathbf{T}(\Sigma_0)_{\mathsf{P}}$. Hence, by the the induction hypothesis, $\sigma(\lambda_i) \in \mathbf{T}(\Sigma_0)_{\overline{\mathsf{P}}}$, $\sigma(y_i) \in \mathbf{T}(\Sigma_0)_{\mathsf{P}}$ and $\mathcal{E}_0 \vdash t_0 \xrightarrow{\alpha} t$.

So this is true for all $y \in Y_d$, and 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 $\sigma(z) \in \mathbf{T}(\Sigma_0)_{\overline{\mathsf{P}}}$ for all $z \in Z \cup Z'$. Since r is pure and label-pure, $\text{Var}(p) \subseteq X \cup Y \cup Z \cup Z'$, therefore $t = \sigma(p) \in \mathbf{T}(\Sigma_0)_{\mathsf{P}}$, and $\text{Var}(\lambda) \subseteq Z \cup Z'$, so $\alpha = \sigma(\lambda) \in \mathbf{T}(\Sigma_0)_{\overline{\mathsf{P}}}$ and $\mathcal{E}_0 \vdash t_0 \xrightarrow{\alpha} t$. ■

Note that the sum in Example 5.2.1 is not a conservative extension, since new transitions are added. Next, I present an example which is a conservative extension.

Example 5.2.2 (Conservative extension)

Consider $\mathcal{E}_{\text{CCSPar}}$ consisting of Σ_{CCSSub} with the added symbol $\mathsf{par} : \mathsf{P}, \mathsf{P} \to \mathsf{P}$ (call this Σ_{CCSPar}) and the rules

$$\frac{x \xrightarrow{z} y}{\mathsf{par}(x, x') \xrightarrow{z} \mathsf{par}(y, x')} \qquad \frac{x \xrightarrow{z} y}{\mathsf{par}(x', x) \xrightarrow{z} \mathsf{par}(x', y)}$$

Then this is an conservative extension of $\mathcal{E}_{\text{CCSPar}}$ since the added rules have the correct form. Clearly these rules cannot add new transitions to the terms of the first signature. ■

5.2.1 Counter-examples

I will now give some counter-examples to show why label-pureness is required.

Counter-example 5.2.1 (Label-pureness)

Consider the many-sorted signature

$$\Sigma = (\ P, s, s';\ \ ok, g, nil;\ \ f, f', f''\ )$$

with $ok :\to s'$, $g :\to s$, $nil :\to P$, $f : P \to P$, $f' : s \to P$ and $f'' : s, P \to P$.

Consider the eTSS $\mathcal{E}$ consisting of Σ and the axiom $f(x) \xrightarrow{z_s} nil$. If I add a new constant $g' :\to s$, then I obtain a new transition $f(nil) \xrightarrow{g'} nil$ which is not in $TS(\mathcal{E})$ since $g' \notin \mathbf{T}(\Sigma)$.

Next, consider the eTSS $\mathcal{E}'$ consisting of Σ and the axiom $f(x) \xrightarrow{ok} f'(z_s)$. If I add a new constant $g' :\to s$, then I obtain a new transition $f(nil) \xrightarrow{ok} f'(g')$ which is not in $TS(\mathcal{E}')$ since $f'(g') \notin \mathbf{T}(\Sigma)$.

Finally consider the eTSS $\mathcal{E}''$ consisting of Σ and the rules

$$\frac{}{f''(z_s, x) \xrightarrow{ok} f'(z_s)} \qquad \frac{f''(z_s, x) \xrightarrow{ok} y}{f(x) \xrightarrow{ok} y}$$

Then if I add a new constant $g' :\to s$, I obtain a new transition $f(nil) \xrightarrow{ok} f'(g')$ by the proof

$$\frac{f''(g', nil) \xrightarrow{ok} f'(g')}{f(nil) \xrightarrow{ok} f'(g')}$$

and this transition is not in $TS(\mathcal{E}'')$ since $f'(g') \notin \mathbf{T}(\Sigma)$.

$$\begin{tikzcd}
& f(nil) \arrow[dl, "g"'] \arrow[dr, dotted, "g'"] & \\
nil & & nil
\end{tikzcd}$$

$$\begin{tikzcd}
& f(nil) \arrow[dl, "ok"'] \arrow[dr, dotted, "ok"] & \\
f'(g) & & f'(g')
\end{tikzcd}$$

Figure 5.2: Counter-example 5.2.1

The terms and transitions are illustrated in Figure 5.2. The solid arrows indicate transitions from the original eTSS, and the dashed arrows indicate transitions that result from the summing of the two eTSSs. The second and third counter-examples have the same diagram. ■

Since all of these counter-examples involve the addition of new terms with an existing sort, it may be possible to have a slightly different result where label-pureness is not required, but there is a restriction on what can be added. I will return to this point later in the chapter.

5.2.2 Related work

Verhoef has given a general extension theorem [Ver94] for the *panth* format which has weaker conditions than the above results. He works with relations and predicates, and by grouping together transitions with the same labels as relations over processes, is able to use a similar mechanism to that which I will use later in the chapter by using sorts. The conditions required for his extension result can be expressed as follows in the extended *tyft/tyxt* format. Note that I am not considering negative premises or predicates here.

Let $\mathcal{E}_0 = (\Sigma_0, R_0)$ be an eTSS in pure, label-pure extended *tyft/tyxt* format and let $\mathcal{E}_1 = (\Sigma_1, R_1)$ be an eTSS in extended *tyft/tyxt* format. Let $\mathcal{E} = \mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ be defined. For any rule $r \in R_1$ in *tyxt* format or *tyft* format with the function symbol from Σ_0

1. r is pure, label-pure and well-founded
2. all terms in the sources of premises come from $\mathbb{T}(\Sigma_0)$
3. there is a premise consisting of only terms from $\mathbb{T}(\Sigma_0)$ in source and target, and where the label term has a sort in $\Sigma_1 - \Sigma_0$.

Then $\mathcal{E}_1$ can be added conservatively to $\mathcal{E}_0$.

So in comparison to Theorem 5.2.1, this result allows *tyxt* rules and *tyft* rules with function symbols from Σ_0, in R_1. However, the third condition requires that one of the premises combines process terms from the first signature and label terms from the second signature, and hence there can be no suitable transitions to fit this premise and hence allow the rule to be used in any proof. In the next section, I will use the differentiating power of the different sorts in a similar fashion to ensure that only the 'right' transitions are added, although I take a slightly different approach in that instead of specifying which terms can appear in the premises, I specify the sorts of label terms appearing in the conclusions. This approach has the advantage that it permits new axioms. Also the results I obtain relate to different notions of extension, specifically to those that preserve transitions up to bisimulation.

5.3 Extensions up to bisimulation

So how does the notion of a conservative extension relate to definitions of bisimulation? A notion of a conservative extension up to bisimulation can be defined in the following manner.

Definition 5.3.1 (Conservative extension up to bisimulation with respect to an equivalence)
Let $\mathcal{E}_i = (\Sigma_i, R_i)$ for $i = 0, 1$ be two eTSSs with $\mathcal{E} = \mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ defined and let

$\mathcal{E} = (\Sigma, R)$. Let $\equiv$ be a congruence on $\mathbf{T}(\Sigma)_{\overline{\mathsf{P}}}$ compatible with $\mathcal{E}$. I say that $\mathcal{E}$ is a *conservative extension of* $\mathcal{E}_0$ *up to bisimulation with respect to* $\equiv$ if for all $t_0, u_0 \in \mathbf{T}(\Sigma_0)_{\mathsf{P}}$,

$$t_0 \sim^{\mathcal{E}}_{\equiv} u_0 \Longleftrightarrow t_0 \sim^{\mathcal{E}_0}_{\equiv} u_0. \qquad \blacksquare$$

This type of definition could be useful in comparing semantic equivalences. In what follows I will present and discuss a number of definitions, as well as looking at whether they are likely to be useful to achieve my goals. The following result is immediate.

Proposition 5.3.1 (Conservative extension implies conservative extension up to bisimulation with respect to an equivalence)
Let $\mathcal{E}_i = (\Sigma_i, R_i)$ for $i = 0, 1$ be two eTSSs with $\mathcal{E} = \mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ defined and let $\mathcal{E} = (\Sigma, R)$. Let $\equiv$ be a congruence on $\mathbf{T}(\Sigma)_{\overline{\mathsf{P}}}$ compatible with $\mathcal{E}$. If $\mathcal{E}$ is a conservative extension of $\mathcal{E}_0$, then $\mathcal{E}$ is a conservative extension of $\mathcal{E}_0$ up to bisimulation with respect to $\equiv$. $\blacksquare$

The proof of this proposition relies on that fact that for every action one process can make, the other process can make the syntactically identical action. I now present a slight different definition of conservative extension—one which is relative to an congruence.

Definition 5.3.2 (Conservative extension with respect to an equivalence)
Let $\mathcal{E}_i = (\Sigma_i, R_i)$ for $i = 0, 1$ be two eTSSs with $\mathcal{E} = \mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ defined and let $\mathcal{E} = (\Sigma, R)$. Let $\equiv$ be a congruence on $\mathbf{T}(\Sigma)_{\overline{\mathsf{P}}}$ compatible with $\mathcal{E}$. I say that $\mathcal{E}$ is a *conservative extension of* $\mathcal{E}_0$ *with respect to* $\equiv$ and that $\mathcal{E}_1$ *can be added conservatively to* $\mathcal{E}_0$ *with respect to* $\equiv$ if for all $t_0 \in \mathbf{T}(\Sigma_0)_{\mathsf{P}}$, $\alpha \in \mathbf{T}(\Sigma)_{\overline{\mathsf{P}}}$, and $t \in \mathbf{T}(\Sigma)_{\mathsf{P}}$, then there exists $\alpha_0 \in \mathbf{T}(\Sigma_0)_{\overline{\mathsf{P}}}$ such that

$$\mathcal{E} \vdash t_0 \xrightarrow{\alpha} t \Rightarrow \mathcal{E}_0 \vdash t_0 \xrightarrow{\alpha_0} t \text{ and } \alpha \equiv \alpha_0. \qquad \blacksquare$$

The reverse implication that there exists $\alpha \in \mathbf{T}(\Sigma)_{\overline{\mathsf{P}}}$ such that $\mathcal{E}_0 \vdash t_0 \xrightarrow{\alpha_0} t \Rightarrow \mathcal{E} \vdash t_0 \xrightarrow{\alpha} t$ and $\alpha \equiv \alpha_0$ is trivial since $\mathcal{E} \vdash t_0 \xrightarrow{\alpha_0} t$. I obtain a new result.

Proposition 5.3.2 (Conservative extension with respect to an equivalence implies conservative extension up to bisimulation with respect to that equivalence)
Let $\mathcal{E}_i = (\Sigma_i, R_i)$ for $i = 0, 1$ be two eTSSs with $\mathcal{E} = \mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ defined and let $\mathcal{E} = (\Sigma, R)$. Let $\equiv$ be a congruence on $\mathbf{T}(\Sigma)_{\overline{\mathsf{P}}}$ compatible with $\mathcal{E}$. If $\mathcal{E}$ is a conservative extension of $\mathcal{E}_0$ with respect to $\equiv$, then $\mathcal{E}$ is a conservative extension of $\mathcal{E}_0$ up to bisimulation with respect to $\equiv$. ■

However, it is not clear that this definition is useful, since the requirement that both transitions result in the same state is a very strong condition. Hence I will work with the notion of conservative extension up to bisimulation with respect to an equivalence, and related definitions which I present immediately.

Definition 5.3.3 (Refining extension up to bisimulation with respect to an equivalence)
Let $\mathcal{E}_i = (\Sigma_i, R_i)$ for $i = 0, 1$ be two eTSSs with $\mathcal{E} = \mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ defined and let $\mathcal{E} = (\Sigma, R)$. Let $\equiv$ be a congruence on $\mathbf{T}(\Sigma)_{\overline{\mathsf{P}}}$ compatible with $\mathcal{E}$. I say that $\mathcal{E}$ is a *refining extension of $\mathcal{E}_0$ up to bisimulation with respect to* $\equiv$ if for all $t_0, u_0 \in \mathbf{T}(\Sigma_0)_{\mathsf{P}}$,

$$t_0 \sim^{\mathcal{E}}_{\equiv} u_0 \Rightarrow t_0 \sim^{\mathcal{E}_0}_{\equiv_0} u_0.$$

I will also write this as $\sim^{\mathcal{E}}_{\equiv} \subseteq \sim^{\mathcal{E}_0}_{\equiv}$. ■

Definition 5.3.4 (Abstracting extension up to bisimulation with respect to an equivalence)
Let $\mathcal{E}_i = (\Sigma_i, R_i)$ for $i = 0, 1$ be two eTSSs with $\mathcal{E} = \mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ defined and let $\mathcal{E} = (\Sigma, R)$. Let $\equiv$ be a congruence on $\mathbf{T}(\Sigma)_{\overline{\mathsf{P}}}$ compatible with $\mathcal{E}$. I say that $\mathcal{E}$ is an *abstracting extension of $\mathcal{E}_0$ up to bisimulation with respect to* $\equiv$ if for all $t_0, u_0 \in \mathbf{T}(\Sigma_0)_{\mathsf{P}}$,

$$t_0 \sim^{\mathcal{E}_0}_{\equiv_0} u_0 \Rightarrow t_0 \sim^{\mathcal{E}}_{\equiv} u_0.$$

I will also write this as $\sim^{\mathcal{E}_0}_{\equiv} \subseteq \sim^{\mathcal{E}}_{\equiv}$. ■

The next result is immediate.

Proposition 5.3.3 (Refining and abstracting extensions up to bisimulation if and only if conservative extension up to bisimulation)
Let $\mathcal{E}_i = (\Sigma_i, R_i)$ for $i = 0, 1$ be two eTSSs with $\mathcal{E} = \mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ defined and let $\mathcal{E} = (\Sigma, R)$. Let $\equiv$ be a congruence on $\mathbf{T}(\Sigma)_{\overline{\mathsf{P}}}$ compatible with $\mathcal{E}$. Then $\mathcal{E}$ 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, I will look at conditions under which I can achieve these types of extensions. As this is fairly complex, I will proceed by making a number of definitions that capture these conditions. I then prove a general lemma that I need for the theorems, and then look at some definitions and results about congruences and sums of eTSSs. Finally I proceed to prove the two theorems.

First I define to different kinds of rule sets for asymmetric sums of eTSSs.

Definition 5.3.5 (Type-1 asymmetric sum)
Let $\Sigma_i = (S_i \cup \{\mathsf{P}\}, F_i)$ for $i = 0, 1$ be two signatures, and let $\mathcal{E}_i = (\Sigma_i, R_i)$ for $i = 0, 1$ be two eTSSs in extended *tyft/tyxt* format such that $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ is defined. Then the asymmetric sum $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ is said to be of *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,
- there is no extended *tyxt* rule in R_1 that has a conclusion label of a sort from S_0. ■

These conditions go beyond those of the conservative extension result, since *tyxt* rules and *tyft* rules with function symbols from Σ_0 are now allowed, but with additional condition that the labels in the conclusion do not have sorts from S_0. However, for one result a stronger condition is required where no *tyft* rules with function symbols from Σ_0 are allowed.

Definition 5.3.6 (Type-0 asymmetric sum)
Let $\Sigma_i = (S_i \cup \{\mathsf{P}\}, F_i)$ for $i = 0, 1$ be two signatures and let $\mathcal{E}_i = (\Sigma_i, R_i)$ for $i = 0, 1$ be two eTSSs in extended *tyft/tyxt* format such that $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ is defined.

Then the asymmetric sum $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ is said to be *type-0* if it is type-1 and there is no extended *tyft* rule in R_1 that 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.

I require the following lemma first before I can prove the main results of this section. This lemma says that given a type-1 sum and a proof of a transition where the last rules used came from R_0 then the transition can be proved in $\mathcal{E}_0$.

Lemma 5.3.1 (Application of rules)
Let $\mathcal{E}_i = (\Sigma_i, R_i)$ for $i = 0, 1$ be two eTSSs in extended *tyft/tyxt* format such that $\mathcal{E} = \mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ is defined. Moreover, let $\mathcal{E}_0$ be pure and label-pure, and 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 $\mathcal{E} = (\Sigma, R)$. Let $t_0 \in \mathbf{T}(\Sigma_0)_{\mathsf{P}}$, $\alpha \in \mathbf{T}(\Sigma)_{\overline{\mathsf{P}}}$, $t \in \mathbf{T}(\Sigma)_{\mathsf{P}}$. 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. With induction on the length of T, I will show that T is also a proof of $t_0 \xrightarrow{\alpha} t$ from $\mathcal{E}_0$ and that $\alpha \in \mathbf{T}(\Sigma_0)_{\overline{\mathsf{P}}}$ and $t \in \mathbf{T}(\Sigma_0)_{\mathsf{P}}$.

Let r be the last rule used in T. I assume it is an extended *tyft/tyxt* rule from R_0. Suppose r is pure, label-pure extended *tyft* (the case that r is pure, label-pure extended *tyxt* can be proved in a similar fashion). I proceed by induction on the length of T. First suppose that r is an axiom, i.e. r is

$$\frac{}{f(\eta_1, \ldots, \eta_m, x_1, \ldots, x_n) \xrightarrow{\lambda} p}$$

and let σ be the substitution used in the last step of the proof T. Hence $\sigma(f(\eta_1, \ldots, \eta_m, x_1, \ldots, x_n)) = t_0$, $\sigma(\lambda) = \alpha$ and $\sigma(p) = t$, so $\sigma(x) \in \mathbf{T}(\Sigma_0)$ for all $x \in \bigcup_{1 \leqslant i \leqslant n} x_i \cup \bigcup_{1 \leqslant k \leqslant m} \mathrm{Var}_{\overline{\mathsf{P}}}(\eta_k)$. Since r is pure, $\mathrm{Var}_{\mathsf{P}}(p) \subseteq \bigcup_{1 \leqslant i \leqslant n} x_i$, and since r is label-pure, $\mathrm{Var}_{\overline{\mathsf{P}}}(p) \cup \mathrm{Var}_{\overline{\mathsf{P}}}(\lambda) \subseteq \bigcup_{1 \leqslant k \leqslant m} \mathrm{Var}_{\overline{\mathsf{P}}}(\eta_k)$. Therefore, it is immediate that $\alpha \in \mathbf{T}(\Sigma_0)_{\overline{\mathsf{P}}}$, $t \in \mathbf{T}(\Sigma_0)_{\mathsf{P}}$ and $\mathcal{E}_0 \vdash t_0 \xrightarrow{\alpha} t$.

Next suppose that r is

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

and let σ be the substitution used in the last step of the proof T. Hence $\sigma(f(\eta_1, \ldots, \eta_m, x_1, \ldots, x_n)) = t_0$, $\sigma(\lambda) = \alpha$ and $\sigma(p) = t$.

By considering the dependency graph G of the premises of r, $\mathrm{depth}(x) \in \mathbb{N}$ can be defined for all $x \in \mathrm{Var}_{\mathsf{P}}(r)$ in a similar fashion to the proof of Theorem 4.3.1. Define

- $X = \{x_i \mid 1 \leqslant i \leqslant n\}$
- $Y = \{y_i \mid i \in I\}$
- $Y_d = \{y \in Y \mid \mathrm{depth}(y) = d\}$ for $n \geqslant 0$.

Observe that for any variable $x \in X$, $\mathrm{depth}(x) = 0$, and the sets Y_d form a partition of Y. Next I need to consider the labels from $V_{\overline{\mathsf{P}}}$ that appear in r. Define

- $Z = \bigcup_{1 \leqslant k \leqslant m} \mathrm{Var}_{\overline{\mathsf{P}}}(\eta_k)$
- $Z' = \bigcup_{i \in I} \mathrm{Var}_{\overline{\mathsf{P}}}(\lambda_i)$
- $Z'_d = \bigcup_{y_i \in Y_d} \mathrm{Var}_{\overline{\mathsf{P}}}(\lambda_i)$.

Note that since r is label-pure, $\mathrm{Var}_{\overline{\mathsf{P}}}(r) = Z \cup Z'$. Also the sets Z'_d form a partition of Z' since variables cannot be shared between labels of premises.

With induction on d, I prove that $\sigma(x) \in \mathbf{T}(\Sigma_0)_{\mathsf{P}}$ for all $x \in X \cup Y$, and that $\sigma(z) \in \mathbf{T}(\Sigma_0)_{\overline{\mathsf{P}}}$ for all $z \in Z \cup Z'$. Because $t_0 \in \mathbf{T}(\Sigma_0)_{\mathsf{P}}$ and because also $\sigma(f(\eta_1, \ldots, \eta_m, x_1, \ldots, x_n)) = t_0$, $\sigma(x) \in \mathbf{T}(\Sigma_0)_{\mathsf{P}}$ for all $x \in X$, and $\sigma(z) \in \mathbf{T}(\Sigma_0)_{\overline{\mathsf{P}}}$ for all $z \in Z$.

Let $d \in \mathbb{N}$ and suppose that $\sigma(x) \in \mathbf{T}(\Sigma_0)_{\mathsf{P}}$ for all $x \in X \cup Y_0 \cup \ldots \cup Y_{d-1}$ and $\sigma(z) \in \mathbf{T}(\Sigma_0)_{\overline{\mathsf{P}}}$ for all $z \in Z \cup Z'_0 \cup \ldots \cup Z'_{d-1}$. Let $y^* \in Y_d$, then there is a unique $i \in I$ such that $y^* = y_i$. Since r is pure, $\mathrm{Var}_{\mathsf{P}}(p_i) \subseteq X \cup Y_0 \cup \ldots \cup Y_{d-1}$, and since r is label-pure, $\mathrm{Var}_{\overline{\mathsf{P}}}(p_i) \subseteq Z$, therefore $\sigma(p_i) \in \mathbf{T}(\Sigma_0)_{\mathsf{P}}$. Consider the transition $\sigma(p_i) \xrightarrow{\sigma(\lambda_i)} \sigma(y_i)$. It is clear that this transition must be generated by a proof whose last step involves a rule from R_0, since $\mathcal{E} = \mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ is type-1 and hence no extended *tyft/tyxt* rule from R_1 can generate a transition that has a label with the same sort as $\sigma(\lambda_i)$. Hence, by the the induction hypothesis, $\sigma(\lambda_i) \in \mathbf{T}(\Sigma_0)_{\overline{\mathsf{P}}}$, $\sigma(y_i) \in \mathbf{T}(\Sigma_0)_{\mathsf{P}}$ and $\mathcal{E}_0 \vdash t_0 \xrightarrow{\alpha} t$.

So this is true for all $y \in Y_d$, and 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 $\sigma(z) \in \mathbf{T}(\Sigma_0)_{\overline{\mathsf{P}}}$ 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 $\alpha = \sigma(\lambda) \in \mathbf{T}(\Sigma_0)_{\overline{\mathsf{P}}}$ and $\mathcal{E}_0 \vdash t_0 \xrightarrow{\lambda} t$. ■

I need some definitions concerning congruences to achieve extension up to bisimulation results. Since I would like each signature to have its own congruence, I define the following way to combine congruences.

Definition 5.3.7 (Sum of two congruences)
Let $\Sigma_i = (S_i \cup \{\mathsf{P}\}, F_i)$ for $i = 0, 1$ be two signatures with $\Sigma = \Sigma_0 \oplus \Sigma_1$ defined. Let $\equiv_0$ and $\equiv_1$ be two congruence defined on $\mathbf{T}(\Sigma_0)$ and $\mathbf{T}(\Sigma_1)$ respectively. Define the *sum of* $\equiv_0$ *and* $\equiv_1$ ($\equiv_0 \oplus \equiv_1$) as the smallest congruence over $\mathbf{T}(\Sigma_0 \oplus \Sigma_1)$ containing both $\equiv_0$ and $\equiv_1$. ■

Notation Let $\equiv$ be an equivalence over a set A, and let $B \subseteq A$. Define

$$\equiv\restriction_B \;=\; \equiv \cap\, (B \times B). \qquad ■$$

But for the main results to work, it is necessary to have a specific relation between the original congruences and their sum.

Definition 5.3.8 (Conservativity of the sum of two congruences)
Let $\Sigma_i = (S_i \cup \{\mathsf{P}\}, F_i)$ for $i = 0, 1$ be two signatures with $\Sigma = \Sigma_0 \oplus \Sigma_1$ defined. Let $\equiv_0$ and $\equiv_1$ be two congruences defined on $\mathbf{T}(\Sigma_0)$ and $\mathbf{T}(\Sigma_1)$ respectively. $\equiv_0 \oplus \equiv_1$ is said to be *conservative with respect to* $\equiv_i$ if $(\equiv_0 \oplus \equiv_1) \restriction_{\mathbf{T}(\Sigma_i)_{\overline{\mathsf{P}}}} \;=\; \equiv_i \restriction_{\mathbf{T}(\Sigma_i)_{\overline{\mathsf{P}}}}$. ■

I also need the following lemma to make the proofs simpler.

Lemma 5.3.2 (Equivalences and sums of congruences)
Let $\Sigma_i = (S_i \cup \{\mathsf{P}\}, F_i)$ for $i = 0, 1$ be two signatures with $\Sigma = \Sigma_0 \oplus \Sigma_1$ defined. Let $\equiv_0$ and $\equiv_1$ be congruences over $\mathbf{T}(\Sigma_0)_{\overline{\mathsf{P}}}$ and $\mathbf{T}(\Sigma_1)_{\overline{\mathsf{P}}}$ respectively and let $\equiv \,=\, \equiv_0 \oplus \equiv_1$. Then for $t_0, u_0 \in \mathbf{T}(\Sigma_0)_{\mathsf{P}}$

1. $t_0 \sim^{\mathcal{E}_0}_{\equiv_0} u_0 \Rightarrow t_0 \sim^{\mathcal{E}_0}_{\equiv} u_0$,

2. if $\equiv$ is conservative with respect to $\equiv_0$ then $t_0 \sim^{\mathcal{E}_0}_{\equiv} u_0 \Rightarrow t_0 \sim^{\mathcal{E}_0}_{\equiv_0} u_0$.

Proof:

1. Assume $t_0 \sim^{\mathcal{E}_0}_{\equiv_0} u_0$. Consider $t_0 \xrightarrow{\alpha} t'$, then there exists $u' \in \mathbf{T}(\Sigma_0)_{\mathsf{P}}$ and $\beta \in \mathbf{T}(\Sigma_0)_{\overline{\mathsf{P}}}$, such that $u_0 \xrightarrow{\beta} u'$ and $\alpha \equiv_0 \beta$. But since $\equiv\, = \,\equiv_0 \oplus \equiv_1$, $\alpha \equiv \beta$ as required.

2. Assume $t_0 \sim^{\mathcal{E}_0}_{\equiv} u_0$. Consider $t_0 \xrightarrow{\alpha} t'$, then there exists $u' \in \mathbf{T}(\Sigma_0)_{\mathsf{P}}$ and $\beta \in \mathbf{T}(\Sigma_0)_{\overline{\mathsf{P}}}$, such that $u_0 \xrightarrow{\beta} u'$ and $\alpha \equiv \beta$. But since $\equiv$ is conservative with respect to $\equiv_0$, and $\alpha, \beta \in \mathbf{T}(\Sigma_0)$, $\alpha \equiv_0 \beta$ as required. ■

In the rest of this section, I prove two theorems about extensions—one for refining extensions and one for abstracting extensions. Both of these results rely on the fact that the congruence I am working with respects sorts, and hence transitions with different sorts cannot be matched. I give examples after the theorems.

I can now prove the following result for refining extensions. The proof requires the conservativity of $\equiv_0 \oplus \equiv_1$ with respect to $\equiv_0$ to ensure that no additional matches can be made under $\equiv_0 \oplus \equiv_1$ on the transitions from terms from $\mathbf{T}(\Sigma_0)_{\overline{\mathsf{P}}}$ generated by rules from R_0. Because new transitions from terms in $\mathbf{T}(\Sigma)_{\overline{\mathsf{P}}}$ generated by extended *tyxt* rules from R_1 are guaranteed to have sorts outside S_0, they cannot match any existing transitions under $\equiv_0 \oplus \equiv_1$. Therefore, terms that are not equated under $\sim^{\mathcal{E}_0}_{\equiv_0}$ are also not equated under $\sim^{\mathcal{E}}_{\equiv_0 \oplus \equiv_1}$.

Theorem 5.3.1 (Refining extension up to bisimulation with respect to a congruence)
Let $\mathcal{E}_i = (\Sigma_i, R_i)$ for $i = 0, 1$ be two eTSSs in extended *tyft/tyxt* format such that $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ is defined. Moreover, let $\equiv_0$ and $\equiv_1$ be congruences over $\mathbf{T}(\Sigma_0)_{\overline{\mathsf{P}}}$ and $\mathbf{T}(\Sigma_1)_{\overline{\mathsf{P}}}$ respectively such that $\equiv_0 \oplus \equiv_1$ is conservative with respect to $\equiv_0$. If $\mathcal{E}_0$ is pure and label-pure, and $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ is type-1, then $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ is a refining extension.

Proof: Let $t_0, u_0 \in \mathbf{T}(\Sigma_0)$ and let $\equiv\, = \,\equiv_0 \oplus \equiv_1$. I wish to show that $\mathcal{E}$ is a refining extension of $\mathcal{E}_0$ up to bisimulation with respect to $\equiv$ so I need to show that $t_0 \sim^{\mathcal{E}}_{\equiv} u_0 \Rightarrow t_0 \sim^{\mathcal{E}_0}_{\equiv_0} u_0$.

I prove this by showing the contrapositive. Assume $t_0 \not\sim^{\mathcal{E}_0}_{\equiv_0} u_0$. Hence, I can assume (without loss of generality) that 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. First note, that since $\equiv$ is conservative with respect to $\equiv_0$, no more transitions can be equated in $\mathcal{E}_0$ under $\equiv$ then under $\equiv_0$, hence I only need to consider the new transitions generated under $\mathcal{E}$. I will prove that none of the new transitions can match this transition. Note also that new transitions can be derived from the new rules and function symbols, but existing transitions cannot be removed, hence I need only consider the new transitions. I have a number of cases to consider for any new transition.

- The transition was generated by the use of an extended *tyft/tyxt* rule from R_0. Then by Lemma 5.3.1, I know that this transition is in fact not a new transition, hence it can have no effect on the bisimilarity of t_0 and u_0 under $\mathcal{E}$.

- The transition was generated by the use of an extended *tyxt* rule from R_1. Then because $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ is type-1 and hence there is the restriction that extended *tyxt* rules in R_1 cannot have a label in the conclusion that has a sort that appears in S_0, any transition generated by such a rule cannot match an old transition from $\mathcal{E}_0$ since matching of actions under a congruence can only occur within a sort.

- The transition was generated by the use of an extended *tyft* rule form R_1. Then because $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ is type-1 and hence there is the restriction that extended *tyft* rules in R_1 with a function symbol from F_0 in the source of the conclusion cannot have a label in the conclusion that has a sort that appears in S_0, any transition generated by such a rule cannot match an old transition from $\mathcal{E}_0$ since matching of actions under a congruence can only occur within a sort.

Hence it is clear that the transition that is not matched by any of the existing transitions cannot be matched by any of the new transitions, therefore $t_0 \not\sim^{\mathcal{E}}_{\equiv} u_0$. ■

I need stronger conditions to show a similar result for abstracting extensions, namely that the sum is type-0 and the second component is well-founded.

Theorem 5.3.2 (Abstracting extension up to bisimulation with respect to a congruence)
Let $\mathcal{E}_i = (\Sigma_i, R_i)$ for $i = 0, 1$ be two eTSSs in extended *tyft/tyxt* format such that $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ is defined. Moreover, let $\equiv_0$ and $\equiv_1$ be congruences over $\mathbf{T}(\Sigma_0)_{\overline{\mathsf{P}}}$ and $\mathbf{T}(\Sigma_1)_{\overline{\mathsf{P}}}$ respectively such that $\equiv_0 \oplus \equiv_1$ is compatible with $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$. If $\mathcal{E}_0$ is pure and label-pure, $\mathcal{E}_1$ is well-founded, and $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ is type-0, then $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ is an abstracting extension.

Proof: The details of this proof are included in Appendix A. It proceeds in a similar fashion to Theorem 4.3.1, but with greater intricacy since the relation that is to be shown a bisimulation is more complex. Given a transition from a state, a substitution must be found to show that an equivalent state has a transition that satisfies the definition of bisimulation. The proof proceeds by induction on the length of the proof tree for a transition and then induction on a partition of the variables that appear in the final rule used in the proof tree. ■

Corollary 5.3.1 (Conservative extension up to bisimulation with respect to a congruence)
Let $\mathcal{E}_i = (\Sigma_i, R_i)$ for $i = 0, 1$ be two eTSSs in extended *tyft/tyxt* format such that $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ is defined. Moreover, let $\equiv_0$ and $\equiv_1$ be congruences over $\mathbf{T}(\Sigma_0)_{\overline{\mathsf{P}}}$ and $\mathbf{T}(\Sigma_1)_{\overline{\mathsf{P}}}$ respectively such that $\equiv_0 \oplus \equiv_1$ is conservative with respect to $\equiv_0$ and compatible with $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$. If $\mathcal{E}_0$ is pure and label-pure, $\mathcal{E}_1$ is well-founded, and $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ is type-0, then $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ is a conservative extension.

Proof: It is clear that $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ is a refining extension because $\mathcal{E}_0$ is pure and label-pure, and $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ is type-0, and hence type-1. It is also clear that $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ is an abstracting extension because if $\mathcal{E}_0$ is pure and label-pure, $\mathcal{E}_1$ is well-founded, and $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ is type-0. Hence $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ is a conservative extension. ■

Example 5.2.1 is neither an refining extension up to bisimulation nor an abstracting extension up to bisimulation, since the sum is not even type-1. I now consider two examples that do give interesting extensions.

Example 5.3.1 (Refining extension up to bisimulation)
Consider the eTSS $\mathcal{E}_{\text{CCSSub}}$ from Example 4.3.1. Consider also the signature, Σ_{CCSTau},

$$(\ \mathsf{P}, \tau\mathsf{Act};\ \ \tau;\ \ \mathsf{plus}\ )$$

with the types $\tau :\rightarrow \tau\mathsf{Act}$ and $\mathsf{plus} : \mathsf{P}, \mathsf{P} \rightarrow \mathsf{P}$. and the rule set R_{CCSTau}

$$\frac{}{\mathsf{plus}(x, x') \xrightarrow{\tau} y}.$$

Let **Id** be the congruence for both eTSSs, then conservativity and compatibility are satisfied. Then by Theorem 5.3.1, $\mathcal{E}_{\text{CCSSub}} \oplus \mathcal{E}_{\text{CCSTau}}$ is a refining extension up to bisimulation with respect to **Id**, since the sum is type-1, and $\mathcal{E}_{\text{CCSSub}}$ is pure and label-pure. Clearly the only transitions added to those terms from the first signature are those with a new sort and hence they cannot cause unidentified terms to become identified. ■

Example 5.3.2 (Abstracting extension up to bisimulation)
Consider again $\mathcal{E}_{\text{CCSSub}}$ and the eTSS $\mathcal{E}_{\text{CCSTauNew}}$ consisting of the signature $\mathcal{E}_{\text{CCSTau}}$ from the previous example and the rule set $R_{\text{CCSTauNew}}$

$$\frac{}{x \xrightarrow{\tau} y}.$$

Let **Id** be the congruence for both eTSSs, then compatibility is satisfied. Then by Theorem 5.3.2, $\mathcal{E}_{\text{CCSSub}} \oplus \mathcal{E}_{\text{CCSTauNew}}$ is an abstracting extension up to bisimulation with respect to **Id**, since the sum is type-0, $\mathcal{E}_{\text{CCSSub}}$ is pure and label-pure, and $\mathcal{E}_{\text{CCSTau}}$ is trivially well-founded. Informally, it can be said that because the new transitions with a new sort are added to every term from the first signature, equivalence is retained. ■

5.3.1 Counter-examples

Again I need counter-examples to show that the conditions in the two main results above cannot be weakened, although as I show in the next section, it is possible to obtain similar results with different conditions.

$$\begin{tikzcd}
f(nil) \arrow[dd, "g"'] & \substack{\not\sim^{\mathcal{E}_0}_{\mathbf{Id}} \\ \sim^{\mathcal{E}_0 \oplus\!\!> \mathcal{E}_0'}_{\mathbf{Id}\oplus\equiv_1}} & f'(nil) \arrow[dd, "g'"] \\
 & \equiv_1 & \\
nil & & nil
\end{tikzcd}$$

Figure 5.3: Counter-example 5.3.1

5.3.1.1 Refining extensions

I wish to show that conservativity, pureness, label-pureness and type-1 sum are necessary conditions. I now present counter-examples for conservativity and type-1 sum. In each of these counter-examples, I will work with two closed terms which are not equivalent, and show how by relaxing the conditions, the result is lost.

In the figures that accompany the counter-examples, my aim is to illustrate the additional transitions that result from the sum of the two eTSSs. These transitions are given by dashed arrows. As in the previous chapter, I do not give all transitions or the LTS associated with terms under consideration.

Counter-example 5.3.1 (Conservativity of equivalence)
This example considers conservativity of $\equiv_0 \oplus \equiv_1$ with respect to $\equiv_0$. First consider the signature

$$\Sigma_0 = (\ \mathsf{P}, s;\ \ g, g', nil;\ \ f, f'\ \ )$$

with $g :\rightarrow s$, $g' :\rightarrow s$, $nil :\rightarrow \mathsf{P}$, $f : \mathsf{P} \rightarrow \mathsf{P}$, and $f' : \mathsf{P} \rightarrow \mathsf{P}$. Assume $\equiv_0$ is the identity relation $\mathbf{Id}$.

Consider the eTSS $\mathcal{E}_0$ consisting of Σ_0 and the axioms $f(x) \xrightarrow{g} x$ and $f'(x) \xrightarrow{g'} x$. Clearly $f(nil) \not\sim^{\mathcal{E}_0}_{\mathbf{Id}} f'(nil)$. Consider summing with this the eTSS $\mathcal{E}_0'$ consisting of the same signature and $\equiv_1$ which is the identity relation plus $g \equiv_1 g'$. Then $\mathbf{Id}\oplus \equiv_1$ is not conservative with respect to $\mathbf{Id}$, and $f(nil) \sim^{\mathcal{E}_0 \oplus\!\!> \mathcal{E}_0'}_{\mathbf{Id}\oplus\equiv_1} f'(nil)$. Figure 5.3 illustrates this counter-example. ■

$$\begin{tikzcd}
f(nil) \arrow[r, phantom, "\begin{array}{c}\not\sim^{\mathcal{E}_1}_{\mathbf{Id}} \\ \sim^{\mathcal{E}_1 \oplus\!\!> \mathcal{E}'_1}_{\mathbf{Id}}\end{array}"] \arrow[d, "g"'] & nil \arrow[d, dotted, "g"] \\
nil & nil
\end{tikzcd}$$

$$\begin{tikzcd}
f(nil) \arrow[r, phantom, "\begin{array}{c}\not\sim^{\mathcal{E}_2}_{\mathbf{Id}} \\ \sim^{\mathcal{E}_2 \oplus\!\!> \mathcal{E}'_2}_{\mathbf{Id}}\end{array}"] \arrow[d, "g"'] & f'(nil) \arrow[d, dotted, "g"] \\
nil & nil
\end{tikzcd}$$

Figure 5.4: Counter-example 5.3.2

Counter-example 5.3.2 (Type-1 sum)

I now look at the requirement for a type-1 sum. This has two parts. Consider first allowing a *tyxt* rule in R_1 which has a sort from S_0. Let $\mathcal{E}_1$ be the signature from the previous example, as well as the rule $f(x) \xrightarrow{g} x$. Then clearly $f(nil) \not\sim^{\mathcal{E}_1}_{\mathbf{Id}} nil$. However if the sum of $\mathcal{E}_1$ and $\mathcal{E}'_1$ is formed where $\mathcal{E}'_1$ consists of Σ_0 and the axiom $x \xrightarrow{g} nil$, then $f(nil) \sim^{\mathcal{E}_1 \oplus\!\!> \mathcal{E}'_1}_{\mathbf{Id}} nil$.

Next, consider allowing a *tyft* rule in R_1 with a function symbol from Σ_0 and a sort from S_0. Let $\mathcal{E}_2$ be the signature from the previous example plus the rule $f(x) \xrightarrow{g} x$. Clearly, $f(nil) \not\sim_{\mathbf{Id}} f'(nil)$. However if $\mathcal{E}_2$ is summed with $\mathcal{E}'_2$ where $\mathcal{E}'_2$ consists of the same signature and the rule $f'(x) \xrightarrow{g} x$, then $f(nil) \sim_{\mathbf{Id}} f'(nil)$. Diagrams of these counter-examples are given in Figure 5.4. ■

I now look at pureness and label-pureness. It is not clear that these conditions are required for the theorem, except for the specific condition of not having a free variable in the source of a premise for which I give a counter-example below. However, it can be shown fairly easily that all of these conditions are required for Lemma 5.3.1 which is used in the theorem. Hence it may be the case that it is

$$\begin{tikzcd} f(nil) \arrow[d, "g"'] & \begin{array}{c} \not\sim^{\mathcal{E}_3}_{\mathbf{Id}} \\ \sim^{\mathcal{E}_3 \oplus\!\!> \mathcal{E}'_3}_{\mathbf{Id}} \end{array} & f'(nil) \arrow[d, dotted, "g"] \\ nil & & nil \end{tikzcd}$$

Figure 5.5: Counter-example 5.3.3

possible to prove the theorem with weaker conditions on the free variables and without the use of Lemma 5.3.1.

Counter-example 5.3.3 (Pureness for refining extensions up to bisimulation)

In this example, I will look at one condition of pureness. A rule is not pure when there is a free variable in the source of a premise. Consider the signature

$$\Sigma_3 = (\ \mathsf{P}, s, s';\ \ g, g', nil;\ \ f, f'\ )$$

with $g :\to s$, $g' :\to s'$, $nil :\to \mathsf{P}$, $f : \mathsf{P} \to \mathsf{P}$ and $f' : \mathsf{P} \to \mathsf{P}$; together with the rules

$$\frac{}{f(x) \xrightarrow{g} x} \qquad\qquad \frac{x_1 \xrightarrow{z'_s} y}{f'(x) \xrightarrow{g} y}$$

and call this eTSS $\mathcal{E}_3$. Clearly the second rule is not pure. It can be shown that $f(nil) \not\sim^{\mathcal{E}_3}_{\mathbf{Id}} f'(nil)$, since $f'(nil)$ has no transitions. However, adding the eTSS $\mathcal{E}'_3$ consisting of Σ plus $f'' :\to \mathsf{P}$ and the axiom $f'' \xrightarrow{g'} nil$ causes $f(nil) \sim^{\mathcal{E}_3 \oplus\!\!> \mathcal{E}'_3}_{\mathbf{Id}} f'(nil)$ since now $f'(nil)$ has a matching transition. This counter-example is illustrated by Figure 5.5. ■

5.3.1.2 Abstracting extensions

I wish to show that pureness, label-pureness and type-0 sum are necessary conditions. Note that the earlier comments about well-foundedness apply to this proof also, as does the counter-example for compatibility in Counter-example 4.3.5.

Counter-example 5.3.4 (Pureness for abstracting extensions up to bisimulation)

In this example, I will look at pureness. 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 the signature Σ_0 used in Counter-example 5.3.1 together with the rule

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

and call this eTSS $\mathcal{E}_4$. Clearly this rule is not pure. It can be shown that $nil \sim^{\mathcal{E}_4}_{\mathbf{Id}} f(nil)$, since neither have any transitions. However, adding the eTSS $\mathcal{E}'_4$ consisting of Σ_0 plus $f'' :\rightarrow \mathsf{P}$ and the axiom $f'' \xrightarrow{g} nil$ causes $nil \not\sim^{\mathcal{E}_4 \oplus\!\!> \mathcal{E}'_4}_{\mathbf{Id}} f(nil)$ since now $f(nil)$ has a transition.

Next consider the signature

$$\Sigma_5 = (\ \mathsf{P}, s, s';\ \ g, g', nil_1, nil_2\ )$$

with $g :\rightarrow s$, $g' :\rightarrow s'$, $nil_1 :\rightarrow \mathsf{P}$ and $nil_2 :\rightarrow \mathsf{P}$.

Let $\mathcal{E}_5$ be the eTSS consisting of Σ_5 and the axioms

$$\frac{}{nil_1 \xrightarrow{g} nil_1} \qquad\qquad \frac{}{nil_2 \xrightarrow{g} x}.$$

The second axiom is not pure. It can be shown that $nil_1 \sim^{\mathcal{E}_5}_{\mathbf{Id}} nil_2$. Let $\mathcal{E}'_5$ be the eTSS consisting of Σ_5 with $nil_3 :\rightarrow \mathsf{P}$ added. Clearly $nil_1 \not\sim^{\mathcal{E}_5 \oplus\!\!> \mathcal{E}'_5}_{\mathbf{Id}} nil_2$. These counter-examples are illustrated in Figure 5.6. ■

Counter-example 5.3.5 (Label-pureness for abstracting extensions up to bisimulation)

This example will deal with label-pureness. There are three ways in which a set of rules can fail the label-pureness condition. Consider the many-sorted signature

$$\Sigma_6 = (\ \mathsf{P}, s, s';\ \ ok, g, nil;\ \ f, f_1, f_2, f', f''\ )$$

with $ok :\rightarrow s'$, $g :\rightarrow s$, $nil :\rightarrow \mathsf{P}$, $f : \mathsf{P} \rightarrow \mathsf{P}$, $f_1 : \mathsf{P} \rightarrow \mathsf{P}$, $f_2 : \mathsf{P} \rightarrow \mathsf{P}$, $f' : s \rightarrow \mathsf{P}$ and $f'' : s, \mathsf{P} \rightarrow \mathsf{P}$.

Consider the eTSS $\mathcal{E}_6$ consisting of Σ_6 and the axioms $f_1(x) \xrightarrow{g} nil$ and $f_2(x) \xrightarrow{z_s} nil$. It is clear that $f_1(nil) \sim^{\mathcal{E}_6}_{\mathbf{Id}} f_2(nil)$. If I add a new constant

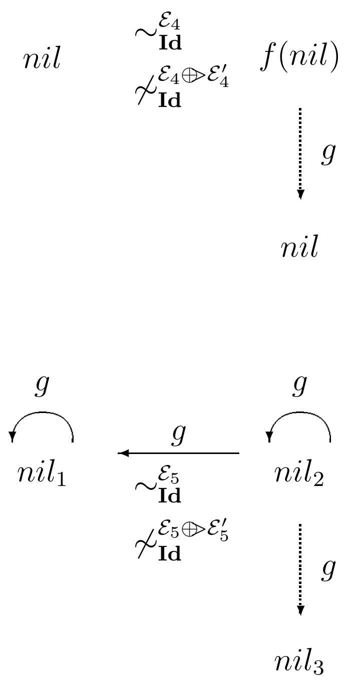
nil
$\sim^{\mathcal{E}_4}_{\mathbf{Id}}$
$\not\sim^{\mathcal{E}_4 \oplus\!\!> \mathcal{E}'_4}_{\mathbf{Id}}$
$f(nil)$
g
nil
g
g
g
nil_1
$\sim^{\mathcal{E}_5}_{\mathbf{Id}}$
$\not\sim^{\mathcal{E}_5 \oplus\!\!> \mathcal{E}'_5}_{\mathbf{Id}}$
nil_2
g
nil_3

Figure 5.6: Counter-example 5.3.4

$g' :\to s$ to form the eTSS $\mathcal{E}'_6$, then I obtain a new transition $f_2(nil) \xrightarrow{g'} nil$, and hence $f_1(nil) \not\sim^{\mathcal{E}_6 \oplus\!\!> \mathcal{E}'_6}_{\mathbf{Id}} f_2(nil)$.

Next, consider the eTSS $\mathcal{E}_7$ consisting of Σ_6 and the axioms $f_1(x) \xrightarrow{ok} f'(z_s)$, $f_2(x) \xrightarrow{ok} f'(g)$, and $f'(z_s) \xrightarrow{z_s} nil$. Clearly, $f_1(nil) \sim^{\mathcal{E}_7}_{\mathbf{Id}} f_2(nil)$. If I add a new constant $g' :\to s$ to create the eTSS $\mathcal{E}'_7$, then I obtain a new transition $f_1(nil) \xrightarrow{ok} f'(g')$, and hence $f_1(nil) \not\sim^{\mathcal{E}_7 \oplus\!\!> \mathcal{E}'_7}_{\mathbf{Id}} f_2(nil)$, since $f_1(nil) \xrightarrow{ok} f'(g')$ and $f'(g') \not\sim^{\mathcal{E}_7 \oplus\!\!> \mathcal{E}'_7}_{\mathbf{Id}} f'(g)$.

Finally consider the eTSS $\mathcal{E}_8$ consisting of Σ_6 and the rules

$$\frac{}{f''(z_s, x) \xrightarrow{ok} f'(z_s)} \qquad \frac{f''(z_s, x) \xrightarrow{ok} y}{f_1(x) \xrightarrow{ok} y} \qquad \frac{f''(g, x) \xrightarrow{ok} y}{f_2(x) \xrightarrow{ok} y} \qquad \frac{}{f'(z_s) \xrightarrow{z_s} nil}.$$

Consider the proofs

$$\frac{f''(g, nil) \xrightarrow{ok} f'(g)}{f_1(nil) \xrightarrow{ok} f'(g)} \qquad \frac{f''(g, nil) \xrightarrow{ok} f'(g)}{f_2(nil) \xrightarrow{ok} f'(g)}.$$

From these, it is clear that $f_1(nil) \sim^{\mathcal{E}_8}_{\mathbf{Id}} f_2(nil)$. If I add a new constant $g' :\to s$ to form the eTSS $\mathcal{E}'_8$, I obtain a new transition $f''(nil) \xrightarrow{ok} f'(g')$, and hence $f_1(nil) \not\sim^{\mathcal{E}_8 \oplus\!\!> \mathcal{E}'_8}_{\mathbf{Id}} f_2(nil)$, since $f_1(nil) \xrightarrow{ok} f'(g')$ and $f'(g') \not\sim^{\mathcal{E}_8 \oplus\!\!> \mathcal{E}'_8}_{\mathbf{Id}} f'(g)$.

$$\begin{tikzcd}
f_1(nil) \arrow[d, "g"'] \arrow[r, phantom, "\begin{array}{c}\sim^{\mathcal{E}_6}_{\mathbf{Id}} \\ \not\sim^{\mathcal{E}_6\oplus\!\!>\mathcal{E}_6'}_{\mathbf{Id}}\end{array}"] & f_2(nil) \arrow[d, "g"'] \arrow[dr, dotted, "g'"] & \\
nil & nil & nil
\end{tikzcd}$$

$$\begin{tikzcd}
f_1(nil) \arrow[d, "ok"'] \arrow[r, phantom, "\begin{array}{c}\sim^{\mathcal{E}_7}_{\mathbf{Id}} \\ \not\sim^{\mathcal{E}_7\oplus\!\!>\mathcal{E}_7'}_{\mathbf{Id}} \\ \sim^{\mathcal{E}_8}_{\mathbf{Id}} \\ \not\sim^{\mathcal{E}_8\oplus\!\!>\mathcal{E}_8'}_{\mathbf{Id}}\end{array}"] & f_2(nil) \arrow[d, "ok"'] \arrow[dr, dotted, "ok"] & \\
f'(g) \arrow[d, "g"'] & f'(g) \arrow[d, "g"'] & f'(g') \arrow[d, dotted, "g'"] \\
nil & nil & nil
\end{tikzcd}$$

Figure 5.7: Counter-example 5.3.5

Figure 5.7 illustrates these counter-examples. The second and third counter-example have the same diagrams. ■

Similarly to the counter-examples for the conservative extension result, these counter-examples all rely on adding new functions of an existing sort. It may be possible to exclude this possibility and achieve a different result. I also need to show why type-0 sums are required. The next counter-example considers what happens if one of the new *tyxt* rules can have a label with a sort from the first signature.

Counter-example 5.3.6 (First example of type-0 sum)

Using the same signature as in the above example, consider the eTSS consisting of Σ_6 and the rules

$$\frac{}{f_1(x) \xrightarrow{g} nil} \qquad \frac{}{f_2(x) \xrightarrow{g} nil} \qquad \frac{f(x) \xrightarrow{z_{s'}} y}{f_2(x) \xrightarrow{z_{s'}} y}$$

$$\begin{tikzcd}
f_1(nil) \arrow[d, "g"'] & \begin{array}{c}\sim^{\mathcal{E}_9}_{\mathbf{Id}} \\ \not\sim^{\mathcal{E}_9 \oplus\!\!> \mathcal{E}_9'}_{\mathbf{Id}}\end{array} & f_2(nil) \arrow[d, "g"'] \arrow[dr, dotted, "ok"] & \\
nil & & nil & nil
\end{tikzcd}$$

Figure 5.8: Counter-example 5.3.6

and call this eTSS $\mathcal{E}_9$. From these rules, it can be seen that $f_1(nil) \sim^{\mathcal{E}_9}_{\mathbf{Id}} f_2(nil)$. If I add a new axiom $x \xrightarrow{ok} nil$ to give the eTSS $\mathcal{E}_9'$, then I can derive a new transition $f_2(nil) \xrightarrow{ok} nil$ by the proof

$$\frac{f(nil) \xrightarrow{ok} nil}{f_2(nil) \xrightarrow{ok} nil}$$

and hence $f_1(nil) \not\sim^{\mathcal{E}_9 \oplus\!\!> \mathcal{E}_9'}_{\mathbf{Id}} f_2(nil)$. This is illustrated in Figure 5.8. ■

I need a counter-example to show that when $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ is type-1, it is not possible to obtain a refining extension and hence a type-0 sum is required.

Counter-example 5.3.7 (Second example of type-0 sum)
Consider the many-sorted signature

$$\Sigma_{10} = (\ \mathsf{P}, s, s';\ \ ok, g, nil;\ \ f, f', f''\ )$$

with $ok :\to s'$, $g :\to s$, $nil :\to \mathsf{P}$, $f : s, \mathsf{P} \to \mathsf{P}$ and $f' : s, \mathsf{P} \to \mathsf{P}$.

Consider the eTSS $\mathcal{E}_{10}$ consisting of Σ_{10} and the axioms $f'(z_s, x) \xrightarrow{z_s} x$ and $f(z_s, x) \xrightarrow{z_s} x$. Also consider the eTSS $\mathcal{E}_{10}'$ consisting of Σ_{10} and the axiom $f'(z_s, x) \xrightarrow{ok} x$. Hence $\mathcal{E}_{10} \oplus\!\!> \mathcal{E}_{10}'$ is type-1 but not type-0.

Then $f(g, nil) \sim^{\mathcal{E}_{10}}_{\mathbf{Id}} f'(g, nil)$, but $f(g, nil) \not\sim^{\mathcal{E}_{10} \oplus\!\!> \mathcal{E}_{10}'}_{\mathbf{Id}} f'(g, nil)$ since $f(g, nil) \xrightarrow{ok} nil$, but $f'(g, nil)$ cannot perform an *ok* action. This counter-example is illustrating in Figure 5.9. ■

$$\begin{tikzcd}
f(g, nil) \arrow[d, "g"'] & \begin{array}{c}\sim^{\mathcal{E}_{10}}_{\mathbf{Id}} \\ \not\sim^{\mathcal{E}_{10}\oplus\!\!>\mathcal{E}'_{10}}_{\mathbf{Id}}\end{array} & f'(g, nil) \arrow[d, "g"'] \arrow[dr, dotted, "ok"] & \\
nil & & nil & nil
\end{tikzcd}$$

Figure 5.9: Counter-example 5.3.7

5.4 A different approach to extensions up to bisimulation

In this section, I will describe how it is possible to change the label-pureness requirement in some of the previous results by giving additional requirements on Σ_1. In the Counter-examples 5.2.1 and 5.3.5 which show that label-pureness was required, a new constant with sort from S_0 was introduced in Σ_1 to show that the required result was 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. I first need a new definition to describe the sort of function symbols I want in this kind of signature.

Definition 5.4.1 (Safety of a signature)
Let $\Sigma_i = (S_i \cup \{\mathsf{P}\}, F_i)$ for $i = 0, 1$ be two signatures. It is said that 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 for the sorts that appear in the intersection (although there are open terms since there are variables with these sorts). This is why I introduced the notion of asymmetric sum earlier in the chapter, so that it was possible to consider non-sensible signatures as extensions.

Also note that from the point of view of this chapter, algebras are associated with arbitrary equivalences. When these results are applied to process algebras, equivalences are induced by an algebra which represents the labels of the process algebras. Hence, signatures which are not sensible are less likely to occur and moreover, it may be necessary to define the equivalence for functions from F_0. So although it would be possible to change Definition 5.4.1 to require that no label in F_1 has a range with a sort from S_0, it is more practical to allow functions from F_0 to appear in F_1. Therefore, the above definition only requires that any new functions added do not have a range with a sort from S_0.

The following technical lemma shows that all closed terms in the sum with a sort from S_0 are in $\mathbb{T}(\Sigma_0)_{\overline{\mathsf{P}}}$.

Lemma 5.4.1 (Implications of safety of a signature)
Let $\Sigma_i = (S_i \cup \{\mathsf{P}\}, F_i)$ for $i = 0, 1$ be two suitable signatures with $\Sigma_0 \oplus\!\!> \Sigma_1$ defined. If Σ_1 is safe for S_0, then for all $t \in \mathbf{T}(\Sigma_0 \oplus\!\!> \Sigma_1)_{\overline{\mathsf{P}}} - \mathbf{T}(\Sigma_0)_{\overline{\mathsf{P}}}$, the sort of t is in $S_1 - S_0$; namely for all $s \in S_0$

$$t \in \mathbf{T}(\Sigma_0 \oplus\!\!> \Sigma_1)_s \Rightarrow t \in \mathbf{T}(\Sigma_0)_{\overline{\mathsf{P}}}.$$

Proof: Let $t' \in \mathbf{T}(\Sigma_0 \oplus\!\!> \Sigma_1)_s$ for $s \in S_0$. Then since t' is not a variable, t' has the form $f(\eta_1, \ldots, \eta_m)$ with $f \in F_0 \cup F_1$ such that $f : s_1 \ldots s_m \to s$ and $s \in S_0$. But since no function in $F_1 - F_0$ has range with sort from S_0, $f \in F_0$ and $s_1, \ldots, s_m \in S_0$. Since η_k $(1 \leqslant k \leqslant m)$ has less structure than t', I can use induction and assume that for $1 \leqslant k \leqslant m$, $\eta_k \in \mathbf{T}(\Sigma_0)_{\overline{\mathsf{P}}}$, since each η_k has a sort from S_0. Hence I can conclude that $f(\eta_1, \ldots, \eta_m) \in \mathbf{T}(\Sigma_0)_{\overline{\mathsf{P}}}$. ■

Theorem 5.2.1 can then be rephrased with the safeness condition added and label-pureness removed. It works because 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 $\mathbf{T}(\Sigma_0)_{\overline{\mathsf{P}}}$.

Theorem 5.4.1 (Conservative extension without label-pureness and with safety)
Let $\Sigma_i = (S_i \cup \{\mathsf{P}\}, F_i)$ for $i = 0, 1$ be two signatures with Σ_1 safe for S_0. Let $\mathcal{E}_0 = (\Sigma_0, R_0)$ be an eTSS in pure extended *tyft/tyxt* format and let $\mathcal{E}_1 = (\Sigma_1, R_1)$ be an eTSS in extended *tyft* format such that there is no rule in R_1 that contains

a function symbol from Σ_0 in the source of the conclusion. Let $\mathcal{E} = \mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ be defined. Then $\mathcal{E}_1$ can be added conservatively to $\mathcal{E}_0$.

Proof: This proof is similar to the proof of the Theorem 5.2.1. The differences are that in the base case here, I use the fact that only terms from $\mathbf{T}(\Sigma_0)$ can be assigned to variables in a rule from R_0 by Lemma 5.4.1, whereas in the earlier proof, label-pureness was used to show that the only variables that could appear were those that already had been shown to have been assigned terms from $\mathbf{T}(\Sigma_0)$. Also in the inductive step here, I use the fact that only terms from $\mathbf{T}(\Sigma_0)$ can be assigned to variables in the source of a premise, whereas previously, the variables that appeared in the source of a premise were shown to appear in the source of the conclusion and hence had been assigned terms from $\mathbf{T}(\Sigma_0)$. In the final step of the proof, I use a similar approach to the earlier proof. ■

Example 5.2.2 cannot be used as an example here since Σ_{CCSPar} is not safe for $\{\mathsf{A}\}$, the set of $\overline{\mathsf{P}}$ sorts for Σ_{CCSSub}.

Example 5.4.1 (Conservative extension with safety)
Consider the eTSS $\mathcal{E}_{\text{CCSSub}}$ and new eTSS $\mathcal{E}_{\text{CCSParTau}}$ consisting of the signature

$$(\ \mathsf{P}, \tau\mathsf{Act};\ \ \tau;\ \ \mathsf{par}\ )$$

with the types $\tau :\rightarrow \tau\mathsf{Act}$ and $\mathsf{par} : \mathsf{P}, \mathsf{P} \rightarrow \mathsf{P}$. and the rule set R_{CCSTau}

$$\frac{}{\mathsf{par}(x, x') \xrightarrow{\tau} y}.$$

Then by the previous theorem, $\mathcal{E}_{\text{CCSSub}} \oplus \mathcal{E}_{\text{CCSParTau}}$ is a conservative extension. Clearly the new rule only allows transitions with a label with the new sort and hence cannot add any transitions with a label of a sort from the first signature. ■

The lemma and the refining, abstracting and conservative extension up to bisimulation results can be rephrased in a similar way. In each of these new statements, I have added the safety condition, and removed the label-pureness condition.

It may look as though it is also 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, since the safety condition prevents this from happening because the terms in $\mathbf{T}(\Sigma_1)_{\overline{\mathsf{P}}}$ have sorts from $S_1 - S_0$. This, however, is not correct since I need to consider terms from $\mathbb{T}(\Sigma_1)_{\overline{\mathsf{P}}}$ when considering rules, and included in this are variables from $V_{\overline{\mathsf{P}}}$ which have sorts from S_0. This could be 'fixed' by insisting that $(S_0 \cap S_1) = \emptyset$, but this is too strong, as I want functions in F_1 which accept arguments with sorts from S_0. So I want to keep the weaker condition and hence I have to keep the condition on extended *tyxt* rules from R_1.

Lemma 5.4.2 (Application of rules without label-pureness and with safety)
Let $\Sigma_i = (S_i \cup \{\mathsf{P}\}, F_i)$ for $i = 0, 1$ be two signatures with Σ_1 safe for S_0. Let $\mathcal{E}_i = (\Sigma_i, R_i)$ for $i = 0, 1$ be two eTSSs in extended *tyft/tyxt* format such that $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ is defined. Moreover, let $\mathcal{E}_0$ be pure, and 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: The proof of this lemma is similar to the proof of Theorem 5.4.1. The differences are that in the base case, the fact that a rule from R_0 is being used is the result of the conditions in the theorem, whereas in this lemma, it is assumed. In the inductive step, in the theorem, I can apply induction directly from the fact that $\sigma(p_i) \in \mathbf{T}(\Sigma_0)_{\mathsf{P}}$, whereas in this lemma, I require some argument from the conditions to show that a rule from R_0 has been used and hence the induction hypothesis can be applied.

I can also rephrase Theorems 5.3.1 and 5.3.2.

Theorem 5.4.2 (Refining extension up to bisimulation with respect to a congruence without label-pureness and with safety))
Let $\Sigma_i = (S_i \cup \{\mathsf{P}\}, F_i)$ for $i = 0, 1$ be two signatures with Σ_1 safe for S_0. Let $\mathcal{E}_i = (\Sigma_i, R_i)$ for $i = 0, 1$ be two eTSSs in extended *tyft/tyxt* format such that $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ is defined. Moreover, let $\equiv_0$ and $\equiv_1$ be congruences over $\mathbf{T}(\Sigma_0)_{\overline{\mathsf{P}}}$ and $\mathbf{T}(\Sigma_1)_{\overline{\mathsf{P}}}$ respectively such that $\equiv_0 \oplus \equiv_1$ is conservative with respect to $\equiv_0$. If $\mathcal{E}_0$ is pure, and $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ is type-1, then $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ is a refining extension.

Proof: This proof is almost identical to the proof of Theorem 5.3.1. The only place where label-pureness is considered in Theorem 5.3.1 is in the first item where Lemma 5.3.1 is used, hence in this proof I can use Lemma 5.4.2 instead. ■

I need stronger conditions to show a similar result for abstracting extensions as before.

Theorem 5.4.3 (Abstracting extension up to bisimulation with respect to a congruence without label-pureness and with safety)
Let $\Sigma_i = (S_i \cup \{\mathsf{P}\}, F_i)$ for $i = 0, 1$ be two signatures with Σ_1 safe for S_0. Let $\mathcal{E}_i = (\Sigma_i, R_i)$ for $i = 0, 1$ be two eTSSs in extended *tyft/tyxt* format such that $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ is defined. Moreover, let $\equiv_0$ and $\equiv_1$ be congruences over $\mathbf{T}(\Sigma_0)_{\overline{\mathsf{P}}}$ and $\mathbf{T}(\Sigma_1)_{\overline{\mathsf{P}}}$ respectively such that $\equiv_0 \oplus \equiv_1$ is compatible with $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$. If $\mathcal{E}_0$ is pure, $\mathcal{E}_1$ is well-founded and $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ is type-0, then $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ is an abstracting extension.

Proof: This proof is almost identical to the proof of Theorem 5.3.2. The only place where label-pureness is considered in Theorem 5.3.2 is when Lemma 5.3.1 is used. Since $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ is type-0 and hence type-1, Lemma 5.4.2 can be used instead.

■

Corollary 5.4.1 (Conservative extension up to bisimulation with respect to a congruence without label-pureness and with safety)
Let $\Sigma_i = (S_i \cup \{\mathsf{P}\}, F_i)$ for $i = 0, 1$ be two signatures with Σ_1 safe for S_0. Let $\mathcal{E}_i = (\Sigma_i, R_i)$ for $i = 0, 1$ be two eTSSs in extended *tyft/tyxt* format such that $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ is defined. Moreover, let $\equiv_0$ and $\equiv_1$ be congruences over $\mathbf{T}(\Sigma_0)_{\overline{\mathsf{P}}}$ and $\mathbf{T}(\Sigma_1)_{\overline{\mathsf{P}}}$ respectively such that $\equiv_0 \oplus \equiv_1$ is compatible with $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$. If $\mathcal{E}_0$ is pure, $\mathcal{E}_1$ is well-founded, and $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ is type-0, then $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ is a conservative extension.

■

Both Example 5.3.1 and Example 5.3.2 can be used as examples for these new theorems since the signatures involved are safe.

In this section, I have given a different approach to dealing with the issue of label-pureness. As will be seen in the next chapter, safety plays an important rôle in comparing equivalences.

5.4.1 Counter-examples

All the previous counter-examples given in Sections 5.2.1 and 5.3.1 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 these sections also apply to the other condition in the theorems since none of them involve adding function symbols with the incorrect sort.

5.5 Conclusion

In this chapter, I have looked at extensions to eTSSs achieved by summing two eTSSs in a particularly manner. This extends previous results, and introduces new definitions and results relating to extension up to bisimulation. I will discuss the application of these results in the next chapter.

Chapter 6

Application of results

6.1 Introduction

This chapter looks at how the results of the previous two chapters can be applied to process algebras. This needs to be done in two parts. The first involves looking at the conditions required for the results of the previous two chapters and seeing how this affects which algebras can be used to represent the labels of the process algebras. The second part looks at how specific process algebras can be represented in the extended *tyft/tyxt* format and how process algebras can be compared.

In the first section, I take conditions from theorems of the previous two chapters, and see what implications these conditions have on algebras that can be used to represent the labels of a process algebra that is to be expressed in extended *tyft/tyxt* format. As described in earlier chapters, the extended *tyft/tyxt* format is purely syntactic, and a mapping is required from the syntactic form of the labels to the semantics of the labels of the process algebras. This is achieved by using a Σ-algebra to represent the labels and then taking the unique homomorphism from the term algebra to this Σ-algebra. This homomorphism induces a congruence over the terms and hence this can be used in the bisimulation that is applied to process terms. My aim in this section is to show that the conditions are not unreasonable, and in fact an additional condition, that of sort-similarity can, in certain situations, ensure some of the conditions are met.

In the second section, I look at how certain process algebras can be expressed in *tyft/tyxt* format. In order to express infinite rule sets, I introduce schemas and schema variables. In the third section, I look at how the results of the previous chapter can be applied to two process algebras.

6.2 Implications of conditions

A number of conditions were required to achieve the results in the previous two chapters. For the congruence result, compatibility is required and for the extension results, sum of congruences are considered. I now investigate how these affect the application of the results.

6.2.1 Implications of congruence compatibility

In the previous two chapters, compatibility (Definition 4.3.8, page 74) was required for a number of results, most notably the congruence result. I now look at conditions on Σ-algebras which will ensure compatibility.

Given a set of rules R, I require that for each $r \in R$, and for any η in $\mathbb{T}(\Sigma)_{\overline{\mathsf{P}}}$ that appears on a transition in a premise of r or as in an argument to the function in the conclusion of r, then

> whenever $\sigma(\eta) \equiv \mu$ for $\mu \in \mathbf{T}(\Sigma)_{\overline{\mathsf{P}}}$, there exists a substitution σ' such that $\mu = \sigma'(\eta)$ and $\sigma(z) \equiv \sigma'(z)$ for all $z \in \mathrm{Var}_{\overline{\mathsf{P}}}(\eta)$.

As an additional definition, for convenience, define a congruence $\equiv$ to be compatible with $\eta \in \mathbb{T}(\Sigma)_{\overline{\mathsf{P}}}$ if the above holds for η.

First, note that I am only interested in those function symbols that appear in rules, since those are the only ones that can be considered when looking at compatibility. Consider $\sigma(\eta) \equiv \mu$, and assume that there is a Σ-algebra $\mathcal{A}$, and $i_{\mathcal{A}}$, the unique homomorphism from $\mathbf{T}(\Sigma)$ to $\mathcal{A}$. The congruence $\equiv$ is defined by

$$i_{\mathcal{A}}(\lambda) = i_{\mathcal{A}}(\lambda') \iff \lambda \equiv \lambda' \quad \forall \lambda, \lambda' \in \mathbf{T}(\Sigma).$$

So if $\sigma(\eta) \equiv \mu$, then $i_{\mathcal{A}}(\sigma(\eta)) = i_{\mathcal{A}}(\mu)$.

I wish to obtain a general result that holds for any signature Σ. Note that I require that the term in question contains no repeated variables. This is not a serious limitation—see the comments after Definition 4.3.8.

Proposition 6.2.1 (Conditions on Σ-algebra ensuring compatibility) Let $\Sigma = (S, F)$ be a signature, and let $\mathcal{A}$ be a Σ-algebra. Let $\equiv$ be the congruence associated with $i_{\mathcal{A}}$ and let $\lambda \in \mathbb{T}(\Sigma)$ be a term with no repeated variables, then $\equiv$ 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 $\alpha \equiv \sigma(\lambda)$ for some $\alpha \in \mathbf{T}(\Sigma)$ and a closed substitution σ. I need to find a substitution σ' such that $\sigma'(\lambda) = \alpha$ and $\sigma(z) \equiv \sigma'(z)$ for all $z \in \text{Var}(\lambda)$. The proof will be proceed by induction on the structure of λ. For the base case, assume $\lambda = z$ for $z \in V$ and define σ' as follows

$$\sigma'(z') = \begin{cases} \alpha & \text{if } z' = z \\ \sigma(z') & \text{otherwise.} \end{cases}$$

Then it is clear that 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

$$\begin{aligned} i_{\mathcal{A}}(\sigma(g(\lambda_1, \ldots, \lambda_n))) &= i_{\mathcal{A}}(\alpha) \\ i_{\mathcal{A}}(g(\sigma(\lambda_1), \ldots, \sigma(\lambda_n))) &= i_{\mathcal{A}}(\alpha) \\ g^{\mathcal{A}}(i_{\mathcal{A}}(\sigma(\lambda_1)), \ldots, i_{\mathcal{A}}(\sigma(\lambda_n))) &= i_{\mathcal{A}}(\alpha). \end{aligned}$$

Fact 1 $g^{\mathcal{A}}(a_1, \ldots, a_n) = i_{\mathcal{A}}(\alpha) \Rightarrow \alpha = g(\mu_1, \ldots, \mu_n)$ for some $\mu_1, \ldots, \mu_n$.

Proof: Suppose not, i.e. $\alpha = g_1(\mu'_1, \ldots, \mu'_m)$ for $g_1 \neq g$ with $g_1 : s'_1 \ldots s'_m \to s$. Then

$$\begin{aligned} g^{\mathcal{A}}(a_1, \ldots, a_n) &= i_{\mathcal{A}}(\alpha) \\ &= i_{\mathcal{A}}(g_1(\mu'_1, \ldots, \mu'_m)) \\ &= g_1^{\mathcal{A}}(i_{\mathcal{A}}(\mu'_1), \ldots, i_{\mathcal{A}}(\mu'_m)). \end{aligned}$$

Contradiction since $Im(g^{\mathcal{A}}) \cap Im(g_1^{\mathcal{A}}) = \emptyset$. ■

So by this fact, there exist $\mu_1, \dots \mu_n$ such that $\alpha = g(\mu_1, \dots, \mu_n)$. Hence

$$\begin{array}{rcll} g^{\mathcal{A}}(i_{\mathcal{A}}(\sigma(\lambda_1)), \dots, i_{\mathcal{A}}(\sigma(\lambda_n))) & = & i_{\mathcal{A}}(g(\mu_1, \dots, \mu_n)) & \\ g^{\mathcal{A}}(i_{\mathcal{A}}(\sigma(\lambda_1)), \dots, i_{\mathcal{A}}(\sigma(\lambda_n))) & = & g^{\mathcal{A}}(i_{\mathcal{A}}(\mu_1), \dots, i_{\mathcal{A}}(\mu_n)) & \\ i_{\mathcal{A}}(\sigma(\lambda_i)) & = & i_{\mathcal{A}}(\mu_i) & \text{for all } 1 \leqslant i \leqslant n \\ & & & \text{(since } g_{\mathcal{A}} \text{ is injective)} \\ \sigma(\lambda_i) & \equiv & \mu_i & \text{for all } 1 \leqslant i \leqslant n. \end{array}$$

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 $\sigma(z) \equiv \sigma_i(z)$ for all $z \in \mathrm{Var}(\lambda_i)$. From these substitutions, construct a substitution that fulfils the compatibility requirements. Let σ' be defined as follows

$$\sigma'(z) \quad = \quad \begin{cases} \sigma_i(z) & \text{if } z \in \mathrm{Var}(\lambda_i) \\ \sigma(z) & \text{otherwise.} \end{cases}$$

This is well-defined since there are no repeated variables. Recall that

$$\begin{array}{rcl} \sigma'(\lambda) & = & \sigma'(g(\lambda_1, \dots, \lambda_n)) \\ & = & g(\sigma'(\lambda_1), \dots, \sigma'(\lambda_n)) \\ & = & g(\sigma_1(\lambda_1), \dots, \sigma_n(\lambda_n)) \\ & = & g(\mu_1, \dots, \mu_n) \\ & = & \alpha. \end{array}$$

Also for any $z \in \mathrm{Var}(\lambda)$, $\sigma'(z) = \sigma_i(z)$ for a unique $1 \leqslant i \leqslant n$, and $\sigma_i(z) \equiv \sigma(z)$, hence $\sigma'(z) \equiv \sigma(z)$. ■

Hence, I have shown some conditions under which compatibility is obtained. To sum up, for the congruence induced over the term algebra by another algebra to be compatible for a certain open term, any function which appears in the open term must be injective and must have an image which is disjoint from the image of any other function.

6.2.2 Sums of congruences and Σ-algebras

In this section, I will prove some important results concerned with sums of congruences and algebras. Although in the material presented so far, I have been working with congruences in an abstract manner, it is also important to investigate them as congruences induced by the Σ-algebras that model the actual process algebra labels. If there are two signatures $\Sigma_i = (S_i, F_i)$, and two Σ_i-algebras, $\mathcal{A}_i$, for $i = 0, 1$, with $\Sigma_0 \oplus\!\!> \Sigma_1$ defined, under what circumstances can I construct a $\Sigma_0 \oplus\!\!> \Sigma_1$-algebra, from $\mathcal{A}_0$ and $\mathcal{A}_1$? Note that here I am considering signatures in general, and do not need to consider the distinguished sort P. I first require some new definitions. The definition of the union of two many-sorted sets is straightforward.

Definition 6.2.1 (Union of many-sorted sets)
Let S_0 and S_1 be two sets, and let $A_i = \{A_{i,s}\}_{s \in S_i}$ be an S_i-sorted set for $i = 0, 1$. Let $S = S_0 \cup S_1$. Then the *S-sorted union* of A_0 and A_1 is defined as $A = \{A_s\}_{s \in S}$ where

- $A_s = A_{0,s}$ if $s \in S_0 - S_1$
- $A_s = A_{1,s}$ if $s \in S_1 - S_0$
- $A_s = A_{0,s} \cup A_{1,s}$ if $s \in S_0 \cap S_1$

and is denoted $\bigcup\!\!\!\!\!\!\!\;\text{s}\;\;_{i=0,1} A_i$. ■

I am interested in a particular class of signatures, namely those whose sums satisfy the following definition.

Definition 6.2.2 (Sort-similar sum of signatures)
Let $\Sigma_i = (S_i, F_i)$ for $i = 0, 1$ be two signatures with $\Sigma_0 \oplus\!\!> \Sigma_1$ defined. $\Sigma_0 \oplus\!\!> \Sigma_1$ is said to be *sort-similar* if for each $s \in S_0 \cap S_1$, $f \in F_0 \cup F_1$ with $f : s_1 \ldots s_n \rightarrow s$ implies $f \in F_0 \cap F_1$. ■

This definition goes beyond that of the sum of two signatures where if a function appears in the intersection of the two signatures, then it has the same type in each signature, since here I require that any function that has a shared result sort, must appear in both signatures. Note that this definition does not contradict safety, since if $f : s_1 \ldots s_n \rightarrow s \in F_1 - F_0$, then it is possible under sort-similarity that $s \notin S_0$.

A result follows from this definition, which I will use later in this section.

Proposition 6.2.2 (Sort-similarity)
Let $\Sigma_i = (S_i, F_i)$ for $i = 0, 1$ be two signatures with $\Sigma_0 \oplus\!\!> \Sigma_1$ defined and sort-similar. If $t \in \mathbf{T}(\Sigma_0 \oplus\!\!> \Sigma_1)$ has sort $s \in S_0 \cap S_1$, then $t \in \mathbf{T}(\Sigma_0) \cap \mathbf{T}(\Sigma_1)$. Moreover $\mathbf{T}(\Sigma_0 \oplus\!\!> \Sigma_1) = \mathbf{T}(\Sigma_0) \cup \mathbf{T}(\Sigma_1)$.

Proof: For the first part of the result, consider $t \in \mathbf{T}(\Sigma_0 \oplus\!\!> \Sigma_1)_s$ for $s \in S_0 \cap S_1$. I proceed by induction on the structure of t. If t is a constant $f :\rightarrow s$ then $f \in F_0 \cap F_1$ and clearly $t \in \mathbf{T}(\Sigma_0) \cap \mathbf{T}(\Sigma_1)$. If $t = f(t_1, \ldots, t_n)$ for $f : s_1 \ldots s_n \rightarrow s$, then $f \in F_0 \cap F_1$ and by the induction hypothesis $t_i \in \mathbf{T}(\Sigma_0) \cap \mathbf{T}(\Sigma_1)$ for $1 \leqslant i \leqslant n$, hence $t \in \mathbf{T}(\Sigma_0) \cap \mathbf{T}(\Sigma_1)$.

Since clearly $\mathbf{T}(\Sigma_0) \cup \mathbf{T}(\Sigma_1) \subseteq \mathbf{T}(\Sigma_0 \oplus\!\!> \Sigma_1)$, I need to show for any $t \in \mathbf{T}(\Sigma_0 \oplus\!\!> \Sigma_1)$ that either $t \in \mathbf{T}(\Sigma_0)$ or $t \in \mathbf{T}(\Sigma_1)$. I will proceed by induction on the structure of t.

If t is a constant symbol, then clearly $t \in \mathbf{T}(\Sigma_0) \cup \mathbf{T}(\Sigma_1)$. Otherwise consider $t = f(t_1, \ldots, t_n)$. Assume that $f \in F_0$ without loss of generality, and that $f : s_1 \ldots s_n \rightarrow s$, then $s_1 \ldots s_n, s \in S_0$. By the induction hypothesis, for $1 \leqslant i \leqslant n$, $t_i \in \mathbf{T}(\Sigma_0) \cup \mathbf{T}(\Sigma_1)$. If any $t_i \in \mathbf{T}(\Sigma_1)$, then since $s_i \in S_0 \cap S_1$, $t_i \in \mathbf{T}(\Sigma_0)$ also. Hence for $1 \leqslant i \leqslant n$, $t_i \in \mathbf{T}(\Sigma_0)$, and hence $t \in \mathbf{T}(\Sigma_0)$. ■

This result shows how under the condition of sort-similarity, no closed terms can be created from operators from both signatures without being in both sets of closed terms. I now look at how the sum of two algebras can be defined.

Definition 6.2.3 (Sum of algebras)
Let $\Sigma_i = (S_i, F_i)$ for $i = 0, 1$ be two signatures with $\Sigma_0 \oplus\!\!> \Sigma_1$ defined and sort-similar. For $i = 0, 1$, let $\mathcal{A}_i = \{\mathcal{A}_{i,s}\}_{s \in S_i}$ together with $\{f^{\mathcal{A}_i} \mid f \in F_i\}$ be Σ_i-algebras such that the following conditions hold.

- for $s \in S_0 \cap S_1$, $\mathcal{A}_{0,s} = \mathcal{A}_{1,s}$,
- for $f \in F_0 \cap F_1$ with $f : s_1 \dots s_n \to s$, $f^{\mathcal{A}_0}(a_1, \dots, a_n) = f^{\mathcal{A}_1}(a_1, \dots, a_n)$ for all $a_j \in \mathcal{A}_{0,s_j}$ for $1 \leqslant j \leqslant n$.

Define $\mathcal{A}_0 \oplus \mathcal{A}_1 = \biguplus^{\mathrm{s}}_{i=0,1} \mathcal{A}_i$ plus the functions $\bigcup_{i=0,1}\{f^{\mathcal{A}_i} \mid f \in F_i\}$. ■

Note that the second condition relies on the fact that since $f \in F_0 \cap F_1$, $s_i \in S_0 \cap S_1$ for $1 \leqslant i \leqslant n$, hence $f^{\mathcal{A}_0}$ and $f^{\mathcal{A}_1}$ are defined on the same sets. Note that because of the sort-similarity, the second condition applies to any function that has a sort in $S_0 \cap S_1$.

It is possible to omit the sort-similarity condition in Definition 6.2.3. Then the second condition is still acceptable in that $f^{\mathcal{A}_0}$ and $f^{\mathcal{A}_1}$ are defined on the same sets; however, it is too weak for the theorem I wish to prove. I give an example.

Example 6.2.1 (Omission of sort-similarity)
Consider the $\{s\}$-sorted signature, $\Sigma_0 = (\{s\}, \{f\})$ with $f :\to s$, and the $\{s\}$-sorted signature, $\Sigma_1 = (\{s\}, \{g\})$ with $g :\to s$. Clearly the sum of these two signatures is not sort-similar. I will use the same set $\{a\}$ for both the Σ_0-algebra $\mathcal{A}_0$ and Σ_1-algebra $\mathcal{A}_1$. Let $f^{\mathcal{A}_0} = a$ and $g^{\mathcal{A}_1} = a$. Then $f \equiv_{\mathcal{A}_0 \oplus \mathcal{A}_1} g$ since $i_{\mathcal{A}_0 \oplus \mathcal{A}_1}(f) = i_{\mathcal{A}_0 \oplus \mathcal{A}_1}(g)$, but f and g are not equated by $\equiv_{\mathcal{A}_0} \oplus \equiv_{\mathcal{A}_1}$. ■

It may be possible to work without sort-similarity—possibly the definition of $\equiv_{\mathcal{A}_0} \oplus \equiv_{\mathcal{A}_0}$ could be modified or the theorem could be changed to have a one-way implication. However, as far as I can tell, sort-similarity is not problematic, except for the fact that I need to show that I can define a Σ-algebra to represent labels without having to consider terms with sort P. The reason I need to consider this is because when giving an extension to a process algebra, I often want to add new function symbols to sort P, however for sorts other than P, I am are only interested in adding new sorts. I will present an argument below to show that this does not cause problems.

There are other choices one could make for $\mathcal{A}_0 \oplus \mathcal{A}_1$, such as defining the carrier set for a sort s as the intersection of $\mathcal{A}_{0,s}$ and $\mathcal{A}_{1,s}$. As far as I can see there is no utility in taking this approach.

I now show that the following result holds.

Proposition 6.2.3 (Sum of algebras)
Let $\Sigma_i = (S_i, F_i)$ for $i = 0, 1$ be two signatures with $\Sigma_0 \oplus\!\!> \Sigma_1$ defined and sort-similar. For $i = 0, 1$, let $\mathcal{A}_i = \{\mathcal{A}_{i,s}\}_{s \in S_i}$ together with $\{f^{\mathcal{A}_i} \mid f \in F_i\}$ be Σ_i-algebras with $\mathcal{A}_0 \oplus \mathcal{A}_1$ defined. Then $\mathcal{A}_0 \oplus \mathcal{A}_1$ is a $(\Sigma_0 \oplus \Sigma_1)$-algebra.

Proof: $\Sigma_0 \oplus\!\!> \Sigma_1 = (S_0 \cup S_1, F_0 \cup F_1)$, and $\biguplus^{s}_{i=0,1} A_i$ is an $S_0 \cup S_1$-sorted family of non-empty carrier sets. Moreover, it is clear that for each $f \in F_0 \cup F_1$ there is a total function $f^{\mathcal{A}_0 \oplus \mathcal{A}_1}$ with the appropriate argument sorts and result sort, regardless of whether f is in the intersection of F_0 and F_1 or not. ■

Since I have a $(\Sigma_0 \oplus\!\!> \Sigma_1)$-algebra, I can consider the unique homomorphism $i_{\mathcal{A}_0 \oplus \mathcal{A}_1}$ from $\mathbf{T}(\Sigma_0 \oplus\!\!> \Sigma_1)$ to $\mathcal{A}_0 \oplus \mathcal{A}_1$. I would like to show that the equivalence induced by this is the same as $\equiv_{\mathcal{A}_0} \oplus \equiv_{\mathcal{A}_1}$, and then I will know that when applying these results to process algebras, I can merely form the sum of the algebras under consideration.

Theorem 6.2.1 (Equivalence induced by sums of algebras)
Let $\Sigma_i = (S_i, F_i)$ for $i = 0, 1$ be two signatures with $\Sigma_0 \oplus\!\!> \Sigma_1$ defined and sort-similar. For $i = 0, 1$, let $\mathcal{A}_i = \{\mathcal{A}_{i,s}\}_{s \in S_i}$ together with $\{f^{\mathcal{A}_i} \mid f \in F_i\}$ be Σ_i-algebras with $\mathcal{A}_0 \oplus \mathcal{A}_1$ defined. Let $\equiv_{\mathcal{A}_0 \oplus \mathcal{A}_1}$ denote the congruence defined by the unique homomorphism from $\mathbf{T}(\Sigma_0 \oplus\!\!> \Sigma_1)$ to $\mathcal{A}_0 \oplus \mathcal{A}_1$; similarly, $\equiv_{\mathcal{A}_0}$ and $\equiv_{\mathcal{A}_1}$. Then $\equiv_{\mathcal{A}_0 \oplus \mathcal{A}_1} \; = \; \equiv_{\mathcal{A}_0} \oplus \equiv_{\mathcal{A}_1}$.

Proof: First note, that for any $t \in \mathbf{T}(\Sigma_0 \oplus\!\!> \Sigma_1)$, $t \in \mathbf{T}(\Sigma_0) \cup \mathbf{T}(\Sigma_1)$ by Proposition 6.2.2. I first wish to show that if $t \in \mathbf{T}(\Sigma_i)$, then $i_{\mathcal{A}_0 \oplus \mathcal{A}_1}(t) = i_{\mathcal{A}_i}(t)$ for $i = 0, 1$. I proceed by induction on the structure of t.

If t is a constant symbol, then this is clearly true. If $t = f(t_1, \ldots, t_n)$, then assume $t \in \mathbf{T}(\Sigma_0)$ without loss of generality. $i_{\mathcal{A}_0 \oplus \mathcal{A}_1}(f(t_1, \ldots, t_n)) = f^{\mathcal{A}_0}(i_{\mathcal{A}_0 \oplus \mathcal{A}_1}(t_1), \ldots, i_{\mathcal{A}_0 \oplus \mathcal{A}_1}(t_n)) = f^{\mathcal{A}_0}(i_{\mathcal{A}_0}(t_1), \ldots, i_{\mathcal{A}_0}(t_n))$ by the induction hypothesis and this gives the required result.

Next, let $t, t' \in \mathbf{T}(\Sigma_0 \oplus\!\!> \Sigma_1)$.

$t \equiv_{\mathcal{A}_0 \oplus \mathcal{A}_1} t' \Rightarrow t \equiv_{\mathcal{A}_0} \oplus \equiv_{\mathcal{A}_1} t'$: Let $t \equiv_{\mathcal{A}_0 \oplus \mathcal{A}_1} t'$, then since I have equivalence this means that t and t' have the same sort and hence they must be both in $\mathbf{T}(\Sigma_0)$ or $\mathbf{T}(\Sigma_1)$. Assume without loss of generality that $t, t' \in \mathbf{T}(\Sigma_0)$. By definition $i_{\mathcal{A}_0 \oplus \mathcal{A}_1}(t) = i_{\mathcal{A}_0 \oplus \mathcal{A}_1}(t')$, hence $i_{\mathcal{A}_0}(t) = i_{\mathcal{A}_0}(t')$. Therefore $t \equiv_{\mathcal{A}_0} t'$ and hence $t \equiv_{\mathcal{A}_0} \oplus \equiv_{\mathcal{A}_1} t'$ as required.

$t \equiv_{\mathcal{A}_0} \oplus \equiv_{\mathcal{A}_1} t' \Rightarrow t \equiv_{\mathcal{A}_0 \oplus \mathcal{A}_1} t'$: I will proceed by induction on the definition of $\equiv_{\mathcal{A}_0} \oplus \equiv_{\mathcal{A}_1}$.

- $t \equiv_{\mathcal{A}_0} t'$ or $t \equiv_{\mathcal{A}_1} t'$. Assume $t \equiv_{\mathcal{A}_0} t'$, then by definition $i_{\mathcal{A}_0}(t) = i_{\mathcal{A}_0}(t')$ and also $t, t' \in \mathbf{T}(\Sigma_0)$. It can be shown by a simple induction proof that $i_{\mathcal{A}_0}(t) = i_{\mathcal{A}_0 \oplus \mathcal{A}_1}(t)$. If t is a constant then this is immediate; otherwise if $t = f(t_1, \ldots, t_n)$ then $i_{\mathcal{A}_0}(f(t_1, \ldots, t_n) = f^{\mathcal{A}_0}(i_{\mathcal{A}_0}(t_1), \ldots, i_{\mathcal{A}}(t_n)) = f^{\mathcal{A}_0 \oplus \mathcal{A}_1}(i_{\mathcal{A}_0 \oplus \mathcal{A}_1}(t_1), \ldots, i_{\mathcal{A}_0 \oplus \mathcal{A}_1}(t_n))$ by the induction hypothesis and the definition of $\mathcal{A}_0 \oplus \mathcal{A}_1$, and this gives the required result.
- Clearly both $t \equiv_{\mathcal{A}_0 \oplus \mathcal{A}_1} t'$ and $t \equiv_{\mathcal{A}_0} \oplus \equiv_{\mathcal{A}_1} t'$ if $t = t'$.
- If $t \equiv_{\mathcal{A}_0} \oplus \equiv_{\mathcal{A}_1} t'$ by a symmetry argument, then $t' \equiv_{\mathcal{A}_0} \oplus \equiv_{\mathcal{A}_1} t$, by a shorter inference, hence $t' \equiv_{\mathcal{A}_0 \oplus \mathcal{A}_1} t$ and therefore $t \equiv_{\mathcal{A}_0 \oplus \mathcal{A}_1} t'$ since $\equiv_{\mathcal{A}_0 \oplus \mathcal{A}_1}$ is symmetric.
- If $t \equiv_{\mathcal{A}_0} \oplus \equiv_{\mathcal{A}_1} t'$ by a transitivity argument, then there exists $t'' \in \mathbf{T}(\Sigma_0 \oplus\!\!> \Sigma_1)$ such that $t \equiv_{\mathcal{A}_0} \oplus \equiv_{\mathcal{A}_1} t''$ and $t'' \equiv_{\mathcal{A}_0} \oplus \equiv_{\mathcal{A}_1} t'$, hence by a shorter inference $t \equiv_{\mathcal{A}_0 \oplus \mathcal{A}_1} t''$ and $t'' \equiv_{\mathcal{A}_0 \oplus \mathcal{A}_1} t'$, and therefore $t \equiv_{\mathcal{A}_0 \oplus \mathcal{A}_1} t'$ since $\equiv_{\mathcal{A}_0 \oplus \mathcal{A}_1}$ is transitive.
- If $t \equiv_{\mathcal{A}_0} \oplus \equiv_{\mathcal{A}_1} t'$ by a congruence argument, then $t = f(t_1, \ldots, t_n)$, $t' = f'(t'_1, \ldots, t'_n)$ and $t_i \equiv_{\mathcal{A}_0} \oplus \equiv_{\mathcal{A}_1} t'_i$ for $1 \leqslant i \leqslant n$, so by a shorter inference $t_i \equiv_{\mathcal{A}_0 \oplus \mathcal{A}_1} t'_i$ for $1 \leqslant i \leqslant n$. Hence $t \equiv_{\mathcal{A}_0 \oplus \mathcal{A}_1} t'$, since $\equiv_{\mathcal{A}_0 \oplus \mathcal{A}_1}$ is a congruence. ■

These are general results for Σ-algebras. However, in my work with formats, I deal with specific signatures that contain a distinguished element P and with specific conditions on function symbols that do not have a result sort P. To recap, given a signature $\Sigma = (S \cup \{\mathsf{P}\}, F)$ where S does not contain P,

> for any function symbol $f \in F$ such that $f : s_1 \ldots s_n \to s$, whenever $s \neq P$ then for all $1 \leqslant i \leqslant n$, $s_i \neq \mathsf{P}$.

Hence

$$\Sigma' = (S, \{f \mid f \in F, f : s_1 \ldots s_n \to s, s \in S\})$$

is a valid signature, and in fact $\mathbf{T}(\Sigma') = \mathbf{T}(\Sigma)_{\overline{\mathsf{P}}}$ because of the fact that all function symbols in Σ' do not refer to P. This is very convenient since I am interested in congruences over $\mathbf{T}(\Sigma)_{\overline{\mathsf{P}}}$ when considering bisimulations over $\mathbf{T}(\Sigma)_{\mathsf{P}}$, and hence I can consider $\mathbf{T}(\Sigma)_{\overline{\mathsf{P}}}$ as the term algebra $\mathbf{T}(\Sigma')$, and therefore consider Σ'-algebras as representations of the labels of process algebras.

6.2.2.1 Conservativity

It can also be shown that $\equiv_{\mathcal{A}_0 \oplus \mathcal{A}_1}$ is conservative with respect to $\equiv_{\mathcal{A}_0}$ and $\equiv_{\mathcal{A}_1}$.

Proposition 6.2.4 (Sum of algebras induces conservative equivalence) Let $\Sigma_i = (S_i, F_i)$ for $i = 0, 1$ be two signatures with $\Sigma_0 \oplus\!\!> \Sigma_1$ defined and sort-similar. For $i = 0, 1$, let $\mathcal{A}_i = \{\mathcal{A}_{i,s}\}_{s \in S_i}$ together with $\{f^{\mathcal{A}_i} \mid f \in F_i\}$ be Σ_i-algebras with $\mathcal{A}_0 \oplus \mathcal{A}_1$ defined. Let $\equiv_{\mathcal{A}_0 \oplus \mathcal{A}_1}$ denote the congruence defined by the unique homomorphism from $\mathbf{T}(\Sigma_0 \oplus\!\!> \Sigma_1)$ to $\mathcal{A}_0 \oplus \mathcal{A}_1$; similarly, $\equiv_{\mathcal{A}_0}$ and $\equiv_{\mathcal{A}_1}$. Then $\equiv_{\mathcal{A}_0} \oplus \equiv_{\mathcal{A}_1}$ is conservative with respect to both $\equiv_{\mathcal{A}_0}$ and $\equiv_{\mathcal{A}_1}$.

Proof: Let $t, t' \in \mathbf{T}(\Sigma_0)$ (without loss of generality) such that $t \equiv_{\mathcal{A}_0} \oplus \equiv_{\mathcal{A}_1} t'$. Hence $t \equiv_{\mathcal{A}_0 \oplus \mathcal{A}_1} t'$, so $i_{\mathcal{A}_0 \oplus \mathcal{A}_1}(t) = i_{\mathcal{A}_0 \oplus \mathcal{A}_1}(t')$. Hence since $t, t' \in \mathbf{T}(\Sigma_0)$, $i_{\mathcal{A}_0}(t) = i_{\mathcal{A}_0}(t')$ and $t \equiv_{\mathcal{A}_0} t'$ as required. ■

6.2.2.2 Compatibility

Finally, I need to consider compatibility in the light of sums of algebras. In the results that involve compatibility, I have only required that $\equiv_0 \oplus \equiv_1$ be compatible with $R_0 \cup R_1$. As I have said previously, it is not clear that this can be dealt with in a more general manner since compatibility depends on the label terms that appear in the rules. One approach could be to require the congruences to be compatible with any label term, but as seen earlier in this chapter, this would impose very strong restrictions on the Σ-algebras that could be used, and is not ideal.

However, it seems in the case of sums of algebras, I can make headway because of the additional condition required on the signature.

Proposition 6.2.5 (Sum of algebras induces compatibility)
Let $\Sigma_i = (S_i, F_i)$ for $i = 0, 1$ be two signatures with $\Sigma_0 \oplus\!\!> \Sigma_1$ defined and sort-similar. For $i = 0, 1$, let $\mathcal{A}_i = \{\mathcal{A}_{i,s}\}_{s \in S_i}$ together with $\{f^{\mathcal{A}_i} \mid f \in F_i\}$ be Σ_i-algebras with $\mathcal{A}_0 \oplus \mathcal{A}_1$ defined. Let $\equiv_{\mathcal{A}_0 \oplus \mathcal{A}_1}$ denote the congruence defined by the unique homomorphism from $\mathbf{T}(\Sigma_0 \oplus\!\!> \Sigma_1)$ to $\mathcal{A}_0 \oplus \mathcal{A}_1$; similarly, $\equiv_{\mathcal{A}_0}$ and $\equiv_{\mathcal{A}_1}$. Moreover, let $\mathcal{E}_i = (\Sigma_i, R_i)$ for $i = 0, 1$ be two eTSSs such that $\equiv_{\mathcal{A}_i}$ is compatible with R_i. Then $\equiv_{\mathcal{A}_0} \oplus \equiv_{\mathcal{A}_1}$ is compatible with $R_0 \cup R_1$.

Proof: Since $\equiv_{\mathcal{A}_i}$ is compatible for R_i for $i = 0, 1$, I have two S_i-sorted sets of terms C_i for $i = 0, 1$ containing terms that appear on a transition in a premise or as an argument to the function in the source of the conclusion of a rule in R_i such that

> whenever $\sigma(\eta) \equiv_{\mathcal{A}_i} \mu$ for $\mu \in \mathbf{T}(\Sigma)_{\overline{\mathsf{P}}}$, there exists a substitution σ' such that $\mu = \sigma'(\eta)$ and $\sigma(z) \equiv_{\mathcal{A}_i} \sigma'(z)$ for all $z \in \mathrm{Var}_{\overline{\mathsf{P}}}(\eta)$.

So I need to consider the following sorts:

$s \in S_0 - S_1$**:** Consider $\eta \in \mathbb{T}(\Sigma_0)_s$. Since $s \notin S_1$, then clearly no terms in $\mathbf{T}(\Sigma_1)$ can be equivalent under $\equiv_{\mathcal{A}_0} \oplus \equiv_{\mathcal{A}_1}$ to $\sigma(\eta)$ for any σ. So consider $\sigma(\eta) \equiv_{\mathcal{A}_0} \oplus \equiv_{\mathcal{A}_1} \mu$ for some $\mu \in \mathbf{T}(\Sigma_0)_s$, then I can show that $\sigma(\eta) \equiv_{\mathcal{A}_0} \mu$ and hence there exists a substitution σ' such that $\mu = \sigma'(\eta)$ and $\sigma(z) \equiv_{\mathcal{A}_0} \sigma'(z)$ for all $z \in \mathrm{Var}_{\overline{\mathsf{P}}}(\eta)$. But since $\equiv_{\mathcal{A}_0}$ is contained in $\equiv_{\mathcal{A}_0} \oplus \equiv_{\mathcal{A}_1}$, then $\sigma(z) \equiv_{\mathcal{A}_0} \oplus \equiv_{\mathcal{A}_1} \sigma'(z)$ for all $z \in \mathrm{Var}_{\overline{\mathsf{P}}}(\eta)$.

$s \in S_1 - S_0$**:** this is proved in a similar way to the previous one.

$s \in S_0 \cap S_1$ Consider $\eta \in \mathbb{T}(\Sigma_0)_s \cap \mathbb{T}(\Sigma_1)_s$. Consider $\sigma(\eta) \equiv_{\mathcal{A}_0} \oplus \equiv_{\mathcal{A}_1} \mu$ for some $\mu \in \mathbf{T}(\Sigma_0)_s \cap \mathbf{T}(\Sigma_1)_s$. It can be shown that $\sigma(\eta) \equiv_{\mathcal{A}_0} \mu$ (and also that

$\sigma(\eta) \equiv_{\mathcal{A}_1} \mu)$. Hence there exists a substitution σ' such that $\mu = \sigma'(\eta)$ and $\sigma(z) \equiv_{\mathcal{A}_0} \sigma'(z)$ for all $z \in \mathrm{Var}_{\overline{\mathsf{P}}}(\eta)$. But since $\equiv_{\mathcal{A}_0}$ is contained in $\equiv_{\mathcal{A}_0} \oplus \equiv_{\mathcal{A}_1}$, then $\sigma(z) \equiv_{\mathcal{A}_0} \oplus \equiv_{\mathcal{A}_1} \sigma'(z)$ for all $z \in \mathrm{Var}_{\overline{\mathsf{P}}}(\eta)$. ■

To see why this does not hold in general for two equivalences $\equiv_0$ and $\equiv_1$ defined on $\mathbf{T}(\Sigma_0)$ and $\mathbf{T}(\Sigma_1)$ compatible with two sets of rules R_0 and R_1 respectively, it may be that for η that appears in the rules of R_0, that $\sigma(\eta)$ is also in $\mathbf{T}(\Sigma_1)$ and there exists $\mu' \in \mathbf{T}(\Sigma_1)$, such that $\sigma(\eta) \equiv_1 \mu'$ but it is not possible to find σ' such that $\mu' = \sigma'(\eta)$ and $\sigma(z) \equiv_1 \sigma'(z)$ for all $z \in \mathrm{Var}_{\overline{\mathsf{P}}}(\eta)$, since η does not appear in the rules of R_1.

The results relating to sums can be summarised as follows. Given the sort-similar sum of two signatures, and two algebras of each signature respectively, then

- the sum of two algebras is an algebra of the sum of the signatures,
- the equivalence induced by the sum of two algebras is the sum of the equivalences induced by each algebra,
- moreover, this equivalence is conservative with respect to each of the other two equivalences,
- furthermore if given the sum of two eTSSs with the two congruences compatible with each eTSS respectively, then the equivalence induced by the sum of the algebras is compatible with the sum of the eTSSs.

Hence, the condition of sort similarity is sufficient to obtain these relationships between algebras and their sums.

In this section, I have looked at what constraints the conditions required for the results of the previous two chapter impose on the algebras that are used to represent structured labels, and I have also defined a new concept which produces some positive results in the situations where it can be applied. It is clear that these are not serious constraints and hence I will now proceed to look at applying these results to process algebras from the literature.

6.3 Using extended *tyft/tyxt* format to express process algebras

In this section, I look at how process algebras can be expressed in the extended *tyft/tyxt* format. Before proceeding with the examples, I need to present a way in which to specify infinite sets of rules by the use of rule schemas. Schemas also allow the description of constants, and rules that do matching on labels. This method of describing rules also is useful for expressing communication in CCS, where because of the restrictions of the format, it is not possible to have repeating variables in the labels of the premises. I assume a set of schema variables $\mathcal{V}$ disjoint from any other variable set and any collection of function symbols.

Definition 6.3.1 (Rule schema)
Let $\Sigma = (S \cup \mathsf{P}, F)$ be a suitable signature. Let $\mathcal{V}$ be an $(S \cup \mathsf{P})$-sorted set of schema variables, and V a $(S \cup \mathsf{P})$-sorted set of variables. Consider $T(\Sigma, V \cup \mathcal{V})$, the set of open terms over variables and schema variables. A rule schema is denoted

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

where I is an index set, $p_i, p'_i, p, p' \in T(\Sigma, V \cup \mathcal{V})_{\mathsf{P}}$, and $\lambda_i, \lambda \in T(\Sigma, V \cup \mathcal{V})_{\overline{\mathsf{P}}}$ for $i \in I$. $C_1, \ldots, C_n$ are conditions on the schema variables involving equality and inequality, and conditions on whether terms are open or closed and which sorts they may have. This notation is understood to mean the set of rules created by replacing each schema variable by any closed term in $\mathbb{T}(\Sigma)$ in accordance with the conditions specified. This can also be viewed as applying all closed substitutions on schema variables which satisfy the conditions. ■

Proposition 6.3.1 (Rule schemas in extended *tyft/tyxt* format)
Let $\Sigma = (S \cup \mathsf{P}, F)$ be a suitable signature. Given a rule schema

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

in extended *tyft/tyxt* format, i.e. the conditions hold for the extended *tyft/tyxt* format on the variables from V and with respect to the form of the rule, but the terms are drawn from $T(\Sigma, V \cup \mathcal{V})$; then the rules generated by the schema are in extended *tyft/tyxt* format. Moreover, if the rule schema is well-founded with respect to the variables in V, then all the rules generated are well-founded.

Proof: The conditions on the variables from V hold, and replacing any schema variable with a closed term, cannot add new variables from V to the rules, hence any rules generated by the rules schema is in extended *tyft/tyxt* format. By a similar argument, it is possible to show that well-founded schema rules generate well-founded rules. ■

It is not possible to do a similar proof for compatibility since a schema variable (with a sort that is not P) may be replaced by a term in the transitions of a premise or as an argument to the function in the source of the conclusion. To check for compatibility with respect to a given congruence it is necessary to ensure that any closed term that may be substituted for an schema variable is not congruent to any other closed term. This may appear to be a strong condition, but since the congruence respects sorts, it may only mean that congruence must be the identity on one sort.

In the earlier work involving formats, most authors have used schemas, although not explicitly. In my 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.

Notation In the following, I use $x, y, \ldots$ to denote variables of sort P, $z, \ldots$ subscripted with a sort to denote variables of sort $\overline{\mathsf{P}}$. I also use $X, Y, \ldots$ to denote schema variables of sort P and $Z, \ldots$ subscripted with a sort to denote schema variables of sort $\overline{\mathsf{P}}$.

6.3.1 CCS

Here I will look at expressing CCS in this format. Let A and Const be two disjoint sets, disjoint from the variables and schema-variables and any other function symbols. I will also use A as a sort name—this does not cause problems. Consider the algebra, Σ_{CCS},

$$\begin{aligned}(\ & \mathsf{A}, \mathsf{Act}, \tau\mathsf{Act}, \mathsf{P}; \\ & \{\mathsf{a}\}_{\mathsf{a}\in\mathsf{A}}, \tau, \mathsf{nil}, \{\mathsf{Cn}\}_{\mathsf{Cn}\in\mathsf{Const}}; \\ & \mathsf{act}, \overline{\mathsf{act}}, \mathsf{pref}, \overline{\mathsf{pref}}, \mathsf{pref}_\tau, \mathsf{plus}, \mathsf{par}, \mathsf{rename}, \mathsf{restrict} \ )\end{aligned}$$

with the sorts

$\mathsf{a} :\to \mathsf{A} \quad \forall \mathsf{a} \in \mathsf{A}$	$\mathsf{act} : \mathsf{A} \to \mathsf{Act}$	$\mathsf{plus} : \mathsf{P}, \mathsf{P} \to \mathsf{P}$
$\tau :\to \tau\mathsf{Act}$	$\overline{\mathsf{act}} : \mathsf{A} \to \mathsf{Act}$	$\mathsf{par} : \mathsf{P}, \mathsf{P} \to \mathsf{P}$
$\mathsf{nil} :\to \mathsf{P}$	$\mathsf{pref} : \mathsf{A}, \mathsf{P} \to \mathsf{P}$	$\mathsf{rename} : \mathsf{A}, \mathsf{A}, \mathsf{P} \to \mathsf{P}$
$\mathsf{Cn} :\to \mathsf{P} \quad \forall \mathsf{Cn} \in \mathsf{Const}$	$\overline{\mathsf{pref}} : \mathsf{A}, \mathsf{P} \to \mathsf{P}$	$\mathsf{restrict} : \mathsf{A}, \mathsf{P} \to \mathsf{P}$
	$\mathsf{pref}_\tau : \mathsf{P} \to \mathsf{P}$	

Also assume that there is a way in which closed terms are assigned to elements of Const, as in the standard CCS approach where the meaning of constant agent A is given by a process P, and this is described by a defining equation $A \overset{\text{def}}{=} P$. The rules and rules schemas are given in Tables 6.1 and 6.2. Let the rules from Table 6.1 and the rules generated by the rule schemas in Table 6.2 be denoted R_{CCS}, and the eTSS defined by Σ_{CCS} and R_{CCS} be $\mathcal{E}_{\text{CCS}}$. I have used names for the operators, as opposed to symbols, for clarity. I have defined the signature so that the non-τ actions and τ actions have different sorts. This is not strictly necessary.

In most cases the rules are straightforward. There are three different prefix operators pref, $\overline{\mathsf{pref}}$ and pref_τ, and two label operators act and $\overline{\mathsf{act}}$ giving the correct form to the action depending on the prefix function. The choice and parallel operator rules in Table 6.1 are straightforward. The parallel communication rule schema in Table 6.2 requires that actions are matched, hence the form. Note that this rule schema could have been written as

$$\left\{ \quad \frac{x \xrightarrow{\mathsf{act}(Z_\mathsf{A})} y \quad x' \xrightarrow{\overline{\mathsf{act}}(Z_\mathsf{A})} y'}{\mathsf{par}(x, x') \xrightarrow{\tau} \mathsf{par}(y, y')} \quad \right\}$$

This may look as though it is not in extended *tyft/tyxt* form because of the repeated variables in the labels of the premises; however since they are schema variables and by the argument above, it is clear that all rules generated by this schema are in extended *tyft/tyxt* format.

$$\frac{}{\mathsf{pref}(z_{\mathsf{A}}, x) \xrightarrow{\mathsf{act}(z_{\mathsf{A}})} x} \qquad \frac{}{\overline{\mathsf{pref}}(z_{\mathsf{A}}, x) \xrightarrow{\overline{\mathsf{act}}(z_{\mathsf{A}})} x} \qquad \frac{}{\mathsf{pref}_{\tau}(x) \xrightarrow{\tau} x}$$

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

$$\frac{x \xrightarrow{\tau} y}{\mathsf{plus}(x, x') \xrightarrow{\tau} y} \qquad \frac{x \xrightarrow{\tau} y}{\mathsf{plus}(x', x) \xrightarrow{\tau} y}$$

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

$$\frac{x \xrightarrow{\tau} y}{\mathsf{par}(x, x') \xrightarrow{\tau} \mathsf{par}(y, x')} \qquad \frac{x \xrightarrow{\tau} y}{\mathsf{par}(x', x) \xrightarrow{\tau} \mathsf{par}(x', y)}$$

$$\frac{x \xrightarrow{\tau} y}{\mathsf{rename}(z_{\mathsf{A}}, z'_{\mathsf{A}}, x) \xrightarrow{\tau} \mathsf{rename}(z_{\mathsf{A}}, z'_{\mathsf{A}}, y)} \qquad \frac{x \xrightarrow{\tau} y}{\mathsf{restrict}(z'_{\mathsf{A}}, x) \xrightarrow{\tau} \mathsf{restrict}(z'_{\mathsf{A}}, y)}$$

Table 6.1: Rules for CCS

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

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

$$\left\{ \quad \frac{x \xrightarrow{\mathsf{act}(Z_{\mathsf{A}})} y}{\mathsf{rename}(Z'_{\mathsf{A}}, Z''_{\mathsf{A}}, x) \xrightarrow{\mathsf{act}(Z''_{\mathsf{A}})} \mathsf{rename}(Z'_{\mathsf{A}}, Z''_{\mathsf{A}}, y)} \quad \middle| \quad Z_{\mathsf{A}} = Z'_{\mathsf{A}} \quad \right\}$$

$$\left\{ \quad \frac{x \xrightarrow{\overline{\mathsf{act}}(Z_{\mathsf{A}})} y}{\mathsf{rename}(Z'_{\mathsf{A}}, Z''_{\mathsf{A}}, x) \xrightarrow{\overline{\mathsf{act}}(Z''_{\mathsf{A}})} \mathsf{rename}(Z'_{\mathsf{A}}, Z''_{\mathsf{A}}, y)} \quad \middle| \quad Z_{\mathsf{A}} = Z'_{\mathsf{A}} \quad \right\}$$

$$\left\{ \quad \frac{x \xrightarrow{\mathsf{act}(Z_{\mathsf{A}})} y}{\mathsf{rename}(Z'_{\mathsf{A}}, Z''_{\mathsf{A}}, x) \xrightarrow{\mathsf{act}(Z_{\mathsf{A}})} \mathsf{rename}(Z'_{\mathsf{A}}, Z''_{\mathsf{A}}, y)} \quad \middle| \quad Z_{\mathsf{A}} \neq Z'_{\mathsf{A}} \quad \right\}$$

$$\left\{ \quad \frac{x \xrightarrow{\overline{\mathsf{act}}(Z_{\mathsf{A}})} y}{\mathsf{rename}(Z'_{\mathsf{A}}, Z''_{\mathsf{A}}, x) \xrightarrow{\overline{\mathsf{act}}(Z_{\mathsf{A}})} \mathsf{rename}(Z'_{\mathsf{A}}, Z''_{\mathsf{A}}, y)} \quad \middle| \quad Z_{\mathsf{A}} \neq Z'_{\mathsf{A}} \quad \right\}$$

$$\left\{ \quad \frac{x \xrightarrow{\mathsf{act}(Z_{\mathsf{A}})} y}{\mathsf{restrict}(Z'_{\mathsf{A}}, x) \xrightarrow{\mathsf{act}(Z_{\mathsf{A}})} \mathsf{restrict}(Z'_{\mathsf{A}}, y)} \quad \middle| \quad Z_{\mathsf{A}} \neq Z'_{\mathsf{A}} \quad \right\}$$

$$\left\{ \quad \frac{x \xrightarrow{\overline{\mathsf{act}}(Z_{\mathsf{A}})} y}{\mathsf{restrict}(Z'_{\mathsf{A}}, x) \xrightarrow{\overline{\mathsf{act}}(Z_{\mathsf{A}})} \mathsf{restrict}(Z'_{\mathsf{A}}, y)} \quad \middle| \quad Z_{\mathsf{A}} \neq Z'_{\mathsf{A}} \quad \right\}$$

$$\left\{ \quad \frac{X \xrightarrow{z_{\mathsf{Act}}} y}{\mathsf{Cn} \xrightarrow{z_{\mathsf{Act}}} y} \quad \middle| \quad \mathsf{Cn} \overset{\mathrm{def}}{=} X, \mathsf{Cn} \in \mathsf{Const} \quad \right\}$$

$$\left\{ \quad \frac{X \xrightarrow{\tau} y}{\mathsf{Cn} \xrightarrow{\tau} y} \quad \middle| \quad \mathsf{Cn} \overset{\mathrm{def}}{=} X, \mathsf{Cn} \in \mathsf{Const} \quad \right\}$$

Table 6.2: Rule schemas for CCS

The τ-forms of the restriction and relabelling operators are straightforward and given in Table 6.1. The non-τ forms given as rule schemas are more complex and require some discussion. Because of the syntactic nature of the rules, and the need to do syntactic matching in the parallel communication rule, relabelling has been difficult to implement and what I have achieved here is a subset of CCS relabelling. The operator has three arguments—the action to be changed, the action that it should be changed to and the process, and the rules are defined in the obvious way. Hence for any finite subset of A, it is possible to define a function, by repeated (but only finite) use of the rename operator. However, in general, it is not possible to define a function on all of A, if A is infinite. There is an exception to this and that involves constant functions and can be done by introducing a new operator $\mathsf{rename'}$ and the rules

$$\frac{x \xrightarrow{\mathsf{act}(z_{\mathsf{A}})} y}{\mathsf{rename'}(z'_{\mathsf{A}}, x) \xrightarrow{\mathsf{act}(z'_{\mathsf{A}})} \mathsf{rename}(z'_{\mathsf{A}}, y)} \qquad \frac{x \xrightarrow{\overline{\mathsf{act}}(z_{\mathsf{A}})} y}{\mathsf{rename'}(z'_{\mathsf{A}}, x) \xrightarrow{\overline{\mathsf{act}}(z'_{\mathsf{A}})} \mathsf{rename}(z'_{\mathsf{A}}, y)}$$

$$\frac{x \xrightarrow{\tau} y}{\mathsf{rename'}(z'_{\mathsf{A}}, x) \xrightarrow{\tau} \mathsf{rename}(z'_{\mathsf{A}}, y)}$$

The restriction rules have a similar limitation—restriction can only be done on a finite subset of A, by repeated use of the restriction operator which takes as arguments the action to be restricted on and the process term. There may be ways to get around these limitations and reasonably remain within the world of rules and rule schemas, but I have not yet seen how to do this.

The next step in the definition is to look for the algebra that will be used to represent the actual process algebra labels. In this case, there is no requirement for an equivalence relation between labels, and hence the term algebra and syntactic equivalence can be used.

In conclusion, it is clear that CCS is not simple to represent in this format, mainly because of the matching required in the communication rules. There is another approach to matching which involves the use of undefined transitions—I will demonstrate this in the next example.

Finally, since all rules in R_{CCS} are in extended *tyft/tyxt* format, $\sim_{\mathbf{Id}}^{\mathcal{E}_{\text{CCS}}}$ is a congruence for $\mathsf{nil}, \mathsf{pref}, \overline{\mathsf{pref}}, \mathsf{pref}_{\tau}, \mathsf{plus}, \mathsf{par}, \mathsf{rename}, \mathsf{rename'}, \mathsf{restrict}$ and $\{\mathsf{Cn}\}_{\mathsf{Cn}\in\mathsf{Const}}$.

6.3.2 Variants of CCS

I now look at how one signature and rule set can be used to represent different process algebras by varying the algebra used to represent the labels.

Let A be a (countably infinite) set of labels (denoted a) disjoint from previously defined sets and let L be a (countably infinite) set of labels (denoted l) disjoint from previously defined sets. I also use A and L as sort names.

Consider the signature Σ_{CCSGen},

$$\begin{array}{l}(\ \mathsf{A}, \mathsf{L}, \mathsf{Act}, \mathsf{P}; \\ \quad \{\mathsf{a}\}_{\mathsf{a}\in\mathsf{A}}, \tau, \{\mathsf{l}\}_{\mathsf{l}\in\mathsf{L}}, \mathsf{nil}, \{\mathsf{Cn}\}_{\mathsf{Cn}\in\mathsf{Const}}, \bot_{\mathsf{A}}, \bot_{\mathsf{Act}}, \bot_{\mathsf{L}}; \\ \quad \mathsf{pref}, \overline{\mathsf{pref}}, \mathsf{pref}_{\tau}, \mathsf{plus}, \mathsf{par}, \mathsf{rename}, \mathsf{restrict}, \mathsf{loc} \\ \quad \mathsf{act}, \overline{\mathsf{act}}, \mathsf{comb}, \mathsf{app}, \mathsf{ren}, \mathsf{restr} \ )\end{array}$$

with the sorts

$\mathsf{a} :\rightarrow \mathsf{A} \quad \forall \mathsf{a} \in \mathsf{A}$	$\mathsf{nil} :\rightarrow \mathsf{P}$
$\mathsf{l} :\rightarrow \mathsf{L} \quad \forall \mathsf{l} \in \mathsf{L}$	$\mathsf{Cn} :\rightarrow \mathsf{P} \quad \forall \mathsf{Cn} \in \mathsf{Const}$
$\tau :\rightarrow \mathsf{Act}$	$\mathsf{pref}_{\tau} : \mathsf{P} \rightarrow \mathsf{P}$
$\mathsf{act} : \mathsf{A}, \mathsf{L} \rightarrow \mathsf{Act}$	$\mathsf{pref} : \mathsf{A}, \mathsf{P} \rightarrow \mathsf{P}$
$\overline{\mathsf{act}} : \mathsf{A}, \mathsf{L} \rightarrow \mathsf{Act}$	$\overline{\mathsf{pref}} : \mathsf{A}, \mathsf{P} \rightarrow \mathsf{P}$
$\mathsf{app} : \mathsf{L}, \mathsf{Act} \rightarrow \mathsf{Act}$	$\mathsf{loc} : \mathsf{L}, \mathsf{P} \rightarrow \mathsf{P}$
$\mathsf{comb} : \mathsf{Act}, \mathsf{Act} \rightarrow \mathsf{Act}$	$\mathsf{par} : \mathsf{P}, \mathsf{P} \rightarrow \mathsf{P}$
$\mathsf{restr} : \mathsf{A}, \mathsf{Act} \rightarrow \mathsf{Act}$	$\mathsf{restrict} : \mathsf{A}, \mathsf{P} \rightarrow \mathsf{P}$
$\mathsf{ren} : \mathsf{A}, \mathsf{A}, \mathsf{Act} \rightarrow \mathsf{Act}$	$\mathsf{rename} : \mathsf{A}, \mathsf{A}, \mathsf{P} \rightarrow \mathsf{P}$
$\bot_{\mathsf{A}} :\rightarrow \mathsf{A}$	$\mathsf{plus} : \mathsf{P}, \mathsf{P} \rightarrow \mathsf{P}$
$\bot_{\mathsf{L}} :\rightarrow \mathsf{L}$	
$\bot_{\mathsf{Act}} :\rightarrow \mathsf{Act}$	

Also assume that there is a way in which closed terms are assigned to elements of Const. The rules are given in Table 6.3 and the rules schemas in Table 6.4. Let the rules given by these rules and rule schemas be denoted R_{CCSGen}, and the eTSS defined by Σ_{CCSGen} and R_{CCSGen} be $\mathcal{E}_{\mathrm{CCSGen}}$.

Note that I have introduced a 'bottom' element for each label sort. I carry this through to the carrier sets for the label algebra, and use these elements to

$$\frac{}{\mathsf{pref}(z_{\mathsf{A}}, x) \xrightarrow{\mathsf{act}(z_{\mathsf{A}},\mathsf{new}(z_{\mathsf{L}}))} \mathsf{loc}(z_{\mathsf{L}}, x)}$$

$$\frac{}{\overline{\mathsf{pref}}(z_{\mathsf{A}}, x) \xrightarrow{\overline{\mathsf{act}}(z_{\mathsf{A}},\mathsf{new}(z_{\mathsf{L}}))} \mathsf{loc}(z_{\mathsf{L}}, x)}$$

$$\frac{}{\mathsf{pref}_{\tau}(x) \xrightarrow{\tau} x}$$

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

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

$$\frac{x \xrightarrow{z_{\mathsf{Act}}} y}{\mathsf{rename}(z_{\mathsf{A}}, z'_{\mathsf{A}}, x) \xrightarrow{\mathsf{ren}(z_{\mathsf{A}},z'_{\mathsf{A}},z_{\mathsf{Act}})} \mathsf{rename}(z_{\mathsf{A}}, z'_{\mathsf{A}}, y)}$$

$$\frac{x \xrightarrow{z_{\mathsf{Act}}} y}{\mathsf{restrict}(z_{\mathsf{A}}, x) \xrightarrow{\mathsf{restr}(z_{\mathsf{A}},z_{\mathsf{Act}})} \mathsf{restrict}(z_{\mathsf{A}}, y)}$$

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

$$\frac{x \xrightarrow{z_{\mathsf{Act}}} y}{\mathsf{loc}(z_{\mathsf{L}}, x) \xrightarrow{\mathsf{app}(z_{\mathsf{L}},z_{\mathsf{Act}})} \mathsf{loc}(z_{\mathsf{L}}, y)}$$

Table 6.3: Rules for CCSGen

$$\left\{ \quad \frac{X \xrightarrow{z_{\mathsf{Act}}} y}{\mathsf{Cn} \xrightarrow{z_{\mathsf{Act}}} y} \quad \middle| \quad \mathsf{Cn} \overset{\text{def}}{=} X, \mathsf{Cn} \in \mathsf{Const} \quad \right\}$$

Table 6.4: Rule schemas for CCSGen

represent labels which are undefined. Any transition with a label term that is equivalent to an undefined constant is not considered in the bisimulation. This allows for a simpler definition of communication. Instead of requiring matching in the communication rule, there is now a term on the transition that is the combination of the two terms from the transitions of each premise. If the combination is meaningful (for example, in CCS, if one label is the complement of the other), then the transition will be considered in the bisimulation. This also permits use of the same signature and rule set for different process algebras.

6.3.2.1 CCS with locations

In CCS with locations, transitions are labelled with an additional string of atomic locations and a location prefixing operator is introduced. Each time an action is performed, a location is associated with it, and the location prefixing operator ensures that location information about past actions with similar locations are added to the transition of a new action. See page 10 for an example.

I need to define a Σ_{CCSGen}-algebra $\mathcal{A}_L$ to represent the actual process algebra labels of CCS with locations. For all the algebras in this section, I will assume that there is a set of actions A (with an action denoted a) such that there is an action in A for each a in A. I will also assume that there is a set of labels L (with a label denoted l) such that there is a label in L for each l in L. I need to define the carrier set for each sort (see Table 6.6), and each function of the algebra (see Table 6.5). The function $\mathsf{comb}^{\mathcal{A}_L}$ illustrates how the labels for communication work.

$$
\begin{array}{rcl}
\tau^{\mathcal{A}_L} & = & \tau \\
\mathsf{a}^{\mathcal{A}_L} & = & a \quad \forall \mathsf{a} \in \mathsf{A} \\
\mathsf{l}^{\mathcal{A}_L} & = & l \quad \forall \mathsf{l} \in \mathsf{L} \\
\mathsf{act}^{\mathcal{A}_L}(\zeta_1,\zeta_2) & = & \begin{cases} (\zeta_1,\zeta_2) & \text{if } \zeta_1 \in A \cup \overline{A} \text{ and } \zeta_2 \in L^+ \\ \bot_{\mathsf{Act}} & \text{otherwise} \end{cases} \\
\overline{\mathsf{act}}^{\mathcal{A}_L}(\zeta_1,\zeta_2) & = & \begin{cases} (\overline{\zeta}_1,\zeta_2) & \text{if } \zeta_1 \in A \text{ and } \zeta_2 \in L^+ \\ \bot_{\mathsf{Act}} & \text{otherwise} \end{cases} \\
\mathsf{app}^{\mathcal{A}_L}(\zeta_1,\zeta_2) & = & \begin{cases} (\zeta_{2,1},\zeta_1\zeta_{2,2}) & \text{if } \zeta_1 \in L \text{ and } \zeta_2 = (\zeta_{2,1},\zeta_{2,2}) \\ & \text{with } \zeta_{2,1} \in A \text{ and } \zeta_{2,2} \in L^+ \\ \bot_{\mathsf{Act}} & \text{otherwise} \end{cases} \\
\mathsf{comb}^{\mathcal{A}_L}(\zeta_1,\zeta_2) & = & \begin{cases} \tau & \text{if for } i = 1,2,\ \zeta_i = (\zeta_{i,1},\zeta_{i,2}),\ \zeta_{i,1} \in A \cup \overline{A}, \\ & \zeta_{i,2} \in L^+ \text{ and } \zeta_{1,1} = \overline{\zeta}_{2,1} \\ \bot_{\mathsf{Act}} & \text{otherwise} \end{cases} \\
\mathsf{ren}^{\mathcal{A}_L}(\zeta_1,\zeta_2,\zeta_3) & = & \begin{cases} (\zeta_2,\zeta_{3,2}) & \text{if } \zeta_1,\zeta_2 \in A,\ \zeta_3 = (\zeta_{3,1},\zeta_{3,2}) \text{ with } \zeta_{3,1} \in A, \\ & \zeta_{3,2} \in L^+ \text{ and } \zeta_1 = \zeta_{3,1} \\ (\overline{\zeta}_2,\zeta_{3,2}) & \text{if } \zeta_1,\zeta_2 \in A,\ \zeta_3 = (\zeta_{3,1},\zeta_{3,2}) \text{ with } \zeta_{3,1} \in \overline{A}, \\ & \zeta_{3,2} \in L^+ \text{ and } \overline{\zeta}_1 = \zeta_{3,1} \\ \zeta_3 & \text{if } \zeta_1,\zeta_2 \in A,\ \zeta_3 = (\zeta_{3,1},\zeta_{3,2}) \text{ with } \zeta_{3,1} \in A \cup \overline{A}, \\ & \zeta_{3,2} \in L^+,\ \zeta_1 \neq \zeta_{3,1} \text{ and } \overline{\zeta}_1 \neq \zeta_{3,1} \\ \bot_{\mathsf{Act}} & \text{otherwise} \end{cases} \\
\mathsf{restr}^{\mathcal{A}_L}(\zeta_1,\zeta_2) & = & \begin{cases} \zeta_2 & \text{if } \zeta_1 \in A,\ \zeta_2 = (\zeta_{2,1},\zeta_{2,2}) \text{ with } \zeta_{2,1} \in A \cup \overline{A}, \\ & \zeta_{2,2} \in L^+,\ \zeta_1 \neq \zeta_{2,1} \text{ and } \overline{\zeta}_1 \neq \zeta_{2,1} \\ \bot_{\mathsf{Act}} & \text{otherwise} \end{cases} \\
\bot_{\mathsf{A}}^{\mathcal{A}_L} & = & \bot_{\mathsf{A}} \\
\bot_{\mathsf{L}}^{\mathcal{A}_L} & = & \bot_{\mathsf{L}} \\
\bot_{\mathsf{Act}}^{\mathcal{A}_L} & = & \bot_{\mathsf{Act}}
\end{array}
$$

Table 6.5: Functions $\mathcal{A}_L$

Sort	Carrier set
A	$A \cup \{\perp_{\mathsf{A}}\}$
L	$L \cup \{\perp_{\mathsf{L}}\}$
Act	$((A \cup \overline{A}) \times L^{+}) \cup \{\tau\} \cup \{\perp_{\mathsf{Act}}\}$

Table 6.6: Carrier sets for $\mathcal{A}_L$

Sort	Carrier set
A	$A \cup \{\perp_{\mathsf{A}}\}$
L	L
Act	$((A \cup \overline{A}) \times L^{+}) \cup \{\tau\} \cup \{\perp_{\mathsf{Act}}\}$

Table 6.7: Carrier sets for $\mathcal{A}_{L_n}$

6.3.2.2 CCS with n locations

A variant on CCS with locations is to only allow a finite set of locations. This can be done by defining a Σ_{CCSGen}-algebra $\mathcal{A}_{L_n}$ to represent the actual process algebra labels. Let num be a bijection from L to $\mathbb{N}$, and assume that n, the number of locations, is given. Denote l such that $num(l) = k$ as l_k. The carrier sets and functions for $\mathcal{A}_{L_n}$ are given in Tables 6.7 and 6.8 respectively. Note that the only difference from $\mathcal{A}_L$ is in $\mathsf{act}^{\mathcal{A}_{L_n}}$ and $\overline{\mathsf{act}}^{\mathcal{A}_{L_n}}$.

There are different choices that can be made for sorts and functions. For example, the carrier for L could be $\{l_1, \ldots, l_n\} \cup \{\perp_{\mathsf{L}}\}$ and the function $\mathsf{l}^{\mathcal{A}_{L_n}}$ could be defined as

$$\mathsf{l}^{\mathcal{A}_{L_n}} = \begin{cases} l_i & \text{if } num(l) = i \text{ and } 1 \leqslant i \leqslant n \\ \perp_{\mathsf{L}} & \text{otherwise} \end{cases}$$

or

$$\mathsf{l}^{\mathcal{A}_{L_n}} = \begin{cases} l_i & \text{if } num(l) = i \text{ and } 1 \leqslant i \leqslant n \\ l_n & \text{if } num(l) = i \text{ and } i \geqslant n \end{cases}$$

$$
\begin{array}{rcl}
\tau^{\mathcal{A}_{L_n}} & = & \tau \\
\mathsf{a}^{\mathcal{A}_{L_n}} & = & a \quad \forall \mathsf{a} \in \mathsf{A} \\
\mathsf{l}^{\mathcal{A}_{L_n}} & = & l \quad \forall \mathsf{l} \in \mathsf{L} \\
\mathsf{act}^{\mathcal{A}_{L_n}}(\zeta_1,\zeta_2) & = & \begin{cases} (\zeta_1,\zeta_2) & \text{if } \zeta_1 \in A \cup \overline{A},\ \zeta_2 \in L \text{ and } num(l) \leqslant n \\ \bot_{\mathsf{Act}} & \text{otherwise} \end{cases} \\
\overline{\mathsf{act}}^{\mathcal{A}_{L_n}}(\zeta_1,\zeta_2) & = & \begin{cases} (\overline{\zeta}_1,\zeta_2) & \text{if } \zeta_1 \in A,\ \zeta_2 \in L \text{ and } num(l) \leqslant n \\ \bot_{\mathsf{Act}} & \text{otherwise} \end{cases} \\
\mathsf{app}^{\mathcal{A}_{L_n}}(\zeta_1,\zeta_2) & = & \begin{cases} (\zeta_{2,1},\zeta_1\zeta_{2,2}) & \text{if } \zeta_1 \in L \text{ and } \zeta_2 = (\zeta_{2,1},\zeta_{2,2}) \\ & \text{with } \zeta_{2,1} \in A \text{ and } \zeta_{2,2} \in L^+ \\ \bot_{\mathsf{Act}} & \text{otherwise} \end{cases} \\
\mathsf{comb}^{\mathcal{A}_{L_n}}(\zeta_1,\zeta_2) & = & \begin{cases} \tau & \text{if for } i = 1,2,\ \zeta_i = (\zeta_{i,1},\zeta_{i,2}),\ \zeta_{i,1} \in A \cup \overline{A}, \\ & \zeta_{i,2} \in L^+ \text{ and } \zeta_{1,1} = \overline{\zeta}_{2,1} \\ \bot_{\mathsf{Act}} & \text{otherwise} \end{cases} \\
\mathsf{ren}^{\mathcal{A}_{L_n}}(\zeta_1,\zeta_2,\zeta_3) & = & \begin{cases} (\zeta_2,\zeta_{3,2}) & \text{if } \zeta_1,\zeta_2 \in A,\ \zeta_3 = (\zeta_{3,1},\zeta_{3,2}) \text{ with } \zeta_{3,1} \in A, \\ & \zeta_{3,2} \in L^+ \text{ and } \zeta_1 = \zeta_{3,1} \\ (\overline{\zeta}_2,\zeta_{3,2}) & \text{if } \zeta_1,\zeta_2 \in A,\ \zeta_3 = (\zeta_{3,1},\zeta_{3,2}) \text{ with } \zeta_{3,1} \in \overline{A}, \\ & \zeta_{3,2} \in L^+ \text{ and } \overline{\zeta}_1 = \zeta_{3,1} \\ \zeta_3 & \text{if } \zeta_1,\zeta_2 \in A,\ \zeta_3 = (\zeta_{3,1},\zeta_{3,2}) \text{ with } \zeta_{3,1} \in A \cup \overline{A}, \\ & \zeta_{3,2} \in L^+,\ \zeta_1 \neq \zeta_{3,1} \text{ and } \overline{\zeta}_1 \neq \zeta_{3,1} \\ \bot_{\mathsf{Act}} & \text{otherwise} \end{cases} \\
\mathsf{restr}^{\mathcal{A}_{L_n}}(\zeta_1,\zeta_2) & = & \begin{cases} \zeta_2 & \text{if } \zeta_1 \in A,\ \zeta_2 = (\zeta_{2,1},\zeta_{2,2}) \text{ with } \zeta_{2,1} \in A \cup \overline{A}, \\ & \zeta_{2,2} \in L^+,\ \zeta_1 \neq \zeta_{2,1} \text{ and } \overline{\zeta}_1 \neq \zeta_{2,1} \\ \bot_{\mathsf{Act}} & \text{otherwise} \end{cases} \\
\bot_{\mathsf{A}}^{\mathcal{A}_{L_n}} & = & \bot_{\mathsf{A}} \\
\bot_{\mathsf{L}}^{\mathcal{A}_{L_n}} & = & \bot_{\mathsf{L}} \\
\bot_{\mathsf{Act}}^{\mathcal{A}_{L_n}} & = & \bot_{\mathsf{Act}}
\end{array}
$$

Table 6.8: Functions for $\mathcal{A}_{L_n}$

Sort	Carrier set
A	$A \cup \{\bot_{\mathsf{A}}\}$
L	$\mathbb{N}$
Act	$A_m = (\underbrace{(A \cup \overline{A} \cup \{\tau\} \cup \{\delta\}) \times \ldots \times (A \cup \overline{A} \cup \{\tau\} \cup \{\delta\})}_{m \text{ times}} \cup \{\bot_{\mathsf{Act}}\}$

Table 6.9: Carrier sets for $\mathcal{A}_m$

Note that in the second definition that the bottom element is not used. Instead all constants that would be mapped to an element outside the finite label set are mapped to l_n.

6.3.2.3 Multiprocessor CCS

Multiprocessor CCS has transition labels which are m-tuples and represent m processors upon which actions can occur. Each element of the tuple can either be idle (represented by δ) or be filled with an action or τ-action. See page 24 for an example. A formal definition of a subset of this process algebra is given in Section 6.4.

I now need to define a Σ_{CCSGen}-algebra $\mathcal{A}_m$ to represent the actual process algebra labels. The carrier sets and functions for $\mathcal{A}_m$ are given in Tables 6.9 and 6.10 respectively. Let $(a_1, \ldots, a_m) +_m (b_1, \ldots, b_m)$ be defined as equal to $(c_1, \ldots, c_m)$ where for all $1 \leqslant i \leqslant n$

$$c_i = \begin{cases} a_i & \text{if } b_i = \delta \\ b_i & \text{if } a_i = \delta \\ \tau & \text{if } a_i = \overline{b}_i \end{cases}$$

This partial function is used in communication.

An important issue is whether these process algebras expressed in this format are actually the same as the original process algebras, or at least whether they result in bisimulations that are identical. For the location process algebras, I have used a slight modification whereby transitions with both actions and locations are

$$
\begin{array}{rcl}
\tau^{\mathcal{A}_m} & = & \tau \\
\mathsf{a}^{\mathcal{A}_m} & = & a \quad \forall \mathsf{a} \in \mathsf{A} \\
\mathsf{l}^{\mathcal{A}_m} & = & num(l) \quad \forall \mathsf{l} \in \mathsf{L} \\
\mathsf{act}^{\mathcal{A}_m}(\zeta_1, \zeta_2) & = & \begin{cases} \overbrace{(\delta, \dots, \zeta_1, \dots, \delta)}^{m} & \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 position of the vector} \\ \bot_{\mathsf{Act}} & \text{otherwise} \end{cases} \\
\overline{\mathsf{act}}^{\mathcal{A}_m}(\zeta_1, \zeta_2) & = & \begin{cases} \overbrace{(\delta, \dots, \overline{\zeta}_1, \dots, \delta)}^{m} & \text{if } \zeta_1 \in A \text{ and } \zeta_2 \in \mathbb{N}^+ \text{ where} \\ & \zeta_1 \text{ is the } \zeta_2\text{-th position of the vector} \\ \bot_{\mathsf{Act}} & \text{otherwise} \end{cases} \\
\mathsf{app}^{\mathcal{A}_m}(\zeta_1, \zeta_2) & = & \begin{cases} \zeta_2 & \text{if } \zeta_1 \in \mathbb{N}^+ \text{ and } \zeta_2 \in A_m - \{\bot_{\mathsf{Act}}\} \\ \bot_{\mathsf{Act}} & \text{otherwise} \end{cases} \\
\mathsf{comb}^{\mathcal{A}_m}(\zeta_1, \zeta_2) & = & \begin{cases} \zeta_1 +_m \zeta_2 & \text{if } \zeta_1, \zeta_2 \in A_m - \{\bot_{\mathsf{Act}}\} \text{ and } \zeta_1 +_m \zeta_2 \text{ defined} \\ \bot_{\mathsf{Act}} & \text{otherwise} \end{cases} \\
\mathsf{ren}^{\mathcal{A}_m}(\zeta_1, \zeta_2, \zeta_3) & = & \begin{cases} \bot_{\mathsf{Act}} & \text{otherwise} \end{cases} \\
\mathsf{restr}^{\mathcal{A}_m}(\zeta_1, \zeta_2) & = & \begin{cases} \zeta_2 & \text{if } \zeta_1 \in A_m - \{\bot_{\mathsf{Act}}\},\ \zeta_2 = (\zeta_{2,1}, \dots, \zeta_{2,n}) \\ & \text{with } \zeta_1 \neq \zeta_{2,i} \text{ and } \overline{\zeta}_1 \neq \zeta_{2,i} \text{ for all } 1 \leqslant i \leqslant n \\ \bot_{\mathsf{Act}} & \text{otherwise} \end{cases} \\
\bot_{\mathsf{A}}^{\mathcal{A}_m} & = & \bot_{\mathsf{A}} \\
\bot_{\mathsf{L}}^{\mathcal{A}_m} & = & 0 \\
\bot_{\mathsf{Act}}^{\mathcal{A}_m} & = & \bot_{\mathsf{Act}}
\end{array}
$$

Table 6.10: Functions for $\mathcal{A}_m$

used in the communication rules, whereas in the original definition, transitions with τ-actions were derived from basic CCS rules. This does not result in a different transition system. For the case of multiprocessor CCS, it can be seen by inspection that when quotiented by the induced equivalence, the same transition system is obtained.

6.3.3 Discussion

In the previous section, I showed three different algebras for representing the labels of three different actual process algebras. Each of these algebras induce an equivalence over the ground terms of the signature, and this equivalence can then be used to define a bisimulation or semantic equivalence. Let the equivalence defined by the algebra $\mathcal{A}_L$ be denoted $\sim_L$ (this is the bisimulation for standard CCS with locations). Let the equivalence defined by $\mathcal{A}_{L_n}$ be denoted $\sim_{L_n}$ (this is the bisimulation for CCS with n locations). Finally, let the equivalence defined by $\mathcal{A}_m$ be denoted $\sim_m$ (this is the bisimulation for CCS with m multiprocessors). Note that $\sim_1$ is the same as Milner's strong equivalence over CCS [Kri96].

It is clear that all of these equivalences are congruences with respect to the process operators, since they are all in extended *tyft/tyxt* format.

I next look at the relationships between these equivalences. First note that for $m = 1$, $\sim_L \subset \sim_1 = \sim$. For $m \geqslant 2$, $\sim_L \neq \sim_m$. The counter-examples are as follows.

Counter-example 6.3.1 (Location equivalence not comparable with multiprocessor equivalence)

$$(a.c \mid \overline{c})\backslash\{c\} + (c \mid \overline{c}.a)\backslash\{c\} \quad \begin{array}{c} \sim_L \\ \not\sim_m \end{array} \quad a \mid \tau$$

$$(a.c + b.d \mid \overline{c}.b + \overline{d}.a)\backslash\{c, d\} \quad \begin{array}{c} \sim_m \\ \not\sim_L \end{array} \quad (a.c + b.d \mid \overline{c}.b + \overline{d}.a)\backslash\{c, d\} + a.\tau.b$$

(I give the processes in slightly different notation to that used earlier in the chapter to aid readability.) ■

The second pair above would seem to indicate multiprocessor bisimulation may be the same as global cause bisimulation (causal bisimulation). However, the following counter-examples show that this is not the case.

Counter-example 6.3.2 (Global cause equivalence not comparable with multiprocessor equivalence)

$$(a.c \mid \overline{c})\backslash\{c\} + (c \mid \overline{c}.a)\backslash\{c\} \quad \underset{\not\sim_m}{\sim_c} \quad a \mid \tau$$

$$a \mid b \quad \underset{\not\sim_c}{\sim_m} \quad a \mid b + a.b$$

(I give the processes in slightly different notation to that used earlier in the chapter to aid readability.) ■

I now look at $\sim_L$ and $\sim_{L_n}$. First note that $\sim_{L_1} \subset \sim$ since $a \mid b \not\sim_{L_1} a.b + b.a$. Also $\sim_L \subset \sim_{L_1}$, for example

$$(a.c.b \mid d.\overline{c}.b)\backslash\{c\} \quad \underset{\not\sim_L}{\sim_{L_1}} \quad (a.c.(b \mid b) \mid d.\overline{c})\backslash\{c\}$$

This example also shows that $\sim_{L_n} \subset \sim_{L_1}$ for all $n \geqslant 2$.

An interesting question relates to how many locations are required to obtain full location equivalence. Intuitively, it seems that at least that two locations are needed to distinguish between different parallel components, and this is supported by the above processes. Because each action may occur at any location, and the greatest arity of any dynamic operator (namely the parallel operator) is two, it may be possible to show that at most two location are needed. This question cannot be answered by the results in Chapter 5 and is an issue for further work.

Finally, I need to consider $\sim_{L_n}$ and $\sim_n$. For $n = 1$, $\sim_{L_1} \subset \sim_1 = \sim$. For $n \geqslant 2$, the processes give in Counter-Example 6.3.1 can be used to show that $\sim_{L_n}$ and $\sim_n$ are not comparable.

Since these process algebras are not comparable for the most part, it is not possible to apply the results of Chapter 6. I will consider some different process algebras in the next section. In this section I have demonstrated that a number of process algebras can be expressed in this format. An obvious question relates to whether there are any process algebras which cannot be expressed. The global process algebra of Kiehn [Kie94] gives some problems. First, in the definition of the parallel operator for communication, syntactic substitution is used to take information from the transitions of the premises into the processes in the target of the conclusion, and this information then affects further transitions made by the target term. However, in this case (but perhaps not generally) it is possible to define a new operator to achieve the same effect. A more serious problem is the fact that the definition of the bisimulation requires that only transitions with fresh causes are to be considered in the bisimulation, and if this is weakened a different equivalence is obtained. It is not clear how this can be dealt with in the new format.

6.4 Comparing two semantic equivalences

In this section, I will compare the multiprocessor equivalence of Krishnan [Kri96] and the pomset bisimulation of Castellani [Cas88]. These equivalences have not been compared previously. The approach I will take involves results from the previous chapter, and is done in two distinct steps. The first step involves to showing that the extension is a refining one, and the second step involves showing two equivalences are the same.

I will work with a similar signature to that of the previous examples; however, it will be somewhat simplified to deal only with the operators for which pomset bisimulation is defined.

Let A be a set of actions disjoint from any collection of variables and let L be a set of labels disjoint from previously defined sets. Consider the signature Σ_{MP} with S_{MP} and F_{MP} as follows,

$$\begin{array}{l}(\ \mathsf{A}, \mathsf{L}, \mathsf{Act}, \mathsf{P};\ \ \{\mathsf{a}\}_{\mathsf{a}\in\mathsf{A}}, \{\mathsf{l}\}_{\mathsf{l}\in\mathsf{L}}, \mathsf{nil}, \{\mathsf{Cn}\}_{\mathsf{Cn}\in\mathsf{Const}}, \bot_{\mathsf{A}}, \bot_{\mathsf{Act}}, \bot_{\mathsf{L}};\\ \quad \mathsf{pref}, \mathsf{plus}, \mathsf{par}, \mathsf{act}, \mathsf{comb},\ \ )\end{array}$$

and

$$\frac{}{\mathsf{pref}(z_{\mathsf{A}}, x) \xrightarrow{\mathsf{act}(z_{\mathsf{A}},\mathsf{new}(z_{\mathsf{L}}))} x}$$

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

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

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

$$\left\{ \quad \frac{X \xrightarrow{z_{\mathsf{Act}}} y}{\mathsf{Cn} \xrightarrow{z_{\mathsf{Act}}} y} \quad \middle| \quad \mathsf{Cn} \overset{\mathrm{def}}{=} X, \mathsf{Cn} \in \mathsf{Const} \quad \right\}$$

Table 6.11: Rules and rule schemas for MP

$\mathsf{a} :\rightarrow \mathsf{A} \quad \forall \mathsf{a} \in \mathsf{A}$	$\mathsf{nil} :\rightarrow \mathsf{P}$
$\mathsf{l} :\rightarrow \mathsf{L} \quad \forall \mathsf{l} \in \mathsf{L}$	$\mathsf{Cn} :\rightarrow \mathsf{P} \quad \forall \mathsf{Cn} \in \mathsf{Const}$
$\mathsf{act} : \mathsf{A}, \mathsf{L} \rightarrow \mathsf{Act}$	$\mathsf{pref} : \mathsf{A}, \mathsf{P} \rightarrow \mathsf{P}$
$\mathsf{comb} : \mathsf{Act}, \mathsf{Act} \rightarrow \mathsf{Act}$	$\mathsf{par} : \mathsf{P}, \mathsf{P} \rightarrow \mathsf{P}$
$\perp_{\mathsf{A}} :\rightarrow \mathsf{A}$	$\mathsf{plus} : \mathsf{P}, \mathsf{P} \rightarrow \mathsf{P}$
$\perp_{\mathsf{L}} :\rightarrow \mathsf{L}$	
$\perp_{\mathsf{Act}} :\rightarrow \mathsf{Act}$	

Also assume that there is a way in which closed terms are assigned to elements of Const. The rules and rule schemas are given in Table 6.11. Let the rules given by these rules and rule schemas be denoted R_{MP}, and the eTSS defined by Σ_{MP} and R_{MP} be $\mathcal{E}_{\mathrm{MP}}$.

6.4.1 Multiprocessor CCS

Multiprocessor CCS was mentioned in the previous section, but here I give the rules for the subset with which I am dealing.

6.4.1.1 Definition of n multiprocessor equivalence

Let A be a set of actions and consider the grammar P

$$P ::= a.P \mid \mathbf{0} \mid P + P \mid P|P$$

for $a \in A$. The elements of A are atomic actions.

Let $\mathcal{O}_n$ denote the set of n-tuples over $A \cup \{\delta\}$, and let $\mathit{Allocate}(a) = \{O \in \mathcal{O}_n \mid \exists i, O(i) = a, \forall j \neq i, O(j) = \delta\}$. Also define the partial function $+_n$ on $\mathcal{O} \times \mathcal{O} \to \mathcal{O}$ as $O_1 +_n O_2 = O$ where

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

(Note that the notation used in this section does not follow the conventions used in the rest of the document.) The rules are given in Table 6.12. The bisimulation is defined in the standard manner.

An n *multiprocessor bisimulation* $\mathcal{R}$ is a binary relation such that for any $(p, q) \in \mathcal{R}$ and $S \in \mathcal{O}_n$, the following holds

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

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

6.4.1.2 Expressing n multiprocessor equivalence

I now need to define a Σ_{MP}-algebra $\mathcal{A}_n$ to represent the actual process algebra labels for multiprocessor CCS. I assume that L is a infinite countable set and num is a bijection from L to $\mathbb{N}$, and moreover that n, the number of processors, is given.

Also let $(a_1, \ldots, a_n) +_n (b_1, \ldots, b_n)$ be defined as equal to $(c_1, \ldots, c_n)$ where for all $1 \leqslant i \leqslant n$

$$c_i \quad = \quad \begin{cases} a_i & \text{if } b_i = \delta \\ b_i & \text{if } a_i = \delta \end{cases}$$

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

$$\frac{p \xrightarrow{S} p'}{p + q \xrightarrow{S} p'} \qquad \frac{p \xrightarrow{S} p'}{q + p \xrightarrow{S} p'}$$

$$\frac{p \xrightarrow{S} p'}{p \mid q \xrightarrow{S} p' \mid q} \qquad \frac{p \xrightarrow{S} p'}{q \mid p \xrightarrow{S} q \mid p'}$$

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

Table 6.12: Rules for n multiprocessor CCS

Sort	**Carrier set**
A	$A \cup \{\bot_{\mathsf{A}}\}$
L	$\mathbb{N}$
Act	$A_n = (\underbrace{(A \cup \{\delta\}) \times \ldots \times (A \cup \{\delta\})}_{n \text{ times}} \cup \{\bot_{\mathsf{Act}}\}$

Table 6.13: Carrier sets for $\mathcal{A}_n$

$$
\begin{array}{rcl}
\mathsf{a}^{\mathcal{A}_n} & = & a \quad \forall \mathsf{a} \in \mathsf{A} \\
\mathsf{l}^{\mathcal{A}_n} & = & num(l) \quad \forall \mathsf{l} \in \mathsf{L} \\
\mathsf{act}^{\mathcal{A}_n}(\zeta_1, \zeta_2) & = & \begin{cases} (\overbrace{\delta, \ldots, \delta}^{\zeta_2 - 1}, \zeta_1, \overbrace{\delta, \ldots, \delta}^{n - \zeta_2}) & \text{if } \zeta_1 \in A \cup \{\tau\} \text{ and } \zeta_2 \in \mathbb{N}^+ \\ & \text{where } \zeta_1 \text{ is in the } \zeta_2\text{-th} \\ & \text{position of the vector} \\ \perp_{\mathsf{Act}} & \text{otherwise} \end{cases} \\
\mathsf{comb}^{\mathcal{A}_n}(\zeta_1, \zeta_2) & = & \begin{cases} \zeta_1 +_n \zeta_2 & \text{if } \zeta_1, \zeta_2 \in A_n - \{\perp_{\mathsf{Act}}\} \text{ and } \zeta_1 +_n \zeta_2 \text{ defined} \\ \perp_{\mathsf{Act}} & \text{otherwise} \end{cases} \\
\perp_{\mathsf{A}}^{\mathcal{A}_n} & = & \perp_{\mathsf{A}} \\
\perp_{\mathsf{L}}^{\mathcal{A}_n} & = & 0 \\
\perp_{\mathsf{Act}}^{\mathcal{A}_n} & = & \perp_{\mathsf{Act}}
\end{array}
$$

Table 6.14: Functions for $\mathcal{A}_n$

The carrier sets and functions for $\mathcal{A}_n$ are given in Tables 6.13 and 6.14.

An important point to note is that the bisimulation equivalences that will be considered in this section will disregard transitions which are labeled with elements that are equivalent to undefined elements. Hence any transition label with sort s that is equivalent under the induced equivalence to $\perp_s$ will be ignored when defining semantic equivalence between process terms. However, I will use the same notation, namely $\sim^{\mathcal{E}}_{\equiv_{\mathcal{A}}}$.

6.4.2 An extension

I now wish to create a new process algebra to represent pomset CCS. I do this by giving an extension to $\mathcal{E}_{\mathrm{MP}}$. I first give the definition of pomset CCS.

6.4.2.1 Definition of pomset equivalence

Let A be a set of actions and consider the grammar P with $a \in A$

$$P ::= a : P \mid \mathrm{NIL} \mid P + P \mid P|P$$

The elements of A are atomic actions. General actions come from the grammar B with $a \in A$

$$B ::= a : B \mid \text{NIL} \mid B|B.$$

These actions are referred to as Act. There are two axioms over Act

$$\begin{aligned} u \mid v &= v \mid u \\ u \mid (v \mid w) &= (u \mid v) \mid w \end{aligned}$$

and they define an equivalence $\equiv$ over Act (Note that the notation used in this section does not follow the conventions of the rest of the document.) The rules are given in Table 6.15. The bisimulation is defined with respect to $\equiv$, but apart from that has the standard definition.

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

6.4.2.2 Expressing pomset equivalence as an extension

Consider the signature Σ_{MPExt} with S_{MPExt} and F_{MPExt} as follows,

$$\begin{aligned} (\ & \mathsf{A}, \mathsf{L}, \mathsf{Act}, \mathsf{Act}_{\mathsf{new}}, \mathsf{P}; \\ & \{\mathsf{a}\}_{\mathsf{a} \in \mathsf{A}}, \{\mathsf{l}\}_{\mathsf{l} \in \mathsf{L}}, \mathsf{nil}, \{\mathsf{Cn}\}_{\mathsf{Cn} \in \mathsf{Const}}, \bot_{\mathsf{A}}, \bot_{\mathsf{Act}}, \bot_{\mathsf{Act}_{\mathsf{new}}}, \bot_{\mathsf{L}}; \\ & \mathsf{pref}, \mathsf{plus}, \mathsf{par}, \mathsf{act}, \mathsf{comb}, \mathsf{comb}_{\mathsf{new}}, \mathsf{concat}\ ) \end{aligned}$$

and

$$\frac{}{a:p \xrightarrow{a:\mathrm{NIL}} p}$$

$$\frac{p \xrightarrow{u} p'}{a:p \xrightarrow{a:u} p'}$$

$$\frac{p \xrightarrow{u} p'}{p+q \xrightarrow{u} p'} \qquad \frac{p \xrightarrow{u} p'}{q+p \xrightarrow{u} p'}$$

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

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

Table 6.15: Rules for pomset CCS

$\mathsf{a} :\to \mathsf{A} \quad \forall \mathsf{a} \in \mathsf{A}$	$\mathsf{nil} :\to \mathsf{P}$
$\mathsf{l} :\to \mathsf{L} \quad \forall \mathsf{l} \in \mathsf{L}$	$\mathsf{Cn} :\to \mathsf{P} \quad \forall \mathsf{Cn} \in \mathsf{Const}$
$\mathsf{act} : \mathsf{A}, \mathsf{L} \to \mathsf{Act}$	$\mathsf{pref} : \mathsf{A}, \mathsf{P} \to \mathsf{P}$
$\mathsf{concat} : \mathsf{A}, \mathsf{Act} \to \mathsf{Act}_{\mathsf{new}}$	
$\mathsf{concat} : \mathsf{A}, \mathsf{Act}_{\mathsf{new}} \to \mathsf{Act}_{\mathsf{new}}$	
$\mathsf{comb} : \mathsf{Act}, \mathsf{Act} \to \mathsf{Act}$	$\mathsf{par} : \mathsf{P}, \mathsf{P} \to \mathsf{P}$
$\mathsf{comb}_{\mathsf{new}} : \mathsf{Act}, \mathsf{Act} \to \mathsf{Act}_{\mathsf{new}}$	
$\mathsf{comb}_{\mathsf{new}} : \mathsf{Act}, \mathsf{Act}_{\mathsf{new}} \to \mathsf{Act}_{\mathsf{new}}$	
$\mathsf{comb}_{\mathsf{new}} : \mathsf{Act}_{\mathsf{new}}, \mathsf{Act} \to \mathsf{Act}_{\mathsf{new}}$	
$\mathsf{comb}_{\mathsf{new}} : \mathsf{Act}_{\mathsf{new}}, \mathsf{Act}_{\mathsf{new}} \to \mathsf{Act}_{\mathsf{new}}$	
$\bot_{\mathsf{A}} :\to \mathsf{A}$	$\mathsf{plus} : \mathsf{P}, \mathsf{P} \to \mathsf{P}$
$\bot_{\mathsf{L}} :\to \mathsf{L}$	
$\bot_{\mathsf{Act}} :\to \mathsf{Act}$	
$\bot_{\mathsf{Act}_{\mathsf{new}}} :\to \mathsf{Act}_{\mathsf{new}}$	

Notice the overloading of concat and $\mathsf{comb_{new}}$. Also assume that there is a way in which closed terms are assigned to elements of Const. The rules and rule schemas are given in Table 6.16. Let the rules given by these rules and rule schemas be denoted R_{MPExt}, and the eTSS defined by Σ_{MPExt} and R_{MPExt} be $\mathcal{E}_{\text{MPExt}}$. Since Σ_{MPExt} contains everything that Σ_{MP} contains, I will use Σ_{MPExt} for $\Sigma_{\text{MP}} \oplus\!\!> \Sigma_{\text{MPExt}}$.

I now need to define a Σ_{MPExt}-algebra $\mathcal{A}_{pom}$ to represent the actual process algebra labels where $\mathcal{C}$ is defined as $c ::= a \mid a : c \mid c|c$ for $a \in A$. Also let $\mathcal{C}'$ be defined as $c' ::= a \mid c'|c'$ for $a \in A$. Clearly $A \subset \mathcal{C}' \subset \mathcal{C}$. I will assume there is an congruence $\equiv_{\mathcal{C}}$ on $\mathcal{C}'$ and $\mathcal{C}$ generated by the following axioms as in Castellani's original definition [Cas88]

$$c \mid c' = c' \mid c \quad \text{and} \quad c \mid (c' \mid c'') = (c \mid c') \mid c''.$$

The carrier sets and functions for $\mathcal{A}_{pom}$ are given in Tables 6.17 and 6.18.

Define $\equiv_{\mathcal{A}_p}$ as follows: if $\alpha, \beta \in \mathbf{T}(\Sigma_{\text{MPExt}})_{\overline{\mathsf{P}}}$, then

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

I would now like to show that $\mathcal{E}_{\text{MP}} \oplus\!\!> \mathcal{E}_{\text{MPExt}}$ together with the equivalence $\equiv_{\mathcal{A}_p}$ provide a refining extension of $\mathcal{E}_{\text{MP}}$. Consider the theorem statement

> Let $\mathcal{E}_i = (\Sigma_i, R_i)$ for $i = 0, 1$ be two eTSSs in extended *tyft/tyxt* format such that $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ is defined. Moreover, let $\equiv_0$ and $\equiv_1$ be congruences over $\mathbf{T}(\Sigma_0)_{\mathsf{P}}$ and $\mathbf{T}(\Sigma_1)_{\mathsf{P}}$ respectively such that $\equiv_0 \oplus \equiv_1$ is conservative with respect to $\equiv_0$. If $\mathcal{E}_0$ is pure and label-pure, and $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ is type-1, then $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ is a refining extension.

However this is problematic since $\equiv_{\mathcal{A}_{\text{n}}} \oplus \equiv_{\mathcal{A}_{\text{p}}}$ is not conservative with respect to $\equiv_{\mathcal{A}_{\text{n}}}$. Conservativity requires that for $\alpha, \beta \in \mathbf{T}(\Sigma_{\text{MP}})_{\overline{\mathsf{P}}}$,

$$\alpha \equiv_{\mathcal{A}_{\text{n}}} \oplus \equiv_{\mathcal{A}_{\text{p}}} \beta \iff \alpha \equiv_{\mathcal{A}_{\text{p}}} \beta.$$

However $\mathsf{act}(\mathsf{a}, \mathsf{l}_1) \equiv_{\mathcal{A}_{\text{n}}} \oplus \equiv_{\mathcal{A}_{\text{p}}} \mathsf{act}(\mathsf{a}, \mathsf{l}_2)$ but $\mathsf{act}(\mathsf{a}, \mathsf{l}_1) \not\equiv_{\mathcal{A}_{\text{n}}} \mathsf{act}(\mathsf{a}, \mathsf{l}_2)$.

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

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

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

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

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

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

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

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

$$\left\{ \quad \frac{X \xrightarrow{z_{\mathsf{Act}_{\mathsf{new}}}} y}{\mathsf{Cn} \xrightarrow{z_{\mathsf{Act}_{\mathsf{new}}}} y} \quad \middle| \quad \mathsf{Cn} \overset{\mathrm{def}}{=} X, \mathsf{Cn} \in \mathsf{Const} \quad \right\}$$

Table 6.16: Rules and rule schemas for MPExt

Sort	**Carrier set**
A	$A \cup \{\bot_{\mathsf{A}}\}$
L	$\mathbb{N}$
Act	$\mathcal{C}' \cup \{\bot_{\mathsf{Act}}\}$
$\mathsf{Act}_{\mathsf{new}}$	$\mathcal{C} \cup \{\bot_{\mathsf{Act}_{\mathsf{new}}}\}$

Table 6.17: Carrier sets for $\mathcal{A}_{pom}$

$$
\begin{array}{rcl}
\mathsf{a}^{\mathcal{A}_{pom}} & = & a \quad \forall \mathsf{a} \in \mathsf{A} \\
\mathsf{l}^{\mathcal{A}_{pom}} & = & num(l) \quad \forall \mathsf{l} \in \mathsf{L} \\
\mathsf{act}^{\mathcal{A}_{pom}}(\zeta_1, \zeta_2) & = & \begin{cases} \zeta_1 & \text{if } \zeta_1 \in A \text{ and } \zeta_2 \in \mathbb{N}^+ \\ \bot_{\mathsf{Act}} & \text{otherwise} \end{cases} \\
\mathsf{comb}^{\mathcal{A}_{pom}}(\zeta_1, \zeta_2) & = & \begin{cases} \zeta_1 \mid \zeta_2 & \text{if } \zeta_1, \zeta_2 \in \mathcal{C}' \\ \bot_{\mathsf{Act}} & \text{otherwise} \end{cases} \\
\mathsf{concat}^{\mathcal{A}_{pom}}(\zeta_1, \zeta_2) & = & \begin{cases} \zeta_1 : \zeta_2 & \text{if } \zeta_1 \in A,\ \zeta_2 \in \mathcal{C} \\ \bot_{\mathsf{Act}_{\mathsf{new}}} & \text{otherwise} \end{cases} \\
\mathsf{comb}_{\mathsf{new}}{}^{\mathcal{A}_{pom}}(\zeta_1, \zeta_2) & = & \begin{cases} \zeta_1 \mid \zeta_2 & \text{if } \zeta_1, \zeta_2 \in \mathcal{C} \\ \bot_{\mathsf{Act}_{\mathsf{new}}} & \text{otherwise} \end{cases} \\
\bot_{\mathsf{A}}^{\mathcal{A}_{pom}} & = & \bot_{\mathsf{A}} \\
\bot_{\mathsf{L}}^{\mathcal{A}_{pom}} & = & 0 \\
\bot_{\mathsf{Act}}^{\mathcal{A}_{pom}} & = & \bot_{\mathsf{Act}} \\
\bot_{\mathsf{Act}_{\mathsf{new}}}^{\mathcal{A}_{pom}} & = & \bot_{\mathsf{Act}_{\mathsf{new}}}
\end{array}
$$

Table 6.18: Functions for $\mathcal{A}_{pom}$

Sort	Carrier set
A	$A \cup \{\bot_{\mathsf{A}}\}$
L	$\mathbb{N}$
Act	$\mathcal{D} = \bigcup_{j \geqslant 1}^{n} \{(c', a_1 \dots a_j) \mid c' \in \mathcal{C}', \mathrm{acts}(c') = j,$ $a_1, \dots, a_j \in \{1, \dots, j\}, \text{pairwise disjoint}\} \cup \{\bot_{\mathsf{Act}}\}$
$\mathsf{Act}_{\mathsf{new}}$	$\mathcal{C} \cup \{\bot_{\mathsf{Act}_{\mathsf{new}}}\}$

Table 6.19: Carrier sets for $\mathcal{A}_{pomn}$

The solution is to work with a different algebra over the labels and show that this algebra generates an equivalence which will result in the same semantic equivalence as is obtained by using $\equiv_{\mathcal{A}_p}$. Notice that the new operators introduced work with a different label set. Clearly the different algebra can be the same on that sort, however on the sort Act, it should be the same as $\mathcal{A}_n$.

I now want a slightly different algebra to represent the labels of a different process algebra based around pomset bisimulation but with some restriction on which processes can contribute to parallel computation for transitions with sort Act. I will call this Σ_{MPExt}-algebra $\mathcal{A}_{pomn}$.

Let N and M be sequences of positive natural numbers without repetition, and let $N \cap M$ indicate the intersection of sequences. Then let $(a, N) +^n (b, M)$ where $a, b \in \mathcal{C}'$, be defined as equal to $(a \mid b, NM)$ whenever $N \cap M = \emptyset$. Also define a function acts : $\mathcal{C}' \to \mathbb{N}$ as

$$\begin{aligned} \mathrm{acts}(c \mid c') &= \mathrm{acts}(c) + \mathrm{acts}(c') \\ \mathrm{acts}(a) &= 1. \end{aligned}$$

Extend $\equiv_{\mathcal{C}}$ to $\mathcal{D}$ in the obvious manner. The carrier sets and functions for $\mathcal{A}_{pomn}$ are given in Tables 6.19 and 6.20.

$$
\begin{aligned}
\mathsf{a}^{\mathcal{A}_{pomn}} &= a \quad \forall \mathsf{a} \in \mathsf{A}\\
\mathsf{l}^{\mathcal{A}_{pomn}} &= num(l) \quad \forall \mathsf{l} \in \mathsf{L}\\
\mathsf{act}^{\mathcal{A}_{pomn}}(\zeta_1,\zeta_2) &= \begin{cases} (\zeta_1,\zeta_2) & \text{if } \zeta_1 \in A \text{ and } \zeta_2 \leqslant n\\ \bot_{\mathsf{Act}} & \text{otherwise} \end{cases}\\
\mathsf{comb}^{\mathcal{A}_{pomn}}(\zeta_1,\zeta_2) &= \begin{cases} \zeta_1 +^n \zeta_2 & \text{if } \zeta_1,\zeta_2 \in \mathcal{D}\\ & \text{and } \zeta_1 +^n \zeta_2 \text{ defined}\\ \bot_{\mathsf{Act}} & \text{otherwise} \end{cases}\\
\mathsf{concat}^{\mathcal{A}_{pomn}}(\zeta_1,\zeta_2) &= \begin{cases} \zeta_1 : \zeta_{2,1} & \text{if } \zeta_1 \in A,\ \zeta_2 \in \mathcal{D} \text{ and}\\ & \zeta_2 = (\zeta_{2,1},\zeta_{2,2})\\ \zeta_1 : \zeta_2 & \text{if } \zeta_1 \in A,\ \zeta_2 \in \mathcal{C}\\ \bot_{\mathsf{Act_{new}}} & \text{otherwise} \end{cases}\\
\mathsf{comb_{new}}^{\mathcal{A}_{pomn}}(\zeta_1,\zeta_2) &= \begin{cases} \zeta_{1,1} \mid \zeta_{2,1} & \text{if } \zeta_1 \in \mathcal{D},\\ & \zeta_2 \in \mathcal{D} \text{ and}\\ & \zeta_1 = (\zeta_{1,1},\zeta_{1,2})\\ & \zeta_2 = (\zeta_{2,1},\zeta_{2,2})\\ \zeta_{1,1} \mid \zeta_2 & \text{if } \zeta_1 \in \mathcal{D},\ \zeta_2 \in \mathcal{C} \text{ and}\\ & \zeta_1 = (\zeta_{1,1},\zeta_{1,2})\\ \zeta_1 \mid \zeta_{2,1} & \text{if } \zeta_1 \in \mathcal{C},\ \zeta_2 \in \mathcal{D} \text{ and}\\ & \zeta_2 = (\zeta_{2,1},\zeta_{2,2})\\ \zeta_1 \mid \zeta_2 & \text{if } \zeta_1,\zeta_2 \in \mathcal{C}\\ \bot_{\mathsf{Act_{new}}} & \text{otherwise} \end{cases}\\
\bot_{\mathsf{A}}^{\mathcal{A}_{pomn}} &= \bot_{\mathsf{A}}\\
\bot_{\mathsf{L}}^{\mathcal{A}_{pomn}} &= 0\\
\bot_{\mathsf{Act}}^{\mathcal{A}_{pomn}} &= \bot_{\mathsf{Act}}\\
\bot_{\mathsf{Act_{new}}}^{\mathcal{A}_{pomn}} &= \bot_{\mathsf{Act_{new}}}
\end{aligned}
$$

Table 6.20: Functions for $\mathcal{A}_{pomn}$

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

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

I will first consider why this gives a refining extension to $\mathcal{E}_{\mathrm{MP}}$, then I will show that this defines an equivalence $\sim_{\equiv_{\mathcal{A}_{pn}}}$ similar to $\sim_{\equiv_{\mathcal{A}_p}}$.

6.4.3 A refining extension

First note that since $\mathcal{E}_{\mathrm{MP}}$ is not label-pure (it is the axiom which is not label-pure), it is necessary to use Theorem 5.4.2

> Let $\Sigma_i = (S_i \cup \{\mathsf{P}\}, F_i)$ for $i = 0, 1$ be two signatures with Σ_1 safe for S_0. Let $\mathcal{E}_i = (\Sigma_i, R_i)$ for $i = 0, 1$ be two eTSSs in extended *tyft/tyxt* format such that $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ is defined. Moreover, let $\equiv_0$ and $\equiv_1$ be congruences over $\mathbf{T}(\Sigma_0)_{\mathsf{P}}$ and $\mathbf{T}(\Sigma_1)_{\mathsf{P}}$ respectively such that $\equiv_0 \oplus \equiv_1$ is conservative with respect to $\equiv_0$. If $\mathcal{E}_0$ is pure, and $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ is type-1, then $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ is a refining extension.

Clearly Σ_{MPExt} is safe for S_{MP} since no function in $F_{\mathrm{MPExt}} - F_{\mathrm{MP}}$ (the functions concat and $\mathsf{comb}_{\mathsf{new}}$) have 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 $\mathsf{act}_{\mathsf{new}}$.

Finally I need to show that $\equiv_{\mathcal{A}_n} \oplus \equiv_{\mathcal{A}_{pn}}$ is conservative with respect to $\equiv_{\mathcal{A}_n}$. First note, that for all $s \in \{\mathsf{L}, \mathsf{A}\}$, $\alpha, \beta \in \mathbf{T}(\Sigma_{\mathrm{MPExt}})_s$

$$\alpha \equiv_{\mathcal{A}_n} \oplus \equiv_{\mathcal{A}_{pn}} \beta \Longleftrightarrow \alpha \equiv_{\mathcal{A}_n} \beta$$

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 is only necessary to consider terms with sort Act. See Appendix B for conservativity on Act. Note that it is not possible to use the results from Section 6.2.2 since the sum of the two algebras is not defined. This is the case because the algebras do not have the same carrier sets for sorts that appear in both signatures.

Hence by applying the above result, I can conclude that for $t, u \in \mathbf{T}(\Sigma_{\mathrm{MP}})_{\mathsf{P}}$

$$t \sim^{\mathcal{E}_{\mathrm{MP}} \oplus\!\!> \mathcal{E}_{\mathrm{MPExt}}}_{\equiv_{\mathcal{A}_n} \oplus \equiv_{\mathcal{A}_{pn}}} u \Longrightarrow t \sim^{\mathcal{E}_{\mathrm{MP}}}_{\equiv_{\mathcal{A}_n}} u$$

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 6.4.1 (Restricted pomset equivalence and multiprocessor equivalence)
For arbitrary n, with $+$ and $\sum$ for plus, and $|$ and $\prod$ for par,

$$\prod_{k=1}^{n} \mathsf{pref}(\mathsf{a}_k, \mathsf{nil}) + \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} \Big(\mathsf{pref}(\mathsf{a}_i, \mathsf{pref}(\mathsf{a}_j, \mathsf{nil})) \mid \prod_{\substack{k=1 \\ k\neq i \\ k \neq j}}^{n} \mathsf{pref}(\mathsf{a}_k, \mathsf{nil})\Big)$$

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

However these processes are not equated by the other equivalence since the first process can perform an 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.

Consider this counter-example in light of Theorem 5.4.3. For the theorem to be applicable, compatibility is required, $\mathcal{E}_{\mathrm{MPExt}}$ is required to be well-founded, and $\mathcal{E}_{\mathrm{MP}} \oplus\!\!> \mathcal{E}_{\mathrm{MPExt}}$ must be type-0. Clearly, compatibility is assured since all terms of sort $\overline{\mathsf{P}}$ that occur on the transitions of premises and in the source of the conclusion are variables. $\mathcal{E}_{\mathrm{MPExt}}$ is clearly well-founded. Hence, the reason why the theorem is not applicable is because 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.

I now need 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. The details of this are given in Appendix B. Hence, I can conclude that for $t, u \in \mathbf{T}(\Sigma_{\mathrm{MP}})_{\mathsf{P}}$

$$t \sim^{\mathcal{E}_{\mathrm{MP}} \oplus\!\!> \mathcal{E}_{\mathrm{MPExt}}}_{\equiv_{\mathcal{A}_n} \oplus \equiv_{\mathcal{A}_p}} u \Longrightarrow t \sim^{\mathcal{E}_{\mathrm{MP}}}_{\equiv_{\mathcal{A}_n}} u$$

In other words, on the process terms of $\mathbf{T}(\Sigma_{\mathrm{MP}})$, pomset equivalence equivalence is a proper subset of n multiprocessor equivalence.

6.5 Discussion

As has been shown in this chapter it is possible to express a number of process algebras with structured labels in the extended *tyft/tyxt* format, as well as use the results from Chapter 5 to compare equivalences. It has not been possible to use the abstracting extension result as it is very strong, and an issue for further work is to look for a weaker result. It has also been shown that the conditions required to achieve the results of Chapter 5 are reasonable.

In terms of the procedure for applying the results, I will describe which areas require work and which come out easily. There are number of decisions to be taken when choosing how to represent the process algebra labels and it may be the case that a few different approaches need to be tried. Usually the carrier sets can be chosen to be those used in the process algebra, but often there are choices about when to introduce terms to represent undefined transitions. Obviously, the rules must be in extended *tyft/tyxt* format, and it can take some time to understand how this can be achieved for a particular process algebra. If one dealing with sums, as in extension results, the results in Section 6.2.2 give conservativity and compatibility if sort-similarity holds and if the two algebras can form a sum; otherwise these need to be shown directly. Compatibility is often easy since most label terms on transitions of premises and in the source of the conclusion consist of a single variable. Sometimes, as in the example in this chapter, it is necessary to show that two algebras give the same equivalence. An area for further work is to find a simpler way to do this, particularly in the case where there are rules which only vary in the sort of the labels on the transitions as in the example here.

6.6 Conclusion

In this chapters, I have looked at some aspects of application of the results from earlier chapters, including the implications of some of the conditions required to obtain the results, and applications of the new format and related results to process algebras.

Chapter 7

Conclusion and further work

In this chapter, I will give a summary of work presented in this thesis, and the conclusions that can be drawn from this research. Finally, I will discuss some items for further work.

7.1 Summary and conclusions

The focus of this thesis has been to look at ways to compare different equivalences for process algebras. This work is motivated by the large number of process algebras that are present in the literature, since to deepen understanding of the different process algebras it is necessary to compare and contrast them and their equivalences.

The second chapter of the thesis gave an overview of different process algebras which have been developed to demonstrate and distinguish different aspects of concurrent behaviour.

In the third chapter, I looked at two approaches to this comparison—one was based on comparing equivalences on CCS terms, and the other looked at the underlying process domains, namely the labelled transition systems, and investigated if it was possible to do the comparison at that level. Neither of these approaches was satisfactory. The first was too limited in that it relied on pure CCS terms and moreover, it required that the terms demonstrated the concurrent behaviour that the equivalence distinguished. The second approach extends existing work on bisimulation homomorphisms by considering label transitions systems with different label sets. As an approach to comparing equivalences, it was not satisfactory since it was unclear how to use the results of the theorem in comparison of equivalences. Issues not resolved include which states to compare, how to find a suitable mapping between label sets and how to compare equivalences over different labelled transitions systems with the same label set.

The rest of the thesis is devoted to looking at a new format and investigating how it can be used to compare equivalences. The approach taken was to extend the syntactic notions used in previous formats for process terms to label terms, and to use an equivalence over these terms to represent the semantics of the labels of the actual process algebras being expressed in the new format. I first showed that some standard results such as congruence and conservative extension hold. I then provided new definitions for extensions up to bisimulation and gave conditions under which they hold. The major results are two theorems relating to refining extensions up to bisimulation with respect to an equivalence and two theorems resulting to abstracting extensions up to bisimulation with respect to an equivalence

The material in Chapters 4 and 5 is presented in an abstract fashion, assuming an equivalence over the label terms. In Chapter 6, I look at how these results can be applied to actual process algebras, and investigate what impact the conditions required for the theorems of the previous chapters have on the process algebras that can be specified in the format. I show that a number of process algebras can be expressed in the new format—CCS, CCS with locations and multiprocessor CCS. Finally I use the results to show that pomset equivalence equivalence is a proper subset of n multiprocessor equivalence.

In conclusion, this research has successfully shown that by introducing a new format which caters for structured labels, equivalence comparison results can be obtained and applied to process algebras from the literature.

7.2 Further work

There are a number of issues for further work, a number of which lead automatically from the fact that I have developed a new format.

- The new format is only concerned with positive premises. An area of further research is negative premises and predicates.

- I have only looked at strong bisimulation. Weak bisimulation has been considered by Bloom [Blo95]. He categorises operators by the form of their rules and uses this to develop an understanding of congruence with respect to weak bisimulation. It may be possible to take a similar approach with the new format.

Another area for further work are some issues that have not been resolved in this thesis.

- An issue to consider is whether well-foundedness is required for the new format. This would involve taking the notion of proof and unification from the work of Fokkink and van Glabbeek [FvG96] and extending it to the new format.

- As yet, I do not know whether the label variables in the sources of premises need to be distinct from those in the terms on the transitions. As seen in Section 4.3.3, when proving congruence in a manner which requires well-foundedness, it is clear that that the distinctness is required in the case where variables are shared across premises. However if the well-foundedness condition can be dropped, it may be possible to drop this condition as well. I also do not know whether a variable can be shared within a source and label of a premise or whether this conflicts with the use of compatibility.

- For the refining extension theorem, it is not clear that all the conditions of pureness and label-pureness are required. An area of further work is to attempt to prove the theorem without the use of the lemma.

Other areas of further work are as follows.

- I wish to see if the new format can be applied to other process algebras outside of those considered in this thesis.

- In this research, I have assumed that processes can only have one specific sort. However, a number of process algebras do not have 'single level' syntax, namely the processes may have more than one sort. An example of this is the way processes are constructed for Kiehn's local/global cause process algebra [KH94] where the only terms that can appear within the scope of a prefix operator are pure CCS terms. An area of further work is to extend the format to take this into account. Operators with label sorts would still require the condition that no arguments can have a process sort.

Finally, there are some more results of interest that would be useful for comparing equivalences.

- As mentioned in Section 6.5, the abstracting extension result is very strong in that no transitions can be added from existing process—for an example of this, see Counter-Example 5.3.7. I wish to find a more applicable result. An approach to take here is obtain conditions that apply to all rules, and ensure that any transitions that are added are also added to equivalent terms. This would involve an understanding of which transitions are possible from the first eTSS. It may be the case that a general result cannot be found.

- In certain situations, such as expressing location CCS as an extension of CCS, it would be desirable to filter out terms of a certain sort. In the example, the τ transitions would be retained, but the non-τ transitions would be discarded. It may be possible to do this in a manner where it can be shown that the added transitions have a similar enough structure to the removed transitions to be able to derive a relationship between the equivalences. It may be possible to use bisimulation homomorphisms in gaining such a result.

In some labelled transition systems there are transitions which do not add any further information to the equivalence and hence it is possible to remove them. If it is possible to do this for a subset of terms that appear on these transitions, it may be possible to obtain a result about the equivalence of the labelled transition systems and hence about the equivalences.

The proof in Section B.3 involves comparing two equivalences for equality, and takes some work. It would be useful to have a general result that permits this to be done more easily. Such a result could be based on the fact that the same rules are present for different sorts. Hence the labelled transition system has similar structure for different sorts, and it may be possible to use this information as well as information about the label equivalences to show that the equivalences are the same.

- Another question of interest look at the tags used in process algebras. By tags, I mean operators and constants (such as location prefixing) that are added to a process when an action occurs, and which appear on later transitions to shown which actions the current action is related to. This is the idea behind both local cause bisimulation and global cause bisimulation. As mentioned in Section 6.3.3, it may be possible to obtain a general results about the number of tags required to retain bisimulation. A number of issues to consider are arity of operators, characteristics of the label equivalences, pureness and label-pureness, and type of operator (static versus dynamic). Arity appears to be a key issue. The label-pureness issue relates to how new tags can be introduced.

Appendix A

Proof of Theorem 5.3.2

A.1 Proof of Theorem 5.3.2

Theorem A.1.1 (Abstracting extension up to bisimulation with respect to a congruence)
Let $\mathcal{E}_i = (\Sigma_i, R_i)$ for $i = 0, 1$ be two eTSSs in extended *tyft/tyxt* format such that $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ is defined. Moreover, let $\equiv_i$ be congruences over $\mathbf{T}(\Sigma_i)_{\overline{\mathsf{P}}}$ for $i = 0, 1$ such that $\equiv_0 \oplus \equiv_1$ is compatible with $\mathcal{E}_0 \oplus\!\!> E_1$. If $\mathcal{E}_0$ is pure and label-pure, $\mathcal{E}_1$ is well-founded, and $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ is type-0, then $\mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ is an abstracting extension.

Proof: Let $t_0, u_0 \in \mathbf{T}(\Sigma_0)$ and let $\equiv\, = \,\equiv_0 \oplus \equiv_1$. I wish to show that $\mathcal{E}$ is an abstracting extension up to bisimulation with respect to $\equiv$ so I need to show that $t_0 \sim^{\mathcal{E}_0}_{\equiv_0} u_0 \Rightarrow t_0 \sim^{\mathcal{E}}_{\equiv} u_0$.

I will exhibit a bisimulation with respect to $\equiv$ under $\mathcal{E}$ containing $\sim^{\mathcal{E}_0}_{\equiv_0}$. Let $\mathcal{R} \subseteq \mathbf{T}(\Sigma_0)_{\mathsf{P}} \times \mathbf{T}(\Sigma_0)_{\mathsf{P}}$ be the least relation satisfying

- $\sim^{\mathcal{E}}_{\equiv} \subseteq \mathcal{R}$,
- $\sim^{\mathcal{E}_0}_{\equiv_0} \subseteq \mathcal{R}$,
- for all $f \in F$ such that $f : s_1 \ldots s_m \mathsf{P} \ldots \mathsf{P} \to \mathsf{P}$, for all terms $\mu_k, \nu_k \in \mathbf{T}(\Sigma)_{\overline{\mathsf{P}}}$ $(1 \leqslant k \leqslant m)$, and for all terms $u_i, v_i \in \mathbf{T}(\Sigma)_{\mathsf{P}}$ $(1 \leqslant i \leqslant n)$,

$$\begin{aligned}&\mu_k \equiv \nu_k \ \ (1 \leqslant k \leqslant m) \ \text{ and } \ u_i \,\mathcal{R}\, v_i \ \ (1 \leqslant i \leqslant n) \Rightarrow \\ &f(\mu_1, \ldots, \mu_m, u_1, \ldots, u_n) \,\mathcal{R}\, f(\nu_1, \ldots, \nu_m, v_1, \ldots v_n).\end{aligned}$$

It is enough to show $\mathcal{R} \subseteq \sim^{\mathcal{E}}_{\equiv}$ since $\sim^{\mathcal{E}}_{\equiv} \subseteq \mathcal{R}$, hence I need to show that $\mathcal{R}$ is a bisimulation with respect to $\equiv$ under $\mathcal{E}$. Assume $u \,\mathcal{R}\, v$. I have three cases—the first is simple since $u \sim^{\mathcal{E}}_{\equiv} v$. The second requires us to show 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 $u' \mathcal{R} v'$.

The third case requires us to show 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 $1 \leqslant k \leqslant m$ and $u_i \mathcal{R} v_i$ for $1 \leqslant i \leqslant n$ then there is a $v' \in \mathbf{T}(\Sigma)_{\mathsf{P}}$ such that $\mathcal{E} \vdash f(\nu_1, \dots, \nu_m, v_1, \dots, v_n)) \xrightarrow{\alpha'} v'$ with $\alpha' \equiv \alpha$ and $u' \mathcal{R} v'$.

I need to consider these two cases together as the proof will proceed by induction on the length of the proof required to prove a transition.

In the second case, I have $u \sim^{\mathcal{E}_0}_{\equiv_0} v$ with $u, v \in \mathbf{T}(\Sigma_0)$. For a given $u \in \mathbf{T}(\Sigma_0)$, I can consider the transitions from it. Either they are the result of extended *tyft/tyxt* rules from R_0 or they are the result of extended *tyxt* rules from R_1, since extended *tyft* rules from R_1 cannot have functions from F_0 in the source of the conclusion.

In the case of extended *tyft/tyxt* rules from R_0, I can apply Lemma 5.3.1 since $\mathcal{E} = \mathcal{E}_0 \oplus\!\!> \mathcal{E}_1$ is type-0 and hence type-1. Therefore, for any transition $u \xrightarrow{\alpha} u'$ then $\alpha \in \mathbf{T}(\Sigma_0)_{\overline{\mathsf{P}}}$ and $u' \in \mathbf{T}(\Sigma_0)_{\mathsf{P}}$, since $u \in \mathbf{T}(\Sigma_0)_{\mathsf{P}}$. Moreover, for any transition of the form $u \xrightarrow{\alpha} u'$ with $\alpha \in \mathbf{T}(\Sigma_0)_{\overline{\mathsf{P}}}$ and $u' \in \mathbf{T}(\Sigma_0)_{\mathsf{P}}$, I can use the fact that $u \sim^{\mathcal{E}_0}_{\equiv_0} v$ to find $\alpha' \in \mathbf{T}(\Sigma_0)_{\overline{\mathsf{P}}}$ 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, I need to do induction on the proof of the transition. I need also to consider the third case in the induction, since I will need to consider the relationship between pairs of processes with the syntactic form of the third case. I will need the following 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 $\mathrm{Var}_{\overline{\mathsf{P}}}(p)$, $\rho(z) \equiv \rho'(z)$. Then $\rho(p) \mathcal{R} \rho'(p)$.

Proof: I proceed by induction on the structure of p. If $p = x$ then I have the result immediately. If $p = f(\eta_1, \dots, \eta_m, p_1, \dots, p_n)$, then I know by the induction hypothesis that $\rho(p_i)\,\mathcal{R}\,\rho'(p_i)$ for $1 \leqslant i \leqslant n$, and also that $\rho(\eta_k) \equiv \rho'(\eta_k)$ for $1 \leqslant k \leqslant m$ (since $\equiv$ is an equivalence), hence $\rho(t)\,\mathcal{R}\,\rho'(t)$ by the definition of $\mathcal{R}$.

■

From Lemma 4.3.1, I know that there is a proof T of $u \xrightarrow{\alpha} u'$ containing only closed transitions. Let r be the last rule used in proof T, in combination with a substitution σ. Assume first that the proof consists of only one step, then I have 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 $z \in \mathrm{Var}_{\overline{\mathsf{P}}}(r)$, 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.

- I have $f(\mu_1, \ldots, \mu_m, u_1, \ldots, u_n)$ and $f(\nu_1, \ldots, \nu_m, v_1, \ldots v_n)$ with $\mu_i \equiv \nu_i$ for all $1 \leqslant k \leqslant m$ and $u_i\,\mathcal{R}\,v_i$ for all $1 \leqslant i \leqslant n$. I have two cases:

  - $r \in R_0$ has the form $x \xrightarrow{\lambda} p$. (Since the terms are drawn from $\mathbf{T}(\Sigma_0)$ it is not possible that an extended *tyft* rule from R_1 could be used.) 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 $1 \leqslant k \leqslant m$, $\sigma(x_i) = u_i$ for $1 \leqslant i \leqslant n$, $\sigma(p) = u'$ and $\sigma(\lambda) = \alpha$. Let $\sigma'(x_i) = v_i$ for $1 \leqslant i \leqslant n$, $\sigma'(z) = \sigma(z)$ for all $z \in \mathrm{Var}_{\overline{\mathsf{P}}}(r) - \bigcup_{1 \leqslant k \leqslant m} \eta_k$, and $\sigma'(y) = \sigma(y)$ for all $y \in \mathrm{Var}_{\mathsf{P}}(r) - \bigcup_{1 \leqslant i \leqslant n} x_i$.

    I need to take more care with η_k ($1 \leqslant k \leqslant m$). Since for a given k such that $1 \leqslant k \leqslant m$, $\sigma(\eta_k) = \mu_k \equiv \nu_k$ and $\equiv$ is compatible with $R_0 \cup R_1$, I know that there is a substitution σ'' such that $\sigma''(\eta_k) = \nu_k$ and for all $z \in \mathrm{Var}_{\overline{\mathsf{P}}}(\eta_k)$, $\sigma(z) \equiv \sigma''(z)$. Hence define $\sigma'(z) = \sigma''(z)$. I can do this 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 $\sigma(p)\,\mathcal{R}\,\sigma'(p)$ as required.

I next assume that the two statements $(*)$ and $(**)$ are true for proofs with n or fewer steps. I will now show that the statements are true for proofs with $n+1$ steps. I will only give the details of the proof for the case of an extended *tyft* rule from R_0 for $(**)$. The cases of 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—they are less complex to prove because they both deal with extended *tyxt* rules and hence there is only a single variable in the source of the conclusion. This makes the assignment of terms to label variables simpler. Note that when dealing with rules from $R_0 \cup R_1$ I treat them as arbitrary non-pure, non-label-pure rules, hence I do not use the fact that rules from R_0 are pure and label-pure in this part of the proof.

Let r be the last rule used in the proof of $u \xrightarrow{\alpha} u'$ in combination with a substitution σ. Assume r is equal to

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

Then I know that $\sigma(\eta_k) = \mu_k$ for $1 \leqslant k \leqslant m$, $\sigma(x_i) = u_i$ for $1 \leqslant i \leqslant n$, $\sigma(p) = u'$ and $\sigma(\lambda) = \alpha$. I want to find a substitution σ' that I can use to show that $v \xrightarrow{\alpha} v'$ with $u' \mathcal{R} v'$.

I proceed with an analysis of the variables that occur in r, and then I use an induction technique to define σ'. By considering the dependency graph G of the premises of r, depth$(x) \in \mathbb{N}$ can be defined for all $x \in$Var$_{\mathsf{P}}(r)$ in a similar fashion to the proof of Theorem 4.3.1. Define

- $X = \{x_i \mid 1 \leqslant i \leqslant n\}$
- $Y = \{y_i \mid i \in I\}$
- $Y_d = \{y \in Y \mid \text{depth}(y) = d\}$ for $n \geqslant 0$.
- $Y_f =$Var$_{\mathsf{P}}(r) - (X \cup Y)$.

Observe that for any variable $x \in X$, depth$(x) = 0$, and the sets Y_d form a partition of Y. Next I consider variables from $V_{\overline{\mathsf{P}}}$. These can be partitioned in four sets.

- $Z = \bigcup_{1 \leqslant k \leqslant m} \mathrm{Var}_{\overline{\mathsf{P}}}(\eta_k)$
- $Z' = \bigcup_{i \in I} \mathrm{Var}_{\overline{\mathsf{P}}}(\lambda_i)$
- $Z'' = \bigcup_{i \in I} \mathrm{Var}_{\overline{\mathsf{P}}}(p_i) - Z$
- $Z_f = (\mathrm{Var}_{\overline{\mathsf{P}}}(\lambda) \cup \mathrm{Var}_{\overline{\mathsf{P}}}(p)) - (Z \cup Z' \cup Z'')$.

I can partition Z' into $Z'_0, Z'_1, \ldots$ by defining $Z'_d = \bigcup_{y_i \in Y_d} \mathrm{Var}_{\overline{\mathsf{P}}}(\lambda_i)$ for each $d \geqslant 0$. This is possible because the variables that appear in the premise labels are disjoint.

I will define a substitution σ' that satisfies the following properties on V_{P} and $V_{\overline{\mathsf{P}}}$

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 $1 \leqslant i \leqslant n$
- $\sigma'(y) = \sigma(y)$ for $y \in V_{\mathsf{P}} - (X \cup \bigcup_{d \geqslant 0} Y_d)$
- $\sigma'(z) = \sigma(z)$ for $z \in V_{\overline{\mathsf{P}}} - (Z \cup \bigcup_{d \geqslant 0} Z'_d \cup Z'')$.

Note therefore that for all variables $z \in V_{\overline{\mathsf{P}}} - (Z \cup \bigcup_{d \geqslant 0} Z'_d \cup Z'')$, $\sigma(z) \equiv \sigma'(z)$. I still have to define σ' on $\bigcup_{d \geqslant 0} Y_d$, $\bigcup_{d \geqslant 0} Z'_d$, Z and Z''.

When σ' is defined for $y \in X \cup Y_0 \cup \ldots \cup Y_d$ and $z \in Z \cup Z_f \cup Z'_0 \cup \ldots \cup Z'_d$ $(d \geqslant 0)$, I will show that $\beta(d)$, $\gamma(d)$ and $\delta(d)$ 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 $z \in Z \cup Z_f \cup Z'_0 \cup \ldots \cup Z'_d$.

If I can show that $\beta(d)$, $\gamma(d)$ and $\delta(d)$ hold for all $d \geqslant 0$, then I know that the second and third properties hold, and that the fourth property will hold for $Z \cup Z_f \cup Z'$. For $z \in Z''$, when dealing with a particular transition $p_i \xrightarrow{\lambda_i} y_i$, I will simply define $\sigma'(z) = \sigma(z)$ for any $z \in \mathrm{Var}_{\overline{\mathsf{P}}}(p_i)$ that has not yet been defined. Hence once I have shown $\beta(d)$, $\gamma(d)$ and $\delta(d)$ hold for all $d \geqslant 0$, I will have defined σ' for all $z \in Z''$ and moreover $\sigma(z) \equiv \sigma'(z)$ for all $z \in Z''$, so the fourth property will be satisfied. I know that property one holds by definition.

I first need to show that $\beta(0)$, $\delta(0)$ and $\gamma(0)$ hold. First note that for any $x \in X$ then $x = x_i$ for some $1 \leqslant i \leqslant n$ and since $\sigma(x_i) = u_i$ and $\sigma'(x_i) = v_i$, and $u_i \mathrel{\mathcal{R}} v_i$, I have $\sigma(x_i) \mathrel{\mathcal{R}} \sigma'(x_i)$. Also $\sigma(y) \mathrel{\mathcal{R}} \sigma'(y)$ for $y \in Y_f$ since $\sigma(y) = \sigma'(y)$.

Then consider Z. Since for a given k such that $1 \leqslant k \leqslant m$, $\sigma(\eta_k) = \mu_k \equiv \nu_k$ and $\equiv$ is compatible with $R_0 \cup R_1$, I know that there is a substitution σ'' such that $\sigma''(\eta_k) = \nu_k$ and for all $z \in \mathrm{Var}_{\overline{\mathsf{P}}}(\eta_k)$, $\sigma(z) \equiv \sigma''(z)$. So let $\sigma'(z) = \sigma''(z)$. I can do this for all $1 \leqslant k \leqslant m$ since there are no variables shared between the terms. Also $\sigma(z) \equiv \sigma'(z)$ for $z \in Z_f$ since $\sigma(z) = \sigma'(z)$.

Next consider $y^* \in Y_0$. There exists $i \in I$ such that $y^* = y_i$, so I can consider the transition $p_i \xrightarrow{\lambda_i} y_i$. If $\mathrm{Var}_{\overline{\mathsf{P}}}(p_i) \cap Z = \emptyset$, then the situation is straightforward. Let $\sigma'(z) = \sigma(z)$ for $z \in \mathrm{Var}_{\overline{\mathsf{P}}}(p_i) \cup \mathrm{Var}_{\overline{\mathsf{P}}}(\lambda_i)$ and let $\sigma'(y_i) = \sigma(y_i)$. Then $\sigma(y_i) \mathrel{\mathcal{R}} \sigma'(y_i)$, and for all $z \in \mathrm{Var}_{\overline{\mathsf{P}}}(p_i) \cup \mathrm{Var}_{\overline{\mathsf{P}}}(\lambda_i)$, $\sigma(z) \equiv \sigma'(z)$ since $\sigma'(z) = \sigma(z)$. Moreover, $\sigma'(p_i \xrightarrow{\lambda_i} y_i) = \sigma'(p_i) \xrightarrow{\sigma'(\lambda_i)} \sigma'(y_i) = \sigma(p_i) \xrightarrow{\sigma(\lambda_i)} \sigma(y_i) = \sigma(p_i \xrightarrow{\lambda_i} y_i)$ Therefore $\mathcal{E} \vdash \sigma'(p_i \xrightarrow{\lambda_i} y_i)$ since $\mathcal{E} \vdash \sigma(p_i \xrightarrow{\lambda_i} y_i)$.

However, if $\mathrm{Var}_{\overline{\mathsf{P}}}(p_i) \cap Z \neq \emptyset$ then I need to take more care. σ' is already defined on $z \in \mathrm{Var}_{\overline{\mathsf{P}}}(p_i) \cap Z$, and I can define $\sigma'(z) = \sigma(z)$ for $z \in \mathrm{Var}_{\overline{\mathsf{P}}}(p_i) - Z$ on which σ' is not defined. Hence I know that $\sigma(z) \equiv \sigma'(z)$ for $z \in \mathrm{Var}_{\overline{\mathsf{P}}}(p_i)$. Since $\mathrm{Var}_{\mathsf{P}}(p_i) = \emptyset$, I know from the fact above that $\sigma(p_i) \mathrel{\mathcal{R}} \sigma'(p_i)$. I have three cases to consider:

- $\sigma(p_i) \sim^{\mathcal{E}}_{\equiv} \sigma'(p_i)$. Since $\mathcal{E} \vdash \sigma(p_i) \xrightarrow{\sigma(\lambda_i)} \sigma(y_i)$, I can find $w \in T(\Sigma)_{\mathsf{P}}$ and $\alpha_i \in T(\Sigma)_{\overline{\mathsf{P}}}$ such that $\mathcal{E} \vdash \sigma'(p_i) \xrightarrow{\alpha_i} w$, $\sigma(\lambda_i) \equiv \alpha_i$ and $\sigma(y_i) \mathrel{\mathcal{R}} w$. So I can define $\sigma'(y^*) = \sigma'(y_i) = w$.

  Moreover, since $\equiv$ is compatible with R_0, I know there exists a substitution σ'' such that $\alpha_i = \sigma''(\lambda_i)$ and $\sigma(z) \equiv \sigma''(z)$ for all $z \in \mathrm{Var}_{\overline{\mathsf{P}}}(\lambda_i)$. Let

$\sigma'(z) = \sigma''(z)$ whence $\alpha_i = \sigma'(\lambda_i)$. (Since $\mathrm{Var}_{\overline{\mathsf{P}}}(p_i) \cap \mathrm{Var}_{\overline{\mathsf{P}}}(\lambda_i) = \emptyset$, it is clear that the use of σ'' does not affect values assigned to $\mathrm{Var}_{\overline{\mathsf{P}}}(p_i)$.)

- $\sigma(p_i) \sim^{\mathcal{E}_0}_{\equiv_0} \sigma'(p_i)$. I have two cases depending on whether the transition is generated by an extended *tyft/tyxt* rule from R_0 or an extended *tyxt* rule from R_1:

  – If $\sigma(\lambda_i) \in \mathbf{T}(\Sigma_0)_{\overline{\mathsf{P}}}$, then since $\mathcal{E}_0 \vdash \sigma(p_i) \xrightarrow{\sigma(\lambda_i)} \sigma(y_i)$, I can find $w \in \mathbf{T}(\Sigma_0)_{\mathsf{P}}$ and $\alpha_i \in T(\Sigma)_{\overline{\mathsf{P}}}$ such that $\mathcal{E}_0 \vdash \sigma(p_i) \xrightarrow{\alpha_i} w$ with $\alpha_i \equiv_0 \sigma(\lambda_i)$ and $\sigma(y_i)\, \mathcal{R}\, w$. So I can define $\sigma'(y^*) = \sigma'(y_i) = w$.

    Since $\equiv\, = \,\equiv_0 \oplus \equiv_1$, $\alpha_i \equiv \sigma(\lambda_i)$. Moreover, since $\equiv$ is compatible with $R_0 \cup R_1$, I know there exists a substitution σ'' such that $\alpha_i = \sigma''(\lambda_i)$ and $\sigma(z) \equiv \sigma''(z)$ for all $z \in \mathrm{Var}_{\overline{\mathsf{P}}}(\lambda_i)$. Again let $\sigma'(z) = \sigma''(z)$ whence $\alpha_i = \sigma'(\lambda_i)$.

  – If $\sigma(\lambda_i) \notin \mathbf{T}(\Sigma_0)_{\overline{\mathsf{P}}}$, then this means that this transition is generated by an extended *tyxt* rule from R_1 and hence I can use the induction hypothesis for proofs of this form. To see why this is the case, consider that the proof of $\sigma(p_i) \xrightarrow{\sigma(\lambda_i)} \sigma(y_i)$ is a subproof of the proof of $u \xrightarrow{\alpha} u'$. Therefore it must have fewer steps than the proof of $u \xrightarrow{\alpha} u'$. But I have assumed that $(*)$ and $(**)$ are true for all proofs of fewer steps, hence I can apply the induction hypothesis. Since $\mathcal{E} \vdash \sigma(p_i) \xrightarrow{\sigma(\lambda_i)} \sigma(y_i)$, I can find $w \in T(\Sigma)_{\mathsf{P}}$ such that $\mathcal{E} \vdash \sigma'(p_i) \xrightarrow{\alpha_i} w$ with $\sigma(\lambda_i) \equiv \alpha_i$ and $\sigma(y_i)\, \mathcal{R}\, w$. So I can define $\sigma'(y^*) = \sigma'(y_i) = w$.

    Moreover, since $\equiv$ is compatible with $R_0 \cup R_1$, I know there exists a substitution σ'' such that $\alpha_i = \sigma''(\lambda_i)$ and $\sigma(z) \equiv \sigma''(z)$ for all $z \in \mathrm{Var}_{\overline{\mathsf{P}}}(\lambda_i)$. Again let $\sigma'(z) = \sigma''(z)$ whence $\alpha_i = \sigma'(\lambda_i)$.

- there is a function symbol $h \in F$ such that $h : s'_1 \ldots s'_m \mathsf{P} \ldots \mathsf{P} \to \mathsf{P}$, and there are terms $\mu'_{k'}, \nu'_{k'} \in \mathbf{T}(\Sigma)_{\overline{\mathsf{P}}}$ for $1 \leqslant k' \leqslant m'$, $w_{i'}, w'_{i'} \in \mathbf{T}(\Sigma)_{\mathsf{P}}$ for $1 \leqslant i' \leqslant n'$ such that

$$\sigma(p_i) = h(\mu'_1, \ldots, \mu'_{m'}, w_1, \ldots, w_{n'}) \text{ and}$$

and

$$\sigma'(p_i) = h(\nu'_1, \ldots, \nu'_{m'}, w'_1, \ldots, w'_{n'})$$

with $\mu'_{k'} \equiv \nu'_{k'}$ $(1 \leqslant k' \leqslant m')$ and $w_{i'} \mathcal{R} w'_{i'}$ $(1 \leqslant i' \leqslant n')$. Now I can apply the induction hypothesis. To see why this is the case, consider that the proof of $h(\mu'_1, \ldots, \mu'_{m'}, w_1, \ldots, w_{n'}) \xrightarrow{\sigma(\lambda_i)} \sigma(y_i)$ is a subproof of the proof of $u \xrightarrow{\alpha} u'$. Therefore it must have fewer steps than the proof of $u \xrightarrow{\alpha} u'$ and so I can apply the induction hypothesis. Since $\mathcal{E} \vdash h(\mu'_1, \ldots, \mu'_{m'}, w_1, \ldots, w_{n'}) \xrightarrow{\sigma(\lambda_i)} \sigma(y_i)$, I can find 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 I can define $\sigma'(y_i) = w$.

Since $\equiv$ is compatible with R_0, I know there exists a substitution σ'' such that $\alpha_i = \sigma''(\lambda_i)$ and $\sigma(z) \equiv \sigma''(z)$. Let $\sigma'(z) = \sigma''(z)$ for all $z \in \mathrm{Var}_{\overline{\mathsf{P}}}(\lambda_i)$ whence $\alpha_i = \sigma'(\lambda_i)$.

Hence I know for $y^* = y_i$, that $\sigma(y_i) \, \mathcal{R} \, \sigma'(y_i)$, $\mathcal{E} \vdash \sigma'(p_i \xrightarrow{\lambda_i} y_i)$ and $\sigma(z) \equiv \sigma'(z)$ for all $z \in \mathrm{Var}_{\overline{\mathsf{P}}}(p_i) \cup \mathrm{Var}_{\overline{\mathsf{P}}}(\lambda_i)$. I can do this for all $y \in Y_0$ and thereby show that $\beta(0)$, $\gamma(0)$ and $\delta(0)$ hold.

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 $Z \cup Z_f \cup Z'_0 \cup \ldots \cup Z'_{d-1}$ such that $\beta(d-1)$, $\gamma(d-1)$ and $\delta(d-1)$ hold. I now define σ' on Y_d and Z'_d such that $\beta(d)$, $\gamma(d)$ and $\delta(d)$ hold. Consider $y^* \in Y_d$. Then there exists $i \in I$ such that $y^* = y_i$, and so I can consider the transition $p_i \xrightarrow{\lambda_i} y_i$. Since $y_i \in Y_d$, then $\mathrm{Var}_{\mathsf{P}}(p_i) \subseteq X \cup Y_f \cup Y_0 \cup \ldots \cup Y_{d-1}$ so $\sigma(y) \, \mathcal{R} \, \sigma'(y)$ for $y \in \mathrm{Var}_{\mathsf{P}}(p_i)$. For $z \in \mathrm{Var}_{\overline{\mathsf{P}}}(p_i)$ such that $\sigma'(z)$ is as yet undefined, let $\sigma'(z) = \sigma(z)$. Hence I know that by the fact above $\sigma'(p_i) \, \mathcal{R} \, \sigma(p_i)$. I have three cases as before and the treatment is identical. From this it is easy to see that $\beta(d)$, $\gamma(d)$ and $\delta(d)$ hold for all $d \geqslant 0$, and hence I know that the four properties hold.

So I know that for all $i \in I$, $\mathcal{E} \vdash \sigma'(p_i) \xrightarrow{\sigma'(\lambda_i)} \sigma'(y_i)$ where $\sigma'(z) \equiv \sigma(z)$ for all $z \in \bigcup_{i \in I} \mathrm{Var}_{\overline{\mathsf{P}}}(\lambda_i)$. Hence I can conclude that

$$\mathcal{E} \vdash \sigma'(f(\eta_1, \ldots, \eta_m, x_1, \ldots, x_n) \xrightarrow{\lambda} p)$$

namely

$$\mathcal{E} \vdash f(\nu_1, \ldots, \nu_m, v_1, \ldots, v_n) \xrightarrow{\sigma'(\lambda)} \sigma'(p).$$

To see that $\sigma'(\lambda) \equiv \alpha$, recall that $\alpha = \sigma(\lambda)$, and I know that for all $z \in \mathrm{Var}_{\overline{\mathsf{P}}}(\lambda)$, $\sigma(z) \equiv \sigma'(z)$ and since $\equiv$ is a congruence, I have the required result. To see that $u \,\mathcal{R}\, \sigma'(p)$, recall that $u = \sigma(p)$, and for all $x \in \mathrm{Var}_{\mathsf{P}}(p)$, $\sigma(x) \,\mathcal{R}\, \sigma'(x)$ and for all $z \in\!\mathrm{Var}_{\overline{\mathsf{P}}}(p)$, $\sigma(z) \equiv \sigma'(z)$, hence by the fact above, $\sigma(p) \,\mathcal{R}\, \sigma'(p)$. ■

Appendix B

Proofs for comparison example

B.1 Introduction

This appendix contains two results which are required for the comparison in Chapter 6.

B.2 Equivalence result required in Section 6.4.3

I wish to show that for $\alpha, \beta \in \mathbf{T}(\Sigma_{\mathrm{MP}})_{\mathsf{Act}}$

$$\alpha \equiv_{\mathcal{A}_n} \oplus \equiv_{\mathcal{A}_{pn}} \beta \Longleftrightarrow \alpha \equiv_{\mathcal{A}_n} \beta$$

Clearly $\alpha \equiv_{\mathcal{A}_n} \beta \Longrightarrow \alpha \equiv_{\mathcal{A}_n} \oplus \equiv_{\mathcal{A}_{pn}} \beta$. To prove the opposite implication, I will provide a homomorphism h from $\mathcal{A}_{pomn}$ to $\mathcal{A}_n$ for the sorts A, L and Act. I also need to show that $h \circ i_{\mathcal{A}_{pomn}} = i_{\mathcal{A}_n}$ since then

$$\begin{array}{rcl} \alpha & \equiv_{\mathcal{A}_{pomn}} & \beta \\ i_{\mathcal{A}_{pomn}}(\alpha) & = & i_{\mathcal{A}_{pomn}}(\beta) \\ h(i_{\mathcal{A}_{pomn}}(\alpha)) & = & h(i_{\mathcal{A}_{pomn}}(\beta)) \\ i_{\mathcal{A}_n}(\alpha) & = & i_{\mathcal{A}_n}(\beta) \\ \alpha & \equiv_{\mathcal{A}_n} & \beta \end{array}$$

and hence $\alpha \equiv_{\mathcal{A}_{pomn}} \beta \Longrightarrow \alpha \equiv_{\mathcal{A}_n} \beta$. I then need to show that $\alpha \equiv_{\mathcal{A}_{pn}} \beta \Longrightarrow \alpha \equiv_{\mathcal{A}_n} \beta$ and then I can conclude that $\alpha \equiv_{\mathcal{A}_n} \oplus \equiv_{\mathcal{A}_{pn}} \beta \Longrightarrow \alpha \equiv_{\mathcal{A}_n} \beta$, since $\equiv_{\mathcal{A}_{pn}}$ can equate no more than $\equiv_{\mathcal{A}_n}$.

Consider the following functions

$$\begin{array}{rcl}
h(a) & = & a \qquad \forall a \in A \\
h(i) & = & i \qquad \forall 0 \leqslant i \leqslant n \\
h(c' \mid c'', N'N'') & = & h(c', N') +_n h(c'', N'') \\
 & & \text{for } N' \cap N'' = \emptyset \text{ and} \\
 & & |N'| = \text{acts}(c'),\ |N''| = \text{acts}(c'') \\
h(a, \{i\}) & = & (\overbrace{\delta, \dots, \delta}^{i-1}, a, \overbrace{\delta, \dots, \delta}^{n-i}) \\
h(\bot_{\mathsf{A}}) & = & \bot_{\mathsf{A}} \\
h(\bot_{\mathsf{Act}}) & = & \bot_{\mathsf{Act}}
\end{array}$$

I need to show that h is a homomorphism, hence I need to show that

$$\begin{array}{rcl}
h(\mathsf{act}^{\mathcal{A}_{pomn}}(\zeta_1, \zeta_2)) & = & \mathsf{act}^{\mathcal{A}_n}(h(\zeta_1), h(\zeta_2)) \\
h(\mathsf{comb}^{\mathcal{A}_{pomn}}(\zeta_1, \zeta_2)) & = & \mathsf{comb}^{\mathcal{A}_n}(h(\zeta_1), h(\zeta_2)).
\end{array}$$

This is straightforward for most cases. I only consider the case of comb when $\zeta_1 = (c', N')$ and $\zeta_2 = (c'', N'')$.

$$\begin{array}{rcl}
h(\mathsf{comb}^{\mathcal{A}_{pomn}}((c', N'), (c'', N''))) & = & h((c', N') +^n (c'', N'')) \\
 & = & \begin{cases} \bot_{\mathsf{Act}} & \text{if } N' \cap N'' \neq \emptyset \\ h(c' \mid c'', N'N'') & \end{cases}
\end{array}$$

$$h(c' \mid c'', N'N'') = h(c', N') +_n h(c'', N'')$$

$$\mathsf{comb}^{\mathcal{A}_n}(h(c', N'), h(c'', N'')) = h(c', N') +_n h(c'', N'')$$

Note that if $N' \cap N'' \neq \emptyset$ then $h(c', N') +_n h(c'', N'')$ is not defined since each operand will have an action at the shared position and hence $+_n$ will not be defined for this case.

I now need to show that $h \circ i_{\mathcal{A}_{pomn}} = i_{\mathcal{A}_n}$. Again this straightforward for most cases and proceeds by induction. I give the proof for comb.

$$\begin{aligned}
h(i_{\mathcal{A}_{pomn}}(\mathsf{comb}(\alpha_1,\alpha_2))) &= h(\mathsf{comb}^{\mathcal{A}_{pomn}}(i_{\mathcal{A}_{pomn}}(\alpha_1), i_{\mathcal{A}_{pomn}}(\alpha_2)) \\
&= h(i_{\mathcal{A}_{pomn}}(\alpha_1) +^n i_{\mathcal{A}_{pomn}}(\alpha_2)) \\
&= h(i_{\mathcal{A}_{pomn}}(\alpha_1)) +_n h(i_{\mathcal{A}_{pomn}}(\alpha_2)) \\
&\qquad \text{by the above proof} \\
&= (i_{\mathcal{A}_n}(\alpha_1)) +_n i_{\mathcal{A}_n}(\alpha_2)) \\
&\qquad \text{by induction} \\
i_{\mathcal{A}_n}(\mathsf{comb}(\alpha_1,\alpha_2)) &= \mathsf{comb}^{\mathcal{A}_n}(i_{\mathcal{A}_n}(\alpha_1), i_{\mathcal{A}_n}(\alpha_2)) \\
&= (i_{\mathcal{A}_n}(\alpha_1)) +_n i_{\mathcal{A}_n}(\alpha_2))
\end{aligned}$$

Finally I need to show that $\equiv_{\mathcal{A}_{pn}} \implies \equiv_{\mathcal{A}_n}$ Recall the definition of $\equiv_{\mathcal{A}_{pn}}$.

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

Since $\equiv_{\mathcal{A}_{pomn}} \implies \equiv_{\mathcal{A}_n}$, I need only consider the case where $i_{\mathcal{A}_{pomn}}(\alpha) \equiv_{\mathcal{C}} i_{\mathcal{A}_{pomn}}(\beta)$. There are two cases:

- $i_{\mathcal{A}_{pomn}}(\alpha) = ((c \mid c') \mid c'', N)$ and $i_{\mathcal{A}_{pomn}}(\beta) = (c \mid (c' \mid c''), N)$. Hence

$$\begin{aligned}
h(((c \mid c') \mid c'', N)) &= (h(c, M) +_n h(c', M'')) +_n h(c'', M'') \\
&\qquad \text{for suitable } MM'M'' = N \\
&= h(c, M) +_n (h(c', M'') +_n h(c'', M'')) \\
&\qquad \text{since } +_n \text{ is assocative} \\
&= h((c \mid (c' \mid c''), N))
\end{aligned}$$

  Therefore $h(i_{\mathcal{A}_{pomn}}(\alpha)) = h(i_{\mathcal{A}_{pomn}}(\beta))$, Hence $i_{\mathcal{A}_n}(\alpha) = i_{\mathcal{A}_n}(\beta)$, and $\alpha \equiv_{\mathcal{A}_n} \beta$ as required.

- $i_{\mathcal{A}_{pomn}}(\alpha) = (c \mid c', N)$ and $i_{\mathcal{A}_{pomn}}(\beta) = (c' \mid c, N)$. This is done in a similar manner to the above case, using the commutativity of $+_n$.

Hence I have shown conservativity as required to apply the refining extension theorem.

B.3 Semantic equivalence result required in Section 6.4.3

Consider the signature Σ_{MPExt},

$$\begin{array}{l}
(\ \ \mathsf{A}, \mathsf{L}, \mathsf{Act}, \mathsf{Act}_{\mathsf{new}}, \mathsf{P};\\
\quad \{\mathsf{a}\}_{\mathsf{a}\in\mathsf{A}}, \{\mathsf{l}\}_{\mathsf{l}\in\mathsf{L}}, \mathsf{nil}, \{\mathsf{Cn}\}_{\mathsf{Cn}\in\mathsf{Const}}, \bot_{\mathsf{A}}, \bot_{\mathsf{Act}}, \bot_{\mathsf{Act}_{\mathsf{new}}}, \bot_{\mathsf{L}};\\
\quad \mathsf{pref}, \mathsf{plus}, \mathsf{par}, \mathsf{act}, \mathsf{comb}, \mathsf{comb}_{\mathsf{new}}, \mathsf{concat}\ \ )
\end{array}$$

with the sorts

$$\begin{array}{ll}
\mathsf{a} :\rightarrow \mathsf{A} \quad \forall \mathsf{a} \in \mathsf{A} & \mathsf{nil} :\rightarrow \mathsf{P}\\
\mathsf{l} :\rightarrow \mathsf{L} \quad \forall \mathsf{l} \in \mathsf{L} & \mathsf{Cn} :\rightarrow \mathsf{P} \quad \forall \mathsf{Cn} \in \mathsf{Const}\\
\mathsf{act} : \mathsf{A}, \mathsf{L} \rightarrow \mathsf{Act} & \mathsf{pref} : \mathsf{A}, \mathsf{P} \rightarrow \mathsf{P}\\
\mathsf{concat} : \mathsf{A}, \mathsf{Act} \rightarrow \mathsf{Act}_{\mathsf{new}} & \\
\mathsf{concat} : \mathsf{A}, \mathsf{Act}_{\mathsf{new}} \rightarrow \mathsf{Act}_{\mathsf{new}} & \\
\mathsf{comb} : \mathsf{Act}, \mathsf{Act} \rightarrow \mathsf{Act} & \mathsf{par} : \mathsf{P}, \mathsf{P} \rightarrow \mathsf{P}\\
\mathsf{comb}_{\mathsf{new}} : \mathsf{Act}, \mathsf{Act} \rightarrow \mathsf{Act}_{\mathsf{new}} & \\
\mathsf{comb}_{\mathsf{new}} : \mathsf{Act}, \mathsf{Act}_{\mathsf{new}} \rightarrow \mathsf{Act}_{\mathsf{new}} & \\
\mathsf{comb}_{\mathsf{new}} : \mathsf{Act}_{\mathsf{new}}, \mathsf{Act} \rightarrow \mathsf{Act}_{\mathsf{new}} & \\
\mathsf{comb}_{\mathsf{new}} : \mathsf{Act}_{\mathsf{new}}, \mathsf{Act}_{\mathsf{new}} \rightarrow \mathsf{Act}_{\mathsf{new}} & \\
\bot_{\mathsf{A}} :\rightarrow \mathsf{A} & \mathsf{plus} : \mathsf{P}, \mathsf{P} \rightarrow \mathsf{P}\\
\bot_{\mathsf{L}} :\rightarrow \mathsf{L} & \\
\bot_{\mathsf{Act}} :\rightarrow \mathsf{Act} & \\
\bot_{\mathsf{Act}_{\mathsf{new}}} :\rightarrow \mathsf{Act}_{\mathsf{new}} &
\end{array}$$

Assume that there is a way in which closed terms are assigned to elements of Const. I give here rules for both $\mathcal{E}_{\mathrm{MP}}$ and $\mathcal{E}_{\mathrm{MPExt}}$. The rules and rule schemas are given in Tables B.1 and B.2. Let the rules given by these rules and rule schemas be denoted R_{MPExt}, and the eTSS defined by Σ_{MPExt} and R_{MPExt} be $\mathcal{E}_{\mathrm{MPExt}}$. Since Σ_{MPExt} contains everything that Σ_{MP} contains, I will use Σ_{MPExt} for $\Sigma_{\mathrm{MP}} \oplus\!\!\!> \Sigma_{\mathrm{MPExt}}$.

$$\frac{}{\mathsf{pref}(z_{\mathsf{A}}, x) \xrightarrow{\mathsf{act}(z_{\mathsf{A}}, \mathsf{new}(z_{\mathsf{L}}))} x}$$

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

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

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

$$\left\{ \quad \frac{X \xrightarrow{z_{\mathsf{Act}}} y}{\mathsf{Cn} \xrightarrow{z_{\mathsf{Act}}} y} \quad \middle| \quad \mathsf{Cn} \overset{\mathrm{def}}{=} X, \mathsf{Cn} \in \mathsf{Const} \quad \right\}$$

Table B.1: Rules and rule schemas for MP

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

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

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

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

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

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

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

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

$$\left\{ \quad \frac{X \xrightarrow{z_{\mathsf{Act_{new}}}} y}{\mathsf{Cn} \xrightarrow{z_{\mathsf{Act_{new}}}} y} \quad \middle| \quad \mathsf{Cn} \overset{\mathrm{def}}{=} X, \mathsf{Cn} \in \mathsf{Const} \quad \right\}$$

Table B.2: Rules and rule schemas for MPExt

Sort	**Carrier set**
A	$A \cup \{\perp_{\mathsf{A}}\}$
L	$\mathbb{N}$
Act	$\mathcal{C}' \cup \{\perp_{\mathsf{Act}}\}$
$\mathsf{Act}_{\mathsf{new}}$	$\mathcal{C} \cup \{\perp_{\mathsf{Act}_{\mathsf{new}}}\}$

Table B.3: Carrier sets for $\mathcal{A}_{pom}$

I define a Σ_{MPExt}-algebra $\mathcal{A}_{pom}$ to represent the actual process algebra labels of pomset CCS, where $\mathcal{C}$ is defined as follows $c ::= a \mid a : c \mid c|c$ for $a \in A$. Also let $\mathcal{C}'$ be defined as follows $c' ::= a \mid c'|c'$ for $a \in A$. Clearly $A \subset \mathcal{C}' \subset \mathcal{C}$.

I will also assume there is an congruence $\equiv_{\mathcal{C}}$ on $\mathcal{C}'$ and $\mathcal{C}$ generated by the following axioms as in Castellani's original definition [Cas88]

$$c \mid c' = c' \mid c \quad \text{and} \quad c \mid (c' \mid c'') = (c \mid c') \mid c''.$$

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

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

The carrier sets and functions for $\mathcal{A}_{pom}$ are given in Tables B.3 and B.4.

I now want a slightly different algebra to represent the labels of a different process algebra based around pomset bisimulation but with some restriction on which processes can contribute to parallel computation. I will call this Σ_{MPExt}-algebra $\mathcal{A}_{pomn}$.

Let N and M be sequences of positive natural numbers without repetition, and let $N \cap M$ indicate the intersection of sequences. Then let $(a, N) +^n (b, M)$ where $a, b \in \mathcal{C}'$, be defined as equal to $(a \mid b, NM)$ whenever $N \cap M = \emptyset$.

$$
\begin{array}{rcl}
\mathsf{a}^{\mathcal{A}_{pom}} & = & a \quad \forall \mathsf{a} \in \mathsf{A} \\
\mathsf{l}^{\mathcal{A}_{pom}} & = & num(l) \quad \forall \mathsf{l} \in \mathsf{L} \\
\mathsf{act}^{\mathcal{A}_{pom}}(\zeta_1, \zeta_2) & = & \begin{cases} \zeta_1 & \text{if } \zeta_1 \in A \text{ and } \zeta_2 \in \mathbb{N}^+ \\ \bot_{\mathsf{Act}} & \text{otherwise} \end{cases} \\
\mathsf{comb}^{\mathcal{A}_{pom}}(\zeta_1, \zeta_2) & = & \begin{cases} \zeta_1 \mid \zeta_2 & \text{if } \zeta_1, \zeta_2 \in \mathcal{C}' \\ \bot_{\mathsf{Act}} & \text{otherwise} \end{cases} \\
\mathsf{concat}^{\mathcal{A}_{pom}}(\zeta_1, \zeta_2) & = & \begin{cases} \zeta_1 : \zeta_2 & \text{if } \zeta_1 \in A,\ \zeta_2 \in \mathcal{C} \\ \bot_{\mathsf{Act_{new}}} & \text{otherwise} \end{cases} \\
\mathsf{comb_{new}}^{\mathcal{A}_{pom}}(\zeta_1, \zeta_2) & = & \begin{cases} \zeta_1 \mid \zeta_2 & \text{if } \zeta_1, \zeta_2 \in \mathcal{C} \\ \bot_{\mathsf{Act_{new}}} & \text{otherwise} \end{cases} \\
\bot_{\mathsf{A}}^{\mathcal{A}_{pom}} & = & \bot_{\mathsf{A}} \\
\bot_{\mathsf{L}}^{\mathcal{A}_{pom}} & = & 0 \\
\bot_{\mathsf{Act}}^{\mathcal{A}_{pom}} & = & \bot_{\mathsf{Act}} \\
\bot_{\mathsf{Act_{new}}}^{\mathcal{A}_{pom}} & = & \bot_{\mathsf{Act_{new}}}
\end{array}
$$

Table B.4: Functions for $\mathcal{A}_{pom}$

Sort	Carrier set
A	$A \cup \{\bot_{\mathsf{A}}\}$
L	$\mathbb{N}$
Act	$\mathcal{D} = \bigcup_{j \geqslant 1}^{n} \{(c', a_1 \ldots a_j) \mid c' \in \mathcal{C}', \mathrm{acts}(c') = j,$ $a_1, \ldots, a_j \in \{1, \ldots, j\}, \text{pairwise disjoint}\} \cup \{\bot_{\mathsf{Act}}\}$
$\mathsf{Act}_{\mathsf{new}}$	$\mathcal{C} \cup \{\bot_{\mathsf{Act}_{\mathsf{new}}}\}$

Table B.5: Carrier sets for $\mathcal{A}_{pomn}$

Also define a function acts : $\mathcal{C}' \to \mathbb{N}$ as

$$\begin{aligned} \mathrm{acts}(c \mid c') &= \mathrm{acts}(c) + \mathrm{acts}(c') \\ \mathrm{acts}(a) &= 1. \end{aligned}$$

Extend $\equiv_{\mathcal{C}}$ to $\mathcal{D}$ in the obvious manner. The carrier sets and functions for $\mathcal{A}_{pomn}$ are given in Tables B.5 and B.6.

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

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

I want to show that for $u, v \in \mathbf{T}(\Sigma_{\text{MPExt}})_{\mathsf{P}}$

$$u \sim_{\equiv_{\mathcal{A}_{pn}}} v \iff u \sim_{\equiv_{\mathcal{A}_p}} v.$$

Here I am working with a definition of bisimulation which does not consider the transitions that are labelled with terms indicating undefined—in this case the terms in the set $\{\bot_{\mathsf{A}}, \bot_{\mathsf{L}}, \bot_{\mathsf{Act}}, \bot_{\mathsf{Act}_{\mathsf{new}}}\}$.

If I can show that $\equiv_{\mathcal{A}_{pn}} \subseteq \equiv_{\mathcal{A}_p}$ then it will follow that $\sim_{\equiv_{\mathcal{A}_{pn}}} \subseteq \sim_{\equiv_{\mathcal{A}_p}}$. However, because some terms are equated to $\bot_{\mathsf{Act}}$ by $\mathcal{A}_{pn}$ and not by $\mathcal{A}_p$, this approach cannot be used.

$$\begin{aligned}
\mathsf{a}^{\mathcal{A}_{pomn}} &= a \quad \forall \mathsf{a} \in \mathsf{A} \\
\mathsf{l}^{\mathcal{A}_{pomn}} &= num(l) \quad \forall \mathsf{l} \in \mathsf{L} \\
\mathsf{act}^{\mathcal{A}_{pomn}}(\zeta_1, \zeta_2) &= \begin{cases} (\zeta_1, \zeta_2) & \text{if } \zeta_1 \in A \text{ and } \zeta_2 \leqslant n \\ \bot_{\mathsf{Act}} & \text{otherwise} \end{cases} \\
\mathsf{comb}^{\mathcal{A}_{pomn}}(\zeta_1, \zeta_2) &= \begin{cases} \zeta_1 +^n \zeta_2 & \text{if } \zeta_1, \zeta_2 \in \mathcal{D} \\ & \text{and } \zeta_1 +^n \zeta_2 \text{ defined} \\ \bot_{\mathsf{Act}} & \text{otherwise} \end{cases} \\
\mathsf{concat}^{\mathcal{A}_{pomn}}(\zeta_1, \zeta_2) &= \begin{cases} \zeta_1 : \zeta_{2,1} & \text{if } \zeta_1 \in A,\ \zeta_2 \in \mathcal{D} \text{ and} \\ & \zeta_2 = (\zeta_{2,1}, \zeta_{2,2}) \\ \zeta_1 : \zeta_2 & \text{if } \zeta_1 \in A,\ \zeta_2 \in \mathcal{C} \\ \bot_{\mathsf{Act_{new}}} & \text{otherwise} \end{cases} \\
\mathsf{comb_{new}}^{\mathcal{A}_{pomn}}(\zeta_1, \zeta_2) &= \begin{cases} \zeta_{1,1} \mid \zeta_{2,1} & \text{if } \zeta_1 \in \mathcal{D}, \\ & \zeta_2 \in \mathcal{D} \text{ and} \\ & \zeta_1 = (\zeta_{1,1}, \zeta_{1,2}) \\ & \zeta_2 = (\zeta_{2,1}, \zeta_{2,2}) \\ \zeta_{1,1} \mid \zeta_2 & \text{if } \zeta_1 \in \mathcal{D},\ \zeta_2 \in \mathcal{C} \text{ and} \\ & \zeta_1 = (\zeta_{1,1}, \zeta_{1,2}) \\ \zeta_1 \mid \zeta_{2,1} & \text{if } \zeta_1 \in \mathcal{C},\ \zeta_2 \in \mathcal{D} \text{ and} \\ & \zeta_2 = (\zeta_{2,1}, \zeta_{2,2}) \\ \zeta_1 \mid \zeta_2 & \text{if } \zeta_1, \zeta_2 \in \mathcal{C} \\ \bot_{\mathsf{Act_{new}}} & \text{otherwise} \end{cases} \\
\bot_{\mathsf{A}}^{\mathcal{A}_{pomn}} &= \bot_{\mathsf{A}} \\
\bot_{\mathsf{L}}^{\mathcal{A}_{pomn}} &= 0 \\
\bot_{\mathsf{Act}}^{\mathcal{A}_{pomn}} &= \bot_{\mathsf{Act}} \\
\bot_{\mathsf{Act_{new}}}^{\mathcal{A}_{pomn}} &= \bot_{\mathsf{Act_{new}}}
\end{aligned}$$

Table B.6: Functions for $\mathcal{A}_{pomn}$

It is possible though to show that for $\alpha, \beta \not\equiv_{\mathcal{A}_{pn}} \gamma$ for $\gamma \in \{\bot_{\mathsf{A}}, \bot_{\mathsf{L}}, \bot_{\mathsf{Act}}, \bot_{\mathsf{Act_{new}}}\}$

$$\alpha \equiv_{\mathcal{A}_{pn}} \beta \Longrightarrow \alpha \equiv_{\mathcal{A}_p} \beta$$

Let $\alpha \equiv_{\mathcal{A}_{pn}} \beta$ then I need to consider the four possible sorts for these terms. Clearly $\alpha, \beta \in \mathbf{T}(\Sigma_{\text{MPExt}})_{\mathsf{A}}$ or $\mathbf{T}(\Sigma_{\text{MPExt}})_{\mathsf{L}}$, the implication holds since both algebras are the same for these terms.

If $\alpha, \beta \in \mathbf{T}(\Sigma_{\text{MPExt}})_{\mathsf{Act}} - \{\bot_{\mathsf{Act}}\}$ then $i_{\mathcal{A}_{pomn}}(\alpha) = (c, N)$ for some $c \in \mathcal{C}'$ and N sequence of natural numbers with no repeats; and $i_{\mathcal{A}_{pom}}(\alpha) = c$. Similarly for β. Since $i_{\mathcal{A}_{pomn}}(\alpha) = i_{\mathcal{A}_{pomn}}(\beta)$, then $i_{\mathcal{A}_{pom}}(\alpha) = i_{\mathcal{A}_{pom}}(\beta)$. A similar argument can be used for $i_{\mathcal{A}_{pomn}}(\alpha) \equiv_{\mathcal{C}} i_{\mathcal{A}_{pomn}}(\beta)$.

If $\alpha, \beta \in \mathbf{T}(\Sigma_{\text{MPExt}})_{\mathsf{Act_{new}}} - \{\bot_{\mathsf{Act_{new}}}\}$ then $i_{\mathcal{A}_{pomn}}(\alpha) = c$ for some $c \in \mathcal{C}'$, $i_{\mathcal{A}_{pom}}(\alpha) = c$ and hence the implication holds. A similar argument can be used for $i_{\mathcal{A}_{pomn}}(\alpha) \equiv_{\mathcal{C}} i_{\mathcal{A}_{pomn}}(\beta)$.

In light of the above, I need to prove $\sim_{\equiv_{\mathcal{A}_{pn}}} \subseteq \sim_{\equiv_{\mathcal{A}_p}}$ by dealing directly with the eTSS and equivalences. Assume $t \sim_{\equiv_{\mathcal{A}_{pn}}} u$. I wish to show that $t \sim_{\equiv_{\mathcal{A}_p}} u$. Consider a transition $t \xrightarrow{\alpha} t'$ with $\alpha \not\equiv_{\mathcal{A}_p} \bot_{\mathsf{Act}}$ and $\alpha \not\equiv_{\mathcal{A}_p} \bot_{\mathsf{Act_{new}}}$. It is not possible to use the argument that $t \sim_{\equiv_{\mathcal{A}_{pn}}} u'$ and hence a matching transition can be found, since it may be the case that $\alpha \equiv_{\mathcal{A}_{pn}} \bot_{\mathsf{Act}}$ or $\alpha \equiv_{\mathcal{A}_{pn}} \bot_{\mathsf{Act_{new}}}$.

Hence I need to work with the proofs which are generated. I will refer to the following rule as the Act *parallel rule.*

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

- First consider $\alpha \in \mathbf{T}(\Sigma_{\text{MPExt}})_{\mathsf{Act}}$. Note that it is possible to show that

  $$t \xrightarrow{\alpha} t', \;\; \alpha \in \mathbf{T}(\Sigma_{\text{MPExt}})_{\mathsf{Act}} \Longleftrightarrow t \xrightarrow{\alpha'} t', \;\; \alpha' \in \mathbf{T}(\Sigma_{\text{MPExt}})_{\mathsf{Act_{new}}}$$

  with $i_{\mathcal{A}_p}(\alpha) = i_{\mathcal{A}_p}(\alpha')$. (Note that α and α' cannot be equivalent because they have different sorts.) Moreover, it can be shown that there is a proof of $t \xrightarrow{\alpha'} t'$ for $\alpha' \in \mathbf{T}(\Sigma_{\text{MPExt}})_{\mathsf{Act_{new}}}$ which does not use the Act parallel rule, and such that no term of the form $\mathsf{act}(\mathsf{a}, \mathsf{l})$ has an l such that $\mathsf{l}^{\mathcal{A}_{pomn}} > n$. Therefore $\alpha' \not\equiv_{\mathcal{A}_{pn}} \bot_{\mathsf{Act_{new}}}$ and $t \xrightarrow{\alpha'} t'$ with $\alpha' \in \mathbf{T}(\Sigma_{\text{MPExt}})_{\mathsf{Act_{new}}}$. Since

$t \sim_{\equiv_{\mathcal{A}_{pn}}} u$, there exist β' and u' such that $u \xrightarrow{\beta'} u'$ with $\alpha' \equiv_{\mathcal{A}_{pn}} \beta'$ and since neither α' nor β' are equivalent to $\perp_{\mathsf{Act}_{\mathsf{new}}}$, $\alpha' \equiv_{\mathcal{A}_p} \beta'$. In other words, $i_{\mathcal{A}_p}(\alpha') = i_{\mathcal{A}_p}(\beta')$.

  By the above, I can find $\beta \in \mathbf{T}(\Sigma_{\mathrm{MPExt}})_{\mathsf{Act}}$ such that $u \xrightarrow{\beta} u'$ and $i_{\mathcal{A}_p}(\beta') = i_{\mathcal{A}_p}(\beta)$. Hence I have found a transition $u \xrightarrow{\beta} u'$ with $\alpha \equiv_{\mathcal{A}_p} \beta$ and $t' \sim_{\equiv_{\mathcal{A}_{pn}}} u'$ as required.

- Next consider $\alpha \in \mathbf{T}(\Sigma_{\mathrm{MPExt}})_{\mathsf{Act}_{\mathsf{new}}}$. By a similar argument it is possible to find $t \xrightarrow{\alpha'} t'$ such that $\alpha \equiv_{\mathcal{A}_p} \alpha'$ and the proof of the new transition does not use the Act parallel rule, and where no term of the form $\mathsf{act}(\mathsf{a}, \mathsf{l})$ has an l such that $\mathsf{l}^{\mathcal{A}_{pomn}} > n$. Hence $\alpha' \not\equiv_{\mathcal{A}_{pn}} \perp_{\mathsf{Act}_{\mathsf{new}}}$.

  Since $t \sim_{\equiv_{\mathcal{A}_{pn}}} u$, there exist β' and u' such that $u \xrightarrow{\beta'} u'$ with $\alpha' \equiv_{\mathcal{A}_{pn}} \beta'$ and since neither α nor β are equivalent to $\perp_{\mathsf{Act}_{\mathsf{new}}}$, $\alpha' \equiv_{\mathcal{A}_p} \beta'$. In other words, $\alpha \equiv_{\mathcal{A}_p} \beta'$ as required.

I also need to show the converse of this, namely that $\sim_{\equiv_{\mathcal{A}_p}} \subseteq \sim_{\equiv_{\mathcal{A}_{pn}}}$. Consider $t \sim_{\equiv_{\mathcal{A}_p}} u$. I wish to show that $t \sim_{\equiv_{\mathcal{A}_{pn}}} u$. Consider a transition $t \xrightarrow{\alpha} t'$ with $\alpha \not\equiv_{\mathcal{A}_{pn}} \perp_{\mathsf{Act}}$ and $\alpha \not\equiv_{\mathcal{A}_{pn}} \perp_{\mathsf{Act}_{\mathsf{new}}}$. It is not possible to use the argument that $t \sim_{\equiv_{\mathcal{A}_p}} u'$ since $\alpha \equiv_{\mathcal{A}_p} \beta \not\Longrightarrow \alpha \equiv_{\mathcal{A}_{pn}} \beta$ for $\alpha, \beta \notin \{\perp_{\mathsf{Act}}, \perp_{\mathsf{Act}_{\mathsf{new}}}\}$. Hence, a different approach is required.

- First consider $\alpha \in \mathbf{T}(\Sigma_{\mathrm{MPExt}})_{\mathsf{Act}}$. Since $\alpha \not\equiv_{\mathcal{A}_{pn}} \perp_{\mathsf{Act}}$, it can be argued that the term l_i appears in α at most n times and moreover all occurrences are distinct and for all i, $\mathsf{l}_i^{\mathcal{A}_{pomn}} \leqslant n$. Since $t \sim_{\equiv_{\mathcal{A}_p}} t'$, there exist β and u' such that $u \sim_{\equiv_{\mathcal{A}_p}} u'$ and $\alpha \equiv_{\mathcal{A}_p} \beta$.

  Moreover, it can be shown that there are the same number of occurrences of terms of the form l_i in β (otherwise it would not be equivalent to α). Hence it is possible (by suitable choice of which l_i's appear in the term) to find a term β' such that $u \xrightarrow{\beta} u'$ and $\beta' \not\equiv_{\mathcal{A}_{pn}} \perp_{\mathsf{Act}}$ with $\beta' \equiv_{\mathcal{A}_{pn}} \alpha$ as required.

- Next consider $\alpha \in \mathbf{T}(\Sigma_{\mathrm{MPExt}})_{\mathsf{Act}_{\mathsf{new}}}$. Since $t \sim_{\equiv_{\mathcal{A}_p}} u$, there exist β and u' such that $u \xrightarrow{\beta} u'$ and $\alpha \equiv_{\mathcal{A}_p} \beta$. As before it is possible to find an β' such that $u \xrightarrow{\beta'} u'$ and such that the Act parallel rule is not used in the proof and no term of the form $\mathsf{act}(\mathsf{a}, \mathsf{l})$ has an l such that $\mathsf{l}^{\mathcal{A}_{pomn}} > n$. Hence $\beta' \not\equiv_{\mathcal{A}_{pn}} \bot_{\mathsf{Act}}$. Moreover it can be shown (by suitable choice of which l_i's appear in the term) that $\alpha \equiv_{\mathcal{A}_{pn}} \beta'$ as required.

Hence I have shown that $\sim_{\equiv_{\mathcal{A}_p}} = \sim_{\equiv_{\mathcal{A}_{pn}}}$ as required.

Bibliography

[ABV94] L. Aceto, B. Bloom, and F. Vaandrager. Turning SOS rules into equations. *Information and Computation*, **111**, 1–52, 1994. (p 30)

[AC96] A. Arnold and I. Castellani. An algebraic characterisation of observation equivalence. *Theoretical Computer Science*, **156**, 289–299, 1996. (pp 28, 47)

[Ace94a] L. Aceto. Deriving complete inference systems for a class of GSOS languages generating regular behaviours. In Jonsson and Parrow [JP94], 449–464. (p 30)

[Ace94b] L. Aceto. GSOS and finite labelled transition systems. *Theoretical Computer Science*, **131**, 181–195, 1994. (p 30)

[Ace94c] L. Aceto. A static view of localities. *Formal Aspects of Computing*, **6**(2), 201–222, 1994. (pp 9, 37)

[AD89] A. Arnold and A. Dicky. An algebraic characterisation of transition system equivalences. *Information and Computation*, **82**, 198–229, 1989. (pp 27, 28, 50)

[ADCRDR89] G. Ausiello, M. Dezani-Ciancaglini, and S. Ronchi Della Rocca, (eds). *ICALP 88*, Lecture Notes in Computer Science **372**. Springer-Verlag, 1989. (p 203)

[ADNF87] L. Aceto, R. De Nicola, and A. Fantechi. Testing equivalences for event structures. In Venturini Zilli [VZ87], 1–20. (p 7)

[AG92] E. Astesiano and Reggio G. Observational structures and their logic. *Theoretical Computer Science*, **96**, 249–283, 1992. (p 31)

[AH93] L. Aceto and M. Hennessy. Towards action refinement in process algebras. *Information and Computation*, **103**, 204–269, 1993. (pp 13, 26, 33, 37)

[AH94] L. Aceto and M. Hennessy. Adding action refinement to a finite process algebra. *Information and Computation*, **115**, 179–247, 1994. (pp 13, 26, 33, 37)

[AI96] L. Aceto and A. Ingólfsdóttir. CPO models for compact GSOS languages. *Information and Computation*, **129**, 107–141, 1996. (p 30)

[AJ95] L. Aceto and A. Jeffrey. A complete axiomatisation of timed bisimulation for a class of timed regular behaviours. *Theoretical Computer Science*, **152**, 251–268, 1995. (p 17)

[AM91a] S. Abramsky and T.S.E. Maibaum, (eds). *TAPSOFT '91*, Lecture Notes in Computer Science **493**. Springer-Verlag, 1991. (p 204)

[AM91b] S. Abramsky and T.S.E. Maibaum, (eds). *TAPSOFT '91*, Lecture Notes in Computer Science **494**. Springer-Verlag, 1991. (p 204)

[AM96] L. Aceto and D. Murphy. Timing and causality in process algebra. *Acta Informatica*, **33**, 317–350, 1996. (pp 13, 36)

[Apt84] K.R. Apt, (ed). *Logics and models of concurrent systems*, Volume **F13** of *Nato ASI Series*. Springer-Verlag, 1984. (p 202)

[Arn90] A. Arnold, (ed). *CAAP 90*, Lecture Notes in Computer Science **431**. Springer-Verlag, 1990. (p 204)

[Arn93] A. Arnold. Verification and comparison of transition systems. In Gaudel and Jouannaud [GJ93], 121–135. (pp 27, 28)

[AS92] C. Autant and Ph. Schnoebelen. Place bisimulations in Petri nets. In Jensen [Jen92], 45–61. (p 7)

[AS94] S. Abiteboul and E. Shamir, (eds). *ICALP '94*, Lecture Notes in Computer Science **820**. Springer-Verlag, 1994. (p 203)

[BB91a] J.C.M. Baeten and J.A. Bergstra. Real space process algebra. In Baeten and Groote [BG91], 96–110. (p 16)

[BB91b] J.C.M. Baeten and J.A. Bergstra. Real time process algebra. *Formal Aspects of Computing*, **3**, 142–188, 1991. (p 16)

[BB93a] J.C.M. Baeten and J.A. Bergstra. Non interleaving process algebra. In Best [Bes93], 308–232. (p 22)

[BB93b] J.C.M. Baeten and J.A. Bergstra. Real space process algebra. *Formal Aspects of Computing*, **5**, 481–529, 1993. (p 12)

[BB95] J.C.M. Baeten and J.A. Bergstra. Real time process algebra with infinitesimals. In Ponse et al. [PVvV95], 148–187. (p 16)

[BB96] J.C.M. Baeten and J.A. Bergstra. Discrete time process algebra. *Formal Aspects of Computing*, **8**, 188–208, 1996. (p 17)

[BBK86] J.C.M. Baeten, J.A. Bergstra, and J.W. Klop. Syntax and defining equations for an interrupt mechanism in basic process algebra. *Fundamenta Informatica*, **IX**, 127–168, 1986. (p 18)

[BBS88] D.B. Benson and O. Ben-Shachar. Bisimulation of automata. *Information and Computation*, **79**, 60–83, 1988. (p 28)

[BC88a] G. Boudol and I. Castellani. Concurrency and atomicity. *Theoretical Computer Science*, **59**, 25–84, 1988. (p 7)

[BC88b] G. Boudol and I. Castellani. A non-interleaving semantics for CCS based on proved transitions. *Fundamenta Informatica*, **XI**, 433–452, 1988. (pp 19, 22)

[BC88c] G. Boudol and I. Castellani. Permutations of transitions: an event structure semantics for CCS and SCCS. In de Bakker et al. [dBdRR88], 411–427. (p 19)

[BC91] G. Boudol and I. Castellani. Observing localities. In Tarlecki [Tar91], 93–102. (p 9)

[BC94] G. Boudol and I. Castellani. Flow models of distributed computations—three equivalent semantics for CCS. *Information and Computation*, **114**, 247–314, 1994. (pp 7, 25)

[BCHK91a] G. Boudol, I. Castellani, M. Hennessy, and A. Kiehn. Observing localities. Technical Report 4/91, Computer Science, University of Sussex, 1991. (p 40)

[BCHK91b] G. Boudol, I. Castellani, M. Hennessy, and A. Kiehn. A theory of processes with localities. Technical Report 13/91, Computer Science, University of Sussex, 1991. (p 42)

[BCHK92] G. Boudol, I. Castellani, M. Hennessy, and A. Kiehn. A theory of processes with localities. In Cleaveland [Cle92], 108–122. (pp 8, 9, 22)

[BCHK93] G. Boudol, I. Castellani, M. Hennessy, and A. Kiehn. Observing localities. *Theoretical Computer Science*, **114**, 31–61, 1993. (pp 9, 33, 35, 40, 42, 56, 59, 66)

[BCHK94] G. Boudol, I. Castellani, M. Hennessy, and A. Kiehn. A theory of processes with localities. *Formal Aspects of Computing*, **6**(2), 165–200, 1994. (pp 8, 9, 33, 35, 36, 37, 41, 56, 57, 59, 66, 94)

[Bed87] M. Bednarczyk. Categories of asynchronous systems. Technical Report 3/87, PhD Thesis, Department of Computer Science, University of Sussex, 1987. (p 22)

[Bes93] E. Best, (ed). *CONCUR '93*, Lecture Notes in Computer Science **715**. Springer-Verlag, 1993. (pp 201, 202, 208, 209)

[BG91] J.C.M. Baeten and J.F. Groote, (eds). *CONCUR '91*, Lecture Notes in Computer Science **527**. Springer-Verlag, 1991. (pp 201, 202, 204, 205, 206, 207, 208)

[BG96] R. Bol and J.F. Groote. The meaning of negative premises in transition system specifications. *Journal of the ACM*, **43**, 863–914, 1996. (p 29)

[BIM95] B. Bloom, S. Istrail, and A.R. Meyer. Bisimulation can't be traced. *Journal of the ACM*, **42**, 232–268, 1995. (p 29)

[BJKB$^+$93] E. Börger, G. Jäger, H. Kleine Büning, S. Martini, and M.M. Ritchter, (eds). *CSL '92*, Lecture Notes in Computer Science **702**. Springer-Verlag, 1993. (p 204)

[BK90] J.C.M. Baeten and J.W. Klop, (eds). *CONCUR '90*, Lecture Notes in Computer Science **458**. Springer-Verlag, 1990. (pp 206, 207, 209)

[BKLL95] E. Brinksma, J.-P. Katoen, R. Langerak, and D. Latella. A stochastic causality-based process algebra. *Computer Journal*, **38**, 552–565, 1995. (p 7)

[Blo95] B. Bloom. Structural operational semantics for weak bisimulations. *Theoretical Computer Science*, **146**, 25–68, 1995. (pp 30, 175)

[BMC94] C. Baier and M.E. Majster-Cederbaum. The connection between event structure semantics and an operational semantics for TCSP. *Acta Informatica*, **31**, 81–104, 1994. (p 7)

[Bou84] G. Boudol. Notes on algebraic calculi of processes. In Apt [Apt84], 261–303. (p 5)

[BS93] A.M. Borzyszkowski and S. Sokolowski, (eds). *MFCS '93*, Lecture Notes in Computer Science **711**. Springer-Verlag, 1993. (pp 202, 206, 207)

[BV93] J.C.M. Baeten and C. Verhoef. A congruence theorem for structured operational semantics with predicates. In Best [Bes93], 477–492. (p 29)

[BVNW87] F.J. Brandenburg, G. Vidal-Naquet, and M. Wirsing, (eds). *STACS '87*, Lecture Notes in Computer Science **247**. Springer-Verlag, 1987. (p 209)

[Cam91] J. Camilleri. A conditional operator for CCS. In Baeten and Groote [BG91], 142–156. (p 18)

[Cas88] I. Castellani. Bisimulations for concurrency. Technical Report CST-51-88, PhD Thesis, Department of Computer Science, University of Edinburgh, 1988. (pp 8, 19, 28, 33, 35, 40, 157, 164, 193)

[Cas93] I. Castellani. Observing distribution in processes. In Borzyszkowski and Sokolowski [BS93], 321–331. (pp 35, 37, 41)

[Cas95] I. Castellani. Observing distribution in processes: static and dynamic localities. *International Journal of Foundations of Computer Science*, **6**, 353–393, 1995. (p 9)

[CdCC92] R.J. Coelha da Costa and J.-P. Courtiat. A causality-based semantics for CCS. In Purushothaman and Zwarico [PZ92]. (p 7)

[CDN94] A. Corradini and R. De Nicola. Distribution and locality of concurrent systems. In Abiteboul and Shamir [AS94], 154–165. (pp 9, 35, 41)

[CDN97] A. Corradini and R. De Nicola. Localitty based semantics for process algebras. *Acta Informatica*, **34**, 291–324, 1997. (pp 8, 9, 35, 36, 40, 41)

[CFM90] A. Corradini, G.L. Ferrari, and U. Montanari. Transition systems with algebraic structure as models of computation. In Guessarian [Gue90], 185–222. (pp 19, 28)

[CH89] I. Castellani and M. Hennessy. Distributed bisimulations. *Journal of the ACM*, **36**(4), 887–911, October 1989. (pp 8, 35)

[CH90] R. Cleaveland and M. Hennessy. Priorities in process algebra. *Information and Computation*, **87**, 58–77, 1990. (p 18)

[Che92] L. Chen. An interleaving model for real-time systems. In Nerode and Taitslin [NT92], 81–92. (p 16)

[Che93] L. Chen. Timed processes: models, axioms and decidability. Technical Report CST-101-93, PhD Thesis, Department of Computer Science, University of Edinburgh, 1993. (p 16)

[Cle92] W.R. Cleaveland, (ed). *CONCUR '92*, Lecture Notes in Computer Science **630**. Springer-Verlag, 1992. (pp 201, 204, 205, 206, 208)

[CLN96] R. Cleaveland, G. Lüttgen, and V. Natarajan. A process algebra with distributed priorities. [MS96], 34–49. (p 18)

[CW95] J. Camilleri and G. Winskel. CCS with priority choice. *Information and Computation*, **116**, 26–37, 1995. (p 18)

[Dan91] M. Daniels. Modeling real-time behaviour with an interval time calculus. In Vytopil [Vyt91], 53–71. (p 16)

[dBdRR88] J.W. de Bakker, W.-P. de Roever, and G. Rozenberg, (eds). *Linear time, branching time and partial orders in logics and models for concurrency.* Lecture Notes in Computer Science **354**. Springer-Verlag, 1988. (pp 201, 203, 205, 206, 207)

[dBHdR92] J.W. de Bakker, C. HUizing, and G. de Roever, W.P. amd Rozenberg, (eds). *Real-time: theory in practice*, Lecture Notes in Computer Science **600**. Springer-Verlag, 1992. (p 207)

[dBNT87] J.W. de Bakker, A.J. Nijman, and P.C. Treleaven, (eds). *PARLE '87*, Lecture Notes in Computer Science **259**. Springer-Verlag, 1987. (p 209)

[DD89] P. Darondeau and P. Degano. Causal trees. In Ausiello et al. [ADCRDR89], 234–248. (pp 7, 9, 35, 66)

[DD90] P. Darondeau and P. Degano. Causal trees, interleaving + causality. In Guessarian [Gue90], 239–255. (pp ix, 7, 8, 66)

[DDNM88a] P. Degano, R. De Nicola, and U. Montanari. A distributed operational semantics for CCS based on condition/event systems. *Acta Informatica*, **26**, 59–91, 1988. (pp 7, 28)

[DDNM88b] P. Degano, R. De Nicola, and U. Montanari. On the consistency of "truly concurrent" operational and denotational semantics. [LIC88], 133–141. (pp 7, 22)

[DDNM88c] P. Degano, R. De Nicola, and U. Montanari. Partial ordering descriptions and observations of nondeterministic concurrent processes. In de Bakker et al. [dB-dRR88], 438–466. (p 7)

[DDNM90] P. Degano, R. De Nicola, and U. Montanari. A partial ordering semantics for CCS. *Theoretical Computer Science*, **75**, 223–262, 1990. (p 22)

[DDNM92] P. Degano, R. De Nicola, and U. Montanari. Observation trees. In Purushothaman and Zwarico [PZ92]. (pp 21, 22, 26, 32)

[DDNM93] P. Degano, R. De Nicola, and U. Montanari. Universal axioms for bisimulations. *Theoretical Computer Science*, **114**, 63–91, 1993. (pp 21, 22, 26, 32)

[DG93] P. Degano and R. Gorrieri. A causal operational semantics of action refinement. *Information and Computation*, **122**, 97–119, 1993. (p 7)

[DM87a] P. Degano and U. Montanari. Concurrent histories: a basis for observing distributed systems. *Journal of Computer and System Sciences*, **34**, 442–461, 1987. (p 22)

[DM87b] P. Degano and U. Montanari. A model of distributed systems based on graph rewriting. *Journal of the ACM*, **34**, 411–449, 1987. (p 7)

[DN87] R. De Nicola. Extensional equivalences for transition systems. *Acta Informatica*, **24**, 211–237, 1987. (p 25)

[DP92] P. Degano and C. Priami. Proved trees. In Kuich [Kui92], 629–640. (pp 21, 22, 32)

[dS85] R. de Simone. Higher-level synchronising devices in MEIJE-SCCS. *Theoretical Computer Science*, **37**, 245–267, 1985. (p 29)

[ES92] D. Etiemble and J.-C. Syre, (eds). *PARLE '92*, Lecture Notes in Computer Science **605**. Springer-Verlag, 1992. (p 204)

[Fan92] J. Fanchon. Dynamic concurrent processes. In Etiemble and Syre [ES92], 859–874. (pp 12, 37)

[Fer93] T. Fernando. Comparative transition system semantics. In Börger et al. [BJKB$^+$93], 146–166. (p 25)

[FGM91] G.L. Ferrari, R. Gorrieri, and U. Montanari. An extended expansion theorem. In Abramsky and Maibaum [AM91a], 29–48. (p 19)

[FM90] G.L. Ferrari and U. Montanari. Toward the unification of models for concurrency. In Arnold [Arn90], 162–176. (pp 19, 28)

[FM91] G.L. Ferrari and U. Montanari. The observation algebra of spatial pomsets. In Baeten and Groote [BG91], 108–122. (p 19)

[FMM91] G.L. Ferrari, U. Montanari, and M. Mowbray. On causality observed incrementally, finally. In Abramsky and Maibaum [AM91b], 26–41. (p 28)

[FvG96] W. Fokkink and R. van Glabbeek. Ntyft/nyxt rules reduce to ntree rules. *Information and Computation*, **126**, 1–10, 1996. (pp 31, 76, 92, 175)

[GJ93] M.-C. Gaudel and J.-P. Jouannaud, (eds). *TAPSOFT '93*, Lecture Notes in Computer Science **668**. Springer-Verlag, 1993. (p 201)

[GKP92] U. Goltz, R. Kuiper, and W. Peczek. Propositional temporal logics and eqivalences. In Cleaveland [Cle92], 222–236. (p 7)

[GL91] R. Gorrieri and C. Laneve. The limit of split$_n$-bisimulations for CCS agents. In Tarlecki [Tar91], 170–180. (pp ix, 13, 26, 27, 28, 33, 38)

[GL94] R. Gerber and I. Lee. A resource-based prioritized bisimulation for real-time systems. *Information and Computation*, **113**, 102–142, 1994. (p 18)

[GL95] R. Gorrieri and C. Laneve. Split and ST bisimulation semantics. *Information and Computation*, **118**, 272–288, 1995. (p 7)

[GM84] U. Goltz and A. Mycroft. On the relationship of CCS and Petri nets. In Paredaens [Par84], 196–208. (p 7)

[Gol90] U. Goltz. CCS and Petri nets. In Guessarian [Gue90], 334–357. (p 7)

[Gro93] J.F. Groote. Transition systems specifications with negative premises. *Theoretical Computer Science*, **118**, 263–299, 1993. (pp ix, 18, 29, 30)

[GRS95] R. Gorrieri, M. Roccetti, and E. Stancampiano. A theory of processes with durational actions. *Theoretical Computer Science*, **140**, 73–94, 1995. (p 13)

[Gue90] I. Guessarian, (ed). *Semantics of Systems of Concurrent Processes.* Lecture Notes in Computer Science **469**. Springer-Verlag, 1990. (pp 203, 205, 206, 209)

[GV92] J.F. Groote and F. Vaandrager. Structured operational semantics and bisimulation as a congruence. *Information and Computation*, **100**, 202–260, 1992. (pp 29, 72, 84, 97, 98)

[Hen88a] M. Hennessy. Axiomatising finite concurrent processes. *SIAM Journal on Computing*, **17**(5), 997–1017, October 1988. (pp 13, 33)

[Hen88b] M. Hennessy. Observing processes. In de Bakker et al. [dBdRR88], 173–200. (p 8)

[Hen91] M. Hennessy. A proof system for weak ST-bisimulation over a finite process algebra. Technical Report 6/91, Computer Science, University of Sussex, 1991. (pp 13, 33, 35, 42)

[Hen95] M. Hennessy. Concurrent testing of processes. *Acta Informatica*, **32**, 509–543, 1995. (p 13)

[Hil96] J. Hillston. *A compositional approach to performance modelling.* Cambridge University Press, 1996. (p 17)

[HR95] M. Hennessy and T. Regan. A process algebra for timed systems. *Information and Computation*, **117**, 221–239, 1995. (p 17)

[HSZM93] C. Ho-Stuart, H.S.M. Zedan, and Fang. M. Congruent weak bisimulation with dense real-time. *Information Processing Letters*, **46**, 55–61, 1993. (p 17)

[IPY93] P. Inverardi, C. Priami, and D. Yankelevich. Extended transition systems for parametric bisimulation. In Lingas et al. [LKC93], 558–569. (pp 22, 32)

[IPY94] P. Inverardi, C. Priami, and D. Yankelevich. Automatizing parametric reasoning on distributed concurrent systems. *Formal Aspects of Computing*, **6**, 676–695, 1994. (pp 22, 26, 32)

[Jan94] T. Janowski. Fault-tolerant bisimulations and process transformations. In Langmaack et al. [LdRV94], 373–392. (p 24)

[Jef91a] A. Jeffrey. Abstract timed observation and process algebra. In Baeten and Groote [BG91], 332–345. (p 17)

[Jef91b] A. Jeffrey. Translating timed process algebra into prioritzed process algebra. In Vytopil [Vyt91], 493–506. (p 18)

[Jef92] A. Jeffrey. A linear time algebra. In Larsen and Skou [LS92a], 432–442. (p 16)

[Jen92] K. Jensen, (ed). *13th International Conference on Application and Theory of Petri Nets*, Lecture Notes in Computer Science **616**. Springer-Verlag, 1992. (p 201)

[JM92] L. Jategaonkar and A. Meyer. Testing equivalence for Petri nets with action refinement. In Cleaveland [Cle92], 17–31. (p 7)

[JP94] B. Jonsson and J. Parrow, (eds). *CONCUR '94*, Lecture Notes in Computer Science **836**. Springer-Verlag, 1994. (pp 200, 206, 207)

[JS90] C.C. Jou and S.A. Smolka. Equivalences, congruences, and complete axiomatisations for probabilitics processes. In Baeten and Klop [BK90], 367–383. (pp 19, 26)

[KB92] B. Krieg-Brückner, (ed). *ESOP '92*, Lecture Notes in Computer Science **582**. Springer-Verlag, 1992. (p 206)

[KH94] A. Kiehn and M. Hennessy. On the decidability of non-interleaving process equivalences. In Jonsson and Parrow [JP94], 18–33. (pp 26, 176)

[Kie] A. Kiehn. Distributed bisimulation for finite CCS. Unpublished document. (p 8)

[Kie89] A. Kiehn. Distributed bisimulations for finite CCS. Technical Report 7/89, Computer Science, University of Sussex, 1989. (pp 8, 35)

[Kie93] A. Kiehn. Proof systems for cause based equivalences. In Borzyszkowski and Sokolowski [BS93], 547–556. (p 36)

[Kie94] A. Kiehn. Comparing locality and causality based equivalences. *Acta Informatica*, **31**(8), 697–718, 1994. (pp 7, 9, 26, 32, 33, 35, 41, 42, 57, 66, 157)

[KLP90] S. Kasangian, A. Labella, and A. Pettorossi. Observers, experiments and agents: a comprehensive approach to parallelism. In Guessarian [Gue90], 393–407. (p 25)

[Kri91] P. Krishnan. Distributed CCS. In Baeten and Groote [BG91], 393–407. (pp 11, 33, 36)

[Kri92] P. Krishnan. A semantics for multiprocessor systems. In Krieg-Brückner [KB92], 307–320. (pp 22, 33, 37)

[Kri94] P. Krishnan. A semantic characterisation for faults in replicated systems. *Theoretical Computer Science*, **128**, 159–177, 1994. (pp 24, 33)

[Kri96] P. Krishnan. Architectural CCS. *Formal Aspects of Computing*, **162**, 162–187, 1996. (pp 11, 22, 33, 36, 37, 155, 157)

[Kui92] W. Kuich, (ed). *ICALP 92*, Lecture Notes in Computer Science **623**. Springer-Verlag, 1992. (pp 204, 207)

[LAMRA91] J. Leach Albert, B. Monien, and M. Rodríguez Artalejo, (eds). *ICALP '91*, Lecture Notes in Computer Science **510**. Springer-Verlag, 1991. (p 209)

[LdRV94] H. Langmaack, W.-P. de Roever, and J. Vytopil, (eds). *Formal Techniques for Real-Time and Fault-Tolerant Systems*, Lecture Notes in Computer Science **863**. Springer-Verlag, 1994. (p 205)

[LIC88] *LICS 88*. Computer Science Press, 1988. (p 203)

[LIC92] *LICS 92*. IEEE Computer Society Press, 1992. (p 208)

[LKC93] A. Lingas, R. Karlsson, and S. Carlsson, (eds). *ICALP '93*, Lecture Notes in Computer Science **700**. Springer-Verlag, 1993. (p 205)

[LRT88] K. Lodaya, R. Ramanujam, and P.S. Thiagarajan. A logic from distributed transition systems. In de Bakker et al. [dBdRR88], 508–522. (p 24)

[LS92a] K.G. Larsen and A. Skou, (eds). *CAV '91*, Lecture Notes in Computer Science **575**. Springer-Verlag, 1992. (p 205)

[LS92b] K.G. Larsen and A. Skou. Compositional verificaton of probabilistic processes. In Cleaveland [Cle92], 456–471. (p 18)

[Lu93] R.Q. Lu. A true concurrency model of CCS semantics. *Theoretical Computer Science*, **113**, 231–258, 1993. (p 7)

[Mes90] J. Meseguer. Rewriting as a unified model of concurrency. In Baeten and Klop [BK90], 384–400. (p 25)

[Mil89] R. Milner. *Communication and concurrency.* Prentice Hall, 1989. (pp 4, 5, 35)

[MN92] M. Mukund and M. Nielsen. CCC, locations and asynchronous transition systems. In Shyamasundar [Shy92], 328–341. (pp ix, 22, 23, 37)

[MS96] U. Montanari and V. Sassone, (eds). *CONCUR '96*, Lecture Notes in Computer Science **1119**. Springer-Verlag, 1996. (p 203)

[MT90] F. Moller and C. Tofts. A temporal calculus of communicating systems. In Baeten and Klop [BK90], 401–415. (pp 16, 17)

[Mur91] D. Murphy. Testing, betting and timed true concurrency. In Baeten and Groote [BG91], 439–454. (p 7)

[Mur93] D. Murphy. Observing located concurrency. In Borzyszkowski and Sokolowski [BS93], 473–484. (pp 7, 37)

[MY89] U. Montanari and D.N. Yankelevich. An algebraic view of interleaving and distributed operational semantics for CCS. In Pitt et al. [PRD$^+$89], 5–20. (p 25)

[MY92] U. Montanari and D.N. Yankelevich. A parametric approach to localities. In Kuich [Kui92], 617–628. (pp 21, 26, 35, 41)

[MY95] U. Montanari and D.N. Yankelevich. Location equivalence in a parametric setting. *Theoretical Computer Science*, **149**, 299–332, 1995. (pp 21, 26)

[NC94] M. Nielsen and C. Clausen. Bisimulation for models in concurrency. In Jonsson and Parrow [JP94], 385–400. (p 22)

[NCCC94] V. Natarajan, I Christoff, L. Christoff, and R. Cleaveland. Priority and abstraction in process algebra. In Thiagarajan [Thi94], 217–230. (p 18)

[NRT92] M. Nielsen, G. Rozenburg, and P.S. Thiagarajan. Elementary transition systems. *Theoretical Computer Science*, **96**, 3–33, 1992. (p 22)

[NS82] M. Nielsen and E.M. Schmidt, (eds). *ICALP 82*, Lecture Notes in Computer Science **140**. Springer-Verlag, 1982. (p 209)

[NS92] X. Nicollin and J. Sifakis. An overview and synthesis on timed process algebras. In de Bakker et al. [dBHdR92], 526–548. (p 17)

[NS94] X. Nicollin and J. Sifakis. The algebra of timed processes, ATP—theory and application. *Information and Computation*, **114**, 131–178, 1994. (p 17)

[NT92] A. Nerode and M. Taitslin, (eds). *Symposuim on Logical Foundations of Computer Science*, Lecture Notes in Computer Science **620**. Springer-Verlag, 1992. (p 203)

[OH86] E.-R. Olderog and C.A.R. Hoare. Specification-oriented semantics for communicating processes. *Acta Informatica*, **23**, 9–66, 1986. (p 4)

[Old88] E.-R. Olderog. Strong bisimilarity on nets: a new concept for comparing net semantics. In de Bakker et al. [dBdRR88], 549–573. (p 7)

[Par84] J. Paredaens, (ed). *ICALP 84*, Lecture Notes in Computer Science **172**. Springer-Verlag, 1984. (p 204)

[Plo88] G. Plotkin. A structural approach to operational semantics. Report DAIMI FN-19, Computer Science Department, Aarhus University, 1988. (p 4)

[Pom85] L. Pomello. Some equivalence notions for concurrent systems—an overview. In Rozenberg [Roz85], 381–400. (p 7)

[Pra95] K.V.S. Prasad. A calculus of broadcasting systems. *Science of Computer Programming*, **25**, 285–327, 1995. (p 18)

[PRD$^+$89] D.H. Pitt, D.E. Rydeheard, P. Dybjer, A.M. Pitts, and A. Poigné, (eds). *Category Theory and Computer Science*. Lecture Notes in Computer Science **389**. Springer-Verlag, 1989. (p 207)

[PRR94] I. Prívara, B. Rovan, and P. Ružička, (eds). *NAPAW 92*, Lecture Notes in Computer Science **841**. Springer-Verlag, 1994. (p 208)

[PRS92] L. Pomello, G. Rozenberg, and C. Simone. A survey of equivalence notions for net based systems. In Rozenberg [Roz92], 410–472. (p 7)

[PVvV95] A. Ponse, C. Verhoef, and S.F.M. van Vlijmen, (eds). *Algebra of Communicating Processes 94*. Springer-Verlag, 1995. (p 201)

[PY94] C. Priami and D. Yankelvich. Read-write causality. In Prívara et al. [PRR94], 567–576. (pp 21, 35, 42)

[PZ92] S. Purushothaman and A. Zwarico, (eds). *Participants' proceedings of NAPAW 92*. Technical Report 92-15, Department of Computer Science, John Hopkins University, 1992. (pp 202, 204)

[QdFA93] J. Quemada, D. de Frutos, and A. Azcorra. TIC: a timed calculus. *Formal Aspects of Computing*, **5**, 224–252, 1993. (p 16)

[Roz85] G. Rozenberg, (ed). *Advances in Petri nets 1985*, Lecture Notes in Computer Science **222**. Springer-Verlag, 1985. (p 207)

[Roz92] G. Rozenberg, (ed). *Advances in Petri nets 1992*, Lecture Notes in Computer Science **609**. Springer-Verlag, 1992. (p 208)

[San96] D. Sangiorgi. Locality and interleaving semantics in calculi for mobile processes. *Information and Computation*, **155**, 39–84, 1996. (p 9)

[Sch91] Ph. Schnoebelen. Experiments on processes with backtracking. In Baeten and Groote [BG91], 480–494. (p 25)

[Sch95] S. Schneider. An operational semantics for Timed CSP. *Information and Computation*, **116**, 193–213, 1995. (p 16)

[Sha92] E. Shapiro. Embeddings among concurrent programming languages. In Cleaveland [Cle92], 486–504. (p 25)

[Shy92] R. Shyamasundar, (ed). *FST&TCS 92*, Lecture Notes in Computer Science **652**. Springer-Verlag, 1992. (p 207)

[SNW93] V. Sassone, M. Neilsen, and G. Winskel. A classification of models for concurrency. In Best [Bes93], 308–232. (p 25)

[SS96] S. Smolka and B. Steffen. Priority as extremal probability. *Formal Aspects of Computing*, **8**, 585–606, 1996. (p 19)

[Sta89] A. Stark. Concurrent transition systems. *Theoretical Computer Science*, **64**, 221–269, 1989. (p 22)

[Tar91] A. Tarlecki, (ed). *MFCS '91*, Lecture Notes in Computer Science **520**. Springer-Verlag, 1991. (pp 201, 204)

[Tau90] D. Taubner. Representing CCS programs by finite predicate nets. *Acta Informatica*, **27**, 533–565, 1990. (p 7)

[Thi94] P.S. Thiagarajan, (ed). *Foundations of Software Technology and Theoretical Computer Science*, Lecture Notes in Computer Science **880**. Springer-Verlag, 1994. (p 207)

[Tof94] C. Tofts. Processes with probabilities, priorities and time. *Formal Aspects of Computing*, **30**, 536–564, 1994. (pp 6, 19)

[Uli92] I. Ulidowski. Equivalences on observable processes. [LIC92], 148–159. (p 30)

[Ver94] C. Verhoef. A general conservative extension theorem in process algebra. *IFIP Transactions A*, **56**, 149–168, 1994. (pp 31, 97, 104)

[Ver95] C. Verhoef. A congruence theorem of structured operational semantics with predicates and negative premises. *Nordic Journal of Computing*, **2**, 274–302, 1995. (p 29)

[vG87] R.J. van Glabbeek. Bounded nondeterminism and the approximation induction principle in process algebras. In Brandenburg et al. [BVNW87], 336–347. (p 4)

[vG90a] R.J. van Glabbeek. *Comparative concurrency semantics and refinement of actions.* PhD thesis, Free University, Amsterdam, 1990. (p 25)

[vG90b] R.J. van Glabbeek. The linear time—branching time spectrum. In Baeten and Klop [BK90], 278–297. (pp 25, 32, 56)

[vG93] R.J. van Glabbeek. The linear time—branching time spectrum II (the semantics of sequential systems with silent moves). In Best [Bes93], 66–81. (pp 25, 32)

[vGG90] R.J. van Glabbeek and U. Goltz. Equivalences and refinement. In Guessarian [Gue90], 309–333. (p 7)

[vGV87] R. van Glabbeek and F. Vaandrager. Petri net models for algebraic theories of concurrency. In de Bakker et al. [dBNT87], 224–242. (pp 7, 13)

[Vyt91] J Vytopil, (ed). *Formal Techniques on Real-Time and Fault-Tolerant Systems*, Lecture Notes in Computer Science **571**. Springer-Verlag, 1991. (pp 203, 205)

[VZ87] M. Venturini Zilli, (ed). *Proceedings of the Advanced School on Mathematical Models for the Semantics of Parallelism*, Lecture Notes in Computer Science **280**. Springer-Verlag, 1987. (p 200)

[Win82] G. Winskel. Event structures for CCS and related languages. In Nielsen and Schmidt [NS82], 561–576. (p 7)

[Yi90] Wang Yi. Real-time behaviour of asynchronous agents. In Baeten and Klop [BK90], 502–520. (pp 16, 17)

[Yi91] Wang Yi. CCS + time = an interleaving model for real-time systems. In Leach Albert et al. [LAMRA91], 217–228. (p 16)