Symbolic Bisimulations and Proof Systems for the π-Calculus

H. Lin

Laboratory for Computer Science

Institute of Software, Chinese Academy of Sciences

E-mail: lhm@ios.ac.cn

Abstract

A theory of symbolic bisimulation for the π-calculus is proposed which captures the conventional notions of bisimulation-based equivalences for this calculus. Proof systems are presented for both *late* and *early* equivalences, and their soundness and completeness are proved. The proof system for early equivalence differs from that for late equivalence only in the inference rule for input prefixing. For the version of π-calculus extended with the *mismatch* construction, complete proof systems can be obtained by adding a rule for mismatch to the proof systems for the π-calculus proper.

1 Introduction

The π-calculus [MPW92] is a special instance of general message-passing process calculi. It is special in that the messages allowed to be passed between processes are *port names*. On one hand this gives the calculus the power to describe *mobile processes* where the communication topology may change dynamically, which is beyond the conventional message-passing process algebras such as value-passing *CCS*. On the other hand, it offers the opportunity to develop a simpler mathmetical theory for the calculus than general message-passing calculi, because the message domain is now just a plain set of *names* upon which tests for equalities or inequalities are the only allowed operations.

The aim of this paper is to provide a symbolic semantic theory as well as sound and complete proof systems for the π-calculus. In the original π-calculus paper [MPW92] *bisimulation congruence* $\sim$ is defined in two steps: first, *ground* bisimulation $\dot{\sim}$ is defined analogously to strong bisimulation in *CCS* [Mil89]. $t \sim u$ is then defined by $t\sigma \dot{\sim} u\sigma$ for every substitution σ. [MPW92] only gave an axiomatisation for the *late* version of ground bisimulation. Subsequently, efforts have been made to formulate complete proof systems for other equivalences for the calculus: [PS93] and [BD94] for both early and late bisimulation congruences, [Hen91] and [BD92] for *testing* equivalence. But all of

these axiomatisations are *not* for *the* π-calculus as proposed in [MPW92, Mil91]; they all require an extension to the calculus: the *mismatch* construction. The only exception is [San93] but the equivalence relation considered there is *open bisimulation* which is strictly finer than late (and hence early) bisimulation.

In this paper we shall first present proof systems for both late and early bisimulations for the π-calculus proper, without mismatch; we then include the mismatch construction into the language and extend our proof systems accordingly with a single inference rule for it. These proof systems are specialisations of those for general message-passing process calculi [HL93]. They consist of a set of inference rules together with some standard equations. The judgements are of the form

$$C \rhd t = u$$

where t, u are terms in the language and C, called a *condition*, is a set of equality or inequality tests between names. It is important to note that C is *not* a construction in the π-calculus; it is a construction used in our meta language in order to reason about bisimulation equivalences for the calculus. To give a taste of the proof system, here is the inference rule for match:

$$\text{MATCH}\frac{C \cup \{x = y\} \rhd t = u \quad C \cup \{x \neq y\} \rhd \mathbf{0} = u}{C \rhd [x = y]t = u}$$

It involves a case analysis: if we can establish $t = u$ under the condition $x = y$, and $\mathbf{0} = u$ under the condition $x \neq y$, then we can conclude $[x = y]t = u$. Here it can be seen clearly how the inequalities in the condition help us to characterise constructions in the calculus.

The proofs of the completeness results for these inference systems rely on the notion of *symbolic bisimulations* [HL92, HL93]. In [MPW92] a general notion of bisimulation, $\sim^D$, called *distinction indexed bisimulation*, is introduced, where the index D is a set of inequations on names such that $t \sim^D u$ *iff* $t\sigma \dot{\sim} u\sigma$ for every substitution σ that satisfies D. It is easy to see that $\sim$ and $\dot{\sim}$ are two extreme cases of $\sim^D$: $t \sim u$ *iff* $t \sim^{\emptyset} u$ while $t \dot{\sim} u$ *iff* $t \sim^{D(fn(t,u))} u$ where $D(fn(t,u)) = \{\, x \neq y \mid x, y \in fn(t,u) \,\}$ and $fn(t,u)$ is the set of free names in t and u. In the setting of π-calculus symbolic bisimulation is a mild generalisation of distinction indexed bisimulation, it is indexed by conditions consisting of name inequations as well as name equations. This generalisation makes it possible to give a direct definition of symbolic bisimulation (in terms of *symbolic transitional semantics*) instead of as substitution closure on top of ground bisimulation, which is itself of interest.

The definition of symbolic bisimulation for general message-passing calculi in [HL92, HL93] involves the phrase "... there exists a partition ...". This is because the languages of message and boolean are taken as parameters to the process language considered there. In the case of the π-calculus, however, the language for the message domain is known: it is simply a plain set of *port names*. To specialise the theory of symbolic bisimulation for general message-passing calculi to the π-calculus, we first develop a simple theory of *conditions* which are sets of equalities and inequalities over names. A *maximally consistent* condition is a *complete*, or *saturated*, condition, in the sense that adding to it anything

not implied by it will result in inconsistency. Given a condition the set of its maximally consistent extensions on a finite set of names constitutes a boolean partition of the condition. Because of this we can use maximally consistent extensions instead of the phrase "... there exists a partition ..." in the definition of symbolic bisimulation. Another important property of maximal consistency is that it characterises substitutions upto an injective substitution. That is, two substitutions satisfying the same maximally consistent condition differ only by an injective substitution. As ground bisimulation in the π-calculus is preserved by injective substitutions, the *finite* set of maximally consistent conditions on $fn(t,u)$ captures *all* the substitutions needed to close up $t \dot{\sim} u$ in order to get $t \sim u$.

The rest of the paper is organised as follows: The calculus and its semantics are introduced in the next section. The inference system is presented in Section 3, along with the completeness proof. Section 4 discusses extensions to the calculus. Section 5 demonstrates how the theory developed for the late equivalence in the previous sections can be carried over to the early case. The paper concluded with Section 6 where the relation with other work are also discussed.

2 The π-Calculus And Bisimulations

2.1 The Language And Operational Semantics

To give the syntax for the π-calculus we first presuppose a countably infinite set $\mathcal{N}$ of *names*, ranging over by $a,\ b,\ x,\ y,\ \ldots$. The language of π-calculus can then be given by the following BNF grammar

$$\begin{array}{rcl} t & ::= & \mathbf{0} \mid \alpha.t \mid t+t \mid [x=y]t \mid (x)t \mid t \mid t \mid A(y_1,...,y_n) \\ \alpha & ::= & \tau \mid a(x) \mid \overline{a}x \end{array}$$

Most of these operators are from *CCS* [Mil89]: $\mathbf{0}$ is a process capable of doing no action, $\alpha.t$ is action prefixing, $+$ is non-deterministic choice, $\mid$ is parallel composition, $(x)t$ is scope restriction. The *match* construction $[x=y]t$ allows the comparison of two names: if x and y are the same name then the process will behave like t; otherwise it will have no action. Each *identifier* A has a defining equation $A(x_1,...,x_n) \stackrel{def}{=} t$ with $fn(t) \subseteq \{x_1,...,x_n\}$.

As in *CCS* τ represents communication. An input-prefixed process $a(x).t$ can receive a name along the port a then behave like t with the received name in place of x. An output-prefixed process $\overline{a}x.t$ can emit the name x along a and continue like t.

In $a(x).t$ and $(x)t$ x is a bound name with scope t. We use $bn(t)$ and $fn(t)$ for the set of bound and free names in t, respectively. Bound names induce the notion of α-equivalence as usual. In the sequel we will not distinguish between α-equivalent terms and will use $\equiv$ for both syntactical equality and α-equivalence.

Substitutions, ranged over by σ and δ, are mappings from $\mathcal{N}$ to $\mathcal{N}$. $[\overline{y}/\overline{x}]$ is the substitution sending $\overline{x}$ to $\overline{y}$ and is identity otherwise. Substitutions are postfix operators, and have higher precedence than the operators in the language. If $\sigma = [\overline{y}/\overline{x}]$ then $n(\sigma) =$

$$\text{pre}\frac{}{\alpha.t \xrightarrow{\alpha} t} \qquad \text{match}\frac{t \xrightarrow{\alpha} t'}{[x=x]t \xrightarrow{\alpha} t'}$$

$$\text{sum}\frac{t \xrightarrow{\alpha} t'}{t+u \xrightarrow{\alpha} t'} \qquad \text{par}\frac{t \xrightarrow{\alpha} t'}{t \mid u \xrightarrow{\alpha} t' \mid u}\ bn(\alpha) \cap fn(u) = \emptyset$$

$$\text{res}\frac{t \xrightarrow{\alpha} t'}{(x)t \xrightarrow{\alpha} (x)t'}\ x \notin n(\alpha) \qquad \text{com}\frac{t \xrightarrow{a(x)} t' \quad u \xrightarrow{\overline{a}y} u'}{t \mid u \xrightarrow{\tau} t'[y/x] \mid u'}$$

$$\text{open}\frac{t \xrightarrow{\overline{a}x} t'}{(x)t \xrightarrow{\overline{a}(x)} t'}\ a \neq x \qquad \text{close}\frac{t \xrightarrow{a(x)} t' \quad u \xrightarrow{\overline{a}(x)} u'}{t \mid u \xrightarrow{\tau} (x)(t' \mid u')}$$

$$\text{id}\frac{t[y_1,...,y_n/x_1,...,x_n] \xrightarrow{\alpha} t'}{A(y_1,...,y_n) \xrightarrow{\alpha} t'}\ A(x_1,...,x_n) \stackrel{def}{=} t$$

Figure 1: π-Calculus Transitional Semantics

$\{\overline{y}\} \cup \{\overline{x}\}$. So for any $x \notin n(\sigma)$, $y\sigma = x$ *iff* $y = x$. Substitutions on actions are defined as usual.

The operational semantics of the language is reported in Figure 1, where a transition is of the form $t \xrightarrow{\alpha} u$ with α ranging over four kinds of actions: τ, $a(x)$, $\overline{a}x$ and $\overline{a}(x)$. We have omitted the symmetric rules for sum and par. Transitions are defined upto α-equivalence, i.e. α-equivalent terms are deemed to have the same transitions. Bound names for actions are defined by: $bn(a(x)) = bn(\overline{a}(x)) = \{x\}$ and $bn(\tau) = bn(\overline{a}x) = \emptyset$.

For the bulk of this paper we will concentrate on *late* bisimulation, and we will sketch in a later section how these results can be carried over to *early* bisimulation in a systematic manner. So until Section 5, the word "bisimulation" means "late bisimulation".

Definition 2.1 A symmetric relation R is a ground bisimulation if $(p,q) \in R$ implies:

- whenever $p \xrightarrow{a(x)} p'$ with $x \notin fn(p,q)$ then $q \xrightarrow{a(x)} q'$ for some q' and for any y $(p'[y/x], q'[y/x]) \in R$.
- whenever $p \xrightarrow{\alpha} p'$ for any other action α with $bn(\alpha) \cap fn(p,q) = \emptyset$ then $q \xrightarrow{\alpha} q'$ for some q' and $(p',q') \in R$.

Write $p \dot{\sim} q$ if there exists a ground bisimulation R s.t. $(p,q) \in R$.
t is bisimular to u, written $t \sim u$, if $t\sigma \dot{\sim} u\sigma$ for any substitution σ. □

Ground bisimulation is preserved by injective substitutions [MPW92]:

Lemma 2.2 *if* $t \dot{\sim} u$ *and* σ *is injective on* $fn(t,u)$ *then* $t\sigma \dot{\sim} u\sigma$

$$\text{Pre}\ \frac{}{\alpha.t \xrightarrow{true,\alpha} t} \qquad \text{Match}\ \frac{t \xrightarrow{M,\alpha} t'}{[x=y]t \xrightarrow{ML,\alpha} t'}\ L = \begin{cases} x=y & \text{if } x \neq y \\ \emptyset & \text{otherwise} \end{cases}$$

$$\text{Sum}\ \frac{t \xrightarrow{M,\alpha} t'}{t+u \xrightarrow{M,\alpha} t'} \qquad \text{Par}\ \frac{t \xrightarrow{M,\alpha} t'}{t \mid u \xrightarrow{M,\alpha} t' \mid u}\ bn(\alpha) \cap fn(u) = \emptyset$$

$$\text{Res}\ \frac{t \xrightarrow{M,\alpha} t'}{(x)t \xrightarrow{M,\alpha} (x)t'}\ x \notin n(M,\alpha) \qquad \text{Com}\ \frac{t \xrightarrow{M,a(x)} t' \quad u \xrightarrow{N,\bar{b}y} u'}{t \mid u \xrightarrow{MNL,\tau} t'[y/x] \mid u'}\ L = \begin{cases} a=b & \text{if } a \neq b \\ \emptyset & \text{otherwise} \end{cases}$$

$$\text{Open}\ \frac{t \xrightarrow{M,\bar{a}x} t'}{(x)t \xrightarrow{M,\bar{a}(x)} t'}\ x \notin n(M,a) \qquad \text{Id}\ \frac{t[y_1,...,y_n/x_1,...,x_n] \xrightarrow{M,\alpha} t'}{A(y_1,...,y_n) \xrightarrow{M,\alpha} t'}\ A(x_1,...,x_n) \stackrel{def}{=} t$$

$$\text{Close}\ \frac{t \xrightarrow{M,a(x)} t' \quad u \xrightarrow{N,\bar{b}(x)} u'}{t \mid u \xrightarrow{MNL,\tau} (x)(t' \mid u')}\ L = \begin{cases} a=b & \text{if } a \neq b \\ \emptyset & \text{otherwise} \end{cases}$$

Figure 2: π-Calculus Symbolic Transitional Semantics

The transitions defined in Figure 1 are *concrete* in the sense that they will always be fired regardless the context in which terms are placed. In our work on general message-passing processes [HL92, HL93] we use a more abstract form of transitions which we called *symbolic transitions*. A symbolic transition takes the form $t \xrightarrow{b,\alpha} u$, where b is a boolean condition. Intuitively b represents the environments under which action α can actually be fired from t. In the setting of the π-calculus b will be a set of *matches*, i.e. equality tests on names. This kind of transition has also been used in the work of open bisimulation by Sangiorgi [San93].

The symbolic transitional semantics of the π-calculus is given in Figure 2, where we use $M,\ N,\ L$ to range over *matches*, namely sets of equality tests on names. For notational convenience we write MN for the union of M and N. Also the symmetric rules for Sum and Par have been omitted.

Lemma 2.3 *If* $t \xrightarrow{M,\alpha} u$ *then* $n(M) \subseteq fn(t)$.

Symbolic bisimulation will be defined in terms of symbolic transitional semantics, but before doing so we need to develop a simple theory of *conditions* on the name set $\mathcal{N}$.

2.2 Conditions

Conditions, ranged over by $C,\ D$, are finite sets of equality or inequality tests on names. We will write $n(C)$ for the set of names appearing in C. We say C is a condition on a name set V if $n(C) \subseteq V$. The empty condition will sometimes be denoted by *true*.

Note that we have been careful in distinguishing between conditions and matches: matches consist only of name equations and is used in the symbolic operational semantics, while conditions may contain *inequations* and will be used in our meta-theory about the π-calculus, namely in the definition of symbolic bisimulations and in the formulation of inference rules. On the other hand, matches are special conditions, so the notations developed for conditions in this subsection apply to matches as well.

A substitution σ satisfies a condition C, written $\sigma \models C$, if $x\sigma = y\sigma$ for any $x = y \in C$ and $x\sigma \neq y\sigma$ for any $x \neq y \in C$. We write $C \Rightarrow D$ to mean that $\sigma \models C$ implies $\sigma \models D$ for any substitution σ.

The elements in a condition are treated as conjuncts. We avoid introducing disjunction into the theory of conditions. The major advantage is that the relation $C \Rightarrow D$ can be tested in linear time w.r.t the size of C and D:

Proposition 2.4 *The relation $C \Rightarrow D$ is linear time decidable.*

Proof: We first generate equivalence classes from the equalities in C, along with a list of pairs of representatives of the equivalence classes such that the pair of representatives of x and y is in the list if and only if $x \neq y$ is in C. This process takes time linear to the size of C. Then for each element e of D if e is an equality $a = b$ we check if $a,\ b$ is in the same equivalence class; if e is an inequality $a \neq b$ we check if the pair consisting of their representatives are in the list. □

Two substitutions σ and σ' are equal on V, written $\sigma =^V \sigma'$, if $x\sigma = x\sigma'$ for all $x \in V$. Two substitutions σ and σ' are *elementarily equivalent* on V if for any $x, y \in V$ $\sigma \models x = y$ if and only if $\sigma' \models x = y$. Clearly equal substitutions are elementarily equivalent. On the other hand elementarily equivalent substitutions are not necessary equal, though they do not differ much, as the following lemma reveals.

Lemma 2.5 *Suppose σ and σ' are substitutions on V. If they are elementarily equivalent on V, then there exists an injective substitution δ such that $\sigma =^V \sigma'\delta$.*

Proof: Let δ be the substitution such that $(x\sigma)\delta = x\sigma'$ and $(x\sigma')\delta = x\sigma$ for all $x \in V$, and is identity otherwise. δ is well defined because σ and σ' are elementarily equivalent on V. Furthermore it is injective and satisfies $\sigma =^V \sigma'\delta$. □

A condition C is *consistent* if there are no $x,\ y \in \mathcal{N}$ such that $C \Rightarrow x = y$ and $C \Rightarrow x \neq y$. C is *maximally consistent* on $V \subset \mathcal{N}$ if for any $x,\ y \in V$ either $C \Rightarrow x = y$ or $C \Rightarrow x \neq y$.

C' is a *maximally consistent extension* of C on V, written $C' \in MCE_V(C)$, if $C \subseteq C'$ and C' is maximally consistent on V. The set of maximally consistent extensions of a given condition on a finite set of names V is finite. We will abbreviate $MCE_V(true)$ as MC_V.

Given a condition C, define $E_V(C) = \{\, x = y \mid x, y \in V,\ C \not\Rightarrow x = y \ \text{ and } \ C \not\Rightarrow x \neq y \,\}$, and for any $I \subseteq E_V(C)$ define $I^* = \{\, x = y \mid x, y \in V,\ I \Rightarrow x = y \,\}$ and $\overline{I} = \{\, x \neq y \mid x = y \in E_V(C) - I^* \,\}$.

Clearly C is maximally consistent on V if and only if $E_V(C) = \emptyset$.

Lemma 2.6 *$D \in MCE_V(C)$ iff $D = C \cup I \cup \overline{I}$ for some $I \subseteq E_V(C)$.*

Proof: For any $I \subseteq E_V(C)$, by construction $C \cup I \cup \overline{I}$ is a maximally consistent extension of C on V. On the other hand, let $D = C \cup I \cup \overline{I}$ for some $I \subseteq E_V(C)$. For any $x,\ y \in V$, there are only three possibilities:

1. $C \Rightarrow x = y$ or $C \Rightarrow x \neq y$;
2. $x = y \in I^*$;
3. $x \neq y \in \overline{I}$.

In each case either $D \Rightarrow x = y$ or $D \Rightarrow x \neq y$. □

Corollary 2.7 $\bigvee MCE_V(C) = C$

Proof:

$$\begin{aligned} \bigvee MCE_V(C) &= \bigvee\{\, C \cup I \cup \overline{I} \mid I \subseteq E_V(C) \,\} \\ &= C \wedge \bigvee\{\, I \cup \overline{I} \mid I \subseteq E_V(C) \,\} \\ &= C \wedge \mathit{true} \\ &= C \end{aligned}$$

□

Corollary 2.7 shows that the set of all consistent extensions of a condition (on a given name set) constitutes a particular *partition*, or *decomposition* of the condition.

Proposition 2.8 *C is maximally consistent on V if and only if all substitutions satisfying C are elementarily equivalent on V.*

Proof: The "only if" part is trivial. For the "if" part, suppose C is not maximally consistent on V. Then there exist $x,\ y \in V$ such that neither $C \Rightarrow x = y$ nor $C \Rightarrow x \neq y$. So there exists a substitution σ such that $\sigma \models C$ and $\sigma \models x \neq y$. Let σ' be the substitution which is the same as σ except that it sends x to $y\sigma$. Then we still have $\sigma' \models C$. But σ' is not elementarily equivalent to σ on V because $\sigma' \models x = y$. A contradiction. □

Corollary 2.9 *Suppose C is maximally consistent on V. If σ and σ' both satisfy C, then $\sigma =^V \sigma'\delta$ for some injective substitution δ.*

In other words, all substitutions satisfying a maximally consistent condition on a given name set are *isomorphic* on that name set. This shows the importance of the notion of maximal consistence: it captures substitutions upto isomorphism. Recalling Lemma 2.2 that isomorphic substitutions make no difference as far as bisimulation is concerned, the set of maximally consistent conditions on $fn(t,u)$ characterises all possible substitutions that may affect the bisimilarity between t and u. Although $t \sim u$ is defined as the closure of $t \dot{\sim} u$ over *all* substitutions, only *finite* number of substitutions need to be checked, one for each maximally consistent condition on $fn(t,u)$.

The notion of maximally consistent condition was first employed in [PS93] to define head normal forms. In the current work we use maximal consistency only in the meta-language. The technique of maximally consistent extension can be traced back to the author's early work on algebraic specifications ([Lin87]).

2.3 Symbolic Bisimulations

Now we are ready to give the definition of symbolic bisimulation.

We write $\alpha =^C \beta$ to mean

$$\begin{array}{l}\text{if } \alpha \equiv \tau \text{ then } \beta \equiv \tau \\ \text{if } \alpha \equiv \overline{a}x \text{ then } \beta \equiv \overline{b}y \text{ and } C \Rightarrow a = b,\ C \Rightarrow x = y \\ \text{if } \alpha \equiv \overline{a}(x) \text{ then } \beta \equiv \overline{b}(x) \text{ and } C \Rightarrow a = b \\ \text{if } \alpha \equiv a(x) \text{ then } \beta \equiv b(x) \text{ and } C \Rightarrow a = b\end{array}$$

Definition 2.10 A condition indexed family of symmetric relations $\mathcal{S} = \{S^C\}$ is a late symbolic bisimulation if $(t,u) \in S^C$ implies

whenever $t \xrightarrow{M,\alpha} t'$ with $bn(\alpha) \cap fn(t,u,C) = \emptyset$, then for each $C' \in MCE_{fn(t,u)}(C \cup M)$ there is a $u \xrightarrow{N,\beta} u'$ such that $C' \Rightarrow N$, $\alpha =^{C'} \beta$, and $(t',u') \in S^{C''}$, where

$$C'' = \begin{cases} C' \cup \{\, x \neq y \mid y \in fn(\alpha.t', \beta.u') \,\} & \textit{if } \alpha \equiv \overline{a}(x) \\ C' & \textit{otherwise} \end{cases}$$

Let $\sim_L$ be the (point-wisely) largest late symbolic bisimulation. □

Note that, in the clause for bound output, inequalities has been added to the indexing condition to keep the restricted name different from any other known names.

Symbolic bisimulation is a generalisation of *distinction indexed bisimulation*, $\sim^D$, as proposed in [MPW92], the main difference being that distinctions contain only inequalities, while in symbolic bisimulation we allow equalities as well as inequalities to appear in the indexing conditions. The notion of symbolic bisimulation was proposed in [HL92, HL93] as an efficient characterisation of conventional notion of bisimulation which, in the setting of message-passing process calculi, is inherently infinite. The current definition is a specialisation of this general notion to the π-calculus. In doing so we have taken

advantage of the particular features presented in the π-calculus: In the above definition, instead of using the phrase "... there exists a partition ..." as in [HL92, HL93], we use the set of maximally consistent extensions which, as shown in Corollary 2.7, constitute a particular partition. It is much easier to work with maximally consistent extensions than an arbitrary partition, as we will see later in the proofs of the completeness results.

Proposition 2.11 *$\sim_L$ is an indexed family of equivalence relations.*

The following proposition reveals the relationship between symbolic bisimulation and the conventional notions of bisimulation equivalence for the π-calculus. where $\dot{\sim}$, $\sim$, and $\sim^D$ are *strong ground bisimulation*, *strong bisimulation* and *distinction indexed bisimulation*, respectively, as defined in [MPW92]. The proof of the proposition requires a technical lemma relating concrete and symbolic transitions:

Lemma 2.12 *1. If $t\sigma \xrightarrow{\alpha} p$ where $bn(\alpha) \cap (fn(t) \cup n(\sigma)) = \emptyset$, then there exist M, β, t' s.t. $\sigma \models M$, $\alpha = \beta\sigma$, $p \equiv t'\sigma$ and $t \xrightarrow{M,\beta} t'$.*

2. If $t \xrightarrow{M,\alpha} t'$ then for any $\sigma \models M$ with $bn(\alpha) \cap (fn(t) \cup n(\sigma)) = \emptyset$, $t\sigma \xrightarrow{\alpha\sigma} p \equiv t'\sigma$.

Proposition 2.13 *1. $t \sim_L^C u$ iff $t\sigma \dot{\sim} u\sigma$ for any $\sigma \models C$.*

2. $t \sim_L^{D(fn(t,u))} u$ iff $t \sim^{D(fn(t,u))} u$, where $D(V) =_{def} \{ x \neq y \mid x, y \in V \text{ and } x \not\equiv y \}$.

3. $t \sim_L^{true} u$ iff $t \sim u$.

Proof: *2* and *3* are direct consequences of *1*. The "only if" part of *1* can be established by defining

$$R = \{ (t\sigma, u\sigma) \mid t \sim_L^C u \text{ for some } C \in MC_{fn(t,u)} \text{ and } \sigma \models C \}$$

and showing that R is a ground bisimulation. Here we will concentrate on the "if" part.

Define

$$S^C = \{ (t, u) \mid n(C) \subseteq fn(t, u),\ C \in MC_{fn(t,u)},\ t\sigma \dot{\sim} u\sigma \text{ for any } \sigma \models C \}$$

and $\mathcal{S} = \{S^C\}$. We show $\mathcal{S}$ is a symbolic bisimulation.

Let $(t, u) \in S^C$ and $t \xrightarrow{M,\alpha} t'$ with $bn(\alpha) \cap fn(t, u) = \emptyset$. Let $C' \in MCE_{fn(t,u)}(C \cup M)$. Take a σ such that $bn(\alpha) \cap n(\sigma) = \emptyset$ and $\sigma \models C'$. Then $\sigma \models C$, $\sigma \models M$. By Lemma 2.12 $t\sigma \xrightarrow{\alpha\sigma} p \equiv t'\sigma$. Since $t\sigma \dot{\sim} u\sigma$, $u\sigma$ must have a matching transition. There are four cases to consider, according to four different types of actions. The most interesting case is when α is a bound output, *i.e.* $\alpha \equiv \overline{a}(x)$.

In this case $u\sigma \xrightarrow{\overline{a\sigma}(x)} q \dot{\sim} p$. By Lemma 2.12 there exist N, b, u' s.t. $\sigma \models N$, $a\sigma = b\sigma$, $q \equiv u'\sigma$ and $u \xrightarrow{N,\overline{b}(x)} u'$. From $C' \in MC_{fn(t,u)}$, $\sigma \models C'$ and $a\sigma = b\sigma$ it follows $\overline{a}(x) =^{C'} \overline{b}(x)$. We need to show $(t', u') \in S^{C''}$ where $C'' = C' \cup \{ x \neq y \mid y \in fn(\overline{a}(x).t', \overline{b}(x).u') \}$.

From $x \notin n(\sigma)$ and $x \notin fn(t,u)$, $\sigma \models x \neq y$ for any $y \in fn(\overline{a}(x).t', \overline{b}(x).u') \subseteq fn(t,u)$. Hence $\sigma \models C''$. Since $fn(t',u') \subseteq fn(t,u) \cup \{x\}$ and $C' \in MC_{fn(t,u)}$, $C'' \in MC_{fn(t',u')}$. We have $t'\sigma \equiv p \dot{\sim} q \equiv u'\sigma$, so by Corollary 2.9 and Lemma 2.2 $t'\sigma' \dot{\sim} u'\sigma'$ for any $\sigma' \models C''$. Therefore $(t',u') \in S^{C''}$ by the definition of $\mathcal{S}$. □

3 The Inference System

This section is devoted to formulating a proof system for late symbolic bisimulation and proving its soundness and completeness. It is well-known that such a proof system does not exist for the full calculus, so we shall restrict ourselves to the finite fragment, *i.t.* leave out recursive definitions.

To give a proof system for finite π-calculus, we first concentrate on a sublanguage without the parallel composition operator $|$. In the next section we will see that this operator can be axiomatised easily.

The inference system for late symbolic bisimulation consists of a set of inference rules in Figure 3 (we have omitted the obvious rules for equivalence relations), together with the standard equations for choice (S1 – S4) and restriction (R1 – R5) in Figure 4. It inherits many features from the proof systems for general message-passing processes presented in [HL93]. The judgements are of the form

$$C \rhd t = u$$

where C is a condition. An important difference is that the inference system of [HL93] relies on some "oracle" to answer questions concerning data, while in the current setting we have a *decidable* theory for the value domain which is simply the set of port names. Therefore in the current inference system questions about names are treated as side conditions to the inference rules.

As we are working modulo α-equivalence, we also assume the following rule

$$\text{ALPHA}\frac{}{true \rhd t = u} \; t \text{ and } u \text{ are } \alpha - \text{equivalent}$$

We write $\vdash C \rhd t = u$ to mean $C \rhd t = u$ can be derived from this inference system.

Proposition 3.1 *1. If $C \Rightarrow M$ and $\vdash C \rhd t = u$ then $\vdash C \rhd Mt = u$.*

2. If $C \cup M \Rightarrow false$ then $\vdash C \rhd Mt = \mathbf{0}$.

Proof: Both can be easily proved by induction on the length of M, using MATCH, CONSEQ and ABSURD. □

The rule PARTITION permits a case analysis on the name space represented by a condition: To see if $t = u$ holds over C, we can decompose C into $C \cup \{x = y\}$ and $C \cup \{x \neq y\}$,

$$\text{AXIOM}\frac{}{true \rhd t = u}\ t = u \text{ is an axiom instance}$$

$$\text{CHOICE}\frac{C \rhd t_i = u_i}{C \rhd t_1 + t_2 = u_1 + u_2}$$

$$\text{L-INPUT}\frac{C \rhd t = u}{C \rhd a(x).t = b(x).u}\ C \Rightarrow a = b, \quad x \notin fn(C)$$

$$\text{OUTPUT}\frac{C \rhd t = u}{C \rhd \overline{a}x.t = \overline{b}y.u}\ C \Rightarrow a = b,\ C \Rightarrow x = y$$

$$\text{TAU}\frac{C \rhd t = u}{C \rhd \tau.t = \tau.u}$$

$$\text{MATCH}\frac{C \cup \{x = y\} \rhd t = u \quad C \cup \{x \neq y\} \rhd \mathbf{0} = u}{C \rhd [x = y]t = u}$$

$$\text{RES}\frac{C \cup \{\, x \neq y \mid y \in fn((x)t, (x)u)\,\} \rhd t = u}{C \rhd (x)t = (x)u}\ x \notin n(C)$$

$$\text{PARTITION}\frac{C \cup \{x = y\} \rhd t = u \quad C \cup \{x \neq y\} \rhd t = u}{C \rhd t = u}$$

$$\text{CONSEQ}\frac{C \rhd t = u}{C' \rhd t = u}\ C' \Rightarrow C$$

$$\text{ABSURD}\frac{}{false \rhd t = u}$$

Figure 3: The Inference Rules for Late Symbolic Bisimulation

and exam each separately. In fact, this rule can be generalised to allow arbitrary decompositions. The following proposition gives the case for a particular decomposition, *i.e.* the decomposition of a condition into its maximally consistent extensions (Corollary 2.7):

Proposition 3.2 *If* $\vdash D \rhd t = u$ *for each* $D \in MCE_V(C)$ *then* $\vdash C \rhd t = u$.

Proof: Suppose $\vdash D \rhd t = u$ for each $D \in MCE_V(C)$. By Lemma 2.6 $D \in MCE_V(C)$ *iff* $D = C \cup I \cup \overline{I}$ for some $I \subseteq E_V(C)$. Apply induction on the cardinality of $E_V(C)$.

S1	$X + \mathbf{0} = X$	R1	$(x)\mathbf{0} = \mathbf{0}$
S2	$X + X = X$	R2	$(x)\alpha.X = \alpha.(x)X$ if $x \notin n(\alpha)$
S3	$X + Y = Y + X$	R3	$(x)\alpha.X = \mathbf{0}$ if x is the port of α
S4	$(X + Y) + Z = X + (Y + Z)$	R4	$(x)(y)X = (y)(x)X$
		R5	$(x)(X + Y) = (x)X + (x)Y$

Figure 4: The Axioms for Choice And Restriction

If $E_V(C)$ is empty then C is the only maximally consistent extension of itself, and the result is trivial.

Otherwise assume $E_V(C)$ has $n+1$ elements and let $x = y \in E_V(C)$. We have

$$MCE_V(C) = MCE_V(C \cup \{x = y\}) \cup MCE_V(C \cup \{x \neq y\})$$

and

$$E_V(C \cup \{x = y\}) \subseteq E_V(C) - \{x = y\}$$
$$E_V(C \cup \{x \neq y\}) \subseteq E_V(C) - \{x = y\}$$

So by induction $\vdash C \cup \{x = y\} \rhd t = u$, $\vdash C \cup \{x \neq y\} \rhd t = u$. By PARTITION $\vdash C \rhd t = u$. □

The following proposition summarises the interaction between the restriction and match operators.

Proposition 3.3 *1.* $\vdash (x)[x = x]t = (x)t$.

2. $\vdash (x)[x = y]t = \mathbf{0}$.

3. $\vdash (x)[y = z]t = [y = z](x)t$.

Proof: Easy applications of RES and MATCH. *2* also uses R1. □

The soundness of the inference system is not difficult to see:

Theorem 3.4 *(Soundness of* $\vdash$*) If* $\vdash C \rhd t = u$ *then* $t \sim_L^C u$.

The rest of this section is devoted to the proof of completeness result for $\vdash$.

The *height* of a term t is defined inductively thus

- $| \mathbf{0} | = 0$
- $| t + u | = max\{| t |, | u |\}$

- $\mid [x=y]t \mid = \mid t \mid$
- $\mid (x)t \mid = \mid t \mid$
- $\mid \alpha.t \mid = 1 + \mid t \mid$

If $a \neq x$ then we abbreviate $(x)\overline{a}x.t$ as $\overline{a}(x).t$. $\overline{a}(x)$ is a derived action and is called *bound output*.

Proposition 3.5 *Suppose* $C \Rightarrow a = b$, $x \notin n(C)$. *If* $\vdash C \cup \{ x \neq y \mid y \in fn(\overline{a}(x).t, \overline{b}(x).u) \} \rhd t = u$ *then* $\vdash C \rhd \overline{a}(x).t = \overline{b}(x).u$.

Proof: Since $\vdash C \cup \{ x \neq y \mid y \in fn(\overline{a}(x).t, \overline{b}(x).u) \} \rhd t = u$ and $C \Rightarrow a = b$, by OUTPUT we get $\vdash C \cup \{ x \neq y \mid y \in fn(\overline{a}(x).t, \overline{b}(x).u) \} \rhd \overline{a}x.t = \overline{b}x.u$. From this and $x \notin n(C)$ an application of RES gives the required $\vdash C \rhd \overline{a}(x).t = \overline{b}(x).u$. □

A term is restriction-free if, using the above abbreviation, it does not explicitly contain any occurrences of the restriction operator.

Lemma 3.6 *For any term* t *there is a restriction-free term* t' *such that* $\vdash t = t'$ *and* $\mid t' \mid \leq \mid t \mid$.

Proof: Using Axioms R1 – R5 and Proposition 3.3, we can push each restriction inwards, until it either disappears or gives rise to a bound output. □

A restriction-free term is a *standard* form if it has the form $\sum_i M_i \alpha_i.t_i$ and each t_i is a standard form.

Lemma 3.7 *For any term* t *there is a standard term* t' *such that* $\vdash t = t'$ *and* $\mid t' \mid \leq \mid t \mid$.

Proof: By Lemma 3.6 we may assume t is restriction-free. The proof is by induction on the structure of t. □

Theorem 3.8 *(Completeness of* $\vdash$*) If* $t \sim_L^C u$ *then* $\vdash C \rhd t = u$.

Proof: By Lemma 3.7 we may assume t, u are standard forms: $t \equiv \sum_i M_i \alpha_i.t_i$, $u \equiv \sum_j N_j \beta_j.u_j$. We may further assume that bound actions in α_i, β_j use the same bound name $z \notin fn(t, u, C)$. The proof is by induction on the joint height of t and u. By Proposition 3.2 we need only to show $\vdash D \rhd t = u$ for each $D \in MCE_{fn(t,u,C)}(C)$, and for this it is sufficient to show $\vdash D \rhd t = t + N_j \beta_j.u_j$ for each j.

If $D \not\Rightarrow N_j$ then, since D is maximally consistent, $D \cup N_j \Rightarrow false$. So by Proposition 3.1 $\vdash D \rhd N_j\beta_j.u_j = \mathbf{0}$. Hence $\vdash D \rhd t = t + N_j\beta_j.u_j$ by S1.

Now assume $D \Rightarrow N_j$ (then $D \in MCE_{fn(t,u)}(C \cup N_j)$), and consider four cases according to the type of β_j.

1. $\beta_j \equiv \tau$. Then $u \xrightarrow{N_j,\tau} u_j$. Since $t \sim_L^C u$ and $D \in MCE_{fn(t,u)}(C \cup N_j)$, $t \xrightarrow{M_i,\tau} t_i$ for some i s.t. $D \Rightarrow M_i$ and $t_i \sim_L^D u_j$. By induction $\vdash D \rhd t_i = u_j$. By TAU $\vdash D \rhd \tau.t_i = \tau.u_j$. By Proposition 3.1, $\vdash D \rhd M_i\tau.t_i = N_j\tau.u_j$. Finally by S2 $\vdash D \rhd t = t + N_j\tau.u_j$.

2. $\beta_j \equiv \overline{a}x$. Then $u \xrightarrow{N_j,\overline{a}x} u_j$, so $t \xrightarrow{M_i,\overline{b}y} t_i$ for some i s.t. $D \Rightarrow M_i$, $\overline{a}x =^D \overline{b}y$ and $t_i \sim_L^D u_j$. By induction $\vdash D \rhd t_i = u_j$. By OUTPUT, $\vdash D \rhd \overline{b}y.t_i = \overline{a}x.u_j$. By Proposition 3.1, $\vdash D \rhd M_i\overline{b}y.t_i = N_j\overline{a}x.u_j$. Hence $\vdash D \rhd t = t + N_j\overline{a}x.u_j$ by S2.

3. $\beta_j \equiv a(z)$. Then $u \xrightarrow{N_j,a(z)} u_j$, so $t \xrightarrow{M_i,b(z)} t_i$ for some i s.t. $D \Rightarrow M_i$, $D \Rightarrow a = b$ and $t_i \sim_L^D u_j$. By induction $\vdash D \rhd t_i = u_j$. Since $z \notin n(D)$, by L-INPUT, $\vdash D \rhd b(z).t_i = a(z).u_j$. By Proposition 3.1 and Axiom S2, $\vdash D \rhd t = t + N_j a(z).u_j$.

4. $\beta_j \equiv \overline{a}(z)$. Then $u \xrightarrow{N_j,\overline{a}(z)} u_j$, so $t \xrightarrow{M_i,\overline{b}(z)} t_i$ for some i s.t. $D \Rightarrow M_i$, $D \Rightarrow a = b$ and $t_i \sim_L^{D \cup \{ z \neq y \mid y \in fn(\overline{b}(z).t_i, \overline{a}(z).u_j) \}} u_j$. By induction $\vdash D \cup \{ z \neq y \mid y \in fn(\overline{b}(z).t_i, \overline{a}(z).u_j) \} \rhd t_i = u_j$. Now $n(D) \subseteq fn(t,u,C)$ and $z \notin fn(t,u,C)$, hence $z \notin n(D)$, so by Proposition 3.5 $\vdash D \rhd \overline{b}(z).t_i = \overline{a}(z).u_j$. By Proposition 3.1 and Axiom S2, $\vdash D \rhd t = t + N_j\overline{a}(z).u_j$.

This completes the proof.

□

4 Other Operators

4.1 Parallel Composition

To deal with parallel composition all we need is a suitable form of expansion law which is presented in Figure 5. With this law it is standard that any term containing parallel operator can be reduced to a term without it, hence the normal form lemma still holds, as does the completeness theorem.

4.2 Mismatch

Mismatch, *i.e.* testing inequality between names, is not included in the original π-calculus. Some later publications, notably [Hen91], [BD92], [PS93] and [BD94], extended the calculus with mismatch in order to give axiomatisations for testing or bisimulation equivalences.

Let $t \equiv \sum_i M_i\alpha_i.t_i$ and $u \equiv \sum_j N_j\beta_j.u_j$ with $bn(\alpha_i) \cap fn(u) = bn(\beta_j) \cap fn(t) = \emptyset$. Then

$$t \mid u = \sum_i M_i\alpha_i.(t_i \mid u) \;+\; \sum_j N_j\beta_j.(t \mid u_j) \;+\; \sum_{\alpha_i \;opp\; \beta_j} M_iN_j[a_i = b_j]\tau.v_{ij}$$

where α_i *opp* β_j and v_{ij} are defined as follows

1. $\alpha_i \equiv a(x), \beta_j \equiv \bar{b}y$; then $v_{ij} \equiv t_i[y/x] \mid u_j$;
2. The converse of the above clause;
3. $\alpha_i \equiv a(x), \beta_j \equiv \bar{b}(y)$; then $v_{ij} \equiv (z)(t_i[z/x] \mid u_j[z/y])$ with $z \notin fn(t, u)$;
4. The converse of the above clause.

Figure 5: The Expansion Law

To include mismatch into the language we first extend the operational semantics by including the following two rules: "mismatch" and "Mismatch" in Figure 1 and Figure 2, respectively (now M ranges over conditions)

$$\text{mismatch}\frac{t \xrightarrow{\alpha} t'}{[x \neq y]t \xrightarrow{\alpha} t'}\; x \neq y \qquad\qquad \text{Mismatch}\frac{t \xrightarrow{M,\alpha} t'}{[x \neq y]t \xrightarrow{M[x\neq y],\alpha} t'}$$

The inference rule for mismatch is dual to that for match:

$$\text{MISMATCH}\frac{C \cup \{x \neq y\} \rhd t = u \quad C \cup \{x = y\} \rhd \mathbf{0} = u}{C \rhd [x \neq y]t = u}$$

With this rule Proposition 3.1 can be generalised to allow M to be an arbitrary condition, not just a match.

Now, in a normal form $\sum_i M_i\alpha_i.t_i$, M_i may contain inequality tests as well as equality tests. This does not affect the proofs of the normal form lemma and the completeness theorem.

5 The Early Case

If we change the clause for input transition in Definition 2.1 to

> whenever $p \xrightarrow{a(x)} p'$ with $x \notin fn(p, q)$ then for any y $q \xrightarrow{a(x)} q'$ for some q' and $(p'[y/x], q'[y/x]) \in R$.

then we get the early version of ground bisimulation, $\dot{\sim}_e$. Early bisimulation congruence $\sim_e$ is then defined as substitution closure in terms of $\dot{\sim}_e$, *i.e.* $t \sim_e u$ *iff* $t\sigma \;\dot{\sim}_e\; u\sigma$ for

any σ. Similarly for distinction indexed early bisimulation: $t \sim_e^D u$ *iff* $t\sigma \dot{\sim}_e u\sigma$ for any $\sigma \models D$.

Early symbolic bisimulation is obtained by separating the clause for input transition from other cases in Definition 2.10:

> whenever $t \xrightarrow{M,a(x)}_L t'$ with $x \notin fn(t,u,C)$, then for each $C' \in MCE_{fn(t,u)\cup\{x\}}(C \cup M)$ there is a $u \xrightarrow{N,b(x)}_L u'$ such that $C' \Rightarrow N,\ C' \models a = b$, and $(t',u') \in S^{C'}$

Let $\sim_E$ be the largest early symbolic bisimulation.

The only difference between early and late bisimulations is that the early version allows to partition the name space over the input name (thus makes it possible for an input transition of one process to be matched by more than one such transitions of the other process), while in the late version this is prohibited (thus guarantees each input transition of one process to be matched by a *single* input transition of the other).

With these definitions the early version of Proposition 2.13 still holds.

The proof system for early bisimulation can be obtained by replacing the L-INPUT rule in Figure 3 with the following one

$$\text{E-INPUT}\frac{C \rhd \sum_{i\in I} \tau.t_i = \sum_{j\in J} \tau.u_j}{C \rhd \sum_{i\in I} a_i(x).t_i = \sum_{j\in J} b_j(x).u_j} \quad \begin{array}{l} C \Rightarrow a_i = b_j,\ i \in I,\ j \in J \\ x \notin fn(C) \end{array}$$

Let us write $\vdash_E C \rhd t = u$ to mean $C \rhd t = u$ can be derived from the new proof system.

The E-INPUT rule has a generalised form:

Proposition 5.1 *Suppose* $x \notin fn(C),\ C \Rightarrow a_i = b_j,\ i \in I,\ j \in J$. *Then from*

$$\vdash_E C \rhd \sum_{i\in I} M_i\tau.t_i = \sum_{j\in J} N_j\tau.u_j$$

infer

$$\vdash_E C \rhd \sum_{i\in I} M_i a_i(x).t = \sum_{j\in J} N_j b_j(x).u$$

Proof: By Proposition 3.2, we only need to show $\vdash_E D \rhd \sum_{i\in I} M_i a_i(x).t = \sum_{j\in J} N_j b_j(x).u$ for each $D \in MCE_V(C)$ where V is the set of free names appearing in the conclusion. Since D is maximally consistent on V, for each i either $C \Rightarrow M_i$ or $C \cup M_i = false$. Hence by Proposition 3.1

$$\vdash_E D \rhd \sum_{i\in I} M_i\tau.t_i = \sum_{i\in I'} M_i\tau.t_i$$

$$\vdash_E D \rhd \sum_{i\in I} M_i a_i(x).t_i = \sum_{i\in I'} M_i a_i(x).t_i$$

where $I' = \{\, i \in I \mid D \Rightarrow M_i \,\}$. Similarly,

$$\vdash_E D \rhd \sum_{j\in J} N_j\tau.u_j = \sum_{j\in J'} N_j\tau.u_j$$

$$\vdash_E D \rhd \sum_{j\in J} N_j b_j(x).u_j = \sum_{j\in J'} N_j b_j(x).u_j$$

where $J' = \{\, j \in J \mid D \Rightarrow N_j \,\}$. Hence, from the assumption we can derive

$$\vdash_E C \rhd \sum_{i\in I'} M_i\tau.t_i = \sum_{j\in J'} N_j\tau.u_j$$

By E-INPUT

$$\vdash_E C \rhd \sum_{i\in I'} M_i a_i(x).t = \sum_{j\in J'} N_j b_j(x).u$$

Therefore

$$\vdash_E C \rhd \sum_{i\in I} M_i a_i(x).t = \sum_{j\in J} N_j b_j(x).u$$

□

We have the following counterpart of Theorems 3.4 and 3.8:

Theorem 5.2 *(Soundness and Completeness of $\vdash_E$) $t \sim_E^C u$ if and only if $\vdash_E C \rhd t = u$.*

Proof: The proof of the completeness result is very similar to that of Theorem 3.8. The only difference is concerning the input case. In this case we want to show

$$\vdash_E D \rhd t = t + N_j b(z).u_j$$

under the assumptions $D \in MCE_{fn(t,u,C)}(C)$ and $D \Rightarrow N_j$. Let $t' \equiv \sum_{i\in I'} M_i a_i(z).t_i$ and $t^\tau \equiv \sum_{i\in I'} M_i\tau.t_i$, where $I' = \{\, i \in I \mid \alpha_i \equiv a_i(z),\ D \Rightarrow a_i = b \,\}$. Then it is sufficient to show

$$\vdash_E D \rhd t' = t' + N_j b(z).u_j$$

which, by Proposition 5.1, can be reduced to

$$\vdash_E D \rhd t^\tau = t^\tau + N_j\tau.u_j$$

Now for each $D' \in MCE_{fn(t,u,C)\cup\{z\}}(D)$ we can prove

$$\vdash_E D' \rhd t^\tau = t^\tau + N_j\tau.u_j$$

in the same way as the τ case in the proof of Theorem 3.8. The proof is then completed by an application of Proposition 3.2. □

6 Conclusions and Related Work

We have introduced symbolic bisimulation equivalences for the π-calculus which subsume the conventional bisimulation equivalences as proposed in [MPW92]. The main advantage of such symbolic equivalences is that they can be defined (in term of symbolic transitional semantics) in a single step. Sound and complete proof systems for symbolic bisimulations have also been presented. A simple theory of *conditions* on names, which is linear time decidable, plays an important rôle in formulating the notion of symbolic bisimulation and associated proof systems.

As a further research topic we would like to extend the current framework to deal with weak bisimulation equivalences. Another interesting subject is the development of a proof tool for the π-calculus, based on syntactical manipulation, similar to VPAM (Value-passing Process Algebra Manipulator) for general message-passing calculi [Lin93].

Related Work. As mentioned in the introduction the current work inherits the main ideas from our previous papers on general message-passing process calculi [HL92, HL93]. As the process calculi considered there take the languages for data and boolean expressions as parameters, the completeness results in those papers are relative to data reasoning. Here, by exploiting the fact that in the π-calculus the message domain is the plain set of port names, we have developed a decidable theory of conditions. As a consequence, in the current proof systems there is no need to refer to an "oracle" for the data domain and the completeness results are no longer relative to reasoning about data.

In [PS93] Parrow and Sangiorgi present complete equational theories for name-passing calculi, *i.e.* extensions of the π-calculus with the mismatch construction. Mismatch plays an essential rôle in at least two places: the definition of normal form and the axiom for early equivalence. The equational characterisation of the restriction operator also relies on mismatch. In this paper mismatch has been kept outside the process language; *conditions* involving mismatches are used as indices for symbolic bisimulation and as guards in the judgements of the proof systems. The notion of normal forms used in the proofs of the completeness results in [PS93] is much more complicated than ours: it involves maximally consistent saturation of boolean conditions inside terms. In our case such saturation is done at the meta-level, so that the complexity of the completeness proofs have been reduced considerably. It is also interesting to note that all the axioms in [PS93] can be easily derived from the corresponding proof systems in this paper.

[BD94] attempts to adapt the symbolic theory of [HL92, HL93] to the setting of π-calculus. As in [HL93] they use the same language for boolean expressions in two different places: the guards in the proof system and the conditional construction in the calculus (this necessitates the inclusion of the mismatch operator in the calculus). As mentioned earlier, in the current paper mismatch is only employed in the meta-language. In fact, a major objective of our work was to achieve complete axiomatisation of the π-calculus proper, with only *external* usage of mismatch. As shown in Section 4.2 the proof system can be extended smoothly to cover the mismatch construction with the addition of a single inference rule. The definition of symbolic bisimulation in [BD94] still uses the phrase

"there exists a decomposition of ..." (as in [HL93]), while the current paper exploits the notion of "maximally consistent extension" which constitutes a particularly interesting decomposition and is sufficient for all theoretical development. As a consequence the completeness proofs of symbolic bisimulation w.r.t concrete bisimulation and of the inference system w.r.t symbolic bisimulation in this paper are much simpler.

[Liu94] gives a symbolic version of distinction-indexed bisimulation for a sub-language of the π-calculus (without the match operator) and show that this notion of bisimulation captures the conventional bisimulation equivalences. But no proof system is considered there.

In [San93] another notion of bisimulation, called *open bisimulation*, for the π-calculus is proposed, along with an equational axiomatisation. A "symbolic characterisation" of open bisimulation is also given. To deal with the restriction operator, distinctions have to be exploited as indices in the definition of open bisimulation as well as in the axiomatisation. We think it is natural to include in the indexing sets not only inequalities but also equalities. By such mild generalisation, direct characterisations of both late and early bisimulation equivalences for the π-calculus become possible and sound and complete proof systems can be formulated, as demonstrated by our results presented here. It is also interesting to characterise open bisimulation using our symbolic approach. Such a characterisation could facilitate the comparisons between open, late and early bisimulations, as they are expressed within the same framework.

Acknowledgment

Thanks to Julian Rathke for carefully reading a draft of this paper and suggesting many improvements.

This work has been supported by research grants from The President Fund of Chinese Academy of Sciences and National Natural Science Foundation of China. The author would also like to thank The Royal Society of U.K. and Chinese Academy of Sciences for an exchange program which enable him to visit the University of Sussex where this paper was finally produced.

References

[BD92] M. Boreale and R. DeNicola. Testing equivalence for mobile processes. In *CONCUR'92*, number 630 in Lecture Notes in Computer Science, pages 2 – 16. Springer–Verlag, 1992.

[BD94] M. Boreale and R. DeNicola. A symbolic semantics for the π-calculus. In *CONCUR'94*, Lecture Notes in Computer Science. Springer–Verlag, 1994.

[Hen91] M. Hennessy. A model for the π-calculus. Technical Report 8/91, CSAI, University of Sussex, 1991.

[HL92] M. Hennessy and H. Lin. Symbolic bisimulations. Technical Report 1/92, CSAI, University of Sussex, 1992. to appear in *Theoretical Computer Science*.

[HL93] M. Hennessy and H. Lin. Proof systems for message-passing process algebras. In *CONCUR'93*, number 715 in Lecture Notes in Computer Science, pages 202–216, 1993.

[Lin87] H. Lin. Relative completeness in algebraic specifications. Technical Report ECS-LFCS-87-43, Department of Computer Science, University of Edinburgh, 1987.

[Lin93] H. Lin. A verification tool for value-passing processes. In *Proceedings of* 13^{th} *International Symposium on Protocol Specificatio, Testing and Verification*, IFIP Transactions. North-Holland, 1993.

[Liu94] X. Liu. Characterizing bisimulation congruence in the π-calculus. In *CONCUR'94*, Lecture Notes in Computer Science. Springer–Verlag, 1994.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[Mil91] R. Milner. The polyadic π-calculus: A tutorial. In *Proceedings of the International Summer School on Logic and Algebra of Specification*, 1991.

[MPW92] R. Milner, J. Parrow, and D. Walker. A calculus of mobile proceses, part I,II. *Information and Computation*, 100:1–77, 1992.

[PS93] J. Parrow and D. Sangiorgi. Algebraic theories for name-passing calculi. Report ECS-LFCS-93-262, LFCS, University of Edinburgh, 1993.

[San93] D. Sangiorgi. A theory of bisimulation for the π-calculus. In *CONCUR'93*, number 715 in Lecture Notes in Computer Science, 1993.