Unique Fixpoint Induction for Message–Passing Process Calculi*

M. Hennessy & H. Lin†
Computer Science
School of Cognitive and Computing Sciences
University of Sussex

Abstract

We present a proof system for message-passing process calculi with recursion. The key inference rule to deal with recursive processes is a version of Unique Fixpoint Induction for *process abstractions*.

We prove the proof system is sound and also complete for a restricted form of regular message-passing processes. We also show that the system is incomplete in general and discuss possible extensions with inductive inference rules.

1 Introduction

The last decade has seen much research into verification techniques for concurrent systems. However in order to realise the potential benefits of this research it is essential that these techniques be applicable to infinite state systems. There are many different sources for the infinite nature of concurrent systems and in a series of papers, [HL95b, HL95a], we have investigated one such source, by focusing on *message-passing processes algebras*. Systems described in these languages are typically infinite state because of the domain of messages transmitted between processes are often infinite.

Our general approach has been to abstract from concrete reasoning about the transmission of actual data values, and its effect on the behaviour of processes, and to develop *symbolic* methods for such reasoning. For example in [HL95b] we have developed symbolic transition systems for message-passing processes and generalised the well-known bisimulation checking algorithms, [CPS89], to work at this symbolic level, while in [HL95a] we have generalised the equational approach to verification, [Mil89], by developing a proof system for deducing that message-passing processes are semantically equivalent. The present paper is a contribution to this latter line of research.

Let us first review this proof system. It uses judgements of the form

$$b \rhd T = U$$

*Supported by the EU KIT Project SYMSEM.
†On leave from the Institute of Software, Chinese Academy of Sciences, Beijing.

where b is a boolean expression and T, U are message-passing process descriptions. This is interpreted as: in all instantiations which satisfy the boolean expression b the processes T and U are semantically equivalent. The proof rules are suitable generalisations of the simple equation based proof systems, which allow the instantiation of a set of equations and the substitution of equals for equals, together with some rules which depend on establishing identities between message expressions. For example consider processes P, Q of the form

$$c?x.\ \mathit{if}\ b\ \mathit{then}\ T\ \mathit{else}\ U\ ,\quad c?x.R$$

respectively. Intuitively P is a process which accepts some message on the channel c and applies the boolean test b. If this evaluates to *true* then T, with the variable x instantiated, is executed and otherwise U is executed. On the other hand Q is a process which simply inputs a message v and executes R with x instantiated to v.

In order to establish

$$\mathit{true} \rhd P = Q,$$

an instance of substitution of equals for equals will reduce it to the proof obligation

$$\mathit{true} \rhd (\ \mathit{if}\ b\ \mathit{then}\ T\ \mathit{else}\ U\ ) = R.$$

A proof rule for case analysis will then reduce this to the two obligations

$$b \rhd T = R \qquad \text{and} \qquad \neg b \rhd U = R.$$

The proof then proceeds by trying to discharge both of these obligations. Presumably the reasoning will be at least a little different in both cases and in general will depend on the messages involved; specifically on the consequences of the boolean expression b. An example of a proof rule which depends on the message domain is

$$\frac{b \models e = e',\ \ b \rhd T = U}{b \rhd c!e.T = c!e'.U}$$

Here $c!e.T$ denotes a process which sends the value of the message expression e along the channel c and then execute T. To establish that two such processes $c!e.T$ and $c!e'.U$ are semantically equivalent under the assumption b it is sufficient to establish two subgoals. The first, $b \models e = e'$, is an assertion about the message-passing domain which must be established with some independent theorem prover, while the second, $b \rhd T = U$, is a subgoal which can be tackled within the main proof system.

In [HL95a] we design a proof system along these lines and show that by choosing appropriate sets of equations complete proof systems can be obtained for a variety of bisimulation based semantic equivalences, specifically early and late strong bisimulation equivalences, and early and late weak bisimulation congruences. Of course these are actually relative completeness theorems, relative to sufficiently powerful theorem provers for establishing properties of data expressions. More importantly they only apply to *finite message-passing processes*, i.e. processes with no recursion or iteration constructs. The aim of this paper is to extend these proof systems to recursively defined message-passing processes and to investigate the extent to which they are complete.

A typical example of a recursively defined process is given by

$$P \Longleftarrow c?x.d!|x|.P.$$

This defines a process which repeatedly inputs a value, say an integer n , on the channel c and outputs its absolute value $|n|$ on the channel d. A natural proof rule to handle such processes is *Unique Fixpoint Induction*, [Mil89]. In general if P is a process defined recursively by a definition

$$P \Longleftarrow D$$

where P is guarded in D, then to prove that Q is semantically equivalent to P it is sufficient to establish that Q is a fixed point of the equation $P = D$. In terms of the proof system of [HL95a] this can be expressed as a proof rule such as

$$\frac{b \rhd Q = D[Q/P]}{b \rhd Q = P}$$

So for example to show that the process Q, defined by

$$Q \Longleftarrow c?x.\ \mathit{if}\ x < 0\ \mathit{then}\ d!(-x).Q\ \mathit{else}\ d!x.Q\ ,$$

is semantically equivalent to P, defined above, it is sufficient, by this rule, to establish

$$\mathit{true} \rhd Q = c?x.d!|x|.Q.$$

This can easily be proved using the proof system of [HL95a], if we are allowed a *Recursion Unfolding* rule for recursively defined processes, whereby the term Q can be freely rewritten to its definition $c?x.\ \mathit{if}\ x < 0\ \mathit{then}\ d!(-x).Q\ \mathit{else}\ d!x.Q$; the proof relies on two facts about the domain of messages,

$$x < 0 \models |x| = -x \qquad \text{and} \qquad \neg(x < 0) \models |x| = x.$$

To have a reasonably expressive language it is natural to allow *parameterised* definition of processes such as

$$P\langle x\rangle \Longleftarrow c!x.c?y.P(y).$$

However in the presence of such definitions the Unique Fixpoint Induction rule, as naively expressed above, is unsound. As an example consider the definition

$$Q\langle x\rangle \Longleftarrow c!|x|.c?y.Q(y).$$

The judgement

$$x \geq 0 \rhd P(x) = Q(x)$$

is obviously untrue but is can be established using the Unique Fixpoint Rule above from the judgement

$$x \geq 0 \rhd P(x) = c!|x|.c?y.P(y).$$

This in turn can be established from the Recursion Unfolding rule together with the standard rules of the proof system in [HL95a].

In this paper we develop a sound version of Unique Fixpoint Induction appropriate to parameterised recursive definitions. The key point is to recognise that definitions such as that for $P\langle x\rangle$ in fact define *abstractions* which more formally could be written as

$$P \Longleftarrow \lambda x\ c!x.c?y.P(y).$$

This means that our language for message-passing processes is extended in a functional manner to allow abstractions over data domains, and the application of these abstractions to data expressions. In this extended language Unique Fixpoint Induction is only applied to abstractions and we show that restricted in this manner it is sound. Although this might appear to limit its applicability we also show that we can derive a version of Unique Fixpoint Induction for arbitrary terms which is also sound; this derived version contains restrictions on the free variables occurring in the assumptions of judgements. We also give a partial completeness theorem for the proof system. In [Mil89] the corresponding proof system for *pure processes* is complete for *regular processes*, where process definitions are only allowed to use action prefixing and choice. Here we show that our proof system is also complete for a suitable extension of the notion of guarded *regular* to message-passing processes.

We end this section with a brief outline of the rest of the paper. The next section contains a description of the process language we consider together with a brief review of the *late* operational semantics. This is followed by a section devoted to an exposition of the proof system which also contains the statement of the main completeness proof. This is the topic of the next section which is then followed by a section showing how to adapt the results in the previous section for late bisimulation to the early one. The final section contains some concluding remarks and a comparison with related work.

2 The Language

We presuppose a set of *base types* for message values, such as *int, bool* etc., ranged over by ι. Then the set of types of process abstractions or terms is given by

$$\pi ::= process \mid \iota \rightarrow \pi.$$

For each base type ι, we assume a set of channel names $Chan_\iota = \{c, ...\}$, a set of message expressions $Exp_\iota = \{e, ...\}$ containing a set of *values* of that type, Val_ι, and a set of data variables $Var_\iota = \{x, y, z, ...\}$. We also assume a set of process identifiers $ID_\pi = \{X, Y, Z, ...\}$, for each abstraction type π.

The formation rules for terms in the language is given in Figure 1. The main type for terms is *process* and the formation rules for these terms should be straightforward for readers familiar with process algebras such as *CCS*; in addition to the usual process operators we have a simple form of boolean guards, $b \rightarrow T$, rather than the more usual *if ... then ... else ...* construct. We also have elementary formation rules concerned with abstraction with respect to base types and application. Throughout the remainder of the paper we assume that all terms are well-typed according to these rules but to aid readability we omit all references to types unless their inclusion is strictly necessary. We will also write $c?x.T$ for $c?\lambda xT$.

In this language we have both data variables and abstraction identifiers and we assume an appropriate notion of substitution for both. A *data substitution* is a family of partial functions from Var_ι to Exp_ι, indexed by base types. We use $\sigma, \ldots$ to denote data substitutions, and write $[\overline{e}/\overline{x}]$ for the data substitution that sends $\overline{x}$ to $\overline{e}$. The application of data substitutions is slightly complicated because an abstraction such as $\lambda x.T$ binds all occurrences of the data variable x in T. This induces the usual notions of bound and free variables of terms; the set of free variables of a term T is denoted by $fv(T)$. We

$$\frac{}{\mathbf{0} : \ process} \qquad \frac{X \in ID_\pi}{X \in \pi}$$

$$\frac{T,\ U : \ process}{T|U : \ process} \qquad \frac{T,\ U : \ process}{T + U : \ process}$$

$$\frac{T : \ process,\ b : bool}{b \rightarrow T : \ process} \qquad \frac{T : \ process,\ c \in Chan}{T\backslash c : \ process}$$

$$\frac{F : \ \iota \rightarrow process,\ c \in Chan_\iota}{c?F : \ process} \qquad \frac{T : \ process,\ e \in Exp_\iota,\ c \in Chan_\iota}{c!eT : \ process}$$

$$\frac{F : \ \iota \rightarrow \pi,\ e \in Exp_\iota}{Fe : \ \pi} \qquad \frac{T : \ \pi,\ x \in Var_\iota}{\lambda xT : \ \iota \rightarrow \pi}$$

Figure 1: The Formation Rules

also have the standard definition of α-equivalence between terms, denoted $\equiv_\alpha$. So when applying a data substitution σ to a term T to obtain the term $T\sigma$ bound variables are renamed when necessary to avoid capture and σ is applied to all occurrences of data expressions. This presupposes a reasonable notion of substitution on data expressions, the application of σ to a data expression e to obtain the data expression $e\sigma$.

The set of abstraction identifiers appearing in T is denoted $Id(T)$. Since there are no binding operators for abstraction identifiers abstraction substitutions are more straightforward. An *abstraction substitution* is a type respecting partial function from process identifiers to *data-closed* terms, i.e. terms with no free occurrences of data variables. For example $[\overline{F}/\overline{X}]$, or $[F_i/X_i|1 \leq i \leq m]$, is the abstraction substitution that sends X_i to F_i for $1 \leq i \leq m$. We will often simply write $[F_i/X_i|i]$ when the range of i is clear from the context. The application of an abstraction substitution to a term T syntactically replaces all occurrences of abstraction identifiers in T by the corresponding terms. The interested reader is referred to [Sto88] for precise definitions of the application of these substitutions. Here we simply state the following facts, where we assume that $\overline{X}$ and $\overline{Y}$ are lists of different abstraction identifiers, and $\overline{F}$ and $\overline{G}$ data-closed terms with appropriate types

1. $T[\overline{F}/\overline{X}][\overline{G}/\overline{Y}] = T[\overline{F}[\overline{G}/\overline{Y}], \overline{G}/\overline{X}, \overline{Y}]$. In particular, if $\overline{Y} \cap Id(\overline{F}) = \emptyset$, then $T[\overline{F}/\overline{X}][\overline{G}/\overline{Y}] \equiv_\alpha T[\overline{F}, \overline{G}/\overline{X}, \overline{Y}]$.
2. $T[\overline{e}/\overline{x}][\overline{F}/\overline{X}] \equiv_\alpha T[\overline{F}/\overline{X}][\overline{e}/\overline{x}]$.

An *data evaluation* ρ is a particular kind of data substitution which maps every variable to a value of the appropriate type. We use $\rho(e)$, or $e\rho$, to denote the result of applying the data evaluation to the data expression e. As in [HL95a] we assume some natural but simplifying properties of the language of data expressions. We assume $\rho(e)$, always yields a value, i.e. the evaluation of data expressions always terminates, and so

our approach is to work modulo these evaluations. We also assume, for example, that each expression e has associated with it a set of variables $fv(e)$ such that if ρ and ρ' agree on $fv(e)$ then $\rho(e) = \rho'(e)$. If an expression e has no variables, *i.e.* it is *closed*, then $\rho(e)$ is independent of ρ and we use $[\![e]\!]$ to denote its value. For expressions of type *bool* we use the suggestive notation $b \models b'$ to indicate that for every evaluation if $\rho(b)$ is true then so is $\rho(b')$. As a consequence, if $b \models b'$ then $b\sigma \models b'\sigma$ for any data substitution σ. We will also write $\rho \models b$ to mean $\rho(b)$ is true.

We now turn our attention to the operational semantics of the language. For readers familiar with [Mil89, HL95a] this is perfectly straightforward. A term T of type *process* may include abstraction identifiers from ID_π and the behaviour of T will depend on *declarations* which associate abstraction terms with these identifiers. Let $Id(T)$ denote the set of abstraction identifier occurring in T. If $X : \pi$, where π has the form $\iota_1 \to \iota_2 \ldots \iota_n \to process$, then a *definition* for X has the form

$$X \Leftarrow F$$

where F is a data–closed term of the form $\lambda x_1 \lambda x_2 \ldots \lambda x_n T$. We will often informally render such a definition as

$$X\langle \overline{x} \rangle \Leftarrow T$$

where $\overline{x}$ represents the vector of variables $x_1, x_2, \ldots x_n$.

A set of definitions

$$D = \{\, X_i \Leftarrow F_i \mid 1 \leq i \leq n \,\}$$

is called a *declaration* if it satisfies the following conditions:

- $Id(F_i) \subseteq \{X_1,\ ...,\ X_n\}$;
- $X_i = X_j$ implies $i = j$.

When such a D is a declaration we say each X_i is declared in D. X_i is *guarded* in D if every occurrence of X_i in any F_j is within some subexpression $\alpha.T$ of F_j. D is guarded if every X_i is guarded in D. We will confine our attention to guarded declarations.

Let $\mathcal{T}_D$ be the set of terms using only the identifiers declared in D. The operational semantics given below will assume a given declaration D and will consequently describe the operational behaviour of terms in $\mathcal{T}_D$ relative to D. As is well-known, in the setting of message–passing process algebras there are two reasonable notions of bisimulation equivalence, namely *late* and *early*, [MPW92, HL95b]. The main body of this paper will concentrate on late bisimulation but we will outline briefly in Section 6 how the theory developed for the late case can be adapted to early bisimulation.

The (late) operational semantics of our language is given in Figure 2, where symmetric rules for + and | have been omitted. It uses three kinds of next state relations:

- $\xrightarrow{\tau}_l$ a relation over data-closed terms of type *process*
- for each value v and channel c of the same type a relation $\xrightarrow{c!v}_l$ also over data-closed terms of type *process*
- for each channel c of type ι a relation $\xrightarrow{c?}_l$ from data-closed terms of type *process* to terms of type $\iota \to process$.

$\tau.P \xrightarrow{\tau}_l P$		
$c!e.P \xrightarrow{c!v}_l P$		where $[\![e]\!] = v$
$c?.F \xrightarrow{c?}_l F$		
$P \xrightarrow{a}_l P'$	implies	$P + Q \xrightarrow{a}_l P'$
$P \xrightarrow{a}_l P'$	implies	$P \mid Q \xrightarrow{a}_l P' \mid Q$
$P \xrightarrow{c?}_l F,\ Q \xrightarrow{c!v}_l Q'$	implies	$P \mid Q \xrightarrow{\tau}_l Fv \mid Q'$
$P \xrightarrow{a}_l P', [\![b]\!] = true$	implies	$b \to P \xrightarrow{a}_l P'$
$P \xrightarrow{a}_l P', c \notin Chan(a)$	implies	$P\backslash c \xrightarrow{a}_l P'\backslash c$
$F\overline{e} \xrightarrow{a}_l P$	implies	$X(\overline{e}) \xrightarrow{a}_l P$
		$X \Leftarrow F$ is a definition
$T[e/X] \xrightarrow{a}_l P$	implies	$(\lambda x T)e \xrightarrow{a}_l P$

Figure 2: Late Operational Semantics

Lemma 2.1 *Suppose X is guarded in P and $P[\overline{F}/\overline{X}] \xrightarrow{a}_l P'$. Then P' is of the form $Q[\overline{F}/\overline{X}]$ for some Q and $P[\overline{G}/\overline{X}] \xrightarrow{a}_l Q[\overline{G}/\overline{X}]$ for any $\overline{G}$.*

Proof: By induction on the structure of P. □

Using these relations we can now define a notion of (strong) bisimulation equivalence.

Definition 2.2 A symmetric relation R between data-closed terms of type *process* is a strong late bisimulation if it satisfies: $(P, Q) \in R$ implies that

$$\text{If } P \xrightarrow{c?}_l F \text{ then } Q \xrightarrow{c?}_l G \text{ and for all } v \in Val\ \ (Fv, Gv) \in R$$
$$\text{If } P \xrightarrow{a}_l P',\ a \in \{\tau, c!v\}, \text{ then } Q \xrightarrow{a}_l Q' \text{ and } (P', Q') \in R$$

We use $\sim_l$ to denote the largest late strong bisimulation. This definition only applies to data-closed terms of type *process*. However it is extended to data-closed terms of arbitrary type by structural induction on types: for F, G of type $\iota \to \pi$ we let $F \sim_l G$ if $Fv \sim_l Gv$ for all $v \in Val_\iota$. Finally it is extended to arbitrary open terms by letting $T \sim_l U$ if $T\rho \sim_l U\rho$ for every data-instantiation ρ. □

The object of the paper is to develop a proof system to deduce statements of the form $T \sim_l U$.

3 The Proof System

The proof system we design is an extension of that used in [HL95a] for finite processes. Judgements are of the form

$$\vdash_D b \rhd T = U$$

where D is a declaration, b is a boolean expression and T, U are terms of the same type. Its intended meaning is: $T\rho \sim_l U\rho$ for every ρ such that $\rho \models b$. We usually abbreviate

EQUIV $\dfrac{}{\vdash_D true \rhd T = T} \quad \dfrac{\vdash_D b \rhd T = U}{\vdash_D b \rhd U = T} \quad \dfrac{\vdash_D b \rhd T = U \quad \vdash_D b \rhd U = V}{\vdash_D b \rhd T = V}$

EQN $\dfrac{}{\vdash_D T\theta = U\theta}$ $T = U$ is an axiom

CONGR $\dfrac{\vdash_D b \rhd T_i = U_i \quad i = 1,2}{\vdash_D b \rhd T_1 + T_2 = U_1 + U_2}$

α-CONV $\dfrac{}{\vdash_D T = U}$ $T \equiv_\alpha U$

INPUT $\dfrac{\vdash_D b \rhd T = U}{\vdash_D b \rhd c?x.T = c?x.U}$ $x \notin fv(b)$

OUTPUT $\dfrac{b \models e = e', \quad \vdash_D b \rhd T = U}{\vdash_D b \rhd c!e.T = c!e'.U}$

TAU $\dfrac{\vdash_D b \rhd T = U}{\vdash_D b \rhd \tau.T = \tau.U}$

GUARD $\dfrac{\vdash_D b \wedge b' \rhd T = U \quad \vdash_D b \wedge \neg b' \rhd \mathbf{0} = U}{\vdash_D b \rhd b' \rightarrow T = U}$

CUT $\dfrac{\vdash_D b \models b_1 \vee b_2, \quad b_1 \rhd T = U \quad b_2 \rhd T = U}{\vdash_D b \rhd T = U}$

ABSURD $\dfrac{}{\vdash_D false \rhd T = U}$

Figure 3: The Old Inference Rules

$\vdash_D true \rhd T = U$ to $\vdash_D T = U$. Strictly speaking we should annotate the terms with their types but again we omit this information for the sake of readability. The inference rules are divided into two groups. The first, for manipulating process terms, are given in Figure 3 and are taken directly from [HL95a]. They are a minimal formal basis for the reasoning outlined in the Introduction. In [HL95a] we have seen that these rules can be extended by a number of natural derived rules which ease the use of the proof system. A typical example is the following rule of consequence:

$$\text{CONSEQ} \qquad \frac{b \models b_1, \quad \vdash_D b_1 \rhd T = U}{\vdash_D b \rhd T = U}$$

which can be derived from the CUT rule.

The rules given in Figure 4 are specifically designed to handle the new constructs in the language, recursive declarations, abstraction and application. The first two are concerned with the introduction of new definitions, a step which can always be carried

dec-I $\quad \dfrac{\vdash_D b \rhd T = U}{\vdash_{D \cup E} b \rhd T = U}$

dec-E $\quad \dfrac{\vdash_{D \cup E} b \rhd T = U}{\vdash_D b \rhd T = U} \quad T,\ U \in \mathcal{T}_D$

UNFOLD $\quad \dfrac{}{\vdash_D \ \rhd X = F} \quad X \Leftarrow F \in D$

UFI $\quad \dfrac{\vdash_D G_i = F_i[\overline{G}/\overline{X}],\ 1 \leq i \leq n}{\vdash_{D \cup E} G_1 = X_1} \quad E = \{\, X_i \Leftarrow F_i \mid 1 \leq i \leq n \,\}$ is a guarded declaration

λ-I $\quad \dfrac{\vdash_D b \rhd Tx = Ux}{\vdash_D b \rhd T = U} \quad x \notin fv(b, T, U)$

λ-E $\quad \dfrac{b \models e = e',\ \vdash_D b \rhd T = U}{\vdash_D Te = Ue'}$

β $\quad \dfrac{}{\vdash_D \ \rhd (\lambda x T)e = T[e/x]}$

Figure 4: The New Inference Rules

out, and their elimination, which is allowed provided the definitions being eliminated do not concern abstraction identifiers which occur in the conclusion. This is followed by the UNFOLD rule, also discussed in the Introduction, and a version of Unique Fixpoint Induction. Finally we have very standard rules for the introduction, application and elimination of λ-abstractions and β-reduction.

As with the rules for process manipulation in Figure 3 these rules form a basis for a proof system for manipulating abstractions and recursive definitions, and on top of which more interesting rules can be derived. Two such examples are:

- $\eta \qquad \vdash_D \lambda x(Tx) = T$
- λ-cong $\quad \dfrac{T = U}{\lambda x T = \lambda x U}$

whose derivation we leave to the reader.

It is interesting to re-examine, in the light of these inference rules, the unsound reasoning in the Introduction which leads to the false conclusion

$$\vdash_D x \geq 0 \rhd P(x) = Q(x)$$

where $P,\ Q$ are defined by

$$P\langle x\rangle \Longleftarrow c!x.c?y.P(y) \quad \text{and} \quad Q\langle x\rangle \Longleftarrow c!|x|c?y.Q(y)$$

respectively. Because of the syntactic form of Unique Fixpoint Induction (UFI) in our proof system it can not be applied to obtain a conclusion of the form

$$\vdash_D x \geq 0 \rhd P(x) = Q(x).$$

As an approximation we could try to derive

$$\vdash_D x \geq 0 \rhd P = Q;$$

to conclude this from an application of UFI we must establish the judgement

$$\vdash_D x \geq 0 \rhd P = \lambda x\ c!|x|.c?y.P(y).$$

This however is not possible. The most likely proof strategy is to apply an instance of λ-I to reduce it to

$$\vdash_D x \geq 0 \rhd Pz = (\lambda x\ c!|x|.c?y.P(y))z.$$

Note that here we can not use x in place of z because of the side condition in λ-I. By β-reduction the above is the same as

$$\vdash_D x \geq 0 \rhd Pz = c!|z|.c?y.P(y).$$

Here the only way forward is to unfold the occurrence of P on the left handside and use β-reduction to obtain the proof obligation

$$\vdash_D x \geq 0 \rhd c!z.c?y.P(y) = c!|z|.c?y.P(y)$$

which of course can not be established; the assumption $x \geq 0$ can not be used (and correctly so) to establish the equality of the output expressions.

As an example of a proof within the system consider the declaration D:

$$\begin{array}{lcl} A\langle x\rangle & \Leftarrow & c!x.c?z.A(x+z) \\ B\langle y\rangle & \Leftarrow & c!(y+1).c?z.B(y+z) \end{array}$$

We show how to derive

$$\vdash_D x = (y+1) \rhd A(x) = B(y),$$

Using the rules λ-I and CONSEQ this can be reduced to

$$\vdash_D A(y+1) = B(y).$$

By λ-E and β this can be further reduced to

$$\vdash_D \lambda y A(y+1) = B.$$

Now UFI can be applied to obtain this conclusion if we can establish

$$\vdash_{D'} \lambda y A(y+1) = \lambda y c!(y+1).c?z.(\lambda y A(y+1))(y+z),$$

i.e.

$$\vdash_{D'} \lambda y A(y+1) = \lambda y\ c!(y+1).c?z.A(y+z+1).$$

where D' contains the definition of A. We can now apply λ-I to reduce this to

$$\vdash_{D'} (\lambda y A(y+1))w = (\lambda y c!(y+1).c?z.A(y+z+1))w,$$

which by β-reduction reduces to

$$\vdash_{D'} A(w+1) = c!(w+1).c?z.A(w+z+1),$$

which follows in a straightforward fashion by an instance of UNFOLD and β-reduction.

This is a somewhat laborious derivation of a relatively simple result but many of the proof steps are trivial applications of β-reduction and λ-introduction and elimination, which can be handled in a semi-automatic way in any implementation of the system. The proof is however complicated by the fact that UFI can only be applied to abstractions, in the sense that one of the terms in the conclusion must be an abstraction identifier. But this restriction can be relaxed a little by using the following derivable proof rule:

If $E = \{ X_i \Leftarrow \lambda \overline{x}_i(b_i \to T_i) \mid 1 \leq i \leq n \}$ is a guarded declaration then

$$\text{UFI-O} \qquad \frac{\vdash_D b_i \rhd U_i = T_i[\overline{F}/\overline{X}],\ 1 \leq i \leq n}{\vdash_{D \cup E} b_1 \rhd U_1 = X_1(\overline{x}_1)}$$

where $F_i \equiv \lambda \overline{x}_i(b_i \to U_i),\ 1 \leq i \leq n$.

Here the conclusion can involve an abstraction identifier, X_1 applied to a list of variables, $\overline{x}_1$, which makes the rule much easier to use. In particular when all b_i are *true* then the rule reduces to

$$\text{UFI-O-t} \qquad \frac{\vdash_D U_i = T_i[\overline{F}/\overline{X}],\ 1 \leq i \leq n}{\vdash_{D \cup E} U_1 = X_1(\overline{x}_1)}$$

where $F_i \equiv \lambda \overline{x}_i U_i,\ 1 \leq i \leq n$.

Now revisiting the proof above,

$$A(y+1) = B(y)$$

can be derived directly by one application of UFI-O-t from the judgement

$$\vdash_D A(y+1) = (c!(y+1).c?z.B(y+z))[\lambda y A(y+1)/B],$$

i.e.

$$\vdash_D A(y+1) = (c!(y+1).c?z.A(y+1+z),$$

which follows immediately from UNFOLD and β-reduction.

Proposition 3.1 *The proof rule UFI-O is derivable.*

Proof: In this proof we assume some familiarity with the capabilities of the basic proof system, that based on the process manipulation rules. All of the properties we require are summarised in Proposition 3.2, stated below.

Suppose

$$\vdash_D b_i \rhd U_i = T_i[\overline{F}/\overline{X}],\ 1 \leq i \leq n.$$

S1	$X + \mathbf{0} = X$
S2	$X + X = X$
S3	$X + Y = Y + X$
S4	$(X + Y) + Z = X + (Y + Z)$

Figure 5: The Axioms $\mathcal{A}$

Using elementary reasoning, as detailed in the just mentioned Proposition, this means we can infer

$$\vdash_D b_i \to U_i = b_i \to T_i[\overline{F}/\overline{X}],\ 1 \leq i \leq n.$$

By λ-cong we have

$$\vdash_D \lambda\overline{x}_i(b_i \to U_i) = \lambda\overline{x}_i(b_i \to T_i[\overline{F}/\overline{X}]),\ 1 \leq i \leq n.$$

Since $\vdash_D \lambda\overline{x}_i(b_i \to T_i[\overline{F}/\overline{X}]) = \lambda\overline{x}_i(b_i \to T_i)[\overline{F}/\overline{X}]$, applying UFI we obtain

$$\vdash_{D\cup E} \lambda\overline{x}_1(b_1 \to U_1) = X_1.$$

Using λ-E, Proposition 3.2 and λ-cong, we can derive $X_1 = \lambda\overline{x}_1(b_1 \to X_1(\overline{x}_1))$. Hence

$$\vdash_{D\cup E} \lambda\overline{x}_1(b_1 \to U_1) = \lambda\overline{x}_1(b_1 \to X_1(\overline{x}_1))$$

Applying λ-E we obtain

$$\vdash_{D\cup E} b_1 \to U_1 = b_1 \to X_1(\overline{x}_1),$$

which, again using Proposition 3.2, gives

$$\vdash_{D\cup E} b_1 \rhd U_1 = X_1(\overline{x}_1).$$

as required. □

This new form of UFI does make the system easier to use but there is still an apparent restriction to the application of UFI-O because conclusions must involve terms of the form $X(\overline{x})$. As an example of where this might cause problems consider the following definitions:

$$\begin{aligned} A\langle x\rangle &\Leftarrow c!(3x).A(x+2) \\ B\langle x\rangle &\Leftarrow c!(2x).B(x+3). \end{aligned}$$

The two terms $A(2x)$ and $B(3x)$ are semantically equivalent but none of our versions of UFI can be used to directly conclude

$$\vdash A(2x) = B(3x).$$

However the way forward is to introduce a new definition

$$C\langle x\rangle \Leftarrow c!(6x).C(x+1)$$

and to use two applications of UFI-O-t to establish

$$\vdash A(2x) = C(x) \text{ and } \vdash B(3x) = C(x).$$

This is an instance of quite a general strategy which indicates the power of the derived rule UFI-O.

In general the usefulness of our proof system depends on the equations which we apply in the inference rule EQN. At the very least the equations in Figure 5 are necessary; these characterise strong bisimulation equivalence for *CCS*. Let $\vdash_D^L b \rhd T = U$ mean that $\vdash_D b \rhd T = U$ can be derived in the proof system using the equations $\mathcal{A}$ (the superscript L stands for "Late"). The following are some simple yet useful facts about $\vdash_D^L$ whose proofs can be found in [HL95a]:

Proposition 3.2 *1.* $\vdash_D^L b \to b' \to T = b \wedge b' \to T$

2. $\vdash_D^L T = T + b' \to T$

3. $b \models b'$ *implies* $\vdash_D^L b \rhd T = b' \to T$

4. $\vdash_D^L b \wedge b' \rhd T = U$ *implies* $\vdash_D^L b \rhd b' \to T = b' \to U$

5. $\vdash_D^L b \to (T + U) = b \to T \ + \ b \to U$

6. $\vdash_D^L b \to U \ + \ b' \to U = b \vee b' \to U$

7. If $fv(b) \cap \ bv(\alpha) = \emptyset$ *then* $\vdash_D^L b \to \alpha.T = b \to \alpha.(b \to T)$

We end this section with another example where in addition the τ-laws of *CCS* and the Expansion Theorem, [Mil89], come into play. We take the version of the expansion theorem from [HL95a] and list it in Figure 6. We will only need the first τ-law in the example to follow:

$$\text{T1} \qquad \alpha.\tau.T = \alpha.T$$

We will also need two sound equations IF-PAR and IF-RES, to distribute the parallel and restriction operators over *if* _ *then* _ *else* _:

$$\begin{array}{ll} \text{IF-PAR} & (\ \mathit{if}\, b\ \mathit{then}\, T\ \mathit{else}\, U\ ) \mid R = \ \mathit{if}\, b\ \mathit{then}\, (T \mid R)\ \mathit{else}\, (U \mid R) \\ \text{IF-RES} & (\ \mathit{if}\, b\ \mathit{then}\, T\ \mathit{else}\, U\ ) \backslash s = \ \mathit{if}\, b\ \mathit{then}\, T \backslash s\ \mathit{else}\, U \backslash s \end{array}$$

where $\mathit{if}\, b\ \mathit{then}\, T\ \mathit{else}\, U$ abbreviates $b \to T + \neg b \to U$.

Consider a personal bank account *Account* which holds the current amount m and can accept two requests *credit* and *withdraw*; when the amount of money to be withdrawn exceeds the current amount the withdraw request will be rejected:

$$\begin{array}{rl} \mathit{Account}\langle m \rangle \Leftarrow & \mathit{credit}?n.\mathit{Account}(m+n) + \\ & \mathit{withdraw}?n.\mathit{if}\ m \geq n\ \mathit{then}\ \mathit{payout}!n.\mathit{Account}(m-n) \\ & \qquad \mathit{else}\ \mathit{reject}!.\mathit{Account}(m) \end{array}$$

(when the content of an output action is immaterial it is omitted, as in the case of *reject*!. Similarly the variable in an input action will also be omitted when it is not used later.)

Let $T,\ U$ denote $\Sigma_i \alpha_i.T_i,\ \Sigma_j \beta_j.U_j,$ with $fv(T) \cap bv(U) = fv(U) \cap bv(T) = \emptyset$ where $fv(T)$ and $bv(T)$ are free data variables and bound data variables in the term T, respectively. Then

$$T \mid U = sync\_move(T,U) + async\_move(T,U)$$

where

$$\begin{array}{ll} sync\_move(T,U) = & \Sigma\{\, \tau.(T_i\{e/x\} \mid U_j) \mid \alpha_i \equiv c?x, \beta_j \equiv c!e \,\} + \\ & \Sigma\{\, \tau.(T_i \mid U_j\{e/x\}) \mid \alpha_i \equiv c!e, \beta_j \equiv c?x \,\} \\ async\_move(T,U) = & \Sigma_i \alpha_i.(T_i \mid U) + \Sigma_j \beta_j.(T \mid U_j) \end{array}$$

Figure 6: The Expansion Theorem

Such an account can be implemented by two processes running in parallel: *Count* interfaces to the account holder and passes all calculating tasks to the process *Calc*. The two processes are synchronised using an internal channel s:

$$\begin{array}{lll} Count\langle m\rangle \Leftarrow & credit?n.add!(m,n).result?r.Count(r) + \\ & withdraw?n.sub!(m,n).(result?r.payout!n.s!.Count(r) + \\ & \qquad\qquad underflow?.reject!.s!.Count(m)) \end{array}$$

$$\begin{array}{lll} Calc \Leftarrow & add?(m,n).result!(m+n).Calc + \\ & sub?(m,n).if\ m \geq n\ then\ result!(m-n).s?.Calc \\ & \qquad\qquad else\ underflow!.s?.Calc \end{array}$$

We show that such an implementation is correct by deriving, in the proof system,

$$\vdash Account(m) = (Count(m) \mid Calc)\backslash R$$

where R denotes the set $\{add, sub, underflow, result, s\}$. By UFI-O this can be reduced to (we use $CC(m)$ to denote $Count(m) \mid Calc$)

$$\begin{array}{l} \vdash CC(m)\backslash R = credit?n.CC(m+n)\backslash R + \\ \qquad\qquad withdraw?n.\ if\ m \geq n\ then\ payout!n.CC(m-n)\backslash R \\ \qquad\qquad\qquad\qquad else\ reject!.CC(m)\backslash R \end{array}$$

Unfolding the recursive definitions in the left handside, then applying the expansion theorem and T1 three times, we get

$$\begin{array}{l} \vdash CC(m)\backslash R = \\ credit?n.CC(m+n)\backslash R + \\ withdraw?n.((\ if\ m \geq n\ then\ result!(m-n).s?.Calc\ else\ underflow!.s?.Calc\ ) \mid \\ \qquad (result?r.payout!n.s!.Count(r) + underflow?.reject!.s!.Count(m)))\backslash R \end{array}$$

So we are done if we can show

$$\begin{array}{ll} \vdash & withdraw?n.((\ if\ m \geq n\ then\ result!(m-n).s?.Calc\ else\ underflow!.s?.Calc\ ) \mid \\ & \qquad (result?r.payout!n.s!.Count(r) + underflow?.reject!.s!.Count(m)))\backslash R \\ = & withdraw?n.\ if\ m \geq n\ then\ payout!n.CC(m-n)\backslash R\ else\ reject!.CC(m)\backslash R\ . \end{array}$$

This can be achieved by using IF-PAR and IF-RES, as well as Proposition 3.2, to push the parallel and restriction operators over *if* _ *then* _ *else* in the left hand-side of the equation, followed by three applications of the expansion theorem and T1.

4 Soundness and Completeness

The soundness of the system is relatively straightforward. The only difficulty is the Unique Fixpoint Induction rule whose soundness depends on the following Proposition, a generalisation of Proposition 14, page 104 of [Mil89].

Proposition 4.1 *Suppose $\overline{H}$ is a sequence of terms of arbitrary type which only use abstraction identifiers from $\overline{X}$, and all occurrences of these identifiers are guarded. Let $\overline{F}$ and $\overline{G}$ be sequences of data-closed terms such that $\overline{F} \sim_l \overline{H}[\overline{F}/\overline{X}]$ and $\overline{G} \sim_l \overline{H}[\overline{G}/\overline{X}]$. Then $\overline{F} \sim_l \overline{G}$.*

Proof: Let

$$R = \{\, (C[\overline{F}/\overline{X}]\rho,\ C[\overline{G}/\overline{X}]\rho) \mid Id(C) \subseteq \overline{X}, C[\overline{F}/\overline{X}], C[\overline{G}/\overline{X}] : process \,\}.$$

First suppose R is a bisimulation. We show that it follows from this that $F_j \sim_l G_j$ for each j. Let the type of X_j be $\iota_1 \to \ldots \to \iota_k \to process$. We need to demonstrate that for all $v_i \in Val_{\iota_i}$, $F_j v_1 \ldots v_k \sim_l G_j v_1 \ldots v_k$. Let $C[\ ]$ be the context $X_j z_1 \ldots z_k$, where z_i are fresh variables, and let ρ map z_i to v_i. Then $C[\overline{F}/\overline{X}]\rho$ is $F_j v_1 \ldots v_k$ and $C[\overline{G}/\overline{X}]\rho$ is $G_j v_1 \ldots v_k$.

So it remains to show that R is a (late) bisimulation. For this we shall show R is a *bisimulation up to* $\sim_l$ ([Mil89]). By symmetry it is enough to prove

$$C[\overline{F}/\overline{X}]\rho \stackrel{a}{\longrightarrow}_l U \text{ implies } C[\overline{G}/\overline{X}]\rho \stackrel{a}{\longrightarrow}_l V \text{ with } U \sim_l R \sim_l V$$

For this we apply induction on why $C[\overline{F}/\overline{X}]\rho \stackrel{a}{\longrightarrow}_l U$. Consider the possible cases for $C[]$.

- $C \equiv X_i(\overline{e})$. Then $C[\overline{F}/\overline{X}]\rho \equiv F_i\overline{e}\rho \stackrel{a}{\longrightarrow}_l U$. Since $F_i \sim_l H_i[\overline{F}/\overline{X}]$, we have $H_i\overline{e}\rho[\overline{F}/\overline{X}] \stackrel{a}{\longrightarrow}_l U' \sim_l U$. Since X_i is guarded in H_i, by Lemma 2.1 U' is of the form $C'[\overline{F}/\overline{X}]\rho$ and $H_i\overline{e}\rho[\overline{G}/\overline{X}] \stackrel{a}{\longrightarrow}_l C'[\overline{G}/\overline{X}]\rho$. But $C[\overline{G}/\overline{X}]\rho \equiv G_i\overline{e}\rho \sim_l H_i\overline{e}\rho[\overline{G}/\overline{X}]$, so $C[\overline{G}/\overline{X}]\rho \stackrel{a}{\longrightarrow}_l V \sim_l C'[\overline{G}/\overline{X}]\rho$. Hence $U \sim_l R \sim_l V$.

- $C_1 \mid C_2$. There are three cases.

  - $C[\overline{F}/\overline{X}]\rho \equiv C_1[\overline{F}/\overline{X}]\rho \mid C_2[\overline{F}/\overline{X}]\rho \stackrel{a}{\longrightarrow}_l U$ is because $C_1[\overline{F}/\overline{X}]\rho \stackrel{a}{\longrightarrow}_l U'$ with $U \equiv U' \mid C_2[\overline{F}/\overline{X}]\rho$. By induction $C_1[\overline{G}/\overline{X}]\rho \stackrel{a}{\longrightarrow}_l V'$ with $U' \sim_l R \sim_l V'$. Hence $C[\overline{G}/\overline{X}]\rho \stackrel{a}{\longrightarrow}_l V \equiv V' \mid C_2[\overline{G}/\overline{X}]\rho$, and $U \equiv U' \mid C_2[\overline{F}/\overline{X}]\rho \sim_l R \sim_l V' \mid C_2[\overline{G}/\overline{X}]\rho \equiv V$.
  - $C[\overline{F}/\overline{X}]\rho \stackrel{a}{\longrightarrow}_l U$ is because $C_2[\overline{F}/\overline{X}]\rho \stackrel{a}{\longrightarrow}_l U'$ with $U \equiv C_1[\overline{F}/\overline{X}]\rho \mid U'$. This case is symmetric to the first case.
  - $C[\overline{F}/\overline{X}]\rho \stackrel{\tau}{\longrightarrow}_l U$ is because $C_1[\overline{F}/\overline{X}]\rho \stackrel{c?}{\longrightarrow}_l F'$, $C_2[\overline{F}/\overline{X}]\rho \stackrel{c!v}{\longrightarrow}_l U'$ and $U \equiv F'v \mid U'$. By induction $C_1[\overline{G}/\overline{X}]\rho \stackrel{c?}{\longrightarrow}_l G'$, $C_2[\overline{G}/\overline{X}]\rho \stackrel{c!v}{\longrightarrow}_l V'$ with $F'v \sim_l R \sim_l G'v$, $U' \sim_l R \sim_l V'$. Then $C[\overline{G}/\overline{X}]\rho \stackrel{\tau}{\longrightarrow}_l G'v \mid V'$ and $F'v \mid U' \sim_l R \sim_l G'v \mid V'$.

The other cases are similar. □

Proposition 4.2 *(Soundness of $\vdash_D^L$) $\vdash_D^L b \rhd T = U$ implies $T\rho \sim_e U\rho$ for any $\rho \models b$.*

Proof: It is sufficient to show that each axiom in $\mathcal{A}$ is sound and each of the proof rules preserves soundness. We concentrate on UFI.

Suppose $\overline{G} \sim_l \overline{F}[\overline{G}/\overline{X}]$. Directly from the operational semantics we can check that $\overline{X} \sim_l \overline{F}[\overline{X}/\overline{X}]$ and so, since the declaration is guarded, we can immediately apply the previous Proposition to conclude $\overline{G} \sim_l \overline{X}$. □

It is unrealistic to expect that the system is complete. Even pure *CCS*, or our language with a trivial one point message-domain, is Turing complete in the presence of the parallel and restriction operators. However in [Mil84, Mil89, BK88] complete proof systems are obtained for *regular* processes, where action prefixing and choice, $+$, are the only operators allowed in declarations. This leads to the following definition.

Definition 4.3 A declaration

$$D = \{ X_i \Leftarrow F_i \mid 1 \leq i \leq n \}$$

is called *regular* if the only operators allowed in F_i are

- action prefixing, $c?x.\ \_$, $c!e.\ \_$ and $\tau.\ \_$;
- choice, $\_ + \_$;
- guards, $b \rightarrow \ \_$.

It is called *restricted regular* if in addition every occurrence of an abstraction identifier X is of the form $X(\overline{v})$: *process*, such that each v_i is either a variable or a data constant. A term is called *restricted regular* if it can be used as part of a restricted regular definition. □

Theorem 4.4 *Let D be a restricted regular declaration and T, U are restricted regular terms in $\mathcal{T}_D$. If $T\rho \sim_l U\rho$ for every ρ such that $\rho \models b$, then $\vdash_D^L b \rhd T = U$.* □

This completeness theorem is not true in general for arbitrary unguarded regular declarations; a counter-example can be found in the conclusion. However the question is still open for guarded regular declarations.

The remainder of the section is devoted to proving this result which follows closely the corresponding result in [Mil84], but technically working at a *symbolic* level.

The first step in the completeness proof is to outline a series of transformations on restricted regular processes which make them easier to handle. We may assume that definitions, and therefore associated terms, are formed by applying prefixing, choice or boolean guard to terms of the form $X(\overline{v})$ or $\mathbf{0}$. Moreover all use of constants can be

eliminated by introducing appropriate new abstraction identifiers with fewer parameters. For example the declaration

$$A\langle x\rangle \;\Leftarrow\; c?y.x > y \to c!y.A(0) \;+ y > x \to c!y.A(y)$$

can be replaced by

$$\begin{aligned} A\langle x\rangle &\Leftarrow c?y.x > y \to c!y.B \;+\; y > x \to c!y.A(y) \\ B &\Leftarrow c?y.0 > y \to c!y.B \;+\; y > 0 \to c!y.A(y) \end{aligned}$$

without affecting the provability relation between the original terms; these are called *equivalent definitions*, which is clarified below. The same technique may be used to eliminate multiple occurrences of the same data variable in $\overline{x}$ of any $X(\overline{x})$. So we may assume that in every declaration all occurrences of abstraction identifiers are of the form $X(\overline{x})$, where $\overline{x}$ is a vector of distinct data variables. However in a given declaration the abstraction identifiers may be of different types. We now outline two transformations which may be applied to declarations to transform them into *uniform declarations* where all declarations have exactly the same type.

The first consists of a transposition of types. For example suppose a declaration D has a definition

$$A \;\Leftarrow\; \lambda x \lambda y T$$

Then this can be replaced by a definition of the form

$$A' \;\Leftarrow\; \lambda y \lambda x (T\sigma)$$

where σ substitutes all occurrences of a term of the form $A(z_1, z_2)$ with $A'(z_2, z_1)$. Let D' be the declaration obtained from D by carrying out this replacement and in addition also applying σ to every other definition in D. Now one can show that

$$\vdash^L_{D\cup D'} \;\rhd A = A'$$

and therefore in order to establish a judgement of the form

$$\vdash^L_D b \rhd T = U$$

it is sufficient to establish

$$\vdash^L_{D'} b \rhd T\sigma = U\sigma.$$

In this case we say that D' is *equivalent* to D.

The second transformation consists of adding on an extra (dummy) parameter to the definition of an abstraction parameter. This consists in replacing a definition

$$A \;\Leftarrow\; F$$

in a declaration D by a definition of the form

$$A' \;\Leftarrow\; \lambda x (F\sigma)$$

with σ replacing all occurrences of A by $A'(z)$ where both $x, z : \iota$ are new variables. Once more if σ is systematically applied to the other definitions in D then we obtain an equivalent declaration.

$\alpha.T \stackrel{true,\alpha}{\longrightarrow} T$		$\alpha \in \{\tau, c!e \mid c \in Chan, e \in Exp\}$
$c?x.T \stackrel{true,c?y}{\longrightarrow} T[y/x]$		$y \notin fv(c?x.T)$
$T \stackrel{b',\alpha}{\longrightarrow} T'$	implies	$b \to T \stackrel{b \wedge b',\alpha}{\longrightarrow} T'$
$T \stackrel{b,\alpha}{\longrightarrow} T'$	implies	$T + U \stackrel{b,\alpha}{\longrightarrow} T'$
		$U + T \stackrel{b,\alpha}{\longrightarrow} T'$
$U[\overline{x}/\overline{z}] \stackrel{b,\alpha}{\longrightarrow} T'$	implies	$X(\overline{x}) \stackrel{b,\alpha}{\longrightarrow} T'$
		$X \Leftarrow \lambda \overline{z} U$ is a definition

Figure 7: Symbolic Operational Semantics

Definition 4.5 A restricted regular declaration $\{ X_i\langle\overline{x}_i\rangle \Leftarrow T_i \mid 1 \leq i \leq n \}$ is called *uniform* if all the $\overline{x}_i$ are of the same length and type. □

Proposition 4.6 *Every restricted regular declaration can be transformed into a uniform declaration.*

Proof: By systematic application of the above transformations. □

So for the rest of this section we can assume that we are working with respect to a uniform declaration, using a fixed sequence of variables $z_1, \ldots z_n$. For terms with respect to these kinds of definitions it is straightforward to develop a version of *symbolic semantics* as defined in [HL95a, HL95b], to which we refer the reader for details. The *symbolic operational semantics* is given in Figure 7 which uses *abstract actions* of the form $\{c?x, c!e, \tau\}$. Based on these relations we define *symbolic bisimulations*, which requires some auxiliary notation. A finite collection of boolean expressions B is called a *b-partition* if $\bigvee B = b$. For two abstract actions α, α' and a boolean b we write $\alpha =^b \alpha'$ to mean: if $\alpha \equiv c!e$ then $\alpha' \equiv c!e'$ and $b \models e = e'$; otherwise $\alpha \equiv \alpha'$. This notation generalises to vectors in the obvious way.

Let $\mathbf{S} = \{ S^b \mid b \in BExp \}$ be a family of relations over terms, indexed by boolean expressions. Then $\mathcal{LSB}(\mathbf{S})$ is the family of symmetric relations defined by:

> $(T, U) \in \mathcal{LSB}(\mathbf{S})^b$ if whenever $T \stackrel{b_1,\alpha}{\longrightarrow} T'$ with $bv(\alpha) \cap fv(b, T, U) = \emptyset$, there is a $b \wedge b_1$-partition B with the property that $fv(B) \subseteq fv(b, T, U)$ and for each $b' \in B$ there exists a $U \stackrel{b_2,\alpha'}{\longrightarrow} U'$ such that $b' \models b_2$, $\alpha =^{b'} \alpha'$ and $(T', U') \in S^{b'}$.

Definition 4.7 (Symbolic Bisimulations)
S is a (late) strong symbolic bisimulation if $\mathbf{S} \subseteq \mathcal{LSB}(\mathbf{S})$, where $\subseteq$ is point-wise inclusion. □

Let $\sim_{\mathbf{L}} = \{\sim_L^b\}$ be the largest (late) strong symbolic bisimulation.

Theorem 4.8 *(Soundness and completeness of $\sim_L$) $T \sim_L^b U$ iff $T\rho \sim_l U\rho$ for every evaluation ρ such that $\rho \models b$.*

Proof: Following the lines in the proofs of **Theorem 4.5** in [HL95b] and **Theorem 3.6** in [HL95a]. □

It can be seen from the above theorem that the free data variables appearing in $T \sim_L^b U$ is interpreted *universally*. This fact is stated in the following proposition which can be easily proved using the theorem.

Proposition 4.9 *$T \sim_L^b U$ implies $T\sigma \sim_L^{b\sigma} U\sigma$ for any data substitution σ.*

If $T \sim_L^b U$ then the definition of symbolic bisimulation requires a boolean partition for each symbolic transition from T or U. As there are only finite many such transitions (modulo α-equivalence), it is possible to find a "uniform" partition which works for *all* symbolic transitions from T or U. Here we show a slightly weaker result: There exists a "uniform" partition from all transitions from T or U which have the same type, as this is sufficient for our purpose. Symbolic actions α, β are of the same type, if either $\alpha \equiv \beta \equiv \tau$, or $\alpha \equiv \beta \equiv c?x$ for some x, or α has the form $c!e$ and β has the form $c!e'$.

Lemma 4.10 *Suppose $T \equiv \sum_{i\in I} \alpha_i.T_i$, $U \equiv \sum_{j\in J} \beta_j.U_j$, where all α_i and β_j are of the same type and $bv(\alpha_i) \cap bv(\beta_j) \cap fv(b,T,U) = \emptyset$. Then $T \sim_L^b U$ iff there exists a b-partition B with $fv(B) \subseteq fv(b,T,U)$ such that for each $b' \in B$ the following hold*

- *For each $i \in I$ there is a $j \in J$ s.t. $\alpha_i =^{b'} \beta_j$ and $T_i \sim_L^{b'} U_j$.*
- *For each $j \in J$ there is an $i \in I$ s.t. $\alpha_i =^{b'} \beta_j$ and $T_i \sim_L^{b'} U_j$.*

Proof: Since $T \sim_L^b U$, for each $i \in I$ there exists a b-partition B_i with $fv(B_i) \subseteq fv(T,\, U)$ such that for each $b_i \in B_i$, $\exists j\ \alpha_i =^{b_i} \beta_j$ and $T_i \sim_L^{b_i} U_j$; for each $j \in J$ there exists a b-partition B'_j with $fv(B'_j) \subseteq fv(b,T,U)$ such that for each $b'_j \in B'_j$, $\exists i\ \alpha_i =^{b_i} \beta_j$ and $T_i \sim_L^{b_i} U_j$.

Let D_I denote the set of booleans $\{ \wedge_{i\in I} b_i \mid b_i \in B_i \}$, D_J the corresponding set $\{ \wedge_{j\in J} b'_j \mid b'_j \in B_j \}$ and let B be the set $\{ b_1 \wedge b_2 \mid b_1 \in D_I,\, b_2 \in D_J \}$. Then $\bigvee B = b$, $fv(B) \subseteq fv(b,T,U)$ and each $b' \in B$ has the form $(\wedge_i b_i) \wedge (\wedge_j b'_j)$ with $b_i \in B_i$, $b'_j \in B'_j$. For each $i \in I$ $b' \models b_i$ for some $b_i \in B_i$, so there is a $j \in B'_j$ s.t. $\alpha_i =^{b'} \beta_j$ and $T_i \sim_L^{b'} U_j$. For each $j \in J$ $b' \models b'_j$ for some $b'_j \in B'_j$, so there is an $i \in B_i$ s.t. $\alpha_i =^{b'} \beta_j$ and $T_i \sim_L^{b'} U_j$. □

Remark 4.11 *The booleans in partition B in the above lemma can be made disjoint as follows: Suppose $B = \{ b_i \mid 1 \le i \le n \}$. Set $B' = \{ b'_i \mid 1 \le i \le n \}$ with $b'_i = b_i \wedge \bigwedge_{1\le j<i} \neg b_j$. It is easy to check that $\bigvee B' = \bigvee B$, $b'_i \wedge b'_j = false$ for any $i \neq j$, and B' enjoys the same property of B mentioned in the lemma.*

From Theorem 4.8 we now know that to prove the completeness of $\vdash^L$ it is sufficient to show

$$T \sim_L^b U \quad \text{implies} \quad \vdash_D^L b \rhd T = U$$

where T, U are defined with respect to the uniform declaration D. There are two major steps in the proof of this statement. The first reduces uniform declarations further to special *standard* forms and the second shows that if $T \sim_L^b U$ then there is a standard declaration which both T and U provably satisfy, with respect to b.

Definition 4.12 A uniform declaration $D = \{X_i\langle\overline{x}_i\rangle \Leftarrow T_i\}_{i\in I}$ is *standard* if each T_i has the form

$$\sum_{k\in K_i} b_{ik} \to \sum_{p\in P_{ik}} \alpha_{ikp}.X_{f(i,k,p)}(\overline{x}_{ikp})$$

where $\bigvee_k b_{ik} = \mathit{true}$ for each i and $b_{ik} \wedge b_{ik'} = \mathit{false}$ for $k \neq k'$. □

Proposition 4.13 *Let D be a guarded declaration. For any $T \in \mathcal{T}_D$ with $fv(T) \subseteq \overline{w}$ there is a standard declaration $E = \{X_i \Leftarrow F_i\}_{i\in I}$ such that $\vdash^L_{D\cup E} T = X_1(\overline{w})$.*

Proof: We first show that any guarded declaration $D = \{Y_i \Leftarrow G_i\}_{i\in I}$ can be transformed into a pre-standard declaration $E = \{Z_j \Leftarrow H_j\}_{j\in J}$ with $\vdash^L_{D\cup E} Y_1 = Z_1$, where each H_j is of the form

$$\lambda\overline{z}_j \sum_{l\in L_j} b_{jl} \to \alpha_{jl}.Z_{f(j,l)}(\overline{z}_{jl})$$

We illustrate the necessary rearrangements by use of an example. Let D be the declaration

$$\begin{array}{rl} Y\langle y\rangle \Leftarrow & c?x.(x \geq y \to (d!x.Y(x) + x \geq 0 \to d!(x-1).\mathbf{0})) + \\ & c!y.Y(y) \end{array}$$

As the first step, using Proposition 3.2 we can derive

$$\begin{array}{rl} \vdash^L_D Y\langle y\rangle = & c?x.(x \geq y \to d!x.Y(x) + x \geq y \wedge x \geq 0 \to d!(x-1).\mathbf{0}) + \\ & c!y.Y(y) \end{array}$$

Now let D' be

$$\begin{array}{rll} Z_1\langle y\rangle & \Leftarrow & c?x.Z_2(y,x) + c!y.Z_1(y) \\ Z_2\langle y,x\rangle & \Leftarrow & x \geq y \to d!x.Z_1(x) + x \geq y \wedge x \geq 0 \to d!(x-1).\mathbf{0} \end{array}$$

Then D' is pre-standard and $\vdash^L_{D\cup D'} Y = Z_1$ by UFI. D' can be further transformed into a uniform declaration D'':

$$\begin{array}{rll} Z_1\langle y,x\rangle & \Leftarrow & c?x.Z_2(y,x) + c!y.Z_1(y,x) \\ Z_2\langle y,x\rangle & \Leftarrow & x \geq y \to d!x.Z_1(x,y) + x \geq y \wedge x \geq 0 \to d!(x-1).\mathbf{0} \end{array}$$

Next we show that a pre-standard declaration E can be transformed into a standard one. Consider a pre-standard definition in E:

$$Z\langle\overline{x}\rangle \Leftarrow \sum_i b_i \to \alpha_i.Z_{f(i)}(\overline{x}_i).$$

Let $b_K = (\bigwedge_{k\in K} b_k) \wedge (\bigwedge_{k'\in I-K} \neg b_{k'})$ for each $K \subseteq I$. Then $\bigvee_{K\subseteq I} b_K = \mathit{true}$ and $b_K \wedge b_{K'} = \mathit{false}$ whenever $K \neq K'$. Hence

$$X\langle\overline{x}\rangle \Leftarrow \sum_{K\subseteq I} b_K \to \sum_{k\in K} \alpha_k.X_{f(k)}(\overline{x}_k)$$

is a standard definition.

Let E' be the declaration obtained by applying the above transformation to each definition in E. Then E' is standard and it is easy to see that $\vdash^L_{E\cup E'} X_1 = Z_1$.

As an illustrative example let us apply this procedure to each of the definitions in D'' above. For convenience we ignore resulting definitions where the body is guarded by the boolean *false*. The first definition $Z_1\langle y, x\rangle$ remains essentially the same, giving rise to

$$X_1\langle y, x\rangle \Leftarrow \quad c?x.X_2(y, x) + c!y.X_1(y, x)$$

while the second, $Z_2\langle y, x\rangle$, gives rise to:

$$\begin{aligned} X_2\langle y, x\rangle \Leftarrow \quad & x \geq y \wedge x \geq 0 \rightarrow (d!x.X_1(x, y) + d!(x-1).\mathbf{0}) + \\ & \neg x \geq 0 \rightarrow d!x.X_1(x, y) + \\ & \neg x \geq y \rightarrow \mathbf{0} \end{aligned}$$

For an arbitrary $T \in \mathcal{T}_D$ with $fv(T) \subseteq \overline{w}$, we can first eliminate any unguarded identifier occurrences in T by unfolding them, obtaining a term T' provably equal to T. We then add a definition $X_0 \Leftarrow \lambda\overline{w}T'$ to D, and finally transform it to standard form. □

We say two vectors of variables $\overline{x} = x_1 x_2 \ldots x_n$ and $\overline{x'} = x'_1 x'_2 \ldots x'_n$ *differ at most at* a (possibly empty) set of variables V, if $x_i = x'_i$ for any $1 \leq i \leq n$ such that $x_i \notin V$. A standard declaration $D = \{X_i\langle\overline{x}_i\rangle \Leftarrow T_i\}_{i\in I}$, where $T_i \equiv \sum_{k\in K_i} b_{ik} \rightarrow \sum_{p\in P_{ik}} \alpha_{ikp}.X_{f(i,k,p)}(\overline{x}_{ikp})$, is *parameter-saturated* if $\overline{x}_{ikp}$ and $\overline{x}_{f(i,k,p)}$ differ at most at $bv(\alpha_{ikp})$ for every $i,\ k,\ p$.

Proposition 4.14 *Every standard declaration $D = \{X_i\langle\overline{x}_i\rangle \Leftarrow T_i\}_{i\in I}$ can be transformed into an equivalent standard and parameter-saturated declaration.*

Proof: For each i and each permutation $\overline{x}'$ of $\overline{x}_i$ add a definition $X'\langle\overline{x}'\rangle \Leftarrow T_i[\overline{x}'/\overline{x}_i]$ into D, and replace each occurrence of $X_i(\overline{x}')$ by $X'(\overline{x}')$ in the enlarged declaration. Let the result be D'. Then D' is standard and parameter-saturated. Moreover it is equivalent to D. □

Proposition 4.15 *Let $D_1 = \{\overline{X} \Leftarrow \overline{G}\}$, $D_2 = \{\overline{Y} \Leftarrow \overline{H}\}$ be two standard and parameter-saturated declarations. If $X_1(\overline{z}) \sim_L^b Y_1(\overline{z})$ then there is a standard declaration $E = \{\overline{Z} \Leftarrow \overline{F}\}$ such that $\vdash^L_{D_1\cup E} b \rhd X_1(\overline{z}) = Z_{11}(\overline{z})$ and $\vdash^L_{D_2\cup E} b \rhd Y_1(\overline{z}) = Z_{11}(\overline{z})$.*

Proof: Let

$$\begin{aligned} X_i\langle\overline{z}\rangle &\Leftarrow \sum_{k\in K_i} c_{ik} \rightarrow \sum_{p\in P_{ik}} \alpha_{ikp}.X_{f(i,k,p)}(\overline{x}_{ikp}) \\ Y_j\langle\overline{z}\rangle &\Leftarrow \sum_{l\in L_j} d_{jl} \rightarrow \sum_{q\in Q_{jl}} \beta_{jlq}.Y_{g(j,l,q)}(\overline{y}_{jlq}) \end{aligned}$$

We assume that all input prefixes in D_1, D_2 use the same data variable z which is different from any other data variables used so far.

For each pair $(i,\ j)$, let b_{ij} with $fv(b_{ij}) \subseteq \overline{z}$ be such that $X_i(\overline{z}) \sim_L^{b_{ij}} Y_j(\overline{z})$ and for any b' with $X_i(\overline{z}) \sim_L^{b'} Y_j(\overline{z})$ it holds $b' \models b_{ij}$. In particular, $b \models b_{11}$. So by CONSEQ to prove the proposition we only need to show $\vdash^L_{D_1\cup E} b_{11} \rhd X_1(\overline{z}) = Z_{11}(\overline{z})$ and $\vdash^L_{D_2\cup E} b_{11} \rhd Y_1(\overline{z}) = Z_{11}(\overline{z})$.

Let $b_{ikjl} = c_{ik} \wedge d_{jl} \wedge b_{ij}$. Since $X_i(\overline{z}) \sim_L^{b_{ij}} Y_j(\overline{z})$, $c_{ik} \wedge c_{ik'} = \mathit{false}$ for $k \neq k'$, $d_{jl} \wedge d_{jl'} = \mathit{false}$ for $l \neq l'$, we know

$$T_{ik} \sim_L^{b_{ikjl}} U_{jl}$$

where

$$\begin{aligned} T_{ik} &\equiv \sum_{p \in P_{ik}} \alpha_{ikp}.X_{f(i,k,p)}(\overline{x}_{ikp}) \\ U_{jl} &\equiv \sum_{q \in Q_{jl}} \beta_{jlq}.Y_{g(j,l,q)}(\overline{y}_{jlq}) \end{aligned}$$

According to the types of the prefixes we group the summands of T_{ik} into T^τ, $T^{c!}$, $T^{c?}$ for each $c \in \mathit{Chan}(T_{ik})$. We have

$$\vdash_{D_1}^L T_{ik} = T^\tau + \sum_c T^{c!} + \sum_c T^{c?}.$$

Similarly for U_{jl}:

$$\vdash_{D_2}^L U_{jl} = U^\tau + \sum_c U^{c!} + \sum_c U^{c?}.$$

We then have $T^\tau \sim_L^{b_{ikjl}} U^\tau$ and $T^{c!} \sim_L^{b_{ikjl}} U^{c!}$, $T^{c?} \sim_L^{b_{ikjl}} U^{c?}$ for each $c \in \mathit{Chan}(T_{ik})$.

Let B^τ, $B^{c!}$, $B^{c?}$ be the b_{ikjl}-partitions guaranteed by Lemma 4.10 for the above symbolic bisimulations.

For each $b' \in B^\tau$ define

$$I_{b'}^\tau = \{\, (p,q) \mid X_{f(i,k,p)}(\overline{x}_{ikp}) \sim_L^{b'} Y_{g(j,l,q)}(\overline{y}_{jlq}), \overline{x}_{ikp} \equiv \overline{y}_{jlq} \,\}.$$

Similarly define $I_{b'}^{c!}$ for $b' \in B^{c!}$ and $I_{b'}^{c?}$ for $b' \in B^{c?}$. $I_{b'}^\tau$, $I_{b'}^{c!}$ and $I_{b'}^{c?}$ are total and surjective because of the property enjoyed by B^τ, $B^{c!}$, $B^{c?}$ and the fact that D_1 and D_2 are parameter-saturated.

Let $\{\, Z_{ij} \mid 1 \leq i \leq m,\ 1 \leq j \leq n \,\}$ be a set of new process identifiers. Also write $\overline{z}_{ikpjlq}$ for $\overline{x}_{ikp} \equiv \overline{y}_{jlq}$. Consider the standard declaration

$$E = \{Z_{ij}\langle\overline{z}\rangle \Leftarrow b_{ij} \rightarrow \sum_{k,l} c_{ik} \wedge d_{jl} \rightarrow (V^\tau + \sum_c V^{c!} + \sum_c v^{c?})\}$$

where

$$\begin{aligned} V^\tau &= \sum_{b' \in B^\tau} b' \rightarrow V_{b'}^\tau \\ V_{b'}^\tau &= \sum_{(p,q) \in I_{b'}^\tau} \tau.Z_{f(i,k,p)g(j,l,q)}(\overline{z}_{ikpjlq}) \\ V^{c!} &= \sum_{b' \in B^{c!}} b' \rightarrow V_{b'}^{c!} \\ V_{b'}^{c!} &= \sum_{(p,q) \in I_{b'}^{c!}} \alpha_{ikp}.Z_{f(i,k,p)g(j,l,q)}(\overline{z}_{ikpjlq}) \\ V^{c?} &= \sum_{b' \in B^{c?}} b' \rightarrow V_{b'}^{c?} \\ V_{b'}^{c?} &= \sum_{(p,q) \in I_{b'}^{c?}} c?z.Z_{f(i,k,p)g(j,l,q)}(\overline{z}_{ikpjlq}) \end{aligned}$$

We are to prove that, for each i, j

$$\vdash_{D_1}^L b_{ij} \rhd X_i(\overline{z}) = (\sum_{k,l} c_{ik} \wedge d_{jl} \rightarrow (V^\tau + \sum_c V^{c!} + \sum_c V^{c?}))\theta \qquad (1)$$

where $\theta \equiv [\lambda\overline{z}(b_{ij} \to X_i(\overline{z}))/Z_{ij}|i,j]$. If this can be done then by UFI-O we obtain the required

$$\vdash^L_{D_1 \cup E} b_{11} \rhd X_1(\overline{z}) = Z_{11}(\overline{z}).$$

By Proposition 3.2, (1) is equivalent to

$$\vdash^L_{D_1} b_{ij} \rhd b_{ij} \to X_i(\overline{z}) = \sum_{k,l} c_{ik} \wedge d_{jl} \wedge b_{ij} \to (V^\tau + \sum_c V^{c!} + \sum_c V^{c?})\theta. \tag{2}$$

Since $\bigvee_{k,l}(c_{ik} \wedge d_{jl}) = (\bigvee_k c_{ik}) \wedge (\bigvee_l d_{jl}) = true$,

$$\vdash^L_{D_1} b_{ij} \rhd b_{ij} \to X_i(\overline{z}) = \sum_{k,l} c_{ik} \wedge d_{jl} \wedge b_{ij} \to T_{ik}.$$

So (2), hence (1), will hold if we can show

$$\vdash^L_{D_1} b_{ij} \rhd \sum_{k,l} b_{ikjl} \to T_{ik} = \sum_{k,l} b_{ikjl} \to (V^\tau + \sum_c V^{c!} + \sum_c V^{c?})\theta.$$

Since both $\{ c_{ik} \mid k \}$ and $\{ d_{jl} \mid l \}$ are sets of disjoint booleans, this reduces to: for each k, l

$$\vdash^L_{D_1} b_{ikjl} \rhd T_{ik} = (V^\tau + \sum_c V^{c!} + \sum_c V^{c?})\theta$$

which further reduces to

$$\vdash^L_{D_1} b_{ikjl} \rhd T^\tau = V^\tau\theta \tag{3}$$
$$\vdash^L_{D_1} b_{ikjl} \rhd T^{c!} = V^{c!}\theta \tag{4}$$
$$\vdash^L_{D_1} b_{ikjl} \rhd T^{c?} = V^{c?}\theta \tag{5}$$

for each $c \in Chan(T_{ik})$.

We first consider (5).

Now

$$V^{c?}\theta \equiv \sum_{b' \in B^{c?}} b' \to V^{c?}_{b'}\theta. \tag{6}$$

By the construction of $V^{c?}_{b'}$, for each $(p,q) \in I^{c?}_{b'}$ it holds that

$$X_{f(i,k,p)}(\overline{z}_{ikpjlq}) \sim^{b'}_L Y_{g(j,l,q)}(\overline{z}_{ikpjlq}).$$

By Proposition 4.9

$$X_{f(i,k,p)}(\overline{z}) \sim^{b'[\overline{z}/\overline{z}_{ikpjlq}]}_L Y_{g(j,l,q)}(\overline{z}).$$

By the definition of b_{ij}

$$b'[\overline{z}/\overline{z}_{ikpjlq}] \models b_{f(i,k,p)g(j,l,q)}.$$

Hence

$$b' \models b_{f(i,k,p)g(j,l,q)}[\overline{z}_{ikpjlq}/\overline{z}]. \tag{7}$$

Therefore

$$\begin{aligned}
\vdash^L_{D_1}\ b'\rhd\ & V^{c?}_{b'}\theta \\
= & \Big(\sum_{(p,q)\in I^{c?}_{b'}} c?z.Z_{f(i,k,p)g(j,l,q)}(\overline{z}_{ikpjlq})\Big)\theta \\
= & \sum_{(p,q)\in I^{c?}_{b'}} c?z.(b_{f(i,k,p)g(j,l,q)}[\overline{z}_{ikpjlq}/\overline{z}] \to X_{f(i,k,p)}(\overline{z}_{ikpjlq})) \\
= & \sum_{(p,q)\in I^{c?}_{b'}} b' \to c?z.(b_{f(i,k,p)g(j,l,q)}[\overline{z}_{ikpjlq}/\overline{z}] \to X_{f(i,k,p)}(\overline{z}_{ikpjlq})) \\
= & \sum_{(p,q)\in I^{c?}_{b'}} b' \to c?z.(b' \to b_{f(i,k,p)g(j,l,q)}[\overline{z}_{ikpjlq}/\overline{z}] \to X_{f(i,k,p)}(\overline{z}_{ikpjlq})) \\
\overset{(7)}{=} & \sum_{(p,q)\in I^{c?}_{b'}} b' \to c?z.(b' \to X_{f(i,k,p)}(\overline{z}_{ikpjlq})) \\
= & \sum_{(p,q)\in I^{c?}_{b'}} b' \to c?z.X_{f(i,k,p)}(\overline{z}_{ikpjlq}) \\
= & \sum_{(p,q)\in I^{c?}_{b'}} c?z.X_{f(i,k,p)}(\overline{z}_{ikpjlq}).
\end{aligned}$$

Since $I^{c?}_{b'}$ is total $T^{c?}$ can be obtained by duplicating and reordering the summands of the last line above, using $S2,\ S3,\ S4$. This means

$$\vdash^L_{D_1}\ b'\rhd V^{c?}_{b'}\theta = T^{c?}. \tag{8}$$

Therefore

$$\begin{aligned}
\vdash^L_{D_1}\ & V^{c?}\theta \\
\overset{(6)}{=} & \sum_{b'\in B^{c?}} b' \to V^{c?}_{b'}\theta \\
\overset{(8)}{=} & \sum_{b'\in B^{c?}} b' \to T^{c?} \\
\overset{3.2}{=} & b_{ikjl} \to T^{c?},
\end{aligned}$$

Using Proposition 3.2 again we then obtain

$$\vdash^L_{D_1}\ b_{ikjl}\rhd V^{c?}\theta = T^{c?},$$

which is the required (5) above. The proofs for (3) and (4) are similar.

This completes the proof of (1). In a symmetric way we can prove $\vdash^L_{D_2\cup E}\ b\rhd Y_1(\overline{z}) = Z_{11}(\overline{z})$. □

Theorem 4.16 *(Completeness of $\vdash^L$) Let $T,\ U\in\mathcal{T}_D$, D a guarded declaration. Then $T\sim^b_L U$ implies $\vdash^L_D\ b\rhd T=U$.*

Proof: Let $fv(T,U)=\overline{z}$. By Propositions 4.13 and 4.14 there exist standard and parameter-saturated declarations $D_1=\{\overline{X}\Leftarrow\overline{G}\}$ and $D_2=\{\overline{Y}\Leftarrow\overline{H}\}$ such that $\vdash^L_{D\cup D_1} T=X_1(\overline{z}),\ \vdash^L_{D\cup D_2} U=Y_1(\overline{z})$.

By Proposition 4.15 there exists a standard declaration $E=\{\overline{Z}\Leftarrow\overline{F}\}$ such that

$$\begin{aligned}
&\vdash^L_{D\cup D_1\cup E}\ b\rhd X_1(\overline{z}) = Z_{11}(\overline{z}) \\
&\vdash^L_{D\cup D_2\cup E}\ b\rhd Y_1(\overline{z}) = Z_{11}(\overline{z})
\end{aligned}$$

Hence $\vdash^L_{D\cup D_1\cup D_2\cup E}\ b \rhd X_1(\overline{z}) = Y_1(\overline{z})$, and so $\vdash^L_{D\cup D_1\cup D_2\cup E}\ b \rhd T = U$. By dec-E $\vdash^L_D\ b \rhd T = U$. □

5 Early Bisimulation

The theory we studied so far is for late bisimulation. In this section we will outline how it can be carried over to the early case with some systematic modifications. It is not surprising that only the parts involving input actions need changing.

The early operational semantics of our language can be obtained by simple changes to the rules in Figure 2. The rule for input transitions is changed to

$$c?F \xrightarrow{c?v}_e Fv \qquad\qquad v \in Val$$

and the communication rule becomes

$$P \xrightarrow{c?v}_e P',\ Q \xrightarrow{c!v}_e Q' \text{ implies } P \mid Q \xrightarrow{\tau}_e P' \mid Q'.$$

In the remaining rules we simply replace $\longrightarrow_l$ by $\longrightarrow_e$. Note that in the early semantics the actions are of the form $\{\tau, c!v, c?v\}$ and the type of input transitions are now from *processes* to *processes*.

Definition 5.1 A symmetric relation R between data–closed terms is a strong early bisimulation if it satisfies: $(P,Q) \in R$ implies that

$$\text{whenever } P \xrightarrow{a}_e P' \text{ then there exists } Q \xrightarrow{a}_e Q' \text{ and } (P',Q') \in R$$

We use $\sim_e$ to denote the largest early strong bisimulation. □

This relation generalizes naturally to open terms by letting $T \sim_e U$ iff $T\rho \sim_e U\rho$ for any ρ, and to abstractions by letting $F \sim_e G$ iff $F\overline{x} \sim_e G\overline{x}$.

For early symbolic operational semantics we use the same set of rules as in Figure 7. Early symbolic bisimulation is defined similarly as in the late case:

Let $\mathbf{S} = \{ S^b \mid b \in BExp \}$ be a family of relations over terms, indexed by boolean expressions. Then $\mathcal{ESB}(\mathbf{S})$ is the family of symmetric relations defined by:

> $(T,U) \in \mathcal{ESB}(\mathbf{S})^b$ if whenever $T \xrightarrow{b_1,\alpha} T'$ with $bv(\alpha) \cap fv(b,T,U) = \emptyset$, there is a $b \wedge b_1$-partition B with the property that $fv(B) \subseteq bv(\alpha) \cup fv(b,T,U)$ and for each $b' \in B$ there exists some $U \xrightarrow{b_2,\alpha'} U'$ such that $b' \models b_2$, $\alpha =^b \alpha'$ and $(T',U') \in S^{b'}$.

Definition 5.2 (Early Symbolic Bisimulations)
$\mathbf{S}$ is a early strong symbolic bisimulation if $\mathbf{S} \subseteq \mathcal{ESB}(\mathbf{S})$. □

Let $\sim_{\mathbf{E}} = \{\sim^b_E\}$ be the largest late strong symbolic bisimulation. Note that the only difference between early and late symbolic bisimulations is in the restriction on the free data variables of the partition B when matching an input action: in the early case B is allowed to have the input variable free, so that the value space for the input variable can be partitioned; while in the late case this is forbidden.

Theorem 5.3 *(Soundness and completeness of $\sim_E$)* $T \sim_E^b U$ *iff* $T\rho \sim_e U\rho$ *for every evaluation* ρ *such that* $\rho \models b$.

Proposition 4.9 and Lemma 4.10 all hold in this new setting, with the expected modification that in Lemma 4.10 it is now required $fv(B) \subseteq bv(\alpha_i) \cup fv(b, T, U)$ instead of $fv(B) \subseteq fv(b, T, U)$.

The inference system for early bisimulation can be obtained by generalising that for late in two different ways: adding the following axiom schema

$$\text{EA} \quad c?x.T + c?x.U = c?x.T + c?x.U + c?x.(b \to T \ + \ \neg b \to U)$$

or replacing the INPUT rule by the more general rule schema

$$\text{E-INPUT} \ \frac{b \rhd \sum_{i\in I} \tau.T_i = \sum_{j\in J} \tau.U_j}{b \rhd \sum_{i\in I} c?x.T_i = \sum_{j\in J} c?x.U_j} \quad x \notin fv(b).$$

It is easy to see that both EA and E-INPUT are sound with respect to early bisimulation. EA is adapted from Parrow and Sangiorgi's axiomatisation for early bisimulation of the π-calculus ([PS93]), while E-INPUT was used in our earlier work on proof systems for recursion–free message–passing processes ([HL95a]). EA can be derived from E-INPUT in the presence of other proof rules.

In what follows we will use the EA extension. So let us write $\vdash_D^E b \rhd T = U$ if $b \rhd T = U$ can be derived from this new inference system.

First we have a generalised form of EA:

Proposition 5.4 *For any finite non-empty collection of booleans* $\{ b_i \mid i \in I \}$ *such that* $\bigvee_{i\in I} b_i = true$ *and* $b_i \wedge b_j = false$ *for* $i \neq j$,

$$\vdash_D^E \sum_{i\in I} c?x.T_i = \sum_{i\in I} c?x.T_i + c?x.\sum_{i\in I} b_i \to T_i.$$

The proof of this Proposition can be found in [HL95a].

To obtain the completeness result for $\vdash_D^E$, only the input case in the proof of Proposition 4.15 needs modifying. Since now z may appear free in $fv(B^{c?})$, the definition of $V^{c?}$ has to be changed. Let

$$\begin{aligned} V^{c?} &= V_1^{c?} + V_2^{c?} \\ V_1^{c?} &= \sum_{\alpha_{ikp} \equiv c?z} \alpha_{ikp}. \sum_{\substack{b' \in B^{c?} \\ (p,q) \in I_{b'}^{c?}}} b' \to Z_{f(i,k,p)g(j,l,q)}(\overline{z}_{ikpjlq}) \\ V_2^{c?} &= \sum_{\beta_{jlq} \equiv c?z} \beta_{jlq}. \sum_{\substack{b' \in B^{c?} \\ (p,q) \in I_{b'}^{c?}}} b' \to Z_{f(i,k,p)g(j,l,q)}(\overline{z}_{ikpjlq}) \end{aligned}$$

Now to show (5), *i.e.*

$$\vdash_{D_1}^E b_{ikjl} \rhd T^{c?} = V^{c?}\theta,$$

We argue as follows: we know

$$T^{c?} \equiv \sum_{\{\,\alpha_{ikp}\equiv c?z \mid p\in P_{ik}\,\}} \alpha_{ikp}.X_{f(i,k,p)}(\overline{z}_{ikpjlq})$$

and we have (7) by the same argument as in the proof of Proposition 4.15. Hence

$$\begin{aligned}
\vdash^E_{D_1}\ & V_1^{c?}\theta \\
= \ & \Big(\sum_{\alpha_{ikp}\equiv c?z} \alpha_{ikp}. \sum_{\substack{b'\in B^{c?}\\(p,q)\in I^{c?}_{b'}}} b' \to Z_{f(i,k,p)g(j,l,q)}(\overline{z}_{ikpjlq})\Big)\theta \\
= \ & \sum_{\alpha_{ikp}\equiv c?z} \alpha_{ikp}. \sum_{\substack{b'\in B^{c?}\\(p,q)\in I^{c?}_{b'}}} b' \to (b_{f(i,k,p)g(j,l,q)}[\overline{z}_{ikpjlq}/\overline{z}_{f(i,k,p)g(j,l,q)}] \to X_{f(i,k,p)}(\overline{z}_{ikpjlq})) \\
\overset{(7)}{=} \ & \sum_{\alpha_{ikp}\equiv c?z} \alpha_{ikp}. \sum_{\substack{b'\in B^{c?}\\(p,q)\in I^{c?}_{b'}}} b' \to X_{f(i,k,p)}(\overline{z}_{ikpjlq}) \\
= \ & \sum_{\alpha_{ikp}\equiv c?z} \alpha_{ikp}.(b_{ikjl} \to X_{f(i,k,p)}(\overline{z}_{ikpjlq}))
\end{aligned}$$

The last step of the above derivation uses the fact that $I^{c?}_{b'}$ is total.

Similarly we can derive

$$\vdash^E_{D_1} V_2^{c?}\theta = \sum_{\beta_{jlq}\equiv c?z} \beta_{jlq}. \sum_{\substack{b'\in B^{c?}\\(p,q)\in I^{c?}_{b'}}} b' \to X_{f(i,k,p)}(\overline{z}_{ikpjlq})$$

Now for each q such that $\beta_{jlq} \equiv c?z$, we know that $\bigvee B^{c?} = b_{ikjl}$. Also by Remark 4.11 we may assume the booleans in $B^{c?}$ are mutual disjoint. And, finally, we know $I^{c?}_{b'}$ is surjective. So we can apply Proposition 5.4 to obtain

$$\begin{aligned}
\vdash^E_{D_1}\ b_{ikjl}\rhd \ & \sum_{\alpha_{ikp}\equiv c?z} \alpha_{ikp}.X_{f(i,k,p)}(\overline{z}_{ikpjlq}) = \\
& \sum_{\alpha_{ikp}\equiv c?z} \alpha_{ikp}.X_{f(i,k,p)}(\overline{z}_{ikpjlq}) + \beta_{jlq}. \sum_{\substack{b'\in B^{c?}\\(p,q)\in I^{c?}_{b'}}} b' \to X_{f(i,k,p)}(\overline{z}_{ikpjlq})
\end{aligned}$$

Repeating this process for each q we obtain

$$\vdash^E_{D_1}\ b_{ikjl} \rhd V_1^{c?}\theta = V_1^{c?}\theta + V_2^{c?}\theta$$

Because $z \notin fv(b_{ikjl})$, from this (5) follows immediately:

$$\begin{aligned}
\vdash^E_{D_1}\ b_{ikjl} \rhd V^{c?}\theta \ &= \sum_{\alpha_{ikp}\equiv c?z} \alpha_{ikp}.(b_{ikjl} \to X_{f(i,k,p)}(\overline{z}_{ikpjlq})) \\
&= \sum_{\alpha_{ikp}\equiv c?z} \alpha_{ikp}.X_{f(i,k,p)}(\overline{z}_{ikpjlq}) \\
&= T^{c?}
\end{aligned}$$

This completes the proof for the early version of Proposition 4.15, thus giving the completeness result for the early case:

Theorem 5.5 *(Completeness of $\vdash^E$) Let D be a guarded declaration and $T,\ U \in \mathcal{T}_D$. Then $T \sim_E^b U$ implies $\vdash_D^E b \rhd T = U$.*

6 Conclusions

In this paper we have suggested one method, based on Unique Fixpoint Induction, of extending the proof system of [HL95a] to recursively defined message-passing processes. We have limited ourselves to considering strong bisimulation equivalence, in its two varieties of early and late, and guarded recursive definitions. We believe that these results can be extended to the *weak* versions of early and late bisimulation congruences by adding the appropriate τ-laws [Mil89], again as in [HL95a]. We have developed an interactive verification tool *VPAM* ([Lin93]) based on our theoretical results, where the implementation of the unique fixpoint induction rule demands special care [Lin95b]. All the examples in this paper have been checked in *VPAM*.

However extending the results to *unguarded* definitions is more complicated. For example with pure processes, if we have the definitions

$$\begin{aligned} X &\Leftarrow X + T \\ Y &\Leftarrow T \end{aligned}$$

in the declaration D then the sound rule

$$\frac{}{\vdash_D \ \rhd X = Y}$$

can be used to convert all unguarded regular declarations to guarded ones. However a simple generalisation of this rule is no longer sound for value-passing processes. For example consider the definitions

$$\begin{aligned} X\langle x,y\rangle &\Leftarrow X(y,x) + c!x.c?z.X(y,z) \\ Y\langle x,y\rangle &\Leftarrow c!x.c?z.Y(y,z). \end{aligned}$$

It is not in general true that $X \sim_l Y$. For example $X(1,0)$ can immediately perform the actions $c!0$ and $c!1$ whereas $Y(1,0)$ can only perform $c!1$. In order to obtain a guarded definition equivalent to X we instead have to use the abstraction defined by

$$Y'\langle x,y\rangle \ \Leftarrow \ c!x.c?z.Y'(y,z) + c!y.c?z.Y'(x,z).$$

A general rule is somewhat complicated to formulate. An unguarded definition of the form

$$X \ \Leftarrow \ \lambda\overline{x}(T + \sum_{i\in I} X(\overline{x}_i)) \qquad X \text{ guarded in } T$$

where each $\overline{x}_i$ is a permutation of $\overline{x}$, can be replaced by a guarded definition of the form

$$X \ \Leftarrow \ \lambda\overline{x}\sum\{\, T\sigma \mid \sigma \in Perm_{\overline{x}}(\{\, \overline{x}_i \mid i \in I \,\})\,\}$$

where $Perm_{\overline{x}}(\{\, \overline{x}_i \mid i \in I \,\})$ is the set of permutations of $\overline{x}$ generated by $\overline{x}_i,\ i \in I$, under permutation composition, and we identify a permutation $\overline{x}'$ of $\overline{x}$ with the substitution

$[\overline{x}'/\overline{x}]$. Note that $Perm_{\overline{x}}(\{\,\overline{x}_i \mid i \in I\,\})$ is always finite and includes $\overline{x}$. Returning to the above example, since $Perm_{(x,y)}(\{(y,x)\}) = \{(x,y),(y,x)\}$, applying this rule to the definition discussed before

$$X\langle x,y\rangle \;\Leftarrow\; X(y,x) + c!x.c?z.X(y,z)$$

we get

$$\begin{aligned} X\langle x,y\rangle &= (c!x.c?z.X(y,z))[x,y/x,y] + (c!x.c?z.X(y,z))[y,x/x,y] \\ &= c!x.c?z.X(y,z) + c!y.c?z.X(x,z). \end{aligned}$$

Despite these extensions there is nevertheless an inherent limitation on proof systems whose only mechanism for deriving judgements on recursive processes is Unique Fixpoint Induction. For example consider the following process declaration over natural numbers

$$\begin{aligned} S &\Leftarrow c?x.d!0.S \\ R &\Leftarrow c?x.D(x) \\ D\langle x\rangle &\Leftarrow x = 0 \to d!0.R \;+\; x \neq 0 \to D(x-1). \end{aligned}$$

The process S inputs any natural number and immediately outputs 0. R, on the other hand, inputs a number, counts down the number to 0 and then outputs 0. It is apparent that these two processes are semantically equivalent but they can not be proved equivalent in our proof system, or indeed in any straightforward extension of it. Intuitively the semantic equivalence between these two processes depends on the inductive nature of the natural numbers which is not reflected anywhere in our proof system. Formally the judgements of our proof system are satisfied in any model of the natural numbers, including non-standard ones; however S and R are not semantically equivalent when the data expressions are interpreted in a non-standard model and therefore $S = R$ can not be a judgement of the system.

In order to develop proof systems in which judgements such as $S = R$ can be derived it seems necessary to have the ability, within the proof system, to derive statements of type process using induction over the data domain. One way to introduce this type of induction, for example for the natural numbers, is as follows:

Assume $T, U : N \to \mathit{process}$. Then

$$\frac{\begin{array}{rl} & \vdash_D \; T(0) = U(0) \\ T(n) = U(n) & \vdash_D \; T(n+1) = U(n+1) \end{array}}{\vdash_D \; T = U}$$

With such a rule one can derive judgements such as $\vdash_D S = R$ although Unique Fixpoint Induction is also required.

One possible method for implementing such a proof system is using a general purpose theorem prover such as *Isabelle* ([Pau94]) and COQ ([DFH+93]). A new type of object would be required, called *process*, and this type would have associated with it particular proof rules, essentially those of our proof system, Figures 3 and 4. Then the extra inductive proof strategies based on the data domains would be automatically inherited from the representation of the data within the general purpose theorem prover.

Nevertheless we conjecture that our restricted proof system will still of considerable use for a large class of problems, particularly those connected with protocol specification where by and large relatively simple use is made of the data being transmitted and received. However this conjecture can only be tested by examining a wide range of case studies and seeing where it is necessary to have the power of induction over the data domains.

This paper has concentrated entirely on extending the approach of [HL95a] to recursively defined message passing processes but we end with some brief pointers to some other approaches to handling data dependent processes. Standard techniques from the theory of algebraic specifications are used in [GP90] to develop a modularised algebraic approach to the process language ACP augmented with message-passing while in [CR94] the theory of abstract interpretation is brought to bear on a language very similar to which we have considered. A much more practical approach, based on similar ideas, is taken in [YY91] to verifying ADA programs. Finally [CGL92] contains an instance of the use of abstraction in model checking.

While this paper was rewritten one of the authors formulated a version of unique fixpoint induction for the π-calculus ([Lin95a]). The **fix** operator is used there and the UFI rule appears the same as in the pure-CCS [Mil89]:

$$\frac{F = G[F/X]}{F = \mathbf{fix}XG}$$

The only difference is that this rule works at a more abstract level: here F, G are (closed) abstractions. In the present work we have used definitions and declarations instead of the **fix** operator, because one objective of the current work is to provide a theoretical basis for practical applications, and recursive definitions with process constants are easier to used in applications than the **fix** operator.

Acknowledgement: The authors would like to thank Julian Rathke for reading a draft of this paper and making many useful suggestions for improvement.

References

[BK88] J. Bergstra and J.W. Klop. A complete inference system for regular processes with silent moves. In *Proceedings, Logic Colloquium'86*, pages 21–81. North-Holland, 1988.

[CGL92] E. Clarke, O. Grumberg, and D. Long. Model checking and abstraction. In *POPL'92*, pages 343 – 354. ACM Press, 1992.

[CPS89] R. Cleaveland, J. Parrow, and B. Steffen. A semantics based verification tool for finite state systems. In *Proceedings of the* 9^{th} *International Symposium on Protocol Specification, Testing and Verification*, North Holland, 1989.

[CR94] R. Cleaveland and J. Riely. Testing-based abstractions for value-passing systems. In *CONCUR'94*, number 836 in Lecture Notes in Computer Science, pages 417 – 432. Springer–Verlag, 1994.

[DFH+93] G. Dowek, A. Felty, H. Herbelin, G. Huet, C. Murthy, C. Parent, C. Paulin-Mohring, and B. Werner. The Cog proof asistant user's guide, version 5.8. Report, INRIA-Rocquencourt and CNRS-ENS Lyon, 1993.

[GP90] L.F. Groote and A. Ponse. The syntax and semantics of μCRL. Report CS-R9076, CWI, Amsterdam, 1990.

[HL95a] M. Hennessy and H. Lin. Proof systems for message-passing process algebras. *To appear in Formal Aspects of Computing*, 1995. Extended abstract in CONCUR'93, LNCS 715, pp. 202-216, 1993.

[HL95b] M. Hennessy and H. Lin. Symbolic bisimulations. *Theoretical Computer Science*, 138:353 – 389, 1995.

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

[Lin95a] H. Lin. Unique fixpoint induction for mobile processes. In *CONCUR'95*, volume 962 of *Lecture Notes in Computer Science*, pages 88–102. Springer–Verlag, 1995.

[Lin95b] H. Lin. On implementing unique fixpoint induction for value-passing processes. In *Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, pages 104–118. Aarhus, Denmark, May 1995.

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

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

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

[Pau94] L.C. Paulson. *Isabelle: A generic theorem prover*, volume 828 of *Lecture Notes in Computer Science*. Springer–Verlag, 1994.

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

[Sto88] A. Stoughton. *Fully Abstract Models of Programming Languages*. Research Notes in Theoretical Computer Science, Pitman/Wiley, 1988.

[YY91] W.J. Yeh and M. Young. Compositional reachability analysis using process algebra. In *TAV'91*, pages 49 – 59. ACM SIGSOFT, ACM Press, 1991.