Unique Fixpoint Induction for Message–Passing Process Calculi*

M. Hennessy†
School of Cognitive and Computing Sciences
University of Sussex, England
matthewh@cogs.susx.ac.uk

H. Lin‡
Institute of Software
Chinese Academy of Sciences, Beijing
lhm@ox.ios.ac.cn

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 more powerful extensions with inductive inference rules.

Keywords Process algebra, Message passing, Bisimulation, Proof system, Unique Fixpoint Induction, Completeness, Symbolic semantics.

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, [HL95a, HL96], 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 [HL95a] 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 [HL96] 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

$$b \rhd T = U$$

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. 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

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

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

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

Here $c!e.T$ denotes a process which sends the value of the message expression e along the channel

*Supported by the EU KIT Project SYMSEM.
†Also supported by EPSRC grant no. GR/K60701
‡Also supported by the National Science Foundation of China.

Proceedings of CATS'97 (Computing: The Australasian Theory Symposium), Sydney, Australia, February 3–4 1997.

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.

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*, [Mil84]. 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 [HL96] 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

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

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

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

This can easily be proved using the proof system of [HL96], if we are allowed a *Recursion Unfolding* rule for recursively defined processes, whereby the term Q can be freely rewritten to its definition $c?x.\ \textit{if } x < 0 \textit{ then } d!(-x).Q \textit{ else } d!x.Q$; the proof relies on two facts about the domain of messages,

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

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 Induction rule above from the judgement

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

This in turn can be established by first unfolding the definition of $P(x)$ then applying the OUTPUT rule mentioned above.

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

$$\frac{}{\mathbf{0} : process} \qquad \frac{T : process,\ b : bool}{b \rightarrow T : process}$$

$$\frac{X \in ID_\pi}{X \in \pi} \qquad \frac{T : process,\ c : Chan}{T \backslash c : process}$$

$$\frac{T,\ U : process}{T + U : process} \qquad \frac{F : \iota \rightarrow process,\ c : Chan_\iota}{c?F : process}$$

$$\frac{T,\ U : process}{T | U : process} \qquad \frac{T : process,\ e : \iota,\ c : Chan_\iota}{c!e.T : process}$$

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

Figure 1: The Formation Rules

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.

Due to space limitation most proofs have been omitted. They can be found in the full version of the paper [HL95b].

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 $c?\lambda xT$.

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 $fv(T)$. We 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 $e\sigma$.

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 $[F_i/X_i | 1 \le i \le m]$, is the abstraction substitution that sends X_i to F_i for $1 \le i \le m$. 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.

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. 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, HL96] 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 $\iota_1 \rightarrow \iota_2 \ldots \iota_n \rightarrow process$, 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

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

$$\frac{}{\tau.P \xrightarrow{\tau}_l P}$$

$$\frac{}{c!e.P \xrightarrow{c!v}_l P} \llbracket e \rrbracket = v$$

$$\frac{}{c?.F \xrightarrow{c?}_l F}$$

$$\frac{P \xrightarrow{a}_l P', \llbracket b \rrbracket = true}{b \to P \xrightarrow{a}_l P'}$$

$$\frac{P \xrightarrow{a}_l P'}{P \mid Q \xrightarrow{a}_l P' \mid Q}$$

$$\frac{P \xrightarrow{c?}_l F,\ Q \xrightarrow{c!v}_l Q'}{P \mid Q \xrightarrow{\tau}_l Fv \mid Q'}$$

$$\frac{P \xrightarrow{a}_l P'}{P + Q \xrightarrow{a}_l P'}$$

$$\frac{P \xrightarrow{a}_l P', c \notin Chan(a)}{P \backslash c \xrightarrow{a}_l P' \backslash c}$$

$$\frac{F\overline{e} \xrightarrow{a}_l P}{X(\overline{e}) \xrightarrow{a}_l P} \quad X \Leftarrow F \text{ is a definition}$$

$$\frac{T[e/X] \xrightarrow{a}_l P}{(\lambda x T)e \xrightarrow{a}_l P}$$

Figure 2: Late Operational Semantics

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.

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, HL95a]. The main body of this paper will concentrate on late bisimulation but we will outline briefly in Section 5 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 $\mid$ 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*.

Definition 2.1 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

$$\text{EQN} \quad \frac{}{\vdash_D T\theta = U\theta} \, T = U \text{ is an axiom}$$

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

$$\alpha\text{-CONV} \quad \frac{}{\vdash_D T = U} \, T \equiv_\alpha U$$

$$\text{INPUT} \quad \frac{\vdash_D b \rhd T = U}{\vdash_D b \rhd c?x.T = c?x.U} \, x \notin fv(b)$$

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

$$\text{TAU} \quad \frac{\vdash_D b \rhd T = U}{\vdash_D b \rhd \tau.T = \tau.U}$$

$$\text{GUARD} \quad \frac{\vdash_D b \wedge b' \rhd T = U \ \ \vdash_D b \wedge \neg b' \rhd \mathbf{0} = U}{\vdash_D b \rhd b' \to T = U}$$

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

$$\text{ABSURD} \quad \frac{}{\vdash_D \mathit{false} \rhd T = U}$$

Figure 3: The Inference Rules for Process Terms

- If $P \xrightarrow{c?}_l F$ then $Q \xrightarrow{c?}_l G$ and for all $v \in Val$ $(Fv, Gv) \in R$;
- If $P \xrightarrow{a}_l P'$, $a \in \{\tau, c!v\}$, then $Q \xrightarrow{a}_l Q'$ 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 [HL96] 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 any $\rho \models b$. We usually abbreviate $\vdash_D$ *true* $\rhd T = U$ to $\vdash_D T = U$. The inference rules are divided into two groups. The first, for manipulating process terms, are given in Figure 3 and are taken directly from [HL96]. They are a minimal formal basis for the reasoning

dec-I	$\dfrac{\vdash_D b \rhd T = U}{\vdash_{D\cup E} b \rhd T = U}$
dec-E	$\dfrac{\vdash_{D\cup E} b \rhd T = U}{\vdash_D b \rhd T = U} T,\ U \in \mathcal{T}_D$
UNFOLD	$\dfrac{}{\vdash_D \ \rhd X = F} X \Leftarrow F \in D$
UFI	$\dfrac{\vdash_D G_i = F_i[\overline{G}/\overline{X}],\ 1 \leq i \leq n}{\vdash_{D\cup E} G_1 = X_1}$ $E = \{ X_i \Leftarrow F_i \mid 1 \leq i \leq n \}$ is a guarded declaration
λ-I	$\dfrac{\vdash_D b \rhd Tx = Ux}{\vdash_D b \rhd T = U} x \notin fv(b, T, U)$
λ-E	$\dfrac{b \models e = e',\ \vdash_D b \rhd T = U}{\vdash_D b \rhd Te = Ue'}$
β	$\dfrac{}{\vdash_D \ \rhd (\lambda x T)e = T[e/x]}$

Figure 4: The Inference Rules for Abstractions

outlined in the Introduction. In [HL96] 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 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:

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

whose derivation we leave to the reader.

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 $\mathcal{A}$

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 following are some simple yet useful facts about $\vdash_D^L$ whose proofs can be found in [HL96]:

Proposition 3.1 *1.* $\vdash_D^L T = T + b' \rightarrow T$

2. $\vdash_D^L b \rightarrow b' \rightarrow T = b \wedge b' \rightarrow T$

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

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

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

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

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

Proposition 3.2 *The following inference rule UFI-O is derivable:*

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

where $E = \{ X_i \Leftarrow \lambda \overline{x}_i(b_i \rightarrow T_i) \mid 1 \leq i \leq n \}$ *is a guarded declaration, and* $F_i \equiv \lambda \overline{x}_i(b_i \rightarrow U_i),\ 1 \leq i \leq n$.

UFI-O is much easier to use than UFI. For example consider the declaration D:

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

Using UFI it may take some efforts to derive

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

while this can be obtained by a straightforward application of UFI-O.

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:

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

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.

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 abstractions such that $\overline{F} \sim_l \overline{H}[\overline{F}/\overline{X}]$ and $\overline{G} \sim_l \overline{H}[\overline{G}/\overline{X}]$. Then $\overline{F} \sim_l \overline{G}$.*

Theorem 4.2 *(Soundness of $\vdash^L_D$) $\vdash^L_D b \triangleright T = U$ 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

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

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 $\vdash^L_D b \triangleright T = U$.* □

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.

For the language in question it is straightforward to develop a version of *symbolic semantics* as defined in [HL96, HL95a], to which we refer the reader for details. The *symbolic operational semantics* is given in Figure 6 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 $T \xrightarrow{b_1,\alpha} T'$ with $bv(\alpha) \cap fv(b,T,U) = \emptyset$, there is a $b \wedge b_1$-partition B with the property that $fv(B) \subseteq fv(b,T,U)$ and for each $b' \in B$ there exists a $U \xrightarrow{b_2,\alpha'} U'$ such that $b' \models b_2$, $\alpha =^{b'} \alpha'$ and $(T',U') \in S^{b'}$.

$$\frac{\alpha \in \{\tau, c!e\}}{\alpha.T \xrightarrow{true,\alpha} T} \qquad \frac{}{c?x.T \xrightarrow{true,c?x} T}$$

$$\frac{T \xrightarrow{b,\alpha} T'}{T+U \xrightarrow{b,\alpha} T'} \qquad \frac{T \xrightarrow{b',\alpha} T'}{b \to T \xrightarrow{b\wedge b',\alpha} T'}$$

$$\frac{U[\overline{x}/\overline{z}] \xrightarrow{b,\alpha} T'}{X(\overline{x}) \xrightarrow{b,\alpha} T'} \quad X \Leftarrow \lambda\overline{z}U \text{ is a definition}$$

Figure 6: Symbolic Operational Semantics

Definition 4.5 (Symbolic Bisimulations)
$\mathbf{S}$ 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 4.6 *(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 [HL95a] and Theorem 3.6 in [HL96]. □

From Theorem 4.6 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 declaration D. There are two major steps in the proof of this statement. The first reduces 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; this second step is a symbolic version of the corresponding result, Theorem 5.10 of [Mil84].

Definition 4.7 A 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 $\bigvee_k b_{ik} = true$ for each i and $b_{ik} \wedge b_{ik'} = false$ for $k \neq k'$, and all input actions in D use the same bound variable. □

Proposition 4.8 *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 $\vdash_{D\cup E}^L T = X_1(\overline{w})$.*

We say two vectors of variables $\overline{x} = x_1x_2\ldots x_n$ 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 $x_i \notin V$.

Definition 4.9 A standard declaration $D = \{X_i\langle\overline{x}_i\rangle \Leftarrow T_i\}_{i\in I}$, where $T_i \equiv \sum_{k\in K_i} b_{ik} \to \sum_{p\in P_{ik}} \alpha_{ikp}.X_{f(i,k,p)}(\overline{x}_{ikp})$, is *parameter-saturated* if

1. $\{\overline{x}_i\} = \{\overline{x}_j\}$ for any $i,\ j$;
2. $\overline{x}_{ikp}$ and $\overline{x}_{f(i,k,p)}$ differ at most at $bv(\alpha_{ikp})$ for any $i,\ k,\ p$. □

Proposition 4.10 *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.*

Proposition 4.11 *Let $D_1 = \{\overline{X} \Leftarrow \overline{G}\}$, $D_2 = \{\overline{Y} \Leftarrow \overline{H}\}$ be two standard and parameter-saturated declarations. If $X_1(\overline{x}_1) \sim_L^b Y_1(\overline{y}_1)$ then there is a standard declaration $E = \{\overline{Z} \Leftarrow \overline{F}\}$ such that $\vdash_{D_1\cup E}^L b \rhd X_1(\overline{x}_1) = Z_{11}(\overline{z}_{11})$ and $\vdash_{D_2\cup E}^L b \rhd Y_1(\overline{y}_1) = Z_{11}(\overline{z}_{11})$.*

Theorem 4.12 *(Completeness of $\vdash^L$) Let D be a restricted regular and guarded declaration, and $T,\ U$ are restricted regular terms in $\mathcal{T}_D$. Then $T \sim_L^b U$ implies $\vdash_D^L b \rhd T = U$.*

Proof: Let $fv(T,U) = \overline{z}$. By Propositions 4.8 there exist standard declarations $D_1 = \{\overline{X} \Leftarrow \overline{G}\}$ and $D_2 = \{\overline{Y} \Leftarrow \overline{H}\}$ such that $\vdash_{D\cup D_1}^L T = X_1(\overline{z})$, $\vdash_{D\cup D_2}^L U = Y_1(\overline{z})$.

By Proposition 4.11 there exists a standard declaration $E = \{\overline{Z} \Leftarrow \overline{F}\}$ such that

$$\vdash_{D\cup D_1\cup E}^L b \rhd X_1(\overline{z}) = Z_{11}(\overline{z})$$
$$\vdash_{D\cup D_2\cup E}^L b \rhd Y_1(\overline{z}) = Z_{11}(\overline{z})$$

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

5 Early Bisimulation

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
whenever $P \xrightarrow{a}_e P'$ then there exists $Q \xrightarrow{a}_e Q'$ 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 6. 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 $\sim_{\mathbf{E}} = \{\sim_E^b\}$ 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 $\rho \models b$.*

The inference system for early bisimulation, $\vdash^E$, can be obtained by generalising that for late in two different ways: adding the following axiom schema

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

or replacing the INPUT rule by the more general rule schema

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

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 ([HL96]).

Theorem 5.4 *(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 [HL96] 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 [HL96]. 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*.

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. 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

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

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

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

we get

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

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

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

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 $T, U : N \rightarrow process$. Then

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

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 [HL96] 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]:

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

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 Coq 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.

[HL96] M. Hennessy and H. Lin. Proof systems for message-passing process algebras. *Formal Aspects of Computing*, 8: 379 – 407, 1996.

[HL95a] M. Hennessy and H. Lin. Symbolic bisimulations. *Theoretical Computer Science*, 138:353 – 389, 1995.

[HL95b] M. Hennessy and H. Lin. Unique Fixpoint Induction for Message-Passing Process Calculi. Report 6/95, Sussex University, December 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.