Proof Systems for Message–Passing Process Algebras[*]

M. Hennessy, H. Lin
University of Sussex

Abstract

We give sound and complete proof systems for a variety of bisimulation based equivalences over a message-passing process algebra. The process algebra is a generalisation of *CCS* where the actions consist of receiving and sending messages or data on communication channels; the standard prefixing operator $a.p$ is replaced by the two operators $c?x.p$ and $c!e.p$ and in addition messages can be tested by a conditional construct. The various proof systems are parameterised on auxiliary proof systems for deciding on equalities or more general boolean identities over the expression language for data. The completeness of these proof systems are thus relative to the completeness of the auxiliary proof systems.

1 Introduction

In standard or *pure* process algebras processes are described in terms of their ability to perform atomic unanalysed actions. For example

$$P \Longleftarrow a.P + b.c.P$$

describes a process which can continually either perform the action a or the sequence of actions b, c . By a *message–passing process algebra* we mean a process algebra in which these actions are given some structure; namely the reception or emission of data values on communication channels. Thus

$$Q \Longleftarrow c?x.\ if\ x \geq 0\ then\ d!x.Q\ else\ c!(x+1).Q$$

describes a process which can cyclically input a value along the channel c and either output it along the channel d unchanged or output its successor along c, depending on whether or not the value concerned is greater than or equal to 0.

The standard approach to providing a semantic basis for these message–passing algebras, advocated for example in [Mil89], is to translate them into an underlying pure algebra. The central feature of this translation, mapping p to $[\![p]\!]$, is that the input expression $c?x.p$ is mapped into the term

$$\sum_{v \in Val} c?v.[\![p[v/x]]\!]$$

[*]This work has been supported by the SERC grant GR/H16537 and the ESPRIT BRA project CONCUR II

where Val is the domain of all data values. Thus the translation of the process Q above is

$$R \Longleftarrow \sum_{n \geq 0} c?n.d!n.R \; + \; \sum_{n<0} c?n.c!(n+1).R$$

This may be taken to be a description in a pure process algebra where we assume that for each channel name c and for every data-value v, in this case every integer, there are atomic actions $c?v$ and $c!v$.

There are two disadvantages in this approach. The first is that descriptions which are in some intuitive sense finite are translated into processes which are inherently infinite, at least if the domain of possible values, *Val*, is infinite; it is necessary to have in the underlying pure process algebra a summation operator Σ_I where I has the same cardinality as the value domain. Such process algebras are difficult to use. For example the standard algorithms and verification tools, see e.g. [CPS89], do not apply and equational reasoning is difficult since any proof system based on this approach is of necessity infinite. The second disadvantage is that with such translations *uniformities* which exist in the object description disappear in the translation. For example the subsequent behaviour of Q above after the reception of an input v is described functionally by the term $\lambda x.if\ x \geq 0 \rightarrow d!x.Q\ else\ c!(x+1).Q$. this uniform treatment of inputs is not apparent in the translation, R. Although this notion of *uniformity* is difficult to define precisely, it should play a central role in proving properties of message passing systems. The object of this paper is to develop a semantic theory of message–passing processes which takes advantage of this uniformity. In particular our semantic theory will apply directly to the syntax of message–passing processes and will not be mediated by a translation into an infinitary language. As a result the associated proof systems will be in some sense finitary.

Such theories already exist for value–passing processes. In [HIar] a fully-abstract denotational model is presented while in [Hen91] a sound finitary proof system is given which is also complete for recursion free processes. However all of this work is with respect to a particular behavioural equivalence called *testing equivalence*, [Hen88]. Here we wish to consider an alternative and much finer behavioural equivalence, bisimulation equivalence from [Mil89]. The main result of the paper is a series of sound and complete proof systems, with respect to a range of bisimulation based equivalences, for a recursion free message–passing process algebra.

For most of the paper we restrict our attention to a very simple language which consists essentially of a notation for the empty process, *nil*, a choice operator $+$ and action prefixing; other operators such as forms of parallel or restriction can easily be accommodated. However, when actions take the form $c?x$ and $c!e$, in order for the language to be of interest we also need to be able to test data and branch on the consequences of the test. Syntactically this could be represented by an *if b then* ... *else* ... construction, where b is a boolean expression, but instead we use the simpler notation of guarded commands, $b \rightarrow t$. Thus

$$c?x.x = 0 \rightarrow d!0.nil \;\; + \;\; x > 0 \rightarrow d!1.nil$$

is a process which inputs a value on c and outputs 0 on d if the input is 0 and 1 if it is greater than 0 and does nothing otherwise. In order to reason about these kinds of processes it is necessary, in general, to reason about data expressions. So following

[Hen91] we design a proof system whose judgements are *guarded equations* of the form

$$b \rhd t = u$$

where b is a boolean expression and t, u are process terms that may contain free data variables. Semantically this should be read as "under any evaluation of free data variables that satisfies b t is semantically equivalent to u". The completeness of the proof system is thus relative to that for the data domain involved. Moreover rather than getting embroiled in the details of an actual proof system for data expressions we simply assume the existence of some all powerful mechanism for answering arbitrary questions about data. On the one hand this enables us to concentrate on the behaviour of processes and on the other it reflects what would be a reasonable implementation strategy for a proof system based on our results; the main proof system would be based on the proof rules whose applicability is determined by the structure of processes and this main system would periodically call auxiliary proof systems to establish facts about data expressions. A simple example of a proof rule from the main system is

$$\frac{b \rhd t_i = u_i \quad i = 1,2}{b \rhd t_1 + t_2 = u_1 + u_2}$$

while

$$\frac{b \models e = e', \quad b \rhd t = u}{b \rhd c!e.t = c!e'.u}$$

is a rule which depends on a call to an auxiliary proof system concerned with the data domain; this is expressed in rather abstract terms, one of the antecedents referring to the semantics of the expressions b, e and e'; as we shall see $b \models e = e'$ is true if the intended meaning of the boolean b always implies the intended meaning of e equals that of e'. Of course the reasoning about processes can not be completely be divorced from the reasoning about data and an example of where they interact is the cut rule

$$\frac{b \models b_1 \vee b_2, \quad b_1 \rhd t = u \quad b_2 \rhd t = u}{b \rhd t = u}$$

This enables a proof to be developed by case analysis on the data.

The soundness of such a proof system depends on having a semantic equivalence for processes and as we have already stated in this paper we are interested in bisimulation-like semantics. As a starting point we use strong bisimulation, [Mil89], but as has been pointed out in [MPW92, HL92] there are at least two natural generalisations of this equivalence to message–passing processes. The first, called *early strong bisimulation equivalence*, is based on the ability of processes to perform actions of the form $c?v$ and $c!v$ while the second is based on the slightly more abstract actions $c?$ and $c!$. Thus the processes

$$c?x.\ even(x) \rightarrow P \ + \ c?x.\ odd(x) \rightarrow P$$

and

$$c?x.P \ + \ c?x.nil$$

are identified by the early version of the equivalence but are differentiated in the late case because the $c?$ move from the first to the abstraction

$$\lambda x.\ even(x) \rightarrow P$$

can not be matched by a corresponding $c?$ move from the second. Each of these generalisations of strong bisimulation equivalence has a corresponding "weak" version in which internal moves are abstracted. Thus in all we have four reasonable semantic equivalences and for each of these we present a corresponding proof system. In the strong cases the difference between early and late is manifested in the slightly different methods for inferring identities involving input prefixes; the early equivalence requires a stronger proof rule. On the other hand the weak version of both equivalences can be obtained by adding to the corresponding proof system the standard τ-laws from [Mil89].

The judgements of the proof systems involve *open* process terms, i.e. terms in which data variables need to be instantiated before any operational significance can be associated with them, but the observational equivalences are only defined on closed terms. Therefore in order to even express the soundness and completeness of the proof systems we need to generalise these equivalences to open terms. For each of these semantic equivalences, $\simeq$, we design a proof system with the property that

$b \rhd t = u$ if and only if $t\rho \simeq u\rho$ for every evaluation ρ satisfying b.

As usual establishing soundness is straightforward but completeness requires some ingenuity. Here we use the approach of [HL92] and introduce symbolic versions of each of the semantic equivalences which are defined directly on open terms. These are expressed in terms of families of relations over open terms parameterised on boolean expressions and we show that, for each semantic equivalence $\simeq$ we consider,

$t \simeq^b u$ if and only if $t\rho \simeq u\rho$ for every evaluation ρ satisfying b.

Thus soundness and completeness of the proof systems can be established relative to the semantic relations $\simeq^b$. Using this approach the completeness theorems in particular now become "symbolic versions" of the standard completeness theorems of [Mil89], although the details are somewhat more complicated.

We now give a brief outline of the content of the remainder of the paper. In the next section we define the simple language, give it a *concrete* operational semantics and define (early) strong bisimulation. This is followed by a discussion of the proof system for proving processes bisimilar. We state the soundness theorem for the system and indicate the difficulty in proving completeness. In the following section, Section 3, we define the symbolic semantics and the associated symbolic bisimulation equivalence and prove that it captures precisely the concrete bisimulation equivalence over processes. We then use this result to show the completeness of the proof system.

In Section 4 we repeat these results for *weak* bisimulation equivalence where again it is necessary to develop an appropriate definition of *weak* symbolic equivalence. The following section outlines corresponding results for a *late* operational semantics and considers both the strong and weak cases. We end by discussing briefly how to extend these results to other language constructs.

Related Work: We end this section with a brief discussion of related work. As stated previously the approach we have taken is based on that of [Hen91] where a sound and complete proof system for *testing equivalence* is developed. Here we tackle various bisimulation based equivalences and an essential ingredient of the completeness theorems is the notion of *symbolic bisimulation equivalence*. This has already been used in [HL92] to develop an algorithm for checking whether two message-passing processes are equivalent and in [HL93] for developing a proof system to verify that such processes satisfy properties described by formulae from a first-order modal logic.

The more standard approach to message-passing processes is to translate them into "pure processes" as outlined at the beginning of this section [Mil89, HR88]. Indeed in [Bur91] a front-end for the Concurrency Workbench is described which translates message-passing processes from a language such as ours into "pure processes" which can be accepted by the Concurrency Workbench and various examples treated using this approach may be found in [Wal89]. However these approaches require the set of values to be finite and even using the boolean value space of two elements leads to an exponential blow-up in the size of descriptions. We hope that with our approach at least some of this complexity can be avoided. In [Lin93] an extension of the *PAM* verification system, [Lin91], based on our results, is described. It offers much the same functionality as the the original *PAM* except that message-passing process algebras can be defined and the proof elaboration scheme is more flexible.

In [GP90] a very general language for describing message-passing, based on *ACP*, is described and in [GP91] a proof theory is given. Although these goals are quite similar to ours their approach is very different. A modular algebraic specification language is used to describe data domains and the description of processes is such that it may be viewed as consisting of another module. They continue to view message-passing processes as universally quantified versions of "pure processes", the quantification being over the domain of messages, but they bring to bear the general framework of algebraic specifications in order to handle proof theoretically this quantification. Nevertheless it would be interesting to compare the two approaches.

Very recently, proof systems for late and early strong bisimulation equivalences over the π-calculus [MPW92] have been given in [PS93]. These proof systems are quite different from ours as they take considerable advantage of the blurring of variables and constants in the π-calculus.

2 A Simple Language

The language we consider can be given by the following BNF grammar

$$\begin{array}{rcl} t & ::= & nil \quad | \quad \alpha.t \quad | \quad t+t \quad | \quad b \rightarrow t \\ \alpha & ::= & \tau \quad | \quad c?x \quad | \quad c!e \end{array}$$

where b is a boolean expression, e data expression, c is a channel name and x a data variable. So this syntax assumes a predefined set of channel names, *Chan*, ranged over by c and a set of data variables, *DVar*, ranged over by $x, y, \ldots$. More importantly it also assumes a language for data expressions *DExp*, ranged over by $e, e', \ldots$ and a similar language *BExp*, ranged over by b, for boolean expressions. At the very least we assume that *DExp* contains the set of data variables *DVar* and also a set of data values *Val* and for every pair of data expressions e, e' we assume that $e = e'$ is a boolean expression. We also assume that all free variables in boolean expressions are data variables. Apart from this we do not worry about the expressive power of these languages although the results on symbolic bisimulations require that the language for boolean expressions is very powerful.

$$\begin{array}{lll}
\tau.p \xrightarrow{\tau}_e p & & \\
c!e.p \xrightarrow{c!v}_e p & & \text{where } [\![e]\!] = v \\
c?x.t \xrightarrow{c?v}_e t[v/x] & & c \in Chan, v \in Val \\
p \xrightarrow{a}_e p' & \text{implies} & p + q \xrightarrow{a}_e p' \\
 & & q + p \xrightarrow{a}_e p' \\
p \xrightarrow{a}_e p', [\![b]\!] = true & \text{implies} & b \rightarrow p \xrightarrow{a}_e p'
\end{array}$$

Figure 1: (Early) Operational Semantics

An *evaluation*, ρ is a mapping from *DVar* to *Val* and we use the standard notation $\rho\{v/x\}$ to denote the evaluation which differs from ρ only in that it maps x to v. An application of ρ to a data expression e, denoted $\rho(e)$, always yields a value from *Val* and similarly for boolean expressions; $\rho(b)$ is either true or false. Thus we assume that evaluation of data and boolean expressions always terminate and our approach is to work modulo these evaluations. We also assume that these evaluations satisfy standard properties; each expression e has associated with it a set of variables $fv(e)$ and, for example, if ρ and ρ' agree on $fv(e)$ then $\rho(e) = \rho'(e)$. If an expression e has no variables, it is *closed*, then $\rho(e)$ is independent of ρ and we use $[\![e]\!]$ to denote it value. Similarly with boolean expressions. We will use the suggestive notation $b \models b'$ to indicate that for every evaluation if $\rho(b)$ is true then so is $\rho(b')$. Of course we could equally well say that $b \rightarrow b'$ is a logical theorem but our notation emphasises the fact that we wish to work modulo the semantics of expressions. We will write $b = b'$ for $b \models b'$ and $b' \models b$.

We will also refer to substitutions, and assume that they satisfy the expected properties; we use $e[e'/x]$ to denote the result of substituting e' for all occurrences of x in e. More generally a substitution σ is a mapping from data variables to expressions and we use $e\sigma$ to denote the result of applying σ to the expression e.

Returning to the process language above, the prefix $c?x$ binds the occurrence of x in the sub-term t of $c?x.t$ and we have as usual the sets of free variables $fv(u)$ and bound variables $bv(u)$ of a term u; of course these depend in general on the variables in the data and boolean expressions contained in u. This leads to the standard definition of α-conversion, $\equiv_\alpha$, over terms and of substitution, $t[e/x]$ denoting the result of substituting all free occurrences of x in t by e, and this relies on the definition of substitution in data expressions. A term is *closed* if it contains no free variables and these we refer to as *processes*, ranged over by $p, q, \ldots$. Throughout the paper *open terms* refer to terms that may contain free occurrences of data variables, but no process variables. Open terms are ranged over by $t, u, \ldots$.

The standard operational semantics of this language is given in Figure 1. It consists of a set of binary relations, $\xrightarrow{a}_e$, between processes, where a ranges over the set $Act = \{\tau, c?v, c!v\}$. In [MPW92, HL92] this is referred to as the *early* operational semantics as when input terms such as $c?x.p$ perform an action the value received is immediately bound to the variable x. In Section 5 we will see a slightly different way of organising input actions where this binding is delayed.

A symmetric relation R between closed terms is a strong bisimulation if it satisfies: $(p, q) \in R$ implies that for every $a \in Act$

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

where a ranges over $\{\tau,\ c?v,\ c!v\}$. We use $\sim_e$ to denote the maximal (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. We then have

Proposition 2.1 *$\sim_e$ is preserved by every operator in the language.* □

We now consider a proof system for deriving statements about $p \sim_e q$. In general we will need to consider open terms because in order to prove a statement such as $c?x.t = c?x.u$ it is necessary to relate the open terms t and u. Also because we allow testing of data we will need to establish statements relative to a boolean expression b. Thus the judgements are *guarded equations* of the form

$$b \rhd t = u$$

where b, the *guard*, is a boolean expression. For brevity we usually write $t = u$ for $true \rhd t = u$.

The rules for the proof system are given in Figure 2, where σ is any mapping from process variables to process terms.. Note that reference is made to the semantics of expressions in the OUTPUT rule, for establishing identities of the form $c!e.t = c!e'.u$, and in the CUT rule which is used to perform proofs by case analysis. The rule GUARD also uses a case analysis; an identity of the form $b \rightarrow t = u$ may be established by considering two cases, one when b is true and the other when it is false. The only rule which may be a little surprising is that for input prefixing. A much simpler rule would be

$$\frac{b \rhd t = u}{b \rhd c?x.t = c?x.u}$$

But without the side condition $x \notin fv(b)$ this is not sound; It could be used to prove

$$x = 1 \rhd c?x.c!1.nil = c?x.c!x.nil$$

because

$$x = 1 \rhd c!1.nil = c!x.nil$$

With this side condition it is sound but not sufficiently powerful to derive all true identities between early bisimilar processes. For example the two processes discussed in the introduction

$$c?x.\ even(x) \rightarrow P\ +\ c?x.\ odd(x) \rightarrow P \text{ and } c?x.P\ +\ c?x.nil$$

are strong bisimulation equivalent but can not be proved equivalent using this restricted rule.

OUTPUT and CUT are the only two rules which relies on checking facts about the data domain but rules such as

$$\text{CONSEQUENCE}\ \frac{b \models b',\ b' \rhd t = u}{b \rhd t = u}$$

can easily be derived.

In order to characterise strong bisimulation equivalence using this proof system we need the standard equations, [Mil89], given in Figure 3. Let us write $\vdash_1 b \rhd t = u$ to mean that $b \rhd t = u$ can be derived from these equations using the rules in Figure 2.

The following are some simple facts about $\vdash_1$:

EQUIV $\dfrac{}{true \rhd t = t} \quad \dfrac{b \rhd t = u}{b \rhd u = t} \quad \dfrac{b \rhd t = u, u = v}{b \rhd t = v}$

EQN $\dfrac{}{true \rhd t\sigma = u\sigma}$ $t = u$ is an axiom

CONGR $\dfrac{b \rhd t_i = u_i \quad i = 1, 2}{b \rhd t_1 + t_2 = u_1 + u_2}$

α-CONV $\dfrac{}{true \rhd c?x.t = c?y.t[y/x]}$ $y \notin fv(t)$

E-INPUT $\dfrac{b \rhd \sum_{i \in I} \tau.t_i = \sum_{j \in J} \tau.u_j}{b \rhd \sum_{i \in I} c?x.t_i = \sum_{j \in J} c?x.u_j}$ $x \notin fv(b)$

OUTPUT $\dfrac{b \models e = e', \quad b \rhd t = u}{b \rhd c!e.t = c!e'.u}$

TAU $\dfrac{b \rhd t = u}{b \rhd \tau.t = \tau.u}$

GUARD $\dfrac{b \wedge b' \rhd t = u \quad b \wedge \neg b' \rhd nil = u}{b \rhd b' \to t = u}$

CUT $\dfrac{b \models b_1 \vee b_2, \quad b_1 \rhd t = u \quad b_2 \rhd t = u}{b \rhd t = u}$

ABSURD $\dfrac{}{false \rhd t = u}$

Figure 2: The Inference Rules

Proposition 2.2 *1.* $\vdash_1 b \to b' \to t = b \wedge b' \to t$

2. $\vdash_1 t = t + b' \to t$

3. $b \models b'$ *implies* $\vdash_1 b \rhd t = b' \to t$

4. $\vdash_1 b \wedge b' \rhd t = u$ *implies* $\vdash_1 b \rhd b' \to t = b' \to u$

5. $\vdash_1 b \to (t + u) = b \to t \ + \ b \to u$

6. $\vdash_1 b \to u \ + \ b' \to u = b \vee b' \to u$

7. *if* $fv(b) \cap \ bv(\alpha) = \emptyset$ *then* $\vdash_1 b \to \alpha.t = b \to \alpha.(b \to t)$

Proof: Left to the reader. □

The E-INPUT rule has a generalised form which will be useful:

S1 $X + nil = X$
S2 $X + X = X$
S3 $X + Y = Y + X$
S4 $(X + Y) + Z = X + (Y + Z)$

Figure 3: The Axioms $\mathcal{A}_1$

Proposition 2.3 *Suppose $x \notin fv(b, c_i, d_j)$, $i \in I, j \in J$. Then from*

$$b \rhd \Sigma_{i \in I} c_i \rightarrow \tau.t_i = \Sigma_{j \in J} d_j \rightarrow \tau.u_j$$

infer

$$b \rhd \Sigma_{i \in I} c_i \rightarrow c?x.t_i = \Sigma_{j \in J} d_j \rightarrow c?x.u_j$$

Proof: For each non-empty $K \subseteq I$ let c_K be the boolean expression $\wedge_{k \in K} c_k \wedge \wedge_{k' \in I-K} \neg c_{k'}$. Then $\bigvee c_K = true, c_K \wedge c_{K'} = false$ whenever $K \neq K'$. Using Proposition 2.2 we can show that

$$\vdash_1 \Sigma_{i \in I} c_i \rightarrow \tau.t_i = t^\tau$$

where t^τ denotes $\Sigma_K c_K \rightarrow t^\tau_K$ and t^τ_K denotes $\Sigma_{k \in K} c_k \rightarrow \tau.t_k$. Let $u^\tau = \Sigma_L d_L \rightarrow u^\tau_L$ be defined in a similar manner.

We know $b \rhd t^\tau = u^\tau$ and therefore for each K, L, $b \wedge c_K \wedge d_L \rhd t^\tau = u^\tau$. Again using Proposition 2.2 we can prove

$$b \wedge c_K \wedge d_L \rhd t^\tau = \Sigma_{k \in K} \tau.t_k$$

and

$$b \wedge c_K \wedge d_L \rhd u^\tau = \Sigma_{l \in L} \tau.u_l.$$

Therefore

$$b \wedge c_K \wedge d_L \rhd \Sigma_{k \in K} \tau.t_k = \Sigma_{l \in L} \tau.u_l.$$

Now we can apply E-INPUT to obtain

$$b \wedge c_K \wedge d_L \rhd \Sigma_{k \in K} c?x.t_k = \Sigma_{l \in L} c?x.u_l.$$

By reversing the above argument we have

$$b \wedge c_K \wedge d_L \rhd t^x = u^x$$

where t^x and u^x denote $\Sigma_K c_K \rightarrow \Sigma_{k \in K} c?x.t_k$ and $\Sigma_L d_L \rightarrow \Sigma_{l \in L} c?x.u_l$, respectively. Since $\bigvee_{K,L} c_K \wedge d_L = true$, we can apply CUT to obtain $b \rhd t^x = u^x$. Finally Proposition 2.2 can be used to transform t^x and u^x into the required form. □

The soundness of the $\vdash_1$ is given by the following proposition:

Proposition 2.4 *If $\vdash_1 b \rhd t = u$ and $\rho \models b$ then $t\rho \sim_e u\rho$*

$$\begin{array}{llll}
\alpha.t \xrightarrow{true,\alpha}_E t & & & \alpha \in \{\, \tau, c!e \mid c \in \ Chan, e \in \ Exp \} \\
c?x.t \xrightarrow{true,c?y}_E t[y/x] & & & y \notin fv(c?x.t) \\
t \xrightarrow{b',\alpha}_E t' & & \text{implies} & b \rightarrow t \xrightarrow{b \wedge b',\alpha}_E t' \\
t \xrightarrow{b,\alpha}_E t' & & \text{implies} & t + u \xrightarrow{b,\alpha}_E t' \\
 & & & u + t \xrightarrow{b,\alpha}_E t'
\end{array}$$

Figure 4: Symbolic Operational Semantics

Proof: The proof is by induction on the derivation of $b \vdash_1 t = u$ and a case analysis on the last rule used. We examine one example, the E-INPUT rule.

Suppose $\rho \models b$. We must show $(\Sigma_{i \in I} c?x.t_i)\rho \sim_e (\Sigma_{j \in J} c?x.u_j)\rho$. For convenience call these P, Q, respectively.

If $P \xrightarrow{a}_e P'$ then a must be $c?v$ for some v and P' must be $t_i\rho\{v/x\}$ for some i. Since $x \notin fv(b)$ we have $\rho\{v/x\} \models b$. So by induction

$$(\Sigma_{i \in I} \tau.t_i)\rho\{v/x\} \sim_e (\Sigma_{j \in J} \tau.u_j)\rho\{v/x\},$$

i.e.

$$\Sigma_{i \in I} \tau.t_i\rho\{v/x\} \sim_e \Sigma_{j \in J} \tau.u_j\rho\{v/x\}.$$

Call the two sides R and S, respectively. Then $R \xrightarrow{\tau}_e t_i\rho\{v/x\}$. So there must be some $j \in J$ such that $S \xrightarrow{\tau}_e u_j\rho\{v/x\}$ and $t_i\rho\{v/x\} \sim_e u_j\rho\{v/x\}$, which means every move of P can be matched by a corresponding move from Q. By symmetry $P \sim_e Q$.

The other rules are similar and easier. □

The converse to this is also true but the proof is far from straightforward. The problem arises because $\sim_e$ is only defined on closed terms whereas the proof system manipulates open terms. So there is no straightforward way to use structural induction on terms. Instead we develop a *symbolic* version of bisimulation equivalence for open terms which captures the standard bisimulation equivalence on all closed instantiations and then prove completeness with respect to this symbolic version.

3 Symbolic Bisimulations

The reader is refered to [HL92] for motivation and discussion on symbolic bisimulations. Here we adapt the definitions, which were originally given for *symbolic graphs*, to our language.

The abstract transition relations are defined to be the least set of relations which satisfy the rules in Figure 4. They take the form of relations $\xrightarrow{b,\alpha}_E$ between arbitrary terms, where b is a boolean expression and α is a prefix, i.e. it has one of the forms $\tau, c?x$ or $c!e$. Intuitively b acts like a trigger: it enables the move when it is true. Based on these abstract actions we can define *symbolic bisimulations* which, for reasons explained in [HL92], must be parameterised on boolean expressions. A finite set of boolean expressions B is called a *b-partition* if $\bigvee B = b$. 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 $t \xrightarrow{b_1,\alpha}_E t'$ with $bv(\alpha) \cap fv(b,t,u) = \emptyset$, there is a $b \wedge b_1$-partition B with the property that for each $b' \in B$ there exists a $u \xrightarrow{b_2,\alpha'}_E u'$ such that $b' \models b_2$ and

1. if $\alpha = c!e$ then $\alpha' = c!e'$, $b' \models e = e'$ and $(t',u') \in S^{b'}$
2. otherwise $\alpha = \alpha'$ and $(t',u') \in S^{b'}$

Definition 3.1 (Symbolic Bisimulations)
$\mathbf{S}$ is an (early) strong symbolic bisimulation if $\mathbf{S} \subseteq \mathcal{ESB}(\mathbf{S})$, where $\subseteq$ is point-wise inclusion. □

Let $\sim_\mathbf{E} = \{\sim_E^b\}$ be the largest (early) strong symbolic bisimulation.

The interest in symbolic bisimulations lies in the fact they are defined with respect to the abstract operational semantics, which for finite terms can be represented as a finite transition graph; in contrast with the standard "concrete" bisimulations are defined over infinite transitions graphs, at least if the set of values is infinite. In [HL92] we give an algorithm for checking for this symbolic equivalence. Here we use it to show completeness of the proof systems. First we relate symbolic and concrete bisimulation equivalence.

Theorem 3.2 *1. (Soundness) $t \sim_E^b u$ implies $t\rho \sim_e u\rho$ for every evaluation ρ such that $\rho \models b$*

2. (Completeness) if $t\rho \sim_e u\rho$ for every evaluation ρ such that $\rho \models b$ then $t \sim_E^b u$

Proof: (Outline) The proof follows the corresponding result in [HL92], Theorem 6.5; it consists in establishing a relationship between symbolic bisimulations and concrete ones. If $\mathbf{S} = \{S^b\}$ is a strong symbolic bisimulation let

$$R_\mathbf{S} = \{\, (t\rho, u\rho) \mid \exists b,\ \rho(b) = \ \mathit{true\ and}\ (t,u) \in S^b \,\}$$

Soundness follows immediately if we can prove that $R_\mathbf{S}$ is a bisimulation. Conversely if R is a strong bisimulation let

$$S^b = \{\, (t,u) \mid \rho \models b \iff (t\rho, u\rho) \in R \,\}$$

for any boolean expression b. Completeness follows if we can show that $\mathbf{S} = \{S^b\}$ is a symbolic bisimulation.

The proof of these two subsidiary results depends on relating the abstract actions to the concrete actions. We simply state the required relationships and leave the proofs to the reader.

1. $t\rho \xrightarrow{\tau}_e q$ if and only if there exist $b,\ t'$ s.t. $\rho \models b,\ q \equiv_\alpha t'\rho$ and $t \xrightarrow{b,\tau}_E t'$.
2. $t\rho \xrightarrow{c!v}_e q$ if and only if there exist $b,\ e,\ t'$ and ρ s.t. $\rho \models b,\ \rho(e) = v,\ q \equiv_\alpha t'\rho$ and $t \xrightarrow{b,c!e}_E t'$.
3. $t\rho \xrightarrow{c?v}_e q$ if and only if there exist $b,\ x,\ t'$ and ρ s.t. $x \notin fv(t),\ \rho \models b,\ r \equiv_\alpha t'\rho\{v/x\}$ and t $\xrightarrow{b,c?x}_E t'$.

□

With this theorem we can now return to the proof system and show its completeness by proving

$$t \sim_E^b u \text{ implies } \vdash_1 b \rhd t = u \qquad (*)$$

This provides the converse to Proposition 2.4. The proof of $(*)$ follows the standard proof of the corresponding "concrete" result, as given in [Mil89], except that now we work at the symbolic level.

The proof is by induction on the size of terms which is defined as follows:

1. $\mid nil \mid = 0$
2. $\mid t + u \mid = \ max\{\mid t \mid, \mid u \mid\}$
3. $\mid b \rightarrow t \mid = \mid t \mid$
4. $\mid \alpha.t \mid = 1 + \mid t \mid$

We also need the notion of normal form:

Definition 3.3 t is a *normal form*, or *in normal form*, if it has the form $\Sigma_i b_i \rightarrow \alpha_i.t_i$ and each t_i is in normal form. □

Lemma 3.4 *For every term t there exists a normal form t' such that $fv(t) = fv(t'), \mid t \mid = \mid t' \mid$ and $\vdash_1 t = t'$.*

Proof: By structural induction on terms using the elementary facts about the proof system given in Proposition 2.2 □

Theorem 3.5 *(Completeness) $t \sim_E^b u$ implies $\vdash_1 b \rhd t = u$*

Proof: By induction on the joint size of t and u. We may assume that both are normal forms, $t \equiv \Sigma_{i \in I} c_i \rightarrow \alpha_i.t_i$ and $u \equiv \Sigma_{j \in J} d_j \rightarrow \beta_j.u_j$. Call a prefix of type $\gamma \in \{\, \tau, c!, c? \mid c \in Chan \,\}$ if it has the form $\tau, c!e, c?x$, respectively. Let $I_\gamma = \{\, i \in I \mid \alpha_i \text{ has type } \gamma \,\}$, $J_\gamma = \{\, j \in J \mid \beta_j \text{ has type } \gamma \,\}$. Let also $t_\gamma = \Sigma_{i \in I_\gamma} c_i \rightarrow \alpha_i.t_i$, $u_\gamma = \Sigma_{j \in J_\gamma} d_j \rightarrow \beta_j.u_j$. We show $b \rhd t_\gamma = u_\gamma$ for each type γ. Clearly $t_\gamma \sim_E^b u_\gamma$. We examine the cases $\gamma = \tau$ and $\gamma = c?$ here and leave the case $\gamma = c!$ to the reader.
(Case $\gamma = \tau$). By symmetry we need only to show

$$b \rhd u_\tau + c_i \rightarrow \tau.t_i = u_\tau.$$

for each $i \in I_\tau$. Note that $b \wedge \neg c_i \rhd c_i \rightarrow \tau.t_i = nil$ so by CUT it is sufficient to show

$$b \wedge c_i \rhd u_\tau + c_i \rightarrow \tau.t_i = u_\tau.$$

Now $t_\tau \xrightarrow{c_i,\tau}_E t_i$. So there exists a $b \wedge c_i$-partition B such that for each $b' \in B$ there is $u_\tau \xrightarrow{d_j,\tau}_E u_j$ such that $b' \models d_j$ and $t_i \sim_E^{b'} u_j$. By induction $b' \rhd t_i = u_j$. By TAU $b' \rhd \tau.t_i = \tau.u_j$. Since $b' \models c_i$ and $b' \models d_j$ we have $b' \rhd c_i \rightarrow \tau.t_i = d_j \rightarrow \tau.u_j$. By S2 $b' \rhd u_\tau + c_i \rightarrow \tau.t_i = u_\tau$. This is true for each b' in B and therefore an application of CUT gives the required

$$b \wedge c_i \rhd u_\tau + c_i \rightarrow \tau.t_i = u_\tau.$$

(Case $\gamma = c?$). Let $z \notin fv(b,t,u)$. We know $\Sigma_{i \in I_{c?}} c_i \rightarrow c?z.t_i[z/x_i] \sim_E^b \Sigma_{j \in J_{c?}} d_j \rightarrow c?z.u_j[z/y_j]$ where $\alpha_i \equiv c?x_i, i \in I_{c?}$ and $\beta_j \equiv c?y_j, j \in J_{c?}$. Therefore $\Sigma_{i \in I_{c?}} c_i \rightarrow \tau.t_i[z/x_i] \sim_E^b \Sigma_{j \in J_{c?}} d_j \rightarrow \tau.u_j[z/y_j]$. By the same argument as in the previous case, we can show $b \rhd \Sigma_{i \in I_{c?}} c_i \rightarrow \tau.t_i[z/x_i] = \Sigma_{j \in J_{c?}} d_j \rightarrow \tau.u_j[z/y_j]$. By Proposition 2.3 $b \rhd \Sigma_{i \in I_{c?}} c_i \rightarrow c?z.t_i[z/x_i] = \Sigma_{j \in J_{c?}} d_j \rightarrow c?z.u_j[z/y_j]$. By α-CONV, $b \rhd \Sigma_{i \in I_{c?}} c_i \rightarrow c?x_i.t_i = \Sigma_{j \in J_{c?}} d_j \rightarrow c?y_j.u_j$. □

4 Weak Bisimulation Equivalence

In this section we outline how to extend the results of the previous two sections to so-called *weak* bisimulations.

The concrete double arrows $\overset{a}{\Longrightarrow}_e$, where $a \in \{\tau, c?v, c!v\}$, are defined as the least relations between closed terms generated by the following rules

- $t \overset{\varepsilon}{\Longrightarrow}_e t$.
- $t \xrightarrow{a}_e u$ implies $t \overset{a}{\Longrightarrow}_e u$.
- $t \xrightarrow{\tau}_e \overset{a}{\Longrightarrow}_e u$ implies $t \overset{a}{\Longrightarrow}_e u$.
- $t \overset{a}{\Longrightarrow}_e \xrightarrow{\tau}_e u$ implies $t \overset{a}{\Longrightarrow}_e u$.

Let $\hat{a}$ to be ε when $a = \tau$, and a otherwise. The early weak bisimulation is then defined as usual (for closed terms):

Definition 4.1 R is an early weak bisimulation if $(t,u) \in R$ implies

- if $t \xrightarrow{a}_e t'$ then $u \overset{\hat{a}}{\Longrightarrow}_e u'$ for some u' and $(t',u') \in R$.
- if $u \xrightarrow{a}_e u'$ then $t \overset{\hat{a}}{\Longrightarrow}_e t'$ for some t' and $(t',u') \in R$.

Let $\approx_e$ be the largest early weak bisimulation. □

Similarly, we define symbolic double arrow by

- $t \overset{true,\varepsilon}{\Longrightarrow}_E t$.
- $t \xrightarrow{b,\alpha}_E u$ implies $t \overset{b,\alpha}{\Longrightarrow}_E u$.
- $t \xrightarrow{b,\tau}_E \overset{b',\alpha}{\Longrightarrow}_E u$ implies $t \overset{b \wedge b',\alpha}{\Longrightarrow}_E u$.
- $t \overset{b,\alpha}{\Longrightarrow}_E \xrightarrow{b',\tau}_E u$ implies $t \overset{b \wedge b',\alpha}{\Longrightarrow}_E u$.

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

> $(t, u) \in \mathcal{EWB}(\mathbf{S})^b$ if whenever $t \xrightarrow{b_1,\alpha}_E t'$ with $bv(\alpha) \cap fv(b, t, u) = \emptyset$, then there is a $b \wedge b_1$-partition B such that for each $b' \in B$ there exists a $u \overset{b_2,\hat{\alpha}'}{\Longrightarrow}_E u'$ such that $b' \models b_2$ and
>
> 1. if $\alpha \equiv c!e$ then $\alpha' \equiv c!e'$, $b' \models e = e'$ and $(t', u') \in S^{b'}$
> 2. otherwise $\alpha \equiv \alpha'$ and $(t', u') \in S^{b'}$

Definition 4.2 (Weak Symbolic Bisimulations)
$\mathbf{S}$ is a weak symbolic bisimulation if $\mathbf{S} \subseteq \mathcal{EWB}(\mathbf{S})$ □

Let $\approx_{\mathbf{E}} = \{\approx_E^b\}$ be the largest (early) weak symbolic bisimulation.

The two versions of weak bisimulation can be related as in the case of strong bisimulation.

Theorem 4.3 *$t \approx_E^b u$ if and only if $t\rho \approx_e u\rho$ for every ρ which satisfies b.*

Proof: Similar to that of Theorem 3.2. □

The aim of this section is to extend the proof system of Section 2 to weak bisimulation equivalence. However it is well-known that $\approx_e$ is not preserved by $+$ and so we have to work with the modified relation:

Definition 4.4 Two closed terms t, u are *early observation equivalent*, written $t \simeq_e u$, if for all $a \in \{\tau, c?v, c!v\}$

- Whenever $t \xrightarrow{a}_e t'$ then $u \overset{a}{\Longrightarrow}_e u'$ for some u and $t' \approx_e u'$.
- Whenever $u \xrightarrow{a}_e u'$ then $t \overset{a}{\Longrightarrow}_e t'$ for some t and $t' \approx_e u'$.

□

As usual $\simeq_e$ is the largest congruence relation contained in $\approx_e$. This relation can be generalized to open terms by letting $t \simeq_e u$ iff $t\rho \simeq_e u\rho$ for any ρ. We then have the standard result

Proposition 4.5 *$\simeq_e$ is preserved by all the operators in the language.*

To give a sound and complete proof system for this relation, it is sufficient to add to the proof system the equations $\mathcal{A}_2$ given in Figure 5. These appear to be slightly weaker than the traditional τ-laws of [Mil89] but in fact we can generalise the third law so that it applies to arbitrary prefixes:

$$\alpha.(X + \tau.Y) + \alpha.Y = \alpha.(X + \tau.Y).$$

When α is τ this follows from T2 and S2 while the case for $c?x$ can be derived using the rule E-INPUT. Let us use $\vdash_2 b \rhd t = u$ to denote that $b \rhd t = u$ can be derived from the proof system using the axioms $\mathcal{A}_2$, and of course $\mathcal{A}_1$. Then we also have the following generalisation of T3:

$$\begin{array}{ll} \mathrm{T1} & \alpha.\tau.X = \alpha.X \\ \mathrm{T2} & X + \tau.X = \tau.X \\ \mathrm{T3} & c!e.(X+\tau.Y) + c!e.Y = c!e.(X+\tau.Y) \end{array}$$

Figure 5: The Axioms $\mathcal{A}_2$

Lemma 4.6 *If* $fv(b) \cap bv(\alpha) = \emptyset$ *then* $\vdash_2 \alpha.(X + b \to \tau.Y) = \alpha.(X + b \to \tau.Y) \ + \ b \to \alpha.Y$

Proof: Since $X = X + b \to X$ we need only to show $b \to \alpha.(X + b \to \tau.Y) = b \to \alpha.(X + b \to \tau.Y) \ + \ b \to \alpha.Y$ which can be derived as follows (using Proposition 2.2):

$$\begin{array}{rlr} & b \to \alpha.(X + b \to \tau.Y) & () \\ = & b \to \alpha.(b \to (X + b \to \tau.Y)) & (2.2.\mathit{7}) \\ = & b \to \alpha.(b \to (X + \tau.Y)) & (2.2.\mathit{5},\mathit{1}) \\ = & b \to \alpha.(X + \tau.Y) & (2.2.\mathit{7}) \\ = & b \to (\alpha.(X+\tau.Y) + \alpha.Y) & (\mathrm{T3}) \\ = & b \to \alpha.(X+\tau.Y) + b \to \alpha.Y & (2.2.\mathit{5}) \\ = & b \to \alpha.(X + b \to \tau.Y) + b \to \alpha.Y & \text{(previous steps reversed)} \end{array}$$

□

The main result of this section is

Theorem 4.7 $\vdash_2 b \rhd t = u$ *if and only if* $t\rho \simeq_e u\rho$ *for every* ρ *such that* $\rho \models b$.

The soundness is straightforward, by induction on the length of the proof of $b \rhd t = u$ and we prove the completeness by relying on the symbolic version of weak bisimulation. Again we have to modify it so that it is preserved by $+$:

Definition 4.8 Two terms t, u are *symbolic observation equivalent* with respect to b, written $t \simeq_E^b u$, if whenever $t \xrightarrow{b_1,\alpha}_E t'$ with $bv(\alpha) \cap fv(b,t,u) = \emptyset$, then there is a $b \wedge b_1$-partition B with the following property: for each $b' \in B$ there exists a $u \overset{b_2,\alpha'}{\Longrightarrow}_E u'$ such that $b' \models b_2$ and

1. if $\alpha \equiv c!e$ then $\alpha' \equiv c!e'$, $b' \models e = e'$ and $t' \approx_E^{b'} u'$
2. otherwise $\alpha \equiv \alpha'$ and $t' \approx_E^{b'} u'$

and symmetrically for u. □

Proposition 4.9 *The relation* $\simeq_E^{true}$ *is preserved by all the operators in the language.*

Theorem 4.10 $t \simeq_E^b u$ *if and only if* $t\rho \simeq_e u\rho$ *for every* ρ *which satisfies* b.

Proof: Similar to that of Theorem 3.2. □

So completeness of the proof system will follow if we can prove $t \simeq_E^b u$ implies $\vdash_2 b \rhd t = u$. This we will do by induction on the *weak size* of terms, $\|t\|$, defined as follows:

1. $\|nil\| = 0$
2. $\|\tau.t\| = \|t\|$
3. $\|\alpha.t\| = 1 + \|t\|$ if $\alpha \not\equiv \tau$
4. $\|t + u\| = \ max\{\|t\|, \|u\|\}$
5. $\|b \to t\| = \|t\|$

The crucial lemma is the following:

Lemma 4.11 *(Absorption) If $t \overset{b,\alpha}{\Longrightarrow}_E t'$ with $fv(b) \cap bv(\alpha) = \emptyset$, then $\vdash_2 t = t + b \to \alpha.t'$.*

Proof: By induction on why $t \overset{b,\alpha}{\Longrightarrow}_E t'$.

1. $t \xrightarrow{b,\alpha}_E t'$.
   - $\alpha'.t_1 \xrightarrow{true,\alpha}_E t'$ with $\alpha'.t_1 \equiv_\alpha \alpha.t'$. Use S3.
   - $b' \to t_1 \xrightarrow{b' \wedge b'',\alpha}_E t'$ because $t_1 \xrightarrow{b'',\alpha}_E t'$. By induction $t_1 = t_1 + b'' \to \alpha.t'$. So
   $$\begin{array}{rcl} b' \to t_1 & = & b' \to (b'' \to \alpha.t') \\ & = & b' \to t_1 + b' \wedge b'' \to \alpha.t' \end{array}$$
   - The other cases are similar.
2. $t \xrightarrow{b',\alpha}_E t_1 \overset{b'',\tau}{\Longrightarrow}_E t'$ with $b \equiv b' \wedge b''$. By induction $t_1 = t_1 + b'' \to \tau.t'$ and $t = t + b' \to \alpha.t_1$. So, since $bv(\alpha) \cap fv(b) = \emptyset$,
   $$\begin{array}{rcl} t & = & t + b' \to \alpha.(t_1 + b'' \to \tau.t') \\ & = & t + b' \to (\alpha.(t_1 + b'' \to \tau.t') + b'' \to \alpha.t') \\ & = & t + b' \to b'' \to \alpha.t' \\ & = & t + b \to \alpha.t' \end{array}$$
3. $t \overset{b',\tau}{\Longrightarrow}_E t_1 \xrightarrow{b'',\alpha}_E t'$ with $b \equiv b' \wedge b''$. Similar to the previous case.

□

Definition 4.12 A normal form $t \equiv \Sigma_i b_i \to \alpha_i.t_i$ is a *full normal* form if

1. $t \overset{b,\alpha}{\Longrightarrow}_E t'$, where $bv(\alpha) \cap fv(b) = \emptyset$, implies $t \xrightarrow{b,\alpha}_E t'$.
2. Each t_i is in full normal form.

□

Lemma 4.13 *For any normal form t there is a full normal form t' such that $fv(t) = fv(t'), \|t\| = \|t'\|$ and $\vdash_2 t = t'$.*

Proof: By structural induction on t using the absorption lemma. □
By this lemma and Lemma 3.4, every term can be transformed into a full normal form of equal weak size.

The following proposition relates symbolic observation equivalence to weak bisimulation. It will be used in the proof of the completeness theorem.

Proposition 4.14 *$t \approx^b_E u$ if and only if there is a b-partition B such that for all $b' \in B$, $t \simeq^{b'}_E u$ or $t \simeq^{b'}_E \tau.u$ or $\tau.t \simeq^{b'}_E u$*

Proof: The "if" part is trivial. For the "only if" part, because of Lemma 3.4 we can assume t, u are in normal form. By the construction similar to that used in the proof of Proposition 2.3, we can further assume $t \equiv \Sigma_{i\in I} c_i \rightarrow \Sigma_{k\in I_i} \alpha_{ik}.t_{ik}$, $u \equiv \Sigma_{j\in J} d_j \rightarrow \Sigma_{l\in J_j} \beta_{jl}.u_{jl}$, where $c_i \wedge c_{i'} = false, i \neq i', d_j \wedge d_{j'} = false, j \neq j', \bigvee_{i\in I} c_i = true$, and $\bigvee_{j\in J} d_j = true$.

Set $B' = \{\, b \wedge c_i \wedge d_j \mid i \in I,\ j \in J \,\}$. Then $\bigvee B' = b$. Consider an arbitrary $b' \equiv b \wedge c_i \wedge d_j \in B'$. Since $b' \models b$, $t \approx^{b'}_E u$. So for every $t \xrightarrow{c_i,\tau}_E t_k$, there exists a b'-partition B_k with the property that for each $b_1 \in B_k$ there is a $u \overset{d_j,\hat{\tau}}{\Longrightarrow}_E u'$ s.t. $t_k \approx^{b_1}_E u'$, and for every $u \xrightarrow{d_j,\tau}_E u_l$ there exists a b'-partition B_l with the property that for each $b_2 \in B_l$ there is a $t \overset{c_i,\hat{\tau}}{\Longrightarrow}_E t'$ s.t. $t' \approx^{b_2}_E u_l$.

Let $B^1 = \{\, \wedge_k b_{k_i} \mid b_{k_i} \in B_k \,\}$, $B^2 = \{\, \wedge_l b_{l_j} \mid b_{l_j} \in B_l \,\}$, and $B_{b'} = \{\, b_1 \wedge b_2 \mid b_1 \in B^1, b_2 \in B^2 \,\}$. Then $\bigvee B_{b'} = b'$. Furthermore $B_{b'}$ has the property that for each $b'' \in B_{b'}$, $t \approx^{b''}_E u$, and whenever $t \xrightarrow{c_i,\tau}_E t'$, there is a $u \overset{d_j,\hat{\tau}}{\Longrightarrow}_E u' \approx^{b''}_E t'$; whenever $u \xrightarrow{d_j,\tau}_E u'$ there is a $t \overset{c_i,\hat{\tau}}{\Longrightarrow}_E t' \approx^{b''}_E u'$. If for some t' $t \xrightarrow{c_i,\tau}_E t' \approx^{b''}_E u$ then $t \simeq^{b''}_E \tau.u$; If for some u' $u \xrightarrow{d_j,\tau}_E u' \approx^{b''}_E t$ then $\tau.t \simeq^{b''}_E u$. Otherwise we can show $t \simeq^{b''}_E u$ as follows. Let $t \xrightarrow{c_i,\alpha}_E t'$; then since $t \approx^{b''}_E u$ we have $u \xrightarrow{d_j,\hat{\alpha}}_E u' \approx^{b''}_E t'$. By assumption u' can not be u itself when $\alpha \equiv \tau$, so $u \xrightarrow{d_j,\alpha}_E u' \approx^{b''}_E t'$ as required. By symmetry, $t \simeq^{b''}_E u$.

Finally, set $B = \cup_{b'\in B'} B_{b'}$. Then B has the required property. □

Theorem 4.15 *(Completeness) $t \simeq^b_E u$ implies $\vdash_2 b \rhd t = u$.*

Proof: We may assume t, u are in full normal form and apply induction on the joint weak size of t and u. The case that the size is 0 is trivial. So let $t \equiv \Sigma_{i\in I} c_i \rightarrow \alpha_i.t_i$, $u \equiv \Sigma_{j\in J} d_j \rightarrow \alpha_j.u_j$. We use the notations $I_\gamma, J_\gamma, t_\gamma, u_\gamma$ as defined in the proof of Theorem 3.5. Consider the case $\gamma \equiv c?$. Let $z \notin fv(b, t, u), i \in I_{c?}, j \in J_{c?}$, and

$$\begin{array}{ll} t^z_{c?} = \Sigma_{i\in I_{c?}} c_i \rightarrow c?z.t_i[z/x_i] & t^\tau_{c?} = \Sigma_{i\in I_{c?}} c_i \rightarrow \tau.t_i[z/x_i] \\ u^z_{c?} = \Sigma_{j\in J_{c?}} d_j \rightarrow c?z.u_j[z/y_j] & u^\tau_{c?} = \Sigma_{j\in J_{c?}} d_j \rightarrow \tau.u_j[z/y_j] \end{array}$$

Since $t \simeq^b_E u$, we have $t^z_{c?} \simeq^b_E u^z_{c?}$ and therefore $t^\tau_{c?} \simeq^b_E u^\tau_{c?}$.

To prove $\vdash_2 b \rhd t_{c?} = u_{c?}$ it is sufficient to establish

$$\vdash_2 b \wedge c_i \rhd u_{c?} + c_i \rightarrow c?x_i.t_i = u_{c?}$$

for each $i \in I_{c?}$.

Now $t_{c?} \xrightarrow{c_i, c?z}_E t_i[z/x_i]$, so there is a $b \wedge c_i$-partition B with the property that for each $b' \in B$ there is $u_{c?} \xrightarrow{d_j, c?z}_E u_j[z/y_j] \overset{d,\hat{\tau}}{\Longrightarrow}_E u'$ s.t. $b' \models d_j \wedge d'$ and $t_i[z/x_i] \approx_E^{b'} u'$. By Proposition 4.14 there exists a b'-partition B', for each $b'' \in B'$ $t_i[z/x_i] \simeq_E^{b''} u'$ or $t_i[z/x_i] \simeq_E^{b''} \tau.u'$ or $\tau.t_i[z/x_i] \simeq_E^{b''} u'$. By induction, together with TAU and T1, in each case we can derive

$$b'' \rhd \tau.u' = \tau.t_i[z/x_i]$$

Applying CUT on B' we get $b' \rhd \tau.u' = \tau.t_i[z/x_i]$.

If $u_j[z/y_j] \equiv u'$, then $b' \rhd \tau.u_j[z/y_j] = \tau.t_i[z/x_i]$ and hence $b' \rhd \tau.u_j[z/y_j] = \tau.u_j[z/y_j] + \tau.t_i[z/x_i]$.
Otherwise, by absorption $u_j[z/y_j] = u_j[z/y_j] + d' \rightarrow \tau.u'$. Therefore

$$\begin{array}{rcl}
b' \rhd \tau.u_j[z/y_j] & = & \tau.(u_j[z/y_j] + d' \rightarrow \tau.u') \\
 & = & \tau.(u_j[z/y_j] + d' \rightarrow \tau.u') + d' \rightarrow \tau.u' \\
 & = & \tau.u_j[z/y_j] + d' \rightarrow \tau.u' \\
 & = & \tau.u_j[z/y_j] + d' \rightarrow \tau.t_i[z/x_i]
\end{array}$$

Since $b' \models d'$, by Proposition 2.2.*3* it follows that

$$b' \rhd \tau.u_j[z/y_j] = \tau.u_j[z/y_j] + \tau.t_i[z/x_i].$$

Similarly, because $b' \models c_i \wedge d_j$, we have

$$b' \rhd d_j \rightarrow \tau.u_j[z/y_j] = d_j \rightarrow \tau.u_j[z/y_j] + c_i \rightarrow \tau.t_i[z/x_i].$$

So

$$b' \rhd u^{\tau}_{c?} = u^{\tau}_{c?} + c_i \rightarrow \tau.t_i[z/x_i].$$

This is true for each b' in the $b \wedge c_i$-partition B. So applying CUT we obtain $b \wedge c_i \rhd u^{\tau}_{c?} = u^{\tau}_{c?} + c_i \rightarrow \tau.t_i[z/x_i]$. By Proposition 2.3 and α-CONV, we get the required $b \wedge c_i \rhd u_{c?} + c_i \rightarrow \alpha_i.t_i = u_{c?}$. $\square$

5 The Late Case

In this section we briefly outline how the theory developed in the previous sections can be carried over to the late case with some systematic modifications. It turns out that only those parts concerning input actions need changing. For brevity we only treat weak equivalence.

The late operational semantics of this language is given in Figure 6. For late symbolic operational semantics we use the same set of rules as in Figure 4, but for notational consistency $\longrightarrow_E$ is repaced by $\longrightarrow_L$.

The late double arrow relations are also defined in the same way as in the early case in Section 4, with $\Longrightarrow_l$ in place of $\Longrightarrow_e$ and $\Longrightarrow_L$ in place of $\Longrightarrow_E$, except that the last clause is only given for non-input actions in each case. So input actions do *not* absorb τ moves after them.

$$
\begin{array}{lll}
a.t \xrightarrow{a}_l t & & a \in \{\tau\} \cup \{\, c!v \mid c \in Chan \,\} \\
c?x.t \xrightarrow{c?x}_l \lambda x.t & & \\
t \xrightarrow{\alpha}_l t' & \text{implies} & t + u \xrightarrow{\alpha}_l t' \\
t \xrightarrow{\alpha}_l t', b = true & \text{implies} & b \to t \xrightarrow{\alpha}_l t'
\end{array}
$$

Figure 6: Late Operational Semantics - closed terms

Definition 5.1 A symmetric relation R between closed terms is a late weak bisimulation if it satisfies: $(p, q) \in R$ implies

- If $p \xrightarrow{c?x}_l \lambda x.t$ then there exists $q \overset{c?y}{\Longrightarrow}_l \lambda y.u$ and for all $v \in Val$ $\exists q'$ s.t. $u[v/y] \Longrightarrow_l q'$ and $(t[v/x], q') \in R$
- For any other action a, if $p \xrightarrow{a}_l p'$ then there exists $q \overset{\hat{a}}{\Longrightarrow}_l q'$ and $(p', q') \in R$

Let $\approx_l$ be the largest late weak bisimulation. □

The late observation equivalence is then defined in terms of late weak bisimulation:

Definition 5.2 Two closed terms is p, q are late observation equivalent, written $p \simeq_l q$, if

- Whenever $p \xrightarrow{c?x}_l \lambda x.t$ then there exists $q \overset{c?y}{\Longrightarrow}_l \lambda y.u$ and for all $v \in Val$ $\exists q'$ s.t. $u[v/y] \Longrightarrow_l q'$ and $t[v/x] \approx_l q'$
- For any other action a, whenever $p \xrightarrow{a}_l p'$ then there exists $q \overset{a}{\Longrightarrow}_l q'$ and $p' \approx_l q'$

And similarly for q.
This relation can be generized to open terms by letting $t \simeq_l u$ iff $t\rho \simeq_l u\rho$ for any ρ. □

It can be shown that $\simeq_l$ is preserved by all operators in our language. In general it is finer that $\simeq_e$; a typical example of a distinction made by $\simeq_l$ but not by $\simeq_e$ is discussed in the Introduction.

Similarly we can define late weak symbolic bisimulation:

Definition 5.3 $\mathbf{S} = \{\, S^b \mid b \in BExp \,\}$ is a late weak symbolic bisimulation if

$(t, u) \in S^b$ implies whenever $t \xrightarrow{b_1, \alpha}_L t'$ with $bv(\alpha) \cap fv(b, t, u) = \emptyset$, then there is a $b \wedge b_1$-partition B such that $fv(B) \subseteq fv(b)$ and for each $b' \in B$ there exists a $u \overset{b_2, \hat{\alpha}'}{\Longrightarrow}_L u'$ such that $b' \models b_2$ and

1. if $\alpha \equiv c!e$ then $\alpha' \equiv c!e'$, $b' \models e = e'$ and $(t', u') \in S^{b'}$
2. if $\alpha \equiv \tau$ then $\alpha' \equiv \tau$ and $(t', u') \in S^{b'}$
3. if $\alpha \equiv c?x$ then $\alpha' \equiv c?x$ and there is a b'-partition B' s.t for each $b'' \in B'$ there is $u' \overset{b_2', \hat{\tau}}{\Longrightarrow}_L u''$ s.t $b'' \models b_2'$ and $(t', u'') \in S^{b''}$.

Let $\approx_{\mathbf{L}} = \{\approx_L^b\}$ be the largest late weak symbolic bisimulation. □

It is important to note that we now require $fv(B) \subseteq fv(b)$; hence when $\alpha \equiv c?x$ it is guaranteed $x \notin fv(B)$. So we can not partition over the value space for an input variable. This makes all the differences between early and late bisimulations!

Late weak symbolic observation equivalence is defined in terms of weak symbolic bisimulation:

Definition 5.4 Two terms t, u are *late weak symbolic observation equivalence* over b, written $t \simeq^b_L u$, if whenever $t \xrightarrow{b_1,\alpha}_L t'$ with $bv(\alpha) \cap fv(b,t,u) = \emptyset$, then there is a $b \wedge b_1$-partition B such that $fv(B) \subseteq fv(b)$ and for each $b' \in B$ there exists a $u \overset{b_2,\alpha'}{\Longrightarrow}_L u'$ such that $b' \models b_2$ and

1. if $\alpha \equiv c!e$ then $\alpha' \equiv c!e'$, $b' \models e = e'$ and $t' \approx^{b'}_L u'$
2. if $\alpha \equiv \tau$ then $\alpha' \equiv \tau$ and $t' \approx^{b'}_L u'$
3. if $\alpha \equiv c?x$ then $\alpha' \equiv c?x$ and there is a b'-partition B' s.t for each $b'' \in B'$ there is $u' \overset{b'_2}{\Longrightarrow}_L u''$ s.t $b'' \models b'_2$ and $t' \approx^{b''}_L u''$.

and symmetrically for u. □

The late versions of Theorems 4.3 and 4.10 can be proved similarly as their early counterparts:

Theorem 5.5 *$t \simeq^b_L u$ if and only if $t\rho \simeq_l u\rho$ for every $\rho \models b$.*

The inference system for late symbolic observation equivalence can be obtained by replacing E-INPUT in Figure 2 with the following simpler rule

$$\text{L-INPUT} \qquad \frac{b \rhd t = u}{b \rhd c?x.t = c?x.u} \quad x \notin fv(b)$$

As the inference system is weakened, the τ-law T3 can no longer be generalised to the case of input prefix. So we have to replace it with

$$\text{T3L} \quad \alpha.(X + \tau.Y) + \alpha.Y = \alpha.(X + \tau.Y)$$

Let $\mathcal{A}_{2L}$ be the set of axioms consisting of T1, T2 and T3L. We write $\vdash_{2L} b \rhd t = u$ to denote $b \rhd t = u$ can be derived from the new inference system using axioms in $\mathcal{A}_1$ and $\mathcal{A}_{2L}$.

We have the soundness theorem:

Theorem 5.6 *(Soundness) $\vdash_{2L} b \rhd t = u$ implies $t\rho \simeq_l u\rho$ for every ρ such that $\rho \models b$.*

For the completeness theorem, we use essentially the same form of full normal form as in the early case (keep in mind that now double input arrows only absorb those τ moves before it):

Definition 5.7 A normal form $t \equiv \Sigma_i b_i \rightarrow \alpha_i.t_i$ is a late full normal form if

1. $t \overset{b,\alpha}{\Longrightarrow}_L t'$ implies $t \xrightarrow{b,\alpha}_L t'$.

2. Each t_i is in late full normal form.

$\square$

The absorption lemma still holds (note that now α can not be an input action in the second case in the proof of the lemma). Every term can be transformed to late normal form and the appropriate version of Proposition 4.14 holds.

Theorem 5.8 *(Completeness) $t \simeq^b_L u$ implies $\vdash_{2L} b \rhd t = u$.*

Proof: Assume t, u are in late full normal form and apply induction on the joint weak size of t and u. For the non-trivial case when the size is not 0 let $t \equiv \Sigma_{i \in I} c_i \rightarrow \alpha_i.t_i$, $u \equiv \Sigma_{j \in J} d_j \rightarrow \beta_j.u_j$. We need to show

$$b \wedge c_i \rhd u + c_i \rightarrow \alpha_i.t_i = u$$

for each $i \in I$. We only consider the case when $\alpha_i \equiv c?x$ here (the other two cases are the same as in the early case). Let z be a fresh variable, i.e. $z \notin fv(b, t, u)$.

Now $t \xrightarrow{c_i, c?z}_L t'_i[z/x]$. So there exists a $b \wedge c_i$-partition B with $fv(B) \subseteq fv(b \wedge c_i)$ s.t for all $b' \in B$, $b' \models c_i$ and there is $u \xrightarrow{d_j, c?z}_L u_j[z/y]$ s.t. $b' \models d_j$ and there exists a b'-partition B' s.t for all $b'' \in B'$ there is $u_j[z/y] \overset{d', \hat{\tau}}{\Longrightarrow}_L u'$ s.t. $b'' \models d'$ and $t_i[z/x] \approx_L u'$.

By Proposition 4.14 and induction, together with TAU and T1, we can derive

$$b'' \rhd \tau.u' = \tau.t_i[z/x]$$

By an argument similar to that used in Theorem 4.15, using CUT on B', we obtain

$$b' \rhd \tau.u_j[z/y] = \tau.u_j[z/y] + \tau.t_i[z/x].$$

Now, since $z \notin fv(b')$, we can apply L-INPUT to get

$$\begin{array}{rcl} b' \rhd c?z.\tau.u_j[z/y] & = & c?z.(\tau.u_j[z/y] + \tau.t_i[z/x]) \\ & = & c?z.(\tau.u_j[z/y] + \tau.t_i[z/x]) + c?x.t_i[z/x] \\ & = & c?z.\tau.u_j[z/y] + c?z.t_i[z/x] \end{array}$$

By T1 and α-CONV, $b' \rhd c?y.u_j = c?y.u_j + c?x.t_i$. Since $b' \models c_i \wedge d_j$, we can derive $b' \rhd d_j \rightarrow c?y.u_j = d_j \rightarrow c?y.u_j + c_i \rightarrow c?x.t_i$. Hence $b' \rhd u = u + c_i \rightarrow c?x.t_i$. Finally, an application of CUT on B gives the required $b \wedge c_i \rhd u = u + c_i \rightarrow c?x.t_i$. $\square$

6 Extensions

So far we have concentrated on the core language of Section 2. As said in the Introduction it can be easily extended by adding the | (parallel) and \ (restriction) operators. The concrete operational semantics for these operators are standard and we only give their symbolic operational semantics (Figure 7, where symmetric rules have been omitted),

$$\begin{array}{lll}
t \xrightarrow{b,\alpha} t' & \text{implies} & t \mid u \xrightarrow{b,\alpha} t' \mid u \\
 & & \alpha \in \{\, \tau, c!e \mid c \in Chan, e \in Exp \,\} \\
t \xrightarrow{b,c?x} t' & \text{implies} & t \mid u \xrightarrow{b,c?x} t' \mid u \\
 & & x \notin fv(u) \\
t \xrightarrow{b,c?x} t',\ u \xrightarrow{b',c!e} u' & \text{implies} & t \mid u \xrightarrow{b\wedge b',\tau} t'[e/x] \mid u' \\
t \xrightarrow{b,\alpha} t' & \text{implies} & t\backslash c \xrightarrow{b,\alpha} t'\backslash c \\
 & & \text{if } chan(\alpha) \neq c
\end{array}$$

Figure 7: Symbolic Operational Semantics – continued

followed by the equational laws reducing them to the core language. As the symbolic operational semantics is the same for both early and late cases, the "E" and "L" subscripts to $\longrightarrow$ have been omitted.

The equations characterizing the restriction operator and the expansion law for the parallel operator are shown in Figure 8. These laws are fairly standard; they are included here just for the sake of completeness.

It is routine to check that all $\sim_e, \simeq_e, \sim_l$ and $\simeq_l$ are preserved by the new operators, and that the new equations are sound for these equivalence relations.

Now if we add these new equations to $\mathcal{A}_1$, then the three normal form lemmas 3.4, 4.13 and 5.7 carry over to the extended language. From these follow the completeness results (Theorem 3.5, 4.15 and 5.8) for the extended language.

Acknowledgements: The authors would like to thank Luca Aceto for carefully reading a draft of this paper and suggesting many improvements.

$$\begin{array}{l}
nil\backslash c = nil \\
(X+Y)\backslash c = X\backslash c + Y\backslash c \\
(\alpha.X)\backslash c = \left\{ \begin{array}{ll} \alpha.(X\backslash c) & \text{if } chan(\alpha) \neq c \\ nil & \text{if } chan(\alpha) = c \end{array} \right.
\end{array}$$

Let X, Y denote $\Sigma_i \alpha_i.X_i$, $\Sigma_j \beta_j.Y_j$, with $fv(X) \cap bv(Y) = fv(Y) \cap bv(X) = \emptyset$ where $fv(X)$ and $bv(X)$ are free data variables and bound data variables in the term X, respectively. Then

$$X \mid Y = sync\_move(X,Y) + async\_move(X,Y)$$

where

$$\begin{array}{rl}
sync\_move(X,Y) = & \Sigma\{\, \tau.(X_i\{e/x\} \mid Y_j) \mid \alpha_i \equiv c?x, \beta_j \equiv c!e \,\} + \\
 & \Sigma\{\, \tau.(X_i \mid Y_j\{e/x\}) \mid \alpha_i \equiv c!e, \beta_j \equiv c?x \,\} \\
async\_move(X,Y) = & \Sigma_i \alpha_i.(X_i \mid Y) + \Sigma_j \beta_j.(X \mid Y_j)
\end{array}$$

Figure 8: New Equations and Expansion Law

References

[Bur91] G. Burns. A language for value-passing CCS. Technical Report ECS-LFCS-91-175, LFCS, University of Edinburgh, August 1991.

[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.

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

[GP91] J.F. Groote and A. Ponse. Proof theory for μCRL. Technical Report CS-R9138, CWI, Amsterdam, 1991.

[Hen88] M. Hennessy. *Algebraic Theory of Processes.* MIT Press, 1988.

[Hen91] M. Hennessy. A proof system for communicating processes with value-passing. *Formal Aspects of Computing*, 3:346–366, 1991.

[HIar] M. Hennessy and A. Ingolfsdottir. A theory of communicating processes with value-passing. *Information and Computation*, To appear.

[HL92] M. Hennessy and H. Lin. Symbolic bisimulations. Technical Report 1/92, Computer Science, University of Sussex, 1992.

[HL93] M. Hennessy and X. Liu. A modal logic for message passing processes. Technical Report 3/93, Computer Science, University of Sussex, 1993.

[HR88] C.A.R. Hoare and A.W. Roscoe. The laws of occam. Technical Report PRG Monograph, Oxford University, 1988.

[Lin91] H. Lin. PAM: A process algebra manipulator. In *Computer Aided Verification*, volume 575 of *Lecture Notes in Computer Science*, pages 136–146. Springer–Verlag, 1991.

[Lin93] H. Lin. A verification tool for value-passing process. Technical Report Forthcoming, Computer Science, University of Sussex, 1993.

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

[MPW92] R. Milner, J. Parrow, and D. Walker. A calculus of mobile proceses, part i. *Information and Computation*, 100(1):1–40, 1992.

[PS93] J. Parrow and D. Sangiorgi. Algebraic theories for value-passing calculi. Report Forthcoming, SICS and Edinburgh University, 1993.

[Wal89] D. Walker. Automated analysis of mutual exclusion algorithms using CCS. *Formal Aspects of Computing*, 1:273–292, 1989.