Symbolic Bisimulations*

M. Hennessy, H. Lin
Computer Science
University of Sussex

Abstract

We re-examine bisimulation equivalence for value-passing process languages in which actions have associated with them values from a possibly infinite value set. Using *symbolic actions* we generalise the standard notion of labelled transition graph to that of *symbolic transition graph*. The advantage of the latter is that the operational semantics of many value-passing processes may be expressed in terms of finite symbolic transition graphs although the underlying (standard) labelled transitions graph is infinite.

A collection of *symbolic bisimulations* parameterised on boolean expressions, $\simeq^b$, are then defined over symbolic transition graphs. These are related to standard bisimulations by proving that $t \simeq^b u$ if and only if in every interpretation which satisfies b t is bisimulation equivalent to u in the standard sense. We then give an algorithm for checking the relation $t \simeq^b u$ which can be applied to a class of finite symbolic transition graphs which we call *standard*.

The results apply to both *early* and *late* bisimulation equivalence, which are the two natural generalisations of the standard bisimulation equivalence to value-passing languages.

1 Introduction

Bisimulation equivalence, [Mil89], provides a useful semantic theory for process description languages. However it has the disadvantage that for value-passing processes, where the values are from an infinite data-space, in order to check for equivalence infinite transition graphs must be compared. The object of this paper is to redefine this equivalence at a more abstract level so that in many cases the checking may be carried out by comparing finite transition graphs. In order to explain our approach we first define some simple concurrent processes..

Consider the two descriptions

$$S \quad \Leftarrow \quad a.\tau.b.\tau.S$$

*The authors would like to acknowledge the financial support of the SERC grants GR/H16537 and GR/H13932.

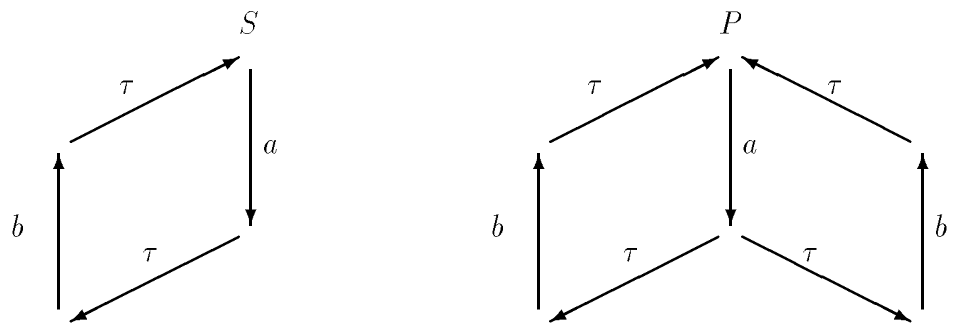

Figure 1: Transition Graphs for S and P

and

$$\begin{array}{rcl} P & \Leftarrow & (Q \mid R \mid R)\backslash\{\alpha, \beta\} \\ Q & \Leftarrow & a.\alpha.\beta.Q \\ R & \Leftarrow & \overline{\alpha}.b.\overline{\beta}.R \end{array}$$

written in the language *CCS*. The first is a simple cyclic process which performs the action a followed by a τ action, a b action and finally another τ action to arrive back at its original start state. Here a and b are some formal uninterpreted actions while τ is a special action which denotes internal unobservable activity. One such activity is an internal communication or synchronisation between two subprocesses which is modelled in *CCS* by the simultaneous occurrence of complementary actions such as a and $\overline{a}$. So in *CCS* synchronisation is a binary operation between exactly two processes. The second process above, P, consists of three subprocesses running in parallel, Q and two copies of R. Q first performs the external action a and then synchronises with one of the copies of R using the action α. That copy now performs the external action b and then synchronises with Q using the other internal action β while the other copy of R is forced to idle. The operator $\backslash\{\alpha, \beta\}$ indicates that the two actions α and β can only be used for internal purposes and are not visible to external users. So in this process their only manifestation is their participation in the τ actions. Although these two descriptions are quite different in nature, semantically they are deemed to be equivalent; according to the definition of bisimulation equivalence $S \sim P$. As another example consider

$$S' \quad \Leftarrow \quad c?x.\tau.d!\lfloor x/2 \rfloor.\tau.S'$$

and

$$\begin{array}{rcl} P' & \Leftarrow & (Q' \mid R' \mid R' \mid T' \mid T')\backslash\{in_1, in_2, \beta\} \\ Q' & \Leftarrow & c?x.(even(x) \rightarrow in_1!x.\beta.Q', in_2!x.\beta.Q') \\ R' & \Leftarrow & in_1?x.d!\lceil x/2 \rceil.\overline{\beta}.R' \\ T' & \Leftarrow & in_2?x.d!\lceil (x-1)/2 \rceil.\overline{\beta}.T' \end{array}$$

Here we use a value-passing version of *CCS* where the actions are interpreted in terms of communication channels. Input of a value for a variable x along a channel c is denoted by

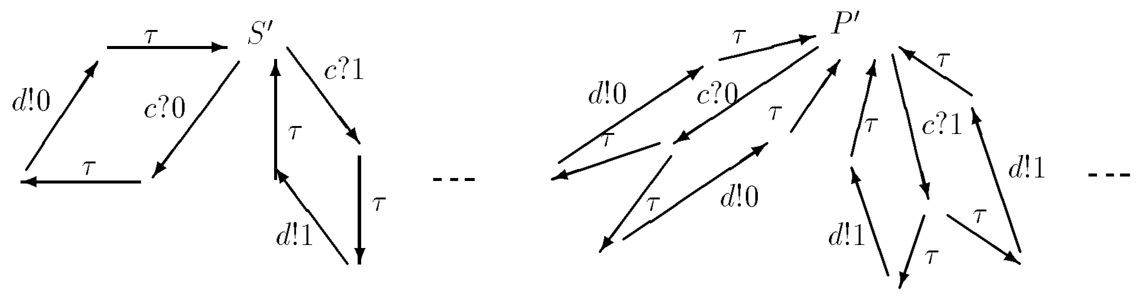

Figure 2: Transition Graphs for S' and P'

$c?x$ while $c!e$ denotes the output of the value of the expression e along c. Communication is modelled as before with τ representing the simultaneous occurrence of complementary actions; with these interpreted actions input and output along the same channel are considered to be complementary. So S describes a process which inputs a value on the channel c, does some internal activity before outputing $\lfloor x/2 \rfloor$ on the channel d, then engages in internal activity again. The description P' is more detailed. This process consists of five process running in parallel. The first, Q' inputs a value on c and outputs it immediately on one of the internal channels in_1 or in_2 depending on whether or not it is even. It then synchronises with the process receiving the output value, which is one of the copies of either R' or T' depending on which internal channel is used. The copy of R' outputs the value $\lceil x/2 \rceil$ on the channel d and then synchronises with Q' while that of T' outputs the value $\lceil (x-1) \rceil /2$.
Once more, although these descriptions are quite different, it turns out that $S' \sim P'$ because they offer essentially the same behaviour to their respective environments.

There are a large number of verification tools which have at their core algorithms for checking bisimulation equivalence between processes, [CPS89, SV89, GLZ89]. By and large these tools do not work directly on syntactic descriptions such as those above but rather on more abstract representations of the behaviour of processes. So for example the operational behaviour of P and S can be represented by the transition graphs in Figure 1 while those for P' and S' are in Figure 2. These graphs are convenient representations of the possible transitions which the processes can perform. The two graphs in Figure 1 are finite and when the standard algorithm is applied to them it returns *true*. However the graphs in Figure 2 are infinite, assuming that the value-space is the set of natural numbers, and therefore when the algorithm is applied to them it will never terminate, although they are bisimulation equivalent.

This is a fundamental limitation of the existing algorithms for bisimulation equivalence; because they only apply to finite transition graphs they are of very limited use for value-passing languages. The aim of this paper is to develop new more powerful algorithms which can be applied to a large class of processes which are defined in these value-passing description languages. The idea is to transfer attention from the standard form of transition graphs to what we call *symbolic transition graphs*. These are more abstract descriptions of processes in terms of *symbolic actions*. For example the symbolic graphs associated with S' and P' are given in Figure 3. These are both finite graphs where the symbolic actions are of the form $c?x$, $d!x/2$, $d!(x-1)/2$ and $d!\lfloor x/2 \rfloor$ and τ,

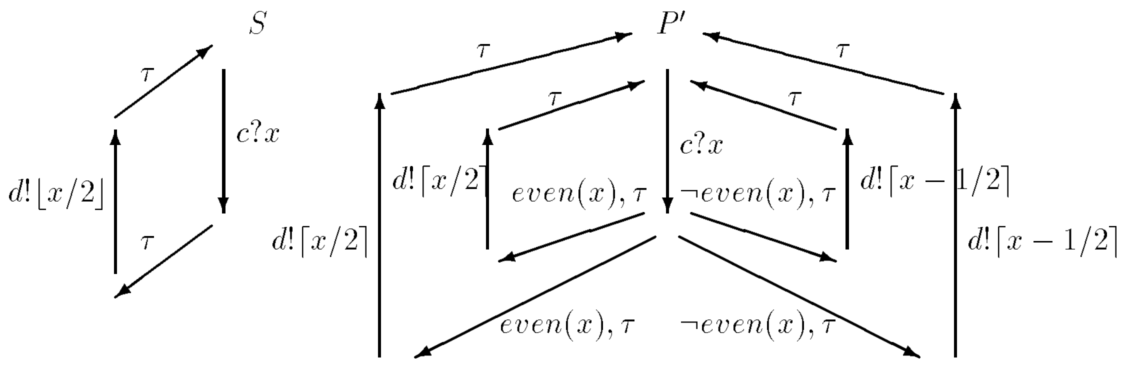

Figure 3: Symbolic Transition Graphs for S' and P'

some of which are guarded by boolean expressions. Our algorithms apply at this level of abstraction and will always return an answer when applied to finite symbolic transition graphs.

Before proceeding further with an outline of the results of the paper we should point out that there are two reasonable variations of bisimulation equivalence which apply to value-passing languages which are often referred to as *early bisimulation equivalence* and *late bisimulation equivalence*, [MPW92]. The variation occurs because of the composite nature of the actions and is best explained using a simple example. Consider the two processes, where again we assume that the value space consists of all the natural numbers:

$$\begin{aligned} P_1 &\Longleftarrow (c?x.even(x) \rightarrow R_1, R_2) + c?x.R_3 \\ P_2 &\Longleftarrow (c?x.even(x) \rightarrow R_1, R_3) + (c?x.odd(x) \rightarrow R_2, R_3) \end{aligned}$$

If we say that the behaviour of these processes is completely determined by their ability to perform actions of the form $c?k$ and $c!k$ where c is a channel and k is a value then it is reasonable to say that P_1 and P_2 are semantically equivalent because each action performed by one can be obviously matched by the other. This is the view taken by early bisimulation equivalence. However there is another view of their behaviour where the set of input actions of the form $c?k$ are replaced by one general action of the form $c?$ which indicates the ability to perform an input action on the channel c without committing to the actual value. Here they are not equivalent since the input move of the form $c?$ from P_1 to $even(x) \rightarrow R_1, R_2$ can not be matched by a corresponding move from P_2. This is the view taken by *late* bisimulation equivalence where for P_2 to be equivalent to P_1 it must be able to make a $c?$ move to a term which, when interpreted as a family of processes indexed by values for x, should be component-wise equivalent to $even(x) \rightarrow R_1, R_2$. For the two possible choices of $c?$ moves, to $even(x) \rightarrow R_1, R_3$ and $odd(x) \rightarrow R_2, R_3$ respectively, there are instantiations for x which violate this requirement, assuming that R_1, R_2 and R_3 are semantically different processes.
Our theory of symbolic bisimulations will be developed for both variations.

Central to our approach is the development of a *symbolic operational semantics* where symbolic actions such as $c?x$, $c!e$ and their associated residuals are associated with terms. So generalising the standard notation for operational semantics [Mil89], we will have for example that

$$c?x.t \xrightarrow{c?x} t \quad \text{and} \quad c!e.t \xrightarrow{c!e} t$$

for arbitrary terms. More generally symbolic actions will have boolean guards associated with them indicating conditions under which they can be performed. Note however that this form of operational semantics must neccessarily be given for *open terms*, i.e. terms which may contain free variables; for example even if $c?x.t$ is a closed term its residual after the symbolic action $c?x$, namely t, will in general contain free occurrences of x. This will complicate to some extent the actual definition of the symbolic operational semantics. Nevertheless we use these formal actions to define two symbolic variants of bisimulation equivalence, a late and early version. It will be convenient to parametrise these on boolean expressions. In this case we will have relations of the form $\simeq_E^b$ and $\simeq_L^b$ between open terms. For example $t \simeq_E^b u$ indicates that with respect to the early version of the symbolic operational semantics t and u are bisimulation equivalent relative to the boolean expression b. Intuitively this is meant to indicate that in every interpretation which satisfies the boolean expression b the processes t and u are bisimulation equivalent. The boolean expressions used to parameterise the equivalences are assumed to be from from some language for describing boolean propositions. Although we do not give any syntax for such language it should be noted that these expressions may contain free variables so that in some interpretations, i.e. assigments of values to variables, a boolean expression may evaluate to true and in others to false.

If we interpret these terms, by assigning values to the free variables, then we can also give concrete operational semantics in terms of the concrete actions $c?v$ and $c!v$ and this in turns leads to a concrete bisimulation equivalence between terms. This level of semantics corresponds to the standard approach as found for example in [Mil89]. Once more there is a late and early version and, if we use ρ to range over assigments of values to free variables, we obtain relations of the form

$$\rho \models t \sim_E u \quad \text{and} \quad \rho \models t \sim_L u.$$

Intuitively these mean that with respect to the assigment ρ t is early/late bisimulation equivalent to u.

Our first major result relates the abstract and concrete versions of these equivalences. We show that

> $t \simeq_i^b u$ if and only if for every assignment ρ which satisfies the boolean b $\rho \models t \sim_i u$, where i is either E or L.

This result underlies the significance of symbolic bisimulations. For example it shows that the standard form of bisimulation equivalence between closed terms, $\sim_i$, coincides with $\simeq_i^{true}$. The crucial difference between these two relations is that the former is defined on concrete transition graphs, which for value-passing languages are nearly always infinite, while the latter is defined on symbolic transition graphs which are frequently finite.

The second part of the paper is devoted to developing algorithms to decide symbolic bisimulation equivalences for finite symbolic transition graphs. For two terms t and u there may be many booleans b for which $t \simeq_i^b u$; for example it turns out that $t \simeq^{false} u$ for all terms t, u. We are interested in calculating the weakest boolean for which $t \simeq_i^b u$. We call this $mgb_i(t, u)$ which has the property that $t \simeq_i^{mgb_i(t,u)} u$ and whenever $t \simeq_i^b u$ then b implies $mgb_i(t, u)$. We also wish to generate a symbolic bisimulation which provides a witness to the fact that $t \simeq_i^{mgb_i(t,u)} u$. Of course even on finite symbolic transition graphs these bisimulations are in general infinite because we must exhibit a suitable relation, R^b, for each boolean expression. However we can easily find a finite representation by using the fact that if b implies b' then $t \simeq_i^{b'} u$ implies $t \simeq_i^b u$. For both the early and late case we present algorithms which given a pair t, u returns a boolean expression logically equivalent to $mgb_i(t, u)$ and the finite representation of a witnessing symbolic bisimulation. The algorithms apply to what we called *standard graphs*, finite symbolic graphs which satisfy some condition on the use of bound variables.

Our algorithms are similar to the bisimulation checking algorithm from [Lar86] in that both follow closely the definition of bisimulations. When given two terms t, u the algorithm will return a boolean expression equivalent to $mgb_i(t, u)$. In this sense we reduce bisimulation equivalence to the logical equivalence of boolean expressions. Of course if the language for expressions is at all complicated bisimulation equivalence will be undecidable as indeed will the equivalence between the corresponding boolean expressions. There is no way of avoiding this problem and our approach at least provides a systematic way of checking bisimulation equivalence which is parameterised on the language for boolean and data expressions.

The algorithms we propose are independent of the language used to define expressions but to be useful we need to be able to simplify the returned expressions into some form of minimal form or at least a readable form. We have implemented the algorithms and a fairly naive set of simplification rules works reasonably well. We hope to develop future versions which will be guided by the user, principally by seeking the user's help in simplifying expressions as they are being generated rather than at present when the only simplification is carried out at the end. We also hope to extend the algorithms to handle other semantic equivalences such as weak bisimulation and testing equivalence.

We now outline the contents of the subsequent sections. In the next section we formally define symbolic transition graphs. By working directly with symbolic transition graphs our results are independent of any particular process description language. However as an example of how to generate such graphs we give a symbolic operational semantics to a value-passing version of *CCS* which associates with each open term of the language such a graph. The next section, Section 3, is devoted to late bisimulation equivalence for symbolic transition graphs and this is followed by a section on late symbolic bisimulation equivalence which includes a proof of the relationship between these two equivalences, as explained above. The algorithm for generating symbolic bisimulations is described in Section 5. Section 6 outlines the changes necessary to handle early bisimulation equivalence and finally some conclusions are drawn in Section 7.

2 Symbolic Transition Graphs

Symbolic transition graphs are parameterised on a number of syntactic categories. The first two is a set of *variables*, *Var*, which we assume to be totally well-ordered, and a set of values V. *Eval*, ranged over by ρ , represents the set of *evaluations*, i.e. the set of total functions from *Var* to V. We use the standard notation $\rho[v/x]$ to denote the evaluation which differs from ρ only in that it maps x to v. A substitution is a partial injective mapping from *Var* to *Var* whose domain is finite. We use *Sub* to represent the set of substitutions and this set is ranged over by σ. We use the notation $\sigma[x \mapsto y]$ to

indicate the obvious modification to the substitution σ.
We also presume a set of *expressions*, *Exp*, ranged over by e, which includes *Var* and V. Each e has associated with it a set of free variables, $fv(e)$, and it is assumed that both evaluations and substitutions behave in a reasonable manner when applied to expressions; the application of ρ to e, denoted $\rho(e)$, yields a value while the application of a substitution, denoted $e\sigma$, yields another expression with the property that $fv(e\sigma) = \sigma(fv(e))$ where the latter is defined in the obvious manner. It is also assumed that if ρ and ρ' agree on $fv(e)$ then $\rho(e) = \rho'(e)$. We also presume a set of boolean expressions, *BExp*, ranged over by b, with similar properties but we will use the more suggestive notation $\rho \models b$ to indicate that $\rho(b) = true$. By and large we do not wish to worry about the expressive power of these expressions but we will assume that boolean expressions are closed under the usual connectives and contains $e = e'$ for every pair of expressions e and e'. In some sections, those concerned with our theoretical results, we will have to assume that the language for boolean expressions is extremely powerful; more or less capable of describing any collection of environments.

After these preliminaries we may now define the class of graphs in which we are interested. Essentially they are arbitrary directed graphs in which the nodes are labelled by a set of variables, intuitively the set of free variables of that node, and the branches are labelled by *guarded actions*, pairs of boolean expressions and actions. An action may be an *input* action, of the form $c?x$ where c is from a set of channels, *Chan*, an output action, of the form $c!e$, or a *neutral* action such as τ. So let *SyAct*, ranged over by α, represent the set of symbolic actions; it has the form

$$SyAct = \{\, c?x,\ c!e \mid c \in Chan \,\} \cup NAct$$

where *NAct* is some set of neutral actions. The set of free and bound variables of these actions are defined in the obvious manner: $fv(c!e) = fv(e)$, $bv(c?x) = \{x\}$ and otherwise both $fv(\alpha)$ and $bv(\alpha)$ are empty. Then the set of guarded actions

$$GuAct = \{\, (b, \alpha) \mid b \in BExp, \alpha \in SyAct \,\}.$$

We use β to range over *GuAct*.

Definition 2.1 (Symbolic Transition Graphs)
A symbolic transition graph is a directed graph in which every node n is labelled by a set of variables $fv(n)$ and every branch is labelled by a guarded action such that if a branch labelled by (b, α) goes from node m to n, which we write as $m \overset{b,\alpha}{\longmapsto} n$, then $fv(b) \cup fv(\alpha) \subseteq fv(m)$, and $fv(n) \subseteq fv(m) \cup bv(\alpha)$. □

We will frequently write $m \overset{\alpha}{\longmapsto} n$ for $m \overset{true,\alpha}{\longmapsto} n$.

A symbolic transition graph may be looked upon as a particularly austere representation of the abstract syntax of a value-passing process algebra. By working at this level of abstraction our results are independent of any particular language. But as an example we show how a symbolic transition graph can be obtained from an example language, based on *CCS*, using the standard approach of structural operational semantics. The abstract syntax of the language is given by

$$\begin{aligned} t \ ::= \ & nil \mid \alpha.t \mid be \rightarrow t, t \\ & t + t \mid t \mid t \mid t \backslash c \mid P(\underline{e}) \end{aligned}$$

$$
\begin{array}{lll}
a.t \overset{true,\alpha}{\longmapsto} t & & \alpha \in \mathit{NAct} \cup \{\, c!e \mid c \in \mathit{Chan}, e \in \mathit{Exp} \,\} \\
c?x.t \overset{true,c?y}{\longmapsto} t[y/x] & & \text{where } y = \mathit{new}(fv(c?x.t)) \\
t \overset{b',\alpha}{\longmapsto} t' & \text{implies} & (b \to t, u) \overset{b \wedge b',\alpha}{\longmapsto} t' \\
u \overset{b',\alpha}{\longmapsto} u' & \text{implies} & (b \to t, u) \overset{\neg b \wedge b',\alpha}{\longmapsto} u' \\
t \overset{b,\alpha}{\longmapsto} t' & \text{implies} & t + u \overset{b,\alpha}{\longmapsto} t' \\
t \overset{b,\alpha}{\longmapsto} t' & \text{implies} & t \mid u \overset{b,\alpha}{\longmapsto} t' \mid u \\
 & & \alpha \in \mathit{NAct} \cup \{\, c!e \mid c \in \mathit{Chan}, e \in \mathit{Exp} \,\} \\
t \overset{b,c?x}{\longmapsto} t' & \text{implies} & t \mid u \overset{b,c?y}{\longmapsto} t'[y/x] \mid u \\
 & & y = \begin{cases} x & \text{if } x \notin fv(u) \\ \mathit{new}(fv(t'|u)) & \text{otherwise} \end{cases} \\
t \overset{b,c?x}{\longmapsto} t',\ u \overset{b',c!e}{\longmapsto} u' & \text{implies} & t \mid u \overset{b \wedge b',\tau}{\longmapsto} t'[e/x] \mid u' \\
t \overset{b,\alpha}{\longmapsto} t' & \text{implies} & t\backslash c \overset{b,\alpha}{\longmapsto} t'\backslash c \\
 & & \text{if } \alpha \text{ does not use the channel } c \\
t[\underline{e}/\underline{x}] \overset{b,\alpha}{\longmapsto} t' & \text{implies} & P(\underline{e}) \overset{b,\alpha}{\longmapsto} t' \\
 & & \text{if } P(\underline{x}) \Longleftarrow t \text{ is a declaration}
\end{array}
$$

Figure 4: Symbolic Operational Semantics of *CCS*

This contains the usual combinators from *CCS* together with a boolean choice mechanism and it assumes a set of process names, ranged over by P. To give a semantics to the terms we assume the existence of a set of declarations of the form

$$P(\underline{x}) \Longleftarrow t,$$

one for each process name which occurs in the terms, where it is assumed that the free variables of t are contained in the list $\underline{x}$.

In this language $c?x$ binds occurrences of the variable x in the sub-term t of $c?x.t$ and we get as usual the set of free variables, $fv(u)$ of a term u. For each $\beta \in \mathit{GuAct}$ let $\overset{\beta}{\longmapsto}$ be the least relation which satisfies the rules in Figure 4 (the symmetric rules for $+$ and $\mid$ have been omitted). This next-state relation uses a function *new*, which when given a set of variables returns a new variable not in that set. Let us assume that the set of variables, Var, is totally ordered and that $\mathit{new}(V)$ returns the least variable not in V. The symbolic transition graph for the language may now be defined by letting the nodes consist of terms t with associated set of free variables $fv(t)$ and $t \overset{\beta}{\longmapsto} t'$ if we can derive this statement from the rules in Figure 4. One can easily check that the requirements of Definition 2.1 are satisfied. An example of a symbolic transition graph generated from the language in this way has already been seen in Figure 3, although the sets of free variables were not shown. In Figure 5 we give another example of a symbolic graph assuming the declaration

$$P(y) \Longleftarrow c?x.x = y \to d!y.P(y),\ c!(x+y).P(y);$$

it is the graph associated with the term $P(y)$, assuming that $\mathit{new}(y) = x$.

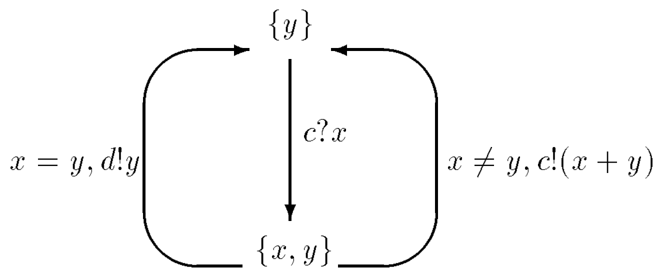

Figure 5: A symbolic transition graph

3 Late Bisimulation Equivalence

The standard definitions of bisimulation equivalence are usually based on an operational semantics which is defined on closed terms of a language. Here the nodes of a symbolic transition system play the role of open terms and therefore we need to define the operational semantics relative to an evaluation. But there is a further complication; the standard definitions of operational semantics rely quite heavily on syntactic substitutions. For example the rules for *CCS* with value-passing would include

$$c?x.p \xrightarrow{c?v} p[v/x]$$

where $p[v/x]$ denotes the closed term obtained by substituting v for all free occurrence of x in p. Working at the more abstract level of symbolic transition systems we have no actual terms into which to make these kind of substitutions. So we have to carry them along in the operational semantics; in other words the proper analogue to an open term in an actual language is a node in a symbolic transition graph together with a substitution, m_σ. Assuming some fixed symbolic transition graph let the set of *terms*

$$\mathcal{T} = \{\, m_\sigma \mid m \text{ is a node}, \ \sigma \in \mathit{Sub}, \ \mathit{domain}(\sigma) \subseteq \mathit{fv}(m) \,\}.$$

We will usually identify the node n with the term $n_\emptyset$ where $\emptyset$ is the empty substitution and use $t, u, \ldots$ to range over $\mathcal{T}$. We will frequently apply notation originally developed for nodes directly to terms and the effect should be obvious. For example the set of free variables of a term m_σ is defined in the obvious way by $\mathit{fv}(m_\sigma) = \{\sigma(\mathit{fv}(m))\}$ and m_σ is said to be *closed* if its set of free variables is empty. We will be somewhat relaxed about the condition that for m_σ to be a term the domain of σ must be contained in the set of free variables of m; but whenever we construct a new term m_σ we assume that it is well-formed in that if necessary the substitution σ is restricted to $\mathit{fv}(m)$. Also if t is a term of the form m_σ, we use $t[x \mapsto z]$ to denote the term $m_{\sigma[x \mapsto z]}$.

A judgement of the late operational semantics then takes the form

$$\rho \models t \xrightarrow{a} u$$

where a is some action from the the set of *late actions*, *LAct*. As explained in the introduction the late operational semantics uses the neutral actions such as τ, the output actions of the form $c!v$ and the more symbolic form of input actions, $c?x$. So *LAct* is defined to be

$$\mathit{NAct} \cup \{\, c!v \mid c \in \mathit{Chan}, \ v \in V \,\} \cup \{\, c?x \mid c \in \mathit{Chan}, \ x \in \mathit{Var} \}.$$

$$\begin{array}{lll}
m \overset{b,a}{\longmapsto} n,\ a \in NAct & \text{implies} & \rho \models m_\sigma \xrightarrow{a} n_\sigma \\
 & & \text{provided } \ \rho \models b\sigma \\[2ex]
m \overset{b,c!e}{\longmapsto} n & \text{implies} & \rho \models m_\sigma \xrightarrow{c!\rho(e\sigma)} n_\sigma \\
 & & \text{provided } \ \rho \models b\sigma \\[2ex]
m \overset{b,c?x}{\longmapsto} n & \text{implies} & \rho \models m_\sigma \xrightarrow{c?x} n_\sigma \\
 & & \text{provided } \ \rho \models b\sigma
\end{array}$$

Figure 6: Late Operational Semantics

The judgements $\rho \models t \xrightarrow{a} u$ are defined to be the least ones which satisfy the rules in Figure 6. These should be more or less self-explanatory; much of the work has already been factored out by the symbolic transition graph and the rules merely interpret the symbolic actions using the evaluation ρ.

The standard definition of a bisimulation is a relation over closed terms and in our setting a closed term is mimiced by a term together with an evaluation. So here a bisimulation will be a collection of relations, parameterised on evaluations. We first define a functional, $\mathcal{LB}$, over such collections of relations. Let $\mathbf{R} = \{ R^\rho \mid \rho \in Eval\}$ be such that each $R^\rho \subseteq \langle \mathcal{T}, \mathcal{T} \rangle$. Then $\mathcal{LB}(\mathbf{R})$ is the *Eval*-indexed family of symmetric relations defined by:
$(t, u) \in \mathcal{LB}(\mathbf{R})^\rho$ if

1. $\rho \models t = m_\sigma \xrightarrow{c?x} m'_\sigma$ implies $\rho \models u = n_\eta \xrightarrow{c?y} n'_\eta$ for some n'_η such that $(m'_{\sigma[x\mapsto z]}, n'_{\eta[y\mapsto z]}) \in R^{\rho[v/z]}$ for all $v \in V$ where z is a fresh variable
2. for any other late action a $\rho \models t \xrightarrow{a} t'$ implies $\rho \models u \xrightarrow{a} u'$ for some u' such that $(t', u') \in R^\rho$

Definition 3.1 (Late Bisimulations)
R is a late bisimulation if $\mathbf{R} \subseteq \mathcal{LB}(\mathbf{R})$, i.e. $R^\rho \subseteq \mathcal{LB}(\mathbf{R})^\rho$, for each ρ. □

We write $\rho \models t \sim_L u$ if there is a late bisimulation **R** such that $(t, u) \in R^\rho$. It will sometimes be more convenient to denote this by $t \sim_L^\rho u$. The standard theory of bisimulations apply here; the functional $\mathcal{LB}$ is (pointwise) monotonic and therefore has a maximal fixpoint and the ρ-th component of this maximal fixpoint coincides with $\sim_L^\rho$. Moreover it is easy to check that each of these relations is an equivalence relation.

We take this semantic equivalence to be the "concrete" behavioural equivalence between processes. It is of course somewhat more abstract than, say, bisimulation equivalence between *CCS* processes as defined in [Mil89] but this is because our definition applies to syntactic transition systems in general. However if we apply the definition to the particular syntactic transition system generated from *CCS* we obtain a "late" version of the standard bisimulation equivalence defined directly on *CCS* terms. This is proved in Appendix A.

We can also show that the equivalence is well-behaved with respect to changes to the free variables; if two terms are equivalent and we apply a substitution then the resulting terms are also equivalent so long as we update the evaluation so as to take the substitution into account:

Proposition 3.2 *If $\rho \models t \sim_L u$ then $\rho \cdot \sigma^{-1} \models t\sigma \sim_L u\sigma$*

A more interesting result is that the equivalence only depends on the free variables of the terms being compared.

Proposition 3.3 *If $\rho(x) = \rho'(x)$ for every $x \in fv(t,u)$ then $\rho \models t \sim_L u$ if and only if $\rho' \models t \sim_L u$.*

Proof: For any $X \subseteq V$ let $\rho =_X \rho'$ if for every $v \in X$ $\rho(x) = \rho'(x)$. Let $\mathbf{R}$ be defined by

$$R^\rho = \{ (t,u) \mid \exists \rho' \,.\, \rho =_{fv(t,u)} \rho' \text{ and } \rho' \models t \sim_L u \}$$

One can prove that if $\rho =_{fv(t)} \rho'$ then $\rho \models t \xrightarrow{a} t'$ if and only if $\rho' \models t \xrightarrow{a} t'$ and from this it follows that $\mathbf{R}$ is a late bisimulation. □

With this *late* operational semantics infinite branching does not necessarily occur because of input moves. But, assuming V is infinite we do in general have to perform an infinite number of comparisons because, intuitively, when matching the move $\rho \models t \xrightarrow{c?x} t'$ with $\rho \models u \xrightarrow{c?x} u'$ we have to ensure that for each $v \in V$ $\rho[v/x] \models t' \sim_L u'$. In the next section we define a symbolic version of the operational semantics where this source of infinite comparisons is eliminated.

4 Symbolic Late Bisimulations

In this section we use the symbolic actions of a symbolic transition system to define a version of bisimulations which captures exactly the collection of concrete relations $\sim_L^\rho$.

Consider the graph in Figure 7, where a, f, g, h are different neutral actions. We have omitted the free variables associated with the nodes and replaced them by tags for ease of reference; the free variables can all be deduced from the fact that $fv(p_0) = fv(q_0) = \emptyset$. It is easy to check that $\rho \models p_0 \sim_L q_0$ for all evaluations ρ.
Let us see how this might be deduced from the symbolic transition graph. The symbolic move $p_0 \overset{c?x}{\longmapsto} p_1$ can be matched by the corresponding move $q_0 \overset{c?x}{\longmapsto} q_1$ if we can show that p_1 and q_1 are symbolically equivalent. Here we come to a problem; we can not always expect, for example, the move $p_1 \overset{a}{\longmapsto} p_{11}$ to be always matched by $q_1 \overset{a}{\longmapsto} q_{11}$ because under some evaluations p_{11} may be behaviourally quite different than q_{11}. To be precise whenever the value associated with x is different than 0 they behave differently. For similar reasons the move can not be matched by $q_1 \overset{a}{\longmapsto} q_{12}$. In general in some interpretations p_{11} can be properly matched by q_{11} and in others by q_{12}.

This leads us to consider a symbolic equivalence parameterised by boolean expressions, $\simeq_L^b$. In the present example we will have $p_{11} \simeq_L^b q_{11}$ whenever b implies $x = 0$ and $p_{11} \simeq_L^b q_{12}$ whenever it implies $x \neq 0$. Let us now reexamine in what way the nodes p_1 and q_1 can be properly matched, i.e. why can we claim $p_1 \simeq_L^b q_1$ for a particular b. The most we can hope for in matching the typical move $p \overset{a}{\longmapsto} p_{11}$ is that the present

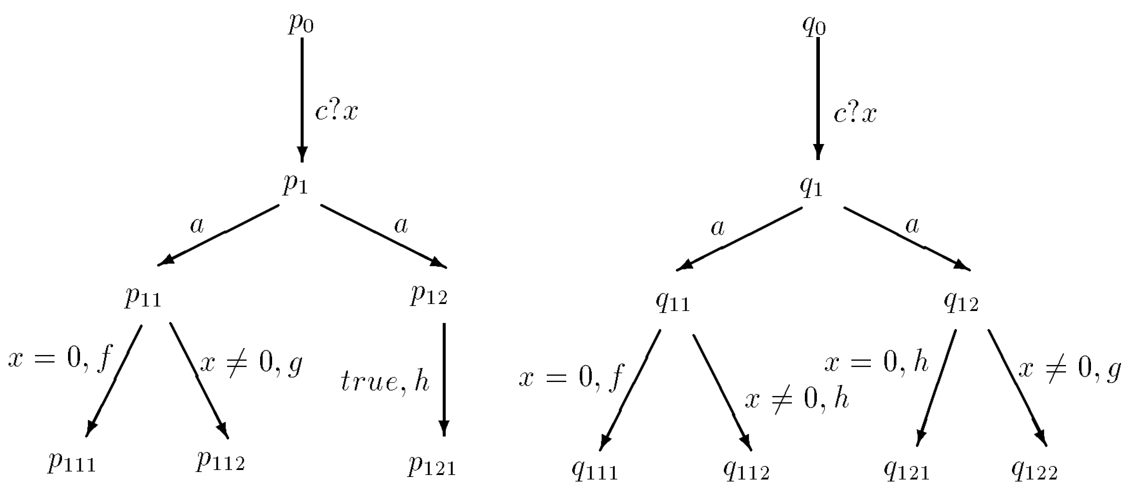

Figure 7: Example graph

circumstance, represented by b, can be divided up into different cases, i.e. there is a set of booleans B such that $b = \bigvee B$, and for each of these individual cases the move can be properly matched. For example if we are trying to check $p_1 \simeq_L^{true} q_1$ then we can let $B = \{x = 0,\ x \neq 0\}$ and in the first case the move is matched by $q_1 \overset{a}{\longmapsto} q_{11}$ while in the second it is matched by $q_1 \overset{a}{\longmapsto} q_{12}$.

We have now explained the ideas behind symbolic bisimulation equivalence. The guards on actions can easily be taken into consideration by demanding some obvious implications between the booleans involved. But unfortunately we can not use the arrows in the transition graphs directly for the same reason as in the previous section; we must work with terms. So the *symbolic late operational semantics* is given as a collection of relations over terms, $\xrightarrow{\beta}_L$, where β is a guarded action. These are defined in Figure 8 and the rules are quite straightforward; essentially they apply the substitution associated with a term when an expression is output or a boolean guard is passed.

With these symbolic actions we now define the symbolic version of bisimulations. This will be a collection of relations over terms, $\{S^b\}$, parameterised by booleans.

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

> $(t, u) \in \mathcal{SLB}(\mathbf{S})^b$ if whenever $t \xrightarrow{b_1, \alpha}_L t'$ there is a collection of booleans B such that $b \wedge b_1 \rightarrow \vee B$ and for each $b' \in B$ there exists a $u \xrightarrow{b_2, \alpha'}_L u'$ such that $b' \rightarrow b_2$ and

1. if α is of the form a where $a \in NAct$ then $\alpha' = a$ and $(t', u') \in S^{b'}$
2. if α is of the form $c!e$ then $\alpha' = c!e'$, $b' \rightarrow e = e'$ and $(t', u') \in S^{b'}$
3. if α is of the form $c?x$ then $\alpha' = c?y$, and $(t'[x \mapsto z], u'[y \mapsto z]) \in S^{b'}$ where z is a fresh variable

Definition 4.1 (Late Symbolic Bisimulations)
$\mathbf{S}$ is a late symbolic bisimulation if $\mathbf{S} \subseteq \mathcal{SLB}(\mathbf{S})$. □

$$
\begin{array}{lll}
m \overset{b,a}{\longmapsto} n,\ a \in NAct & \text{implies} & m_\sigma \xrightarrow{b\sigma,a}_L n_\sigma \\[2ex]
m \overset{b,c!e}{\longmapsto} n & \text{implies} & m_\sigma \xrightarrow{b\sigma,c!e\sigma}_L n_\sigma \\[2ex]
m \overset{b,c?x}{\longmapsto} n & \text{implies} & m_\sigma \xrightarrow{b\sigma,c?x}_L n_\sigma
\end{array}
$$

Figure 8: Late symbolic operational semantics

We write $t \simeq_L^b u$ if there is a symbolic late bisimulation $\mathbf{S}$ such that $(t, u) \in S^b$. As usual the standard theory applies because $\mathcal{SLB}$ is pointwise monotonic. So $\{ \simeq_L^b \mid b \in BExp \}$ is the maximal symbolic late bisimulation. We can also show that each $\simeq_L^b$ is an equivalence relation, using the same approach as with $\sim_L^\rho$.

As an example consider the graph in Figure 7 and let A, B, C be the following pairs of sets:

$$
\begin{aligned}
A &= \{(p_0, q_0), (p_1, q_1)\} \\
B &= \{(p_{11}, q_{11}), (p_{12}, q_{12}), (p_{111}, q_{111}), (p_{121}, q_{121})\} \\
C &= \{(p_{11}, q_{12}), (p_{12}, q_{11}), (p_{111}, q_{122}), (p_{121}, q_{112})\}
\end{aligned}
$$

Then the following is a symbolic bisimulation:

$$
\begin{aligned}
S^{true} &= A \cup A^{-1} \\
S^{x=0} &= B \cup B^{-1} \\
S^{x\neq 0} &= C \cup C^{-1}
\end{aligned}
$$

The remainder of this section is devoted to determining the relationship between symbolic late bisimulations and concrete late bisimulations. We first show the connection between the symbolic actions and the concrete actions.

Proposition 4.2

1. *$\rho \models t \xrightarrow{c?x} t'$ if and only if $t \xrightarrow{b,c?x}_L t'$ for some b such that $\rho \models b$*
2. *$\rho \models t \xrightarrow{c!v} t'$ if and only if $t \xrightarrow{b,c!e}_L t'$ for some b and e such that $\rho \models b$ and $\rho(e) = v$*
3. *$\rho \models t \xrightarrow{a} t'$ if and only if $t \xrightarrow{b,a}_L t'$ for some b such that $\rho \models b$*

Proof: Follows in straightforward manner from the definitions of the two arrows. □

Now let $\mathbf{S}$ be an arbitrary late symbolic bisimulation. Define an *Eval*-indexed collection of relations over terms, $\mathbf{R_S}$, by

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

Proposition 4.3 *If* $\mathbf{S}$ *is a late symbolic bisimulation then* $\mathbf{R_S}$ *is a late bisimulation.*

Proof: Let $(t, u) \in R_{\mathbf{S}}^\rho$, i.e. $(t, u) \in S^b$ for some b such that $\rho \models b$. We must show that the possible moves from t and u are properly matched.

1. Suppose $\rho \models t \xrightarrow{a} t'$ where $a \in NAct$. Then by Proposition 4.2 it follows that $t \xrightarrow{b_1,a}_L t'$ for some b_1 such that $\rho \models b_1$. Therefore there exists a set of booleans B such that $b \wedge b_1 \to \vee B$ and for each $b' \in B$ there is a $u \xrightarrow{b_2,a}_L u'$ such that $b' \to b_2$ and $(t', u') \in S^{b'}$. Since $\rho \models b \wedge b_1$ and $b \wedge b_1 \to \vee B$, there must be $b' \in B$ such that $\rho \models b'$, and hence $\rho \models b_2$. Now apply Proposition 4.2, we have $\rho \models u \xrightarrow{a} u'$ for the u' associated with this b'. Since $\rho \models b'$, $(t', u') \in R^{\rho}_{\mathbf{S}}$ as required.

2. Suppose $\rho \models t \xrightarrow{c!v} t'$. Arguing as in the previous case we get $t \xrightarrow{b_1,c!e_1}_L t'$ for some b_1 and e such that $\rho \models b_1$ and $\rho(e_1) = v$, and there exists a b', $\rho \models b'$ and $u \xrightarrow{b_2,c!e'}_L u'$ such that $(t', u') \in S^{b'}$, $b' \to b_2 \wedge e = e'$, and $\rho(e_2) = v$. This means that $\rho \models u \xrightarrow{c!v} u'$ and $(t', u') \in R^{\rho}_{\mathbf{S}}$.

3. Suppose $\rho \models t \xrightarrow{c?x} t'$. Again arguing as in the previous case this means that at the symbolic level $t \xrightarrow{b_1,c?x}_L t'$ for some b_1 such that $\rho \models b_1$. Furthermore there exists a boolean b' such that $\rho \models b'$ and $u \xrightarrow{b_2,c?y}_L u'$ such that $\rho \models b_2$ and $(m'_{\sigma[x\mapsto z]}, n'_{\eta[y\mapsto z]}) \in S^{b'}$ with z a fresh variable, where t', u' have the forms m'_{σ}, n'_{η} respectively. Applying Proposition 4.2, we obtain $\rho \models u \xrightarrow{c?y} u'$. Moreover $z \notin fv(b')$ because z is fresh. Hence $\rho[v/z] \models b'$ for every v. Therefore $(t'_{\sigma[x\mapsto z]}, u'_{\eta[y\mapsto z]}) \in R^{\rho[v/z]}_{\mathbf{S}}$ for all v, as required.

□

Conversely starting with a late bisimulation $\mathbf{R}$ we can construct a symbolic bisimulation. Here we need to assume that the language for booleans is very expressive; more or less expressive enough to characterise equivalence with particular terms. This is quite a strong requirement but as we will see in the next section at least for finite graphs these booleans can be automatically generated. Let $\mathbf{S_R}$ be defined by

$$S^b_{\mathbf{R}} = \{\, (t, u) \mid \rho \models b \text{ implies } (t, u) \in R^{\rho} \,\}.$$

Proposition 4.4 *If* $\mathbf{R}$ *is a late bisimulation then* $\mathbf{S_R}$ *is a symbolic late bisimulation.*

Proof: Let $(t, u) \in S^b_{\mathbf{R}}$. We show that their symbolic actions can be matched in the appropriate manner.

1. Suppose $t \xrightarrow{b_1,c!e}_L t'$. We must find a set of booleans B such that $b \wedge b_1 \to \vee B$ and for each $b' \in B$ there must exist a move $u \xrightarrow{b_2,c!e'}_L u'$ such that $b' \to b_2 \wedge e = e'$ and $(t', u') \in S^{b'}_{\mathbf{R}}$. For convenience let us assume that for each u' there is at most one β such that $u \xrightarrow{\beta}_L u'$; the same argument hold even without this assumption but we would have to introduce even clumsier notation. Let U be the set $\{\, u' \mid u \xrightarrow{b(u'),c!e(u')}_L u' \,\}$ and for each $u' \in U$ let $b'_{u'}$ be a boolean expression which satisfies

$$\rho \models b'_{u'} \text{ if and only if } (t', u') \in R^{\rho}$$

and let $b_{u'}$ be $b'_{u'} \wedge b(u') \wedge e = e(u')$. Finally let $B = \{\, b_{u'} \mid u' \in U \,\}$.

We first check that $b \wedge b_1 \to \vee B$, i.e. $\rho \models b \wedge b_1$ implies $\rho \models b_{u'}$ for some $u' \in U$. Assuming $\rho \models b \wedge b_1$, it follows that $(t, u) \in R^\rho$ and $\rho \models t \xrightarrow{c!v} t'$ where $\rho(e_1) = v$ (by Proposition 4.2). So this move can be matched by some $\rho \models u \xrightarrow{c!v} u'$ such that $(t', u') \in R^\rho$. This means $\rho \models b'_{u'}$. Applying the same Proposition we obtain $u \xrightarrow{b(u'),c!e(u')}_L u'$ with $\rho \models b(u')$ and $\rho(e(u')) = v$; the latter implies $\rho \models e = e(u')$ and therefore $\rho \models b_{u'}$.

Now for any $b_{u'} \in B$ there exists $u \xrightarrow{b(u'),c!e(u')}_L u'$. It is obvious that $b_{u'} \to b(u') \wedge e = e(u')$ and $(t', u') \in S_{\mathbf{R}}^{b_{u'}}$ follows from the definition of $b_{u'}$.

2. The case $t \xrightarrow{a}_L t'$ is similar.

3. Suppose $t = m_\sigma \xrightarrow{b_1,c?x}_L m'_\sigma$. Again we must find a set of booleans B such that $b \to \vee B$ and for every b' in B there exists a matching move $u = n_\eta \xrightarrow{b_2,c?y}_L n'_\eta$ such that $b' \to b_2$ and $(m'_{\sigma[x\mapsto z]}, n'_{\eta[y\mapsto z]}) \in S_{\mathbf{R}}^{b'}$ with z a fresh variable. We proceed as in the previous case by letting $U = \{\, n'_\eta \mid n_\eta \xrightarrow{b(n'),c?y}_L n'_\eta \,\}$ and defining B to be the set $\{\, b_{n'} \mid n' \in U \,\}$ where $b_{n'} = b'_{n'} \wedge b(n')$ with $b'_{n'}$ a boolean expression satisfying

   $\rho \models b'_{n'}$ if and only if for every value v $(m'_{\sigma[x\mapsto z]}, n'_{\eta[y\mapsto z]}) \in R^{\rho[v/z]}$ with z a fresh variable.

   Again we check that $b \wedge b_1 \to \vee B$, i.e. $\rho \models b \wedge b_1$ implies $\rho \models b_{n'}$ for some $n' \in U$. Assuming $\rho \models b \wedge b_1$ we have $(m_\sigma, n_\eta) \in R^\rho$ and $\rho \models m_\sigma \xrightarrow{c?x} m'_\sigma$ by Proposition 4.2. So there exists $\rho \models u_\eta \xrightarrow{c?y} n'_\eta$ such that $(m'_{\sigma[x\mapsto z]}, n'_{\eta[y\mapsto z]}) \in R^{\rho[v/z]}$ for all v where z is a fresh variable, i.e. $\rho \models b'_{n'}$. Applying Proposition 4.2, there exists $b_{n'}$ such that $n_\eta \xrightarrow{b_{n'},c?y}_L n'_\eta$ and $\rho \models b(n')$; the latter implies $\rho \models b_{n'}$.

   Now for any $b_{n'} \in B$, there is a $n_\eta \xrightarrow{b(u'),c?y}_L n'_\eta$, and by construction both $b_{n'} \to b(n')$ and $(m'_{\sigma[x\mapsto z]}, n'_{\eta[y\mapsto z]}) \in S_{\mathbf{R}}^{b_{n'}}$ are immediate.

$\square$

As an immediate corollary to these two propositions we get the main result of this section.

Theorem 4.5 *$t \simeq_L^b u$ if and only if $\rho \models t \sim_L u$ for every ρ such that $\rho \models b$.*

In particular it follows that for closed terms $\rho \models t \sim_L u$ if and only $t \simeq_L^b u$ for some $b = true$. This means that we have reduced the checking of late bisimulation equivalence to the checking of symbolic late bisimulations which are defined on the more abstract symbolic transition graphs. Of course these bisimulations are more difficult to check because of the requirement to find decompositions of the booleans each time a move is to be matched. However one can easily check that if the underlying symbolic transition system is finite branching then the sets of booleans B can always be taken to be finite. Moreover one could envisage an interactive system for checking or generating symbolic bisimulations where the user suggests the required decompositions.
In the next section we take a different approach by defining an algorithm which automatically generates these decompositions and for a given pair of terms t, u calculates the booleans b such $t \simeq_L^b u$.

5 The Algorithm

In this section we confine our attention to finite symbolic transition graphs, i.e. graphs with a finite number of nodes; they may of course contain infinite paths representing infinite computation sequences. However they are finite branching and, as remarked previously, for such graphs it is sufficient to restrict attention to *finite* sets of booleans B in the definition of $\simeq_L$. The graphs that the algorithm applies takes the form of two disjoint finite rooted symbolic transition graphs which satisfy an additional constraint that we called *standard.* The roots of these graphs, which we denote by r and r' respectively, represent finite state terms from a language such as *CCS.* A *direct path* in such a graph is a path from a root which contains at most one occurrence of each node, i.e. no loops are allowed, and a node m is a *direct ancestor* of n if m occurs on a direct path from the root to n. A graph is *standard* if whenever $m \overset{c?x}{\longmapsto} n$ then x is not in the set of free variables of any direct ancestor of m.

Here we describe an algorithm which given two terms t, u, calculates a boolean b such that $t \simeq_L^b u$. This is trivial in general since $t \simeq_L^{false} u$ for all terms t, u but we are interested in calculating the *most general boolean* b such that $t \simeq_L^b u$. A boolean is the most general boolean for a pair of terms t, u, written as $mgb_L(t,u)$, if $t \simeq_L^{mgb_L(t,u)} u$ and whenever $t \simeq_L^b u$ then $b \to mgb_L(t,u)$.

The algorithm for computing late symbolic bisimulation is shown in Figure 9, where $NAct(t,u)$, $Chan(t,u)$ are the sets of neutral actions and channel names, respectively, that appear in the next transitions from t, u. It calculates $mgb_L(t,u)$ and in addition exhibits a finite representation, in terms of a *table*, of a symbolic late bisimulation equivalence which witnesses the fact that $t \simeq_L^{mgb(t,u)} u$. The principle procedure $bisim(t,u)$ calls $close(t,u,true,\emptyset)$ and this returns two values, M a boolean which will turn out to be $mgb(t,u)$ and a *table* T used to construct the witnessing bisimulation. In general a table is a function $T\colon \langle \mathcal{T}, \mathcal{T} \rangle \longmapsto 2^{BExp}$ – it is convenient to use (finite) sets of boolean expressions rather than simply boolean expressions. We also need some notation for tables: $T \sqsubseteq T'$ iff $T(t,u) \subseteq T'(t,u)$ for all (t,u), $T \sqcup T'$ is defined by $(T \sqcup T')(t,u) = T(t,u) \cup T'(t,u)$ for all (t,u) and we write $b \epsilon T(t,u)$ to mean $b \to b'$ for some $b' \in T(t,u)$. The procedure *close* has four parameters, t and u, the current terms being compared, b a boolean expression which represents the constraints accumulated by previous calls to *close* and inherited by the current call, and finally W a set of pairs of nodes which have already been visited; each pair of nodes will be visited at most once by the algorithm and therefore is guaranteed to halt. A call to $close(t,u,b,W)$ uses the procedure *match* to compare each possible matching move from t and u. Each such comparison returns a boolean and a table and these are used to construct M and T, the values returned from the call to *close*. It is important to note that W is a set of pairs of nodes rather than terms but for convenience we will use notation such as $(t,u) \in W$ etc. to mean that $(m,n) \in W$ where t, u are the terms m_σ, n_η respectively. It should also be noted that in the procedure *match* with the parameter $c?$ boolean expressions of the form $\forall z.M_{ij}$ are used, where z is a fresh variable. Since the z does not occur elsewhere the eventual boolean expression returned by *bisim* can be considered to be of the form $\forall \underline{z}.M$ where M is quantifier free, and an evaluation ρ satisfies this expression if for all $v \in V$ $\rho[\underline{v}/\underline{z}] \models M[\underline{v}/\underline{z}]$.

The correctness of the algorithm is stated in the following two theorems:

Theorem 5.1 *(Soundness of Late Algorithm)*

```
$bisim(t,u) = close(t,u,true,\emptyset)$

$close(t,\ u,\ b,\ W) =$
if $(t,\ u) \in W$ then $(true,\ \emptyset)$
else let $(M_\gamma,\ T_\gamma) = match(\gamma,\ t,\ u,\ b,\ W)$
                              for $\gamma \in \{\, a, c!, c? \mid a \in NAct(t,u), c \in Chan(t,u)\,\}$
    in $(\bigwedge_\gamma M_\gamma,\ \bigsqcup_\gamma T_\gamma \sqcup \{(t,u) \mapsto \{b \wedge \bigwedge_\gamma M_\gamma\}\})$

$match(a,\ t,\ u,\ b,\ W) =$
let $(M_{ij},\ T_{ij}) = close(t_i,\ u_j,\ b \wedge b_i \wedge b'_j, \{(t,u)\} \cup W)$
                  for $t \xrightarrow{b_i,a}_L t_i, u \xrightarrow{b'_j,a}_L u_j$
in $(\bigwedge_i(b_i \rightarrow \bigvee_j(b'_j \wedge M_{ij})) \wedge (\bigwedge_j(b'_j \rightarrow \bigvee_i(b_i \wedge M_{ij})), \bigsqcup_{ij} T_{ij}))$

$match(c!,\ t,\ u,\ b,\ W) =$
let $(M_{ij},\ T_{ij}) = close(t_i,\ u_j,\ b \wedge b_i \wedge b'_j \wedge e_i = e'_j, \{(t,u)\} \cup W)$
                  for $t \xrightarrow{b_i,c!e_i}_L t_i, u \xrightarrow{b'_j,c!e'_j}_L u_j$
in $(\bigwedge_i(b_i \rightarrow \bigvee_j(b'_j \wedge e_i = e'_j \wedge M_{ij})) \wedge (\bigwedge_j(b'_j \rightarrow \bigvee_i(b_i \wedge e_i = e'_j \wedge M_{ij})), \bigsqcup_{ij} T_{ij}))$

$match(c?,\ t,\ u,\ b,\ W) =$
let $(M_{ij},\ T_{ij}) =$ let $z = newVar()$
                      $(M'_{ij},\ T'_{ij}) = close(t_i[x \mapsto z],\ u_j[y \mapsto z],\ b \wedge b_i \wedge b'_j, \{(t,u)\} \cup W)$
                                  for $t \xrightarrow{b_i,c?x}_L t_i, u \xrightarrow{b'_j,c?y}_L u_j$
                  in $(\forall z.M'_{ij},\ T'_{ij})$
in $(\bigwedge_i(b_i \rightarrow \bigvee_j(b'_j \wedge M_{ij})) \wedge (\bigwedge_j(b'_j \rightarrow \bigvee_i(b_i \wedge M_{ij})), \bigsqcup_{ij} T_{ij}))$
```

Figure 9: The Algorithm for Computing Late Symbolic Bisimulation

For standard graphs if $bisim(r,r') = (M,T)$ *then* $T(r,r') = \{M\}$ *and* $r \simeq_L^M r'$.

Theorem 5.2 *(Completeness of Late Algorithm)*
For standard graphs if $r \simeq_L^b r'$ *and* $bisim(r,r') = (M,T)$*, then* $b \rightarrow M$.

The proofs of these two theorems use some auxiliary notions and are quite lengthy, so we have put them in Appendix B.

We have implemented the algorithm and run it on some example problems. The boolean expressions returned by the algorithm are usually complex and hard to read. But with a small set of reduction rules on boolean expressions, they can be reduced to simple and readable forms. We give two examples here.

The example shown in Figure 7 can be written in *CCS* syntax as

```
P = c?x.((a.(IF x=0 THEN f.NIL ELSE g.NIL)) + a.h.NIL)

Q = c?x.(a.(IF x=0 THEN f.NIL ELSE h.NIL) +
         a.(IF x=0 THEN h.NIL ELSE g.NIL))
```

Running the algorithm on it produces:

```
The reduced LATE characteristic formula is

        true

with the bisimulation table:

L_1 R_1: true
L_2 R_2: true
L_3 R_3: v_1=0
L_3 R_4: not(v_1=0)
L_4 R_3: not(v_1=0)
L_4 R_4: v_1=0

where

L_1 = c?x.(a.IF x=0 THEN f.NIL ELSE g.NIL + a.h.NIL)
L_2 = a.IF x=0 THEN f.NIL ELSE g.NIL + a.h.NIL
L_3 = IF x=0 THEN f.NIL ELSE g.NIL
L_4 = h.NIL

R_1 = c?x.(a.IF x=0 THEN f.NIL ELSE h.NIL + a.IF x=0 THEN h.NIL ELSE g.NIL)
R_2 = a.IF x=0 THEN f.NIL ELSE h.NIL + a.IF x=0 THEN h.NIL ELSE g.NIL
R_3 = IF x=0 THEN f.NIL ELSE h.NIL
R_4 = IF x=0 THEN h.NIL ELSE g.NIL
```

Note that `v_1` is a fresh variable generated by the algorithm, and all terms, except `L_1` and `R_1`, have the substitution `x` $\mapsto$ `v_1` associated with them.

For the second example

```
P1 = c?x.(c!ABS(x).P1 + c!(-ABS(x)).P1)
Q1 = c?x.(c!x.Q1 + c!ABS(x).Q1 + c!(-ABS(x)).Q1)
```

where `ABS(x)` is the absolute value of `x`, the algorithm generates non-trivial bisimulation condition for `P1` and `Q1`:

```
The reduced LATE characteristic formula is

        forall v_1.ABS(v_1)=v_1 or -ABS(v_1)=v_1

with the bisimulation table:

L_1 R_1: forall v_1.ABS(v_1)=v_1 or -ABS(v_1)=v_1
L_2 R_2: ABS(v_1)=v_1 or -ABS(v_1)=v_1

where

L_1 = c?x.(c!ABS(x).P1 + c!-ABS(x).P1)
```

```
L_2 = c!ABS(x).P1 + c!-ABS(x).P1

R_1 = c?x.(c!x.Q1 + c!ABS(x).Q1 + c!-ABS(x).Q1)
R_2 = c!x.Q1 + c!ABS(x).Q1 + c!-ABS(x).Q1
```

A conventional transition graph can be viewed as a degenerated symbolic graph where all actions are neutral actions and all booleans guards are simply *true*. For such a graph the algorithm will return a boolean which is either *true* or *false*, and in case it is *true* the pairs of terms with entries *true* in the returned table constitute a bisimulation.

6 The Early Case

In this section we turn to early symbolic bisimulation. As before we will give an early concrete as well as an early symbolic semantics to symbolic transition graphs. Two definitions of bisimulation, one for early concrete and the other for early symbolic, will be presented, and a result relating them will be established. Finally an algorithm computing early symbolic bisimulation will be given. It turns out that we only need some minor (and systematic) modifications to the late case, and all results in the previous sections carry over to the new setting.

For the early concrete operational semantics we use the same form of judgements as used for late operational semantics, and we only need to change the rule concerning guarded input in Figure 6 to:

$$m \overset{b,c?x}{\longmapsto} n \quad \text{implies} \quad \rho \models m_\sigma \xrightarrow{c?y} n_{\sigma[x\mapsto y]}$$
$$\text{provided} \ \ \rho \models b\sigma, \ y \notin fv(m_\sigma)$$

This rule allows changing bound variables in input actions while infering transitions.

Now the definition of early concrete bisimulation can be given by slightly modifying Definition 3.1. As before we first define a functional over collections of relations parameterised on evaluations. Let $\mathbf{R} = \{ R^\rho \subseteq \langle \mathcal{T}, \mathcal{T} \rangle \mid \rho \in Eval\}$. Then $\mathcal{EB}(\mathbf{R})$ is the *Eval*-indexed family of symmetric relations defined by:
$(t, u) \in \mathcal{EB}(\mathbf{R})^\rho$ if

1. $\rho \models t \xrightarrow{c?z} t'$, where z is a fresh variable, implies that for each $v \in V$ $\rho \models u \xrightarrow{c?z} u'$ for some u' and $(t', u') \in R^{\rho[v/z]}$

2. for any other action a $\rho \models t \xrightarrow{a} t'$ implies $\rho \models u \xrightarrow{a} u'$ for some u' such that $(t', u') \in R^\rho$

Definition 6.1 (Early Bisimulations)
$\mathbf{R}$ is an early bisimulation if $R^\rho \subseteq \mathcal{EB}(\mathbf{R})^\rho$ for each ρ. □

We can now adapt the notation developed for the late case to this new setting. We write $\rho \models t \sim_E u$ if there is an early bisimulation $\mathbf{R}$ such that $(t, u) \in R^\rho$ and as before the standard theory applies; $\sim_E^\rho$, defined as $t \sim_E^\rho u$ if $\rho \models t \sim_E u$ is the ρ-th component of the maximal fixpoint of the functional $\mathcal{EB}$.
In Appendix A we show that when this definition is applied to the symbolic transition

graph for *CCS* we obtain the standard notion of (early) bisimulation equivalence as defined in [Mil89].

Similarly the early symbolic semantics may be obtained by changing the input rule in Figure 8 to

$$m \overset{b,c?x}{\longmapsto} n \quad \text{implies} \quad m_\sigma \xrightarrow{b\sigma,c?y}_E n_{\sigma[x\mapsto y]} \quad \text{provided } y \notin fv(m_\sigma)$$

(The $\longrightarrow_L$ arrows in the other rules are changed to $\longrightarrow_E$ as well.)
To define early symbolic bisimulation let $\mathbf{S} = \{ S^b \mid b \in BExp \}$ be a parameterised family of relations over terms. Then $\mathcal{SEB}(\mathbf{S})$ is the *BExp*-indexed family of symmetric relations defined by:

> $(t, u) \in \mathcal{SEB}(\mathbf{S})^b$ if $t \xrightarrow{b_1,\alpha}_E t'$ where $bv(\alpha)$ is a fresh variable, then there is a collection of booleans B such that $b \wedge b_1 \rightarrow \vee B$ and for each $b' \in B$ there exists a $u \xrightarrow{b_2,\alpha'}_E u'$ such that $b' \rightarrow b_2$ and
>
> 1. if $\alpha = c!e$ then $\alpha' = c!e'$, $b' \rightarrow e = e'$ and $(t', u') \in S^{b'}$
> 2. otherwise $\alpha = \alpha'$ and $(t', u') \in S^{b'}$

It is important to note that the set of booleans B may contain occurrences of the new variable $bv(\alpha)$.

Definition 6.2 (Early Symbolic Bisimulations)
$\mathbf{S}$ is an early symbolic bisimulation if $\mathbf{S} \subseteq \mathcal{SEB}(\mathbf{S})$ □

Again adapting the notation already developed we write $t \simeq^b_E u$ if there is a symbolic early bisimulation $\mathbf{S}$ such that $(t, u) \in S^b$ and as usual the standard theory implies that $\{ \simeq^b_E \mid b \in BExp \}$ is the maximal symbolic early bisimulation and that each $\simeq^b_E$ is an equivalence relation.

We now outline the relationship between these two semantic equivalences. First, as in the late case, early symbolic actions and early concrete actions can be related in a natural way.

Proposition 6.3

1. *$\rho \models t \xrightarrow{c?x} t'$ if and only if $t \xrightarrow{b,c?x}_E t'$ for some b such that $\rho \models b$*
2. *$\rho \models t \xrightarrow{c!v} t'$ if and only if $t \xrightarrow{b,c!e}_E t'$ for some b and e such that $\rho \models b$ and $\rho(e) = v$*
3. *$\rho \models t \xrightarrow{a} t'$ if and only if $t \xrightarrow{b,a}_E t'$ for some b and $\rho \models b$*

In analogy with Propositions 4.3 and 4.4, we have the following constructions:
Let $\mathbf{S}$ be an arbitrary early symbolic bisimulation. Define an *Eval*-indexed collection of relations over terms, $\mathbf{R_S}$, by letting

$$R^\rho_{\mathbf{S}} = \{ (t, u) \mid \exists b \,.\, \rho \models b \text{ and } (t, u) \in S^b \}$$

Conversely given an early bisimulation $\mathbf{R}$ we can construct a symbolic bisimulation $\mathbf{S_R}$ by letting

$$S^b_{\mathbf{R}} = \{ (t, u) \mid \rho \models b \text{ implies } (t, u) \in R^\rho \}.$$

Now we have

Proposition 6.4

1. *If* $\mathbf{S}$ *is an early symbolic bisimulation then* $\mathbf{R_S}$ *is an early bisimulation.*

2. *If* $\mathbf{R}$ *is an early bisimulation then* $\mathbf{S_R}$ *is an early symbolic bisimulation.*

Proof:

1. Let $(t,u) \in R^{\rho}_{\mathbf{S}}$, i.e. $(t,u) \in S^b$ for some b such that $\rho \models b$. We must show that all derivations from t and u are properly matched. The proofs for neutral and output actions are the same as for Proposition 4.3, and we only consider input actions here.

   Suppose $\rho \models t \xrightarrow{c?z} t'$, where z is a fresh variable. By Proposition 6.3 it follows that $t \xrightarrow{b_1,c?z}_E t'$ for some b_1 such that $\rho \models b_1$. Therefore there exists a set of booleans B such that $b \wedge b_1 \rightarrow \vee B$ and for each $b' \in B$ there is a $u \xrightarrow{b_2,c?z}_E u'$ such that $b' \rightarrow b_2$ and $(t',u') \in S^{b'}$. Let $v \in V$. Since z is a fresh variable $\rho[v/z] \models b \wedge b_1$. Thus there must be $b' \in B$ such that $\rho[v/z] \models b'$, and hence $\rho[v/z] \models b_2$ for the $u \xrightarrow{b_2,c?z}_E u'$ associated with this b'. As $z \notin fv(b_2)$, $\rho \models b_2$. Now applying the same Proposition we have $\rho \models u \xrightarrow{c?z} u'$. Since $(t',u') \in S^{b'}$ and $\rho[v/z] \models b'$ it follows that $(t',u') \in R^{\rho[v/x]}_{\mathbf{S}}$, as required.

2. Suppose $(t,u) \in S^b_{\mathbf{R}}$. We show that their symbolic actions can be matched in the appropriate manner. Again we only consider input action here as the cases of neutral and output actions are the same as Proposition 4.4.

   Let $t \xrightarrow{b_1,c?z}_E t'$, where z is a fresh variable. We must find a set of booleans B such that $b \wedge b_1 \rightarrow \vee B$ and for each $b' \in B$ there must exists a move $u \xrightarrow{b_2,c?z}_E u'$ such that $b' \rightarrow b_2$ and $(t',u') \in S^{b'}$. As in Proposition 4.4 we simplify the notation somewhat by assuming that for each u' there is at most one β such that $u \xrightarrow{\beta}_E u'$. Let U be the set $\{\, u' \mid u \xrightarrow{b(u'),c?z}_E u' \,\}$ and for each $u' \in U$ let $b'_{u'}$ be a boolean expression which satisfies

   $$\rho \models b'_{u'} \text{ if and only if } (t',u') \in R^{\rho}$$

   and let $b_{u'}$ be $b'_{u'} \wedge b(u')$. Finally let $B = \{\, b_{u'} \mid u' \in U \,\}$.
   We first check that $b \wedge b_1 \rightarrow \vee B$, i.e. $\rho \models b \wedge b_1$ implies $\rho \models b_{u'}$ for some $u' \in U$. Since $\rho \models b$ it follows that $(t,u) \in R^{\rho}$ and since $\rho \models b_1$ it follows from Proposition 6.3 that $\rho \models t \xrightarrow{c?z} t'$. So for every $v \in V$ there exists some u_v such that $\rho \models u \xrightarrow{c?z} u_v$ and $(t',u') \in R^{\rho[v/z]}$. So the required u' is $u_{\rho(z)}$ since in this case $(t',u') \in R^{\rho}$ and again by Proposition 6.3 $u \xrightarrow{b(u'),c?z} u'$ with $\rho \models b(u')$; therefore $\rho \models b_{u'}$.
   Also by the construction for any $b_{u'} \in B$ there exists $u \xrightarrow{b(u'),c?z}_E u'$ where $b \rightarrow b(u')$ and $(t',u') \in S^{b'}_{\mathbf{R}}$.

□

As a direct corollary we have

Theorem 6.5 $\rho \models t \sim_E u$ *if and only if* $t \simeq^b_E u$ *for every boolean* b *such that* $\rho \models b$.

The algorithm for computing late symbolic bisimulation presented in Figure 9 can also be modified to calculate early symbolic bisimulation. As may be expected, we only need to change the case dealing with input within function *match*:

```
$match(c?, t, u, b, W) =$
let $x = newVar()$
    $(M_{ij}, T_{ij}) = close(t_i, u_j, b \wedge b_i \wedge b'_j, \{(t,u)\} \cup W)$
                    for $t \xrightarrow{b_i, c?x}_E t_i, u \xrightarrow{b'_j, c?x}_E u_j$
in $(\forall x. \bigwedge_i (b_i \rightarrow \bigvee_j (b'_j \wedge M_{ij})) \wedge \bigwedge_j (b'_j \rightarrow \bigvee_i (b_i \wedge M_{ij})), \bigsqcup_{ij} T_{ij})$
```

This algorithm is sound and complete with respect to early symbolic bisimulation:

Theorem 6.6

1. *For standard graphs if* $bisim(r, r') = (M, T)$ *then* $T(r, r') = \{M\}$ *and* $r \simeq_E^M r'$.

2. *For standard graphs if* $r \simeq_E^M r'$ *and* $bisim(r, r') = (M, T)$ *then* $b \rightarrow M$.

The proof is similar to the proofs for the soundness and completeness of the late algorithm, and we leave it to the reader.

The early algorithm has also been implemented. Running it on the example presented in the Introduction

```
P1 = c?x.(IF EVEN(x) THEN R1 ELSE R2) + c?x.R3
P2 = c?x.(IF EVEN(x) THEN R1 ELSE R3) + c?x.IF EVEN(x) THEN R3 ELSE R2
R1 = NIL
R2 = tau.NIL
R3 = tau.tau.NIL
```

shows the difference between late and early bisimulations. The early algorithm returns:

```
The reduced EARLY characteristic formula is

        true

with the bisimulation table:

L_1 R_1: true
L_2 R_2: EVEN(v_1)
L_2 R_3: not(EVEN(v_1))
L_3 R_4: false
L_4 R_2: not(EVEN(v_1))
L_4 R_3: EVEN(v_1)
L_5 R_4: true
L_3 R_5: true
L_5 R_5: false

where
```

```
L_1 = c?x.IF EVEN(x) THEN R1 ELSE R2+c?x.R3
L_2 = IF EVEN(x) THEN R1 ELSE R2
L_3 = NIL
L_4 = tau.tau.NIL
L_5 = tau.NIL

R_1 = c?x. IF EVEN(x) THEN R1 ELSE R3+c?x.IF EVEN(x) THEN R3 ELSE R2
R_2 = IF EVEN(x) THEN R1 ELSE R3
R_3 = IF EVEN(x) THEN R3 ELSE R2
R_4 = tau.NIL
R_5 = NIL
```

But the boolean M returned by the late algorithm is

```
forall v_1,v_2,v_3,v_4.
(EVEN(v_1) or (not(EVEN(v_2)))) and ((not(EVEN(v_3))) or EVEN(v_4)) and
((EVEN(v_1) or (not(EVEN(v_3)))) and ((not(EVEN(v_2))) or EVEN(v_4)))
```

which is equivalent to false.

7 Conclusion

We have presented a new approach to bisimulation equivalence which works at the symbolic level rather than the more usual level of concrete operational semantics. At this level of abstraction many value-passing processes have a finite representation although semantically they are in some sense infinite. We have developed algorithms to compute symbolic bisimulations for a class of finite symbolic transition graphs called standard. The algorithms are independent of the language used to define expressions but to be useful, even for the restricted class of graphs to which they apply, we need to be able to simplify the returned expressions into some form of minimal form or at least a readable form. We have implemented the algorithms and a fairly naive set of simplification rules works reasonably well. They are adequate for simple examples but there is considerable room for improvement. For example the users help could be requested to simplify expressions as they are being generated rather than at present when the only simplification is carried out at the end. We believe that the algorithms may also be easily adapted to handle other semantic equivalences such as weak bisimulation and testing equivalence.

However one disadvantage of the present situation is that the class of processes to which the algorithms apply appears to be too restrictive. There are many simple processes which when expressed in a language such as *CCS* appear to be symbolically finite but generate symbolic graphs which are not standard. One example is an updatable memory such as

$$M(x) \Longleftarrow r!x.M(x) + w?y.M(y).$$

The symbolic graph associated with $M(x)$ is not standard and therefore it can not be used as input to our algorithm. However it is possible to adapt the technique developed in [JP92] in order to handle such processes. This will be discussed below.

The standard approach to value-passing in process algebras is to interpret the process $c?x.p$ as the nondeterministic sum $\sum_{v \in V} c?v.p[v/t]$. This is the approach suggested in [Mil89] and pursued, for example, in [Wal89, Bur91]. This results in a calculus with an infinite sum operator which may be satisfactory from a theoretical point of view but is outside the scope of existing verification tools. The only work of which we are aware which attempts to generalise bisimulation checking to value-passing languages is reported in [JP92]. There they consider a language similar to our extension of *CCS* which allows input and output guards of the form $c?x$, $c!e$ where e is either a data value or a single variable. However the processes are not allowed to test or modify the data received; the processes they consider are *data independent*. By using "schematic variables" they are able to show that (early) bisimulation equivalence between such processes can be reduced to bisimulation equivalence between finite state processes. The idea is to translate data-independent processes into processes which use schematic variables; these variables are treated differently from ordinary variables in that the operational semantics considers them to be data-values. The translation preserves bisimulation equivalence and data-independent processes are mapped into processes with finite transition graphs. A typical example of such a process is the updatable memory $M(x)$ above and this is translated into the new process

$$\begin{array}{lcl} M(x) & \Leftarrow & r!x.M(x) + w?v.M(v) \\ M(v) & \Leftarrow & r!v.M(v) + w?v.M(v) \end{array}$$

This process is now finite state because in the action $w?v$ the symbol v is not treated as a variable but as a constant.

In a straightforward manner we can extend our symbolic semantics to include these schematic variables v and the associated special actions of the form $w?v$. Thus by using their translation we can also apply our algorithms to arbitrary data-independent processes. However it remains to be seen if our algorithm can be modified so as to apply to arbitrary finite symbolic graphs. It would also be interesting to see if this symbolic approach can be applied to "weak" bisimulation equivalence.

For a different approach to checking properties of infinite state processes the reader is referred to [Wol86, BS90] where techniques are developed for checking that such processes have properties expressed in temporal and modal logics.

Appendices

A Concrete Bisimulations and *CCS*-Bisimulations

In this appendix we argue that, for the *CCS*-like language given in Section 2, the concrete bisimulations defined in Section 3 and Section 6 using symbolic transition graphs coincide with appropriate versions of these equivalences defined directly on the syntax of the language.

We first consider the late case.

A *late* version of the operational semantics for this example language is given in Figure 10. The relations $\xrightarrow{a}$ are defined over closed terms and the obvious symmetric rules for $+$ and $|$ have been omitted.
A *CCS*-late bisimulation is a symmetric relation between closed terms that satisfies: $(p, q) \in R$ implies

1. $p \xrightarrow{c?x} \lambda x.p' \Longrightarrow$ there exists $q \xrightarrow{c?y} \lambda y.q'$ such that for all $v \in V, (p'[v/x], q'[v/y]) \in R$
2. for any other actions $a, p \xrightarrow{a} p' \Longrightarrow$ there exists $q \xrightarrow{a} q'$ such that $(p', q') \in R$

Let $\sim$ be the maximal *CCS*-late bisimulation. We will show that it coincides with the late bisimulation obtained by viewing *CCS* as a symbolic transition system generated by rules in Figure 4. For convenience we will denote the term $t_\emptyset$ of the *CCS* symbolic transition system simply by t.

The situation is a little complicated by the fact that the symbolic transition system is defined between nodes which are pairs of the form (t, U) with t a *CCS* term and U a set of variables. But we can identify a term t with the pair $(t, fv(t))$. We then have the theorem

Theorem A.1 *$p \sim q$ iff $\rho \models p \sim_L q$ (i.e. $\rho \models p_\emptyset \sim_L q_\emptyset$) for every evaluation ρ.*

This follows from two more general results.

Proposition A.2 *Let $S^\rho = \{ (t_\sigma, u_\eta) \mid t\rho\sigma \sim u\rho\eta \}$. Then $\{S^\rho\}$ is a late bisimulation.*

As an immediate corollary we have

Corollary A.3 *$p \sim q$ implies $\rho \models p \sim_L q$ for all ρ*

Proof: The pair $p_\emptyset, q_\emptyset$ are in any S^ρ because $fv(p) = \emptyset$ and $p\rho\sigma = p$ for closed terms p. □

Proposition A.4 *Let R be the set of all pairs $(t\rho\sigma, u\rho\eta)$ such that $\rho \models t_\sigma \sim_L u_\eta$. Then R is a* CCS*-late bisimulation.*

Corollary A.5 *$\rho \models p \sim_L q$ for all ρ implies $p \sim q$*

$a.p \xrightarrow{a} p$		$a \in NAct \cup \{\, c!v \mid c \in Chan \,\}$
$c?x.p \xrightarrow{c?x} \lambda x.p$		$c \in Chan$
$p \xrightarrow{\alpha} p'$	implies	$p + q \xrightarrow{\alpha} p'$
$p \xrightarrow{\alpha} p', \alpha \in \{\tau\} \cup \{c!v\}$	implies	$p \mid q \xrightarrow{\alpha} p' \mid q$
$p \xrightarrow{c?x} \lambda x.p'$	implies	$p \mid q \xrightarrow{c?x} \lambda x.(p' \mid q)$
$p \xrightarrow{c?x} \lambda x.p', q \xrightarrow{c!v} q'$	implies	$p \mid q \xrightarrow{\tau} p'[v/x] \mid q'$
$p \xrightarrow{\alpha} p', \alpha \in \{\tau\} \cup \{c!v\}$	implies	$p\backslash c \xrightarrow{\alpha} p'\backslash c$
		if α does not use the channel c
$p \xrightarrow{c?x} \lambda x.p'$	implies	$p\backslash d \xrightarrow{\alpha} \lambda x.(p'\backslash d)$
		if $c \neq d$
$t[\underline{v}/\underline{x}] \xrightarrow{\alpha} p'$	implies	$P(\underline{v}) \xrightarrow{\alpha} p'$
		if $P(\underline{x}) \Longleftarrow t$ is a declaration
$p \xrightarrow{\alpha} p', b \equiv true$	implies	$(b \rightarrow p, q) \xrightarrow{\alpha} p'$
$p \xrightarrow{\alpha} p', b \equiv false$	implies	$(b \rightarrow q, p) \xrightarrow{\alpha} p'$

Figure 10: Late Operational Semantics of *CCS* - closed terms

Proof: Since $\rho \models p_\emptyset \sim_L q_\emptyset$, it follows that $(p, q) \in R$. □

The proofs of the two propositions depend on relating the arrows which underly the two different definitions of equivalence. The relationship is captured in three lemmas, namely A.6, A.7 and A.8.

Lemma A.6

1. *If $\rho \models b$ then $t \stackrel{b,c?x}{\longmapsto} t'$ implies $t\rho \xrightarrow{c?y} \lambda y.r$ for some r such that for all v $r[v/y] \equiv t'[v/x]\rho$.*

2. *If $t\rho \xrightarrow{c?y} \lambda y.r$ then there exist b and t' such that $\rho \models b$ and $t \stackrel{b,c?x}{\longmapsto} t'$ for some x with $r[v/y] \equiv t'[v/x]\rho$ for all v.*

Proof:

1. By induction on why $t \stackrel{b,c?x}{\longmapsto} t'$.

   - $c?y.t \stackrel{true,c?x}{\longmapsto} t[x/y]$ where $x \notin fv(c?y.t)$. Then $(c?y.t)\rho \xrightarrow{c?y} (\lambda y.t)\rho$ and the requirement is satisfied.
   - $t \mid u \stackrel{b,c?x}{\longmapsto} t'[x/z] \mid u$ because $t \stackrel{b,c?z}{\longmapsto} t'$. By induction $t\rho \xrightarrow{c?y} \lambda y.r$ for some r such that for all v $r[v/y] \equiv t'[v/z]\rho$. Now $(t \mid u)\rho \xrightarrow{c?y} \lambda y.(r \mid u\rho)$ and $(r \mid u\rho)[v/y] = r[v/y] \mid u\rho \equiv t'[v/z]\rho \mid u\rho$. By the definition of the operational semantics x is either z or else it is not in $fv(t')$. In either case we have $t'[v/z] \equiv t'[x/z][v/x]$. Therefore $r[v/y] \mid u\rho \equiv t'[x/z][v/x]\rho \mid u\rho$
   $\equiv (t'[x/z] \mid u)[v/x]\rho$ because $x \notin fv(u)$.
   - $(b' \rightarrow t, u) \stackrel{b \wedge b',c?x}{\longmapsto} t'$ because $t \stackrel{b,c?x}{\longmapsto} t'$. By induction $t\rho \xrightarrow{c?y} \lambda y.r$ such that $r[v/y] \equiv t'[v/x]\rho$ for all v. Since $\rho \models b'$ we also have $(b' \rightarrow t, u)\rho \xrightarrow{c?y} \lambda y.r$.

- $P(\underline{e}) \overset{b,c?x}{\longmapsto} t'$ because $t[\underline{e}/\underline{x}] \overset{b,c?x}{\longmapsto} t'$ where $P(\underline{x}) \Longleftarrow t$. By induction $t[\underline{e}/\underline{x}]\rho \xrightarrow{c?y} \lambda y.r$ s.t. $r[v/y] \equiv t'[v/x]\rho$ for all v. Furthermore $t[\underline{e}/\underline{x}]\rho = t[\underline{e\rho}/\underline{x}]$ because $fv(t) \subseteq \{\underline{x}\}$. Therefore $P(\underline{e\rho}) \xrightarrow{c?y} \lambda y.r$, i.e. $P(\underline{e})\rho \xrightarrow{c?y} \lambda y.r$.
- Remaining cases are similar.

2. By induction on why $t\rho \xrightarrow{c?y} \lambda y.r$.

- $(c?y.t)\rho \xrightarrow{c?y} (\lambda y.t)\rho$. In this case $r[v/y]$ is $t[v/y]$ and obviously $c?y.t \overset{true,c?x}{\longmapsto} t[x/y]$ where $x = new(fv(c?y.t))$. Since $x \in fv(t) - \{y\}$ it follows that $t[v/y]\rho \equiv t[x/y][v/x]\rho$ for all v.
- $(t \mid u)\rho \xrightarrow{c?y} \lambda y.(r \mid u\rho)$ because $t\rho \xrightarrow{c?y} \lambda y.r$. By induction there exist b and t', $\rho \models b$ and $t \overset{b,c?z}{\longmapsto} t'$ s.t. $r[v/y] \equiv t'[v/z]\rho$ for all v. So $t \mid u \overset{b,c?x}{\longmapsto} t'[x/z] \mid u$ and $(r \mid u\rho)[v/y] \equiv r[v/y] \mid u\rho \equiv t'[v/z]\rho \mid u\rho \equiv t'[x/z][v/x]\rho \mid u\rho \equiv (t'[x/z] \mid u)[v/x]\rho$ because $x \notin fv(u) \cup (fv(t') - \{z\})$.
- $(b' \to t, u)\rho \xrightarrow{c?y} \lambda y.r$ because $\rho \models b'$ and $t\rho \xrightarrow{c?y} \lambda y.r$. By induction there exist b and t', $\rho \models b$ and $t \overset{b,c?x}{\longmapsto} t'$ s.t. $r[v/y] \equiv t'[v/x]\rho$ for all v. So $(b' \to t, u) \overset{b\wedge b',c?x}{\longmapsto} t'$ and $\rho \models b \wedge b'$.
- $P(\underline{e})\rho \xrightarrow{c?y} \lambda y.r$ because $t[\underline{e\rho}/\underline{x}] \xrightarrow{c?y} \lambda y.r$ where $P(x) \Longleftarrow t$ and $fv(t) \subseteq \{\underline{x}\}$. In this case $t[\underline{e}/\underline{x}]\rho = t[\underline{e\rho}/\underline{x}]$. By induction there exist b and t', $\rho \models b$ and $t[\underline{e}/\underline{x}] \overset{b,c?x}{\longmapsto} t'$ s.t. $r[v/y] \equiv t'[v/x]\rho$ for all v. Therefore $P(\underline{e}) \overset{b.c?x}{\longmapsto} t'$.
- remaining cases are similar.

□

Lemma A.7

1. *If $\rho(b) = true$ and $\rho(e) = v$ then $t \overset{b,c!e}{\longmapsto} t'$ implies there exists r, $t\rho \xrightarrow{c!v} r \equiv t'\rho$.*

2. *If $t\rho \xrightarrow{c!v} r$ then there exist b, t' and e s.t. $\rho(b) = true, \rho(e) = v,$ and $t \overset{b,c!e}{\longmapsto} t'$ with $r \equiv t'\rho$.*

Proof: By induction on the derivation of $t \overset{b,c!e}{\longmapsto} t'$ and $t\rho \xrightarrow{c!v} r$, respectively. □

Lemma A.8 1. *If $\rho(b) = true$ then $t \overset{b,\tau}{\longmapsto} t'$ implies there exists r, $t\rho \xrightarrow{\tau} r \equiv t'\rho$.*

2. *If $t\rho \xrightarrow{\tau} r$ then there exist b, t' s.t. $\rho(b) = true$ and $t \overset{b,\tau}{\longmapsto} t'$ with $r \equiv t'\rho$.*

Proof: By induction on the derivation of $t \overset{b,\tau}{\longmapsto} t'$ and $t\rho \xrightarrow{\tau} r$, respectively. □

Proof of Proposition A.2. We have to prove that $\{S^\rho\}$ is a late bisimulation.
Suppose $(t_\sigma, u_\eta) \in S^\rho$

1. Let $\rho \models t_\sigma \xrightarrow{c?x} t'_\sigma$. By the definition of the late operational semantics (Figure 6)

$$t \xmapsto{b,c?x} t' \text{ for some } b \text{ with } \rho \models b\sigma.$$

Now from Lemma A.6

$$t\rho\sigma \xrightarrow{c?y} \lambda y.r \text{ for some } r \text{ s.t. for all } v,\ r[v/y] \equiv t'[v/x]\rho\sigma$$

Since $t\rho\sigma \sim u\rho\eta$, we have

$$u\rho\eta \xrightarrow{c?z} \lambda z.s \text{ for some } s \text{ s.t. for all } v,\ r[v/y] \sim s[v/z]$$

Again applying Lemma A.6 we get

$$u \xmapsto{b',c?w} u'$$

for some w s.t $\rho\eta \models b'$, i.e. $\rho \models b'\eta$, and for all v, $s[v/z] \equiv u'[v/w]\rho\eta$. This means $u_\eta \xrightarrow{c?w} u'_\eta$. We need to show that for all v,

$$(t'_{\sigma[x\mapsto z]}, u'_{\eta[w\mapsto z]}) \in S^{\rho[v/z]}$$

where z is a fresh variable. The only non-trivial condition is that

$$t'\rho[v/z]\sigma[x \mapsto z] \sim u'\rho[v/w]\eta[w \mapsto z]$$

This follows because $t'\rho[v/z]\sigma[x \mapsto z] \equiv t'[v/x]\rho\sigma$ and $u'\rho[v/z]\eta[w \mapsto z] \equiv u'[v/w]\rho\eta$.

2. Let $\rho \models t_\sigma \xrightarrow{c!e} t'_\sigma$. This must be because $t \xmapsto{b,c!e} t'$ for some b, e s.t. $\rho\sigma(b) = true$ and $\rho\sigma(e) = v$. By Lemma A.7,

$$t\rho\sigma \xrightarrow{c!v} r \equiv t'\rho\sigma$$

Since $t\rho\sigma \sim u\rho\eta$, we have

$$u\rho\eta \xrightarrow{c!v} r' \sim r$$

Again by Lemma A.7

$$u \xmapsto{b',c!e'} u'$$

s.t. $\rho\eta \models b', \rho\eta(e') = v$, and $r' \equiv u'\rho\eta$. So $\rho \models u_\eta \xrightarrow{c!v} u'$ and $u'\rho\eta \equiv r' \sim r \equiv t'\rho\sigma$, i.e. $(t', u') \in S^\rho$.

3. The final case, $\rho \models t_\sigma \xrightarrow{\tau} t'_\sigma$, is similar. □.

The proof of Proposition A.4 is similar: to show R is a *CCS*-late bisimulation suppose $(t\rho\sigma, u\rho\eta) \in R$. Again there are three different kinds of moves to consider. As an example suppose $t\rho\sigma \xrightarrow{c?x} \lambda x.r$. We must show that $u\rho\eta \xrightarrow{c?y} \lambda y.r'$ s.t. for all v, $(r[v/x], r'[v/y]) \in R$. By Lemma A.6,

$$t \xmapsto{b,c?x'} t'$$

$$\begin{array}{lll}
a.p \xrightarrow{a} p & & a \in \{\tau\} \cup \{\, c!v \mid c \in Chan \,\} \\
c?x.p \xrightarrow{c?v} p[v/x] & & c \in Chan, v \in V \\
p \xrightarrow{\alpha} p' & \text{implies} & p + q \xrightarrow{\alpha} p' \\
p \xrightarrow{\alpha} p' & \text{implies} & p \mid q \xrightarrow{\alpha} p' \mid q \\
p \xrightarrow{c?v} p', q \xrightarrow{c!v} q' & \text{implies} & p \mid q \xrightarrow{\tau} p'[v/x] \mid q \\
p \xrightarrow{\alpha} p' & \text{implies} & p\backslash c \xrightarrow{\alpha} p'\backslash c \\
 & & \text{if } \alpha \text{ does not use the channel } c \\
t[\underline{v}/\underline{x}] \xrightarrow{\alpha} p' & \text{implies} & P(\underline{v}) \xrightarrow{\alpha} p' \\
 & & \text{if } P(\underline{x}) \Longleftarrow t \text{ is a declaration} \\
p \xrightarrow{\alpha} p', b \equiv true & \text{implies} & (b \rightarrow p, q) \xrightarrow{\alpha} p' \\
p \xrightarrow{\alpha} p', b \equiv false & \text{implies} & (b \rightarrow q, p) \xrightarrow{\alpha} p'
\end{array}$$

Figure 11: Early Operational Semantics of *CCS* - closed terms

s.t. $\rho \models b\sigma$ and for all v, $r[v/x] \equiv t'[v/x']\rho\sigma$. This in turn means

$$\rho \models t_\sigma \xrightarrow{c?x'} t'_\sigma.$$

Since $\rho \models t_\sigma \sim_L u_\eta$ we must have

$$u_\eta \xrightarrow{c?w} u'_\eta$$

s.t. for all v, $\rho[v/z] \models t'_{\sigma[x' \mapsto z]} \sim_L u'_{\eta[w \mapsto z]}$ with z fresh. First note that $u \overset{b',c?w}{\longmapsto} u'$ for some b' with $\rho \models b'\eta$. So by Lemma A.6

$$u\rho\eta \xrightarrow{c?y} \lambda y.r'$$

s.t. for all v $r'[v/y] \equiv u'[v/w]\rho\eta$. Now

$$r[v/x] \equiv t'[v/x']\rho\sigma \equiv t'\rho[v/z]\sigma[x' \mapsto z] \;\text{ and }\; r'[v/y] \equiv u'[v/w]\rho\eta \equiv u'\rho[v/z]\eta[w \mapsto z]$$

Hence $(r[v/x], r'[v/y]) \in R$ (up to $\equiv$).

We now turn our attention to the early case. This is very similar to the late case and so we only outline the corresponding results.

The standard operational semantics for the example language is given in Figure 11 where again the symmetric rules for $+$ and $\mid$ have been omitted. This corresponds to an *early* interpretation of the language. So we call any symmetric relation R between closed terms a *CCS*-early bisimulation if it satisfies: $(p, q) \in R$ implies

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

where a ranges over $\{\tau,\ c?v,\ c!v\}$ Now let us use $\sim$ to denote the maximal *CCS*-early bisimulation.

Again viewing *CCS* as a symbolic transition system generated by rules in Figure 4, we show that early concrete bisimulation (as defined in Section 6) coincides with $\sim$:

$p \sim q$ iff $\rho \models p \sim_E q$ (i.e. $\rho \models p_\emptyset \sim_E q_\emptyset$) for every evaluation ρ.

The proof follows the same pattern as before. Define

$$S^\rho = \{ (t_\sigma, u_\eta) \mid t\rho\sigma \sim u\rho\eta \}$$

and

$$R = \{ (t\rho\sigma, u\rho\eta) \mid \rho \models t_\sigma \sim_E u_\eta \}.$$

Then one can show that $\{S^\rho\}$ is an early bisimulation and R is a *CCS*-early bisimulation, from which the result follows immediately.
The proof of these two results depends on relating the two different arrow relations:

1. $t\rho \xrightarrow{c?v} r$ if and only if there exist b and t' such that $\rho \models b$ and $t \overset{b,c?x}{\longmapsto} t'$ for some x with $r \equiv t'\rho[v/x]$.

2. $t\rho \xrightarrow{c!v} r$ if and only if there exist b, t' and e s.t. $\rho(b) = true, \rho(e) = v$ and $t \overset{b,c!e}{\longmapsto} t'$ with $r \equiv t'\rho$.

3. $t\rho \xrightarrow{\tau} r$ if and only if there exist b, t' s.t. $\rho(b) = true$ and $t \overset{b,\tau}{\longmapsto} t'$ with $r \equiv t'\rho$.

B Proving Correctness of The Late Algorithm

This appendix is devoted to the proof of correctness of the late bisimulation algorithm. We show that if r, r' are the roots of two finite standard graphs then $bisim(r, r')$ always halts and if it returns the pair (M, T) then $M = mgb(r, r')$ and T can be used to construct a symbolic bisimulation $\mathbf{S}$ such that $(r, r') \in S^M$. So for the remainder of this section let us fix such a pair of graphs, G, H.

Let $\mathcal{T}_G$ and $\mathcal{T}_H$ be the sets of terms associated with G and H, respectively. We use $t, t', t'', \ldots$ to range over $\mathcal{T}_G$ and $u, u', u'', \ldots$ to range over $\mathcal{T}_H$. The main problem is to come up with a verification condition for the procedure *close* which will of course involve verification conditions for the auxiliary procedure *match*. This involves, among other things, characterising the domain of the table T which is returned by a call to $close(t, u, b, W)$. This will have an entry for each pair t', u' such that there are "matching" derivations from t, u to t', u' respectively, provided these derivations do not involve the pairs of nodes in W that have already been visited. The first series of definitions will formalise this idea.

We say a pair of guarded symbolic transitions β, β' are of the same type $\gamma \in \{ a, c!, c? \mid a \in NAct, c \in Chan \}$, and associate a boolean $C(\beta, \beta')$ with them, if

1. $\beta = (b, a)$ and $\beta' = (b', a)$ with $a \in NAct$, and $C(\beta, \beta') = b \wedge b'$

2. $\beta = (b, c?x)$ and $\beta' = (b', c?y)$ with $c \in Chan$, and $C(\beta, \beta') = b \wedge b'$

3. $\beta = (b, c!e)$ and $\beta' = (b', c!e')$ with $c \in Chan$, and $C(\beta, \beta') = b \wedge b' \wedge e = e'$

A matching derivation from (m_σ, n_η) to $(m'_{\sigma'}, n'_{\eta'})$ is a pair of derivations

$$m_\sigma \xrightarrow{\beta}_L m'_{\sigma'}, n_\eta \xrightarrow{\beta'}_L n'_{\eta'}$$

such that β and β' are of the same type, and

$$(\sigma', \eta') = \begin{cases} (\sigma[x \mapsto z], \eta[y \mapsto z]) & \text{if } \beta = (b, c?x) \text{ and } \beta' = (b', c?y) \\ (\sigma, \eta) & \text{otherwise} \end{cases}$$

where z is a fresh variable. We will sometimes write this as $d : (m_\sigma, n_\eta) \xrightarrow{\beta,\beta'}_L (m'_{\sigma'}, n'_{\eta'})$. This is generalised to sequences in the obvious way: a matching path, written $p : (t, u) \longrightarrow^* (t', u')$, is defined inductively by

1. $\varepsilon : (t, u) \longrightarrow^* (t, u)$ is a matching path from (t, u) to (t, u) with $\varepsilon^C = \mathit{true}$ and $\varepsilon^S = \emptyset$

2. If $p : (t, u) \longrightarrow^* (t'', u'')$ is a matching path and $d : (t'', u'') \xrightarrow{\beta,\beta'}_L (t', u')$ is a matching derivation then $p^\frown d : (t, u) \longrightarrow^* (t', u')$ is a matching path with $(p^\frown d)^C = p^C \wedge C(\beta, \beta')$ and $(p^\frown d)^S = p^S \cup \{(t'', u'')\}$

For a matching path $p : (t_0, u_0) \longrightarrow \cdots \longrightarrow (t_k, u_k)$ and a table T, let $p^T = \{ \bigwedge_{0 \le i \le k} b_i \mid b_i \in T(t_i, u_i) \}$.

We write $p : (t, u) \longrightarrow^*_W (t', u')$ if $p : (t, u) \longrightarrow^* (t', u')$ and the pairs of nodes used in this derivation, other than the last pair of terms (t', u'), are not contained in $\mathit{domain}(W)$. We also write $(t, u) \longrightarrow^*_W (t', u')$ to mean there exists such a matching derivation.

We use $\mathcal{B}_\gamma(t, u, b, W, T)$ to mean the following condition is satisfied:
Whenever $t \xrightarrow{b_1,\alpha}_L t'$ is a derivation of type γ there is a set of booleans B such that $b \wedge b_1 \rightarrow \vee B$ and for each $b' \in B$ there exists a $u \xrightarrow{b_2,\alpha'}_L u'$ such that $b' \rightarrow b_2$ and

- if $\gamma = a \in \mathit{NAct}$ then $\alpha = \alpha' = a$ and $(t', u') \notin W \Rightarrow b' \,\epsilon\, T(t', u')$
- if $\gamma = c!$ then α has the form $c!e$ and α' has the form $c!e'$, $b' \rightarrow e = e'$ and $(t', u') \notin W \Rightarrow b' \,\epsilon\, T(t', u')$
- if $\gamma = c?$ then α has the form $c?x$ and α' has the form $c?y$, and $(t', u') \notin W \Rightarrow b' \,\epsilon\, T(t'[x \mapsto z], u'[y \mapsto z])$ where z is a fresh variable

and similarly for u.

Finally,

$$\mathcal{B}(t, u, b, W, T) =_{def} \bigwedge_{\gamma \in \{\, a, c!, c? \mid a \in \mathit{NAct}(t,u), c \in \mathit{Chan}(t,u) \,\}} \mathcal{B}_\gamma(t, u, b, W, T)$$

It is easy to see that $\mathcal{B}_\gamma(t, u, b, W, T)$, and hence $\mathcal{B}(t, u, b, W, T)$, is anti-monotonic in its third argument and monotonic in its fifth, that is if $b' \rightarrow b$ and $T \sqsubseteq T'$ then $\mathcal{B}_\gamma(t, u, b, W, T)$ implies $\mathcal{B}_\gamma(t, u, b', W, T')$.

We are now ready to define the major components of the verification conditions of the procedures *close* and *match*.

Definition B.1 Let $H(t, u, b, W, T)$ be true if the following conditions are satisfied:

$(H1)$ $(r, r') \longrightarrow^*_W (t, u)$

$(H2)$ $W \cap (\mathit{domain}(T) - \{(t, u)\} = \emptyset$

$(H3)$ $p : (t, u) \longrightarrow^*_W (t', u')$ and $(t', u') \notin W$ implies $p^S \cup \{(t', u')\} \subseteq \mathit{domain}(T)$ and for every $d \in p^T$ $\mathcal{B}(t', u', b \wedge d, W, T)$

$\mathit{CLOSE}(t, u, b, W, M, T) =_{def} H(t, u, b, W, T)$ and $(t, u) \notin W \Longrightarrow T(t, u) = \{b \wedge M\}$. $\Box$

Definition B.2 For each γ let $H_\gamma(t, u, b, W, M, T)$ be true if

$(H_\gamma 1)$ $(r, r') \longrightarrow^*_W (t, u)$

$(H_\gamma 2)$ $(\{(t, u)\} \cup W) \cap \mathit{domain}(T) = \emptyset$

$(H_\gamma 3)$ if $(t, u) \xrightarrow{\beta, \beta'} (t'', u'')$ is a matching derivation of type γ, $p : (t'', u'') \longrightarrow^*_W (t', u')$ and $(t', u') \notin W \cup \{(t, u)\}$ then $p^S \cup \{(t', u')\} \subseteq \mathit{domain}(T)$ and for every $d \in p^T$ $\mathcal{B}(t', u', b \wedge M \wedge d, W, T)$

$(H_\gamma 4)$ $\mathcal{B}_\gamma(t, u, b \wedge M, \{(t, u)\} \cup W, T)$

$\mathit{MATCH}_\gamma(t, u, b, W, M, T) =_{def} H_\gamma(t, u, b, W, M, T)$. $\Box$

There now follow two propositions which show that these verification conditions imply each other when instantiated to the parameters which correspond to the way in which the two procedures *close* and *match* call each other. The first one shows that the verification condition of *match*, namely H_γ, implies that of *close*.

Proposition B.3 *If $H_\gamma(t, u, b, W, M_\gamma, T_\gamma)$ for each type γ then $H(t, u, b, W, T)$, where $T = \bigsqcup_\gamma T_\gamma \sqcup \{(t, u) \mapsto \{b \wedge \bigwedge_\gamma M_\gamma\}\}$.*

Proof: The only non-trivial condition is $H3$. Let $p : (t, u) \longrightarrow^*_W (t', u')$ and $(t', u') \notin W$. It is obvious that (t', u') is in the domain of T. So suppose $d \in p^T$, we must prove $\mathcal{B}(t', u', b \wedge d, W, T)$. There are two cases to consider.

1. $(t', u') = (t, u)$. Then since $\mathcal{B}$ is anti-monotonic in its third argument we may assume d has the form $b \wedge \bigwedge_\gamma M_\gamma$. So we must show $\mathcal{B}(t, u, d, W, T)$. As an example we show $\mathcal{B}_{c?}(t, u, d, W, T)$. Let $t \xrightarrow{b_1, c?x}_L t''$. We know $H_{c?}(t, u, b, W, M_{c?}, T_{c?})$ and so there is a set of booleans B' such that $b \wedge b_1 \rightarrow \vee B'$ with the properties guaranteed by $H_\gamma 4$, i.e. $\mathcal{B}_{c?}(t, u, b \wedge M_{c?}, \{(t, u)\} \cup W, T_{c?})$. Let $B = \{\, b' \wedge d \mid b' \in B' \,\}$. Then $d \wedge b_1 \rightarrow \vee B$. Now consider an arbitrary element of B, $b' \wedge d$. Since $b' \in B'$ there must exist a move $u \xrightarrow{b_2, c?y}_L u''$ such that $b' \rightarrow b_2$ and therefore $b' \wedge d \rightarrow b_2$ and if $(t'', u'') \notin W \cup \{(t, u)\}$ then $b' \mathrel{\epsilon} T_{c?}(t''[x \mapsto z], u''[y \mapsto z])$.
   We must show $(t'', u'') \notin W$ implies $b' \wedge d \mathrel{\epsilon} T(t''[x \mapsto z], u''[y \mapsto z])$. This follows immediately unless $(t'', u'') \in \{(t, u)\}$ or more accurately $n(t) = n(t'')$ and $n(u) = n(u'')$. In this case, since the graphs are assumed to be standard $x \notin \mathit{fv}(n(t))$ and $y \notin \mathit{fv}(n(u))$ and so $t = t''$, $u = u''$. It follows that $T(t'', u'') = \{d\}$ and therefore $b' \wedge d \mathrel{\epsilon} T(t'', u'')$.

2. $(t', u') \neq (t, u)$. Then for some type γ (t', u') is in the domain of T_γ and $H_\gamma(t, u, b, W, M_\gamma, T_\gamma)$. Also d must have the form $b \wedge \bigwedge_\gamma M_\gamma \wedge d'$ for some d' such that $H_\gamma 3$, i.e. $\mathcal{B}(t', u', b \wedge M_\gamma \wedge d', W, T_\gamma)$, holds. Since $b \wedge M_\gamma \wedge d' = b \wedge d$ and $T_\gamma \sqsubseteq T$, we obtain $\mathcal{B}(t', u', b \wedge d, W, T)$ by the monotonicity of $\mathcal{B}$ in its fifth argument.

□

We now show that appropriate instances of the verification condition of *close* imply those of *match*.

Proposition B.4 *Suppose $(t,u) \notin W$. If for all type γ derivations $(t \xrightarrow{b_i,\alpha}_L t_i, u \xrightarrow{b'_j,\alpha'}_L u_j)$, $H(t_i,u_j,b\wedge b_i \wedge b'_j, \{(t,u)\}\cup W, T_{ij})$ and*

$$(t_i,u_j) \notin \{(t,u)\} \cup W \Longrightarrow b \wedge b_i \wedge b'_j \wedge \forall z.M_{ij} \in T_{ij}(t'_i,u'_j)$$

where

$$(t'_i,u'_j) = \begin{cases} (t_i[x \mapsto z], u_j[y \mapsto z]) & \textit{if } \alpha = c?x, \alpha' = c?y \\ (t_i,u_j) & \textit{otherwise} \end{cases}$$

with z a fresh variable, then $H_\gamma(t,u,b,W,M,T)$ where $M = \bigwedge_i (b_i \to \bigvee_j (b'_j \wedge \forall z.M_{ij})) \wedge \bigwedge_j (b'_j \to \bigvee_i (b_i \wedge \forall z.M_{ij}))$ and $T = \bigsqcup_{ij} T_{ij}$

Proof: Again the first two conditions of H_γ are straightforward.
To show $H_\gamma 3$, let $d : (t \xrightarrow{b_i,\alpha}_L t_i, u \xrightarrow{b'_j,\alpha'}_L u_j)$ is a type γ matching derivation and $p : (t_i,u_j) \longrightarrow^*_W (t',u')$ where $(t',u') \notin W$. It follows from $H(t_i,u_j,b\wedge b_i \wedge b_j, \{(t,u)\}\cup W, T_{ij})$ that $(t',u') \in \mathit{domain}(T_{ij}) \subseteq \mathit{domain}(T)$. Suppose $d \in p^T$. We must show

$$\mathcal{B}(t',u',b\wedge b_i \wedge b_j \wedge M \wedge d, W, T).$$

But we know that $\mathcal{B}(t',u',b\wedge d, W, T_{ij})$ and so it follows immediately since $\mathcal{B}$ is anti-monotonic in its third argument and monotonic in its fifth.

The final condition we must establish is $H_\gamma 4$, i.e. $\mathcal{B}_\gamma(t,u,b\wedge M, \{(t,u)\}\cup W, T)$. As an example consider the move $t \xrightarrow{b_i,a}_L t_i$. Set $B' = \{\, b\wedge b_i \wedge b'_j \wedge \forall z.M_{ij} \mid u \xrightarrow{b'_j,a}_L u_j \,\}$. Then $b_i \wedge b \wedge b'_j \wedge M \to \vee B'$ and each $b\wedge b_i \wedge b'_j \wedge \forall z.M_{ij}$ in B' has the move $u \xrightarrow{b'_j,a}_L u_j$ associated with it which satisfies $(t_i,u_j) \notin \{(t,u)\}\cup W \Longrightarrow b' \to b\wedge b_i \wedge b'_j \wedge \forall z.M_{ij} \in T_{ij}(t_i,u_j) \subseteq T(t_i,u_j)$. □

Putting these two results together we have

Proposition B.5 *If $(r,r') \longrightarrow^*_W (t,u)$ then*

1. *$\mathit{close}(t,u,b,W) = (M,T)$ implies $CLOSE(t,u,b,W,M,T)$*

2. *$\mathit{match}(\gamma,t,u,b,W) = (M,T)$ implies $MATCH_\gamma(t,u,b,W,M,T)$*

Proof: Both statements are proved simultaneously and the proof proceeds by induction on the number of recursive calls to the procedures. Proposition B.3 and induction are used to establish the first while Proposition B.4 and induction establishes the second. □

Proof of Theorem 5.1:
To show $r \simeq_L^M r'$, we construct a boolean indexed family of relations $\mathbf{S}$ by letting

$$S^b = \{\, (t,u) \mid there\ exists\ p : (r,r') \longrightarrow^* (t,u),\ b \rightarrow d\ for\ some\ d \in p^T \,\}$$

Note that $(r,r') \in S^M$ as $\varepsilon : (r,r') \longrightarrow^* (r,r')$ and $\varepsilon^T = \{M\}$ because $T(r,r') = \{M\}$ follows from $CLOSE(r,r',true,\emptyset,M,T)$. So if we can prove $\mathbf{S}$ is a late symbolic bisimulation then we are done.

Suppose $(t,u) \in S^b$ and $t \xrightarrow{b_1,\alpha}_L t'$. We have to find matching derivations from u. We only consider $\alpha = a \in NAct$ here. The other cases are similar.
By the definition of $\mathbf{S}$ there exists $p : (r,r') \longrightarrow^* (t,u)$ and $b \rightarrow d$ for some $d \in p^T$. Moreover since $H(r,r',true,\emptyset,T)$ it follows that $\mathcal{B}(t,u,d,\emptyset,T)$. Let B'' be the set of booleans guaranteed by $\mathcal{B}(t,u,d,\emptyset,T)$ and define B' to be $\{\, d \wedge b'' \mid b'' \in B'' \,\}$. Then $b \wedge b_1 \rightarrow \vee B'$ since $d \wedge b_1 \rightarrow \vee B''$. Also for each $d \wedge b'' \in B'$ there exists a $u \xrightarrow{b_2,a}_L u'$ such that $b'' \rightarrow b_2$ and therefore $d \wedge b'' \rightarrow b_2$. It remains to show that $(t',u') \in S^{d \wedge b''}$ which is straightforward. Let p' be the obvious prolongation of the path p. Then $p' : (r,r') \longrightarrow^* (t',u')$, $d \in p'^T$ and $d \wedge b'' \rightarrow d$. □

Now we turn to the completeness of the algorithm. Again we emphasise that W consists of pairs of nodes rather than terms and therefore for finite graphs $close(t,u,b,W)$ will eventually terminate. This is because each call of *close* either terminates immediately or leads to another call of the form $close(t',u',b',\{(t,u)\} \cup W)$ with $(t,u) \notin W$ and this can not go on forever. So in particular $bisim(r,r')$ will always terminate. We show that the boolean it returns is $mgb(r,r')$. Here it is convenient to consider a modification to the definition of late symbolic bisimulations where the condition "$b \wedge b_1 \rightarrow \vee B$" is replaced with the stronger condition "$b \wedge b_1 = \vee B$". We leave it to the reader to show, using Theorem 4.5 that this determines exactly the same equivalence.

Proposition B.6 *Suppose $t \simeq_L^b u$ and $p : (r,r') \longrightarrow^* (t',u')$, $b \rightarrow p^C$ and $close(t,u,p^C,p^S) = (M,T)$. Then $b \rightarrow M$.*

Proof: By induction on the length of the computation of *close*. From the definition of $CLOSE$ we know that $T(t,u) = \{p^C \wedge M\}$ and M has the form $p^C \wedge \bigwedge_\gamma M_\gamma$ where each M_γ is returned from a call to the procedure $match(\gamma,t,u,p^C,p^S)$. So it is sufficient to show that for each $\gamma \in \{a,\ c?,\ c!\}$ that $b \rightarrow M_\gamma$. As an example we consider the case when γ is a; the other cases are similar.

We have to show $b \rightarrow M_a$ where $M_a = \bigwedge_i (b_i \rightarrow \bigvee_j (b'_j \wedge M_{ij})) \wedge \bigwedge_j (b'_j \rightarrow \bigvee_i (b_i \wedge M_{ij}))$ and $close(t_i,u_j,p^C \wedge b_i \wedge b'_j,\{(t,u)\} \cup p^S) = (M_{ij},T_{ij})$ for $t \xrightarrow{b_i,a}_L t_i, u \xrightarrow{b'_j,a}_L u_j$.

Let $t \xrightarrow{b_i,a}_L t_i$. Since $t \simeq_L^b u$ there exists a set of booleans B such that $b \wedge b_i = \vee B$ and for every $b' \in B$ there exists $u \xrightarrow{b'_j,a}_L u_j$, $b' \rightarrow b'_j$ and $t_i \simeq_L^b u_j$. If $(t_i,u_j) \in \{(t,u)\} \cup p^S$ then $M_{ij} = true$, hence $b' \rightarrow M_{ij}$; Otherwise let p_0 be the matching path to (t_i,u_j) formed by appending $(t \xrightarrow{b_i,a}_L t_i, u \xrightarrow{b'_j,a}_L u_j)$ at the end of p. Then $p_0^C = p^C \wedge b_i \wedge b'_j$ and $p_0^S = \{(t,u)\} \cup p^S$. If we can show $b' \rightarrow p_0^C$ then by induction $b' \rightarrow M_{ij}$. But this is trivial because from $b' \in B, b \wedge b_i = \vee B$, we know $b' \rightarrow b \wedge b_i$. Since $b \rightarrow p^C$ and $b' \rightarrow b'_j$, it follows that $b' \rightarrow p^C \wedge b_i \wedge b'_j = p_0^C$.

Now in either case we have $b' \rightarrow b'_j \wedge M_{ij}$. This is true for each $b' \in B$ and so $\vee B \rightarrow \bigvee_j (b'_j \wedge M_{ij})$. Since $b \wedge b_i = \vee B$ it follows that $b \rightarrow (b_i \rightarrow \bigvee_j (b'_j \wedge M_{ij}))$. This argument holds for every move $t \xrightarrow{b_i, a}_L t'$ and therefore $b \rightarrow \bigwedge_i (b_i \rightarrow \bigvee_j (b'_j \wedge M_{ij}))$ and by symmetry we can conclude that $b \rightarrow M_a$. □

Proof of Thereom 5.2:
An immediate corollary to the above proposition. □

Acknowledgements: The authors would like to thank Alan Jeffrey and Xinxin Liu for carefully reading a draft of this paper and suggesting many improvements. We would also like to thank Uffe Engberg for his detailed and constructive criticism.

References

[BS90] J. Bradfield and C. Stirling. Local model checking for infinite state spaces. Technical Report ECS-LFCS-90-115, University of Edinburgh, June 1990.

[Bur91] G. Burns. A language for value-passing ccs. Technical Report ECS-LFCS-91-175, 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.

[GLZ89] J. Godskesen, K. Larsen, and M. Zeeberg. Tav user manual. Report R89-19, Aalborg University, 1989.

[JP92] B. Jonsson and J. Parrow. Deciding bisimulation equivalences for a class of non-finite-state programs. *Information and Computation*, 1992. to appear. Also available as SICS research Report R-89/8908.

[Lar86] K. G. Larsen. *Context-Dependent Bisimulation Between Processes.* Ph.D. thesis, Edinburgh University, 1986.

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

[MPW92] R. Milner, J. Parrow, and D. Walker. Mobile logics for mobile processes. *Theoretical Computer Science*, 1992. to appear.

[SV89] R. De Simon and D. Vergamimi. Aboard auto. Report RT111, INRIA, 1989.

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

[Wol86] P. Wolper. Expressing interesting properties of programs in propositional temporal logic (extended abstract). In *Proc. 13th ACM POPL*, pages 184–193, January 1986.