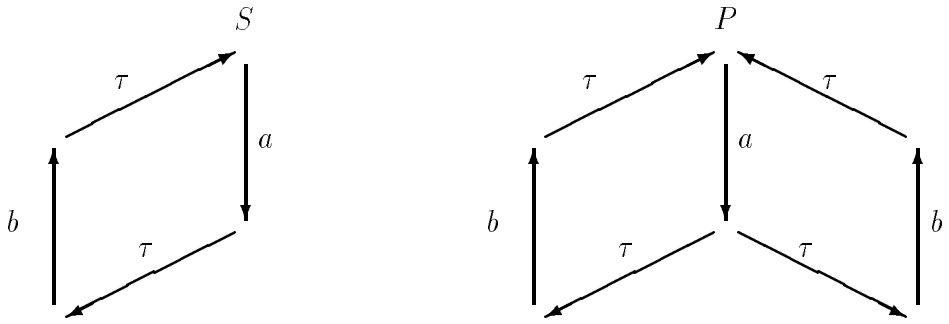


Figure 1: Transition Graphs for S and P

and

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

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

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

and

$$\begin{aligned} P' &\Longleftarrow (Q' \mid R' \mid R' \mid T' \mid T')\backslash\{in_1, in_2, \beta\} \\ Q' &\Longleftarrow c?x.(even(x \end{aligned}$$

$$.\tau \; \underset{\mathsf{J}}{in} \quad .\tau.Q$$

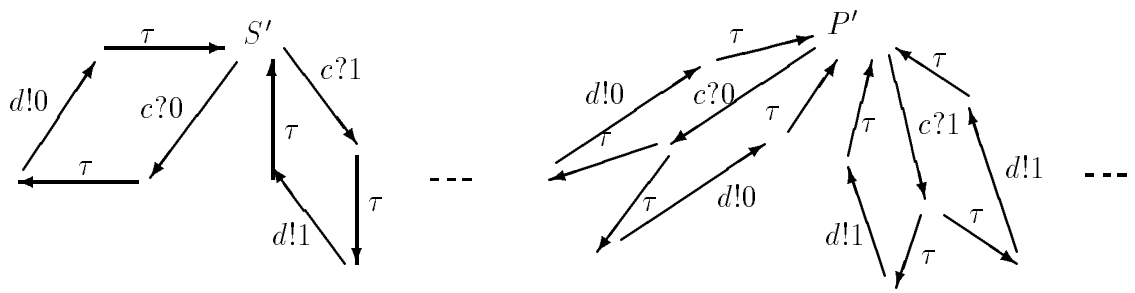


Figure 2: Transition Graphs for S' and P'

$c?x$ 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 . These are both finite graphs where the symbolic actions

and

$$d\,bx/2\rfloor\Downarrow \quad \tau$$

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 $\simeq^b_E$ and $\simeq^b_L$ between open terms. For example $t \simeq^b_E u$ 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

> $t \simeq^b_i u$ 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 $\simeq^{true}_i$. 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 $t \simeq^b_i u$; 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 $t \simeq^b_i u$. We call this $mgb_i(t,u)$ which has the property that $t \simeq^{mgb_i(t,u)}_i u$ and whenever $t \simeq^b_i u$ 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

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

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

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

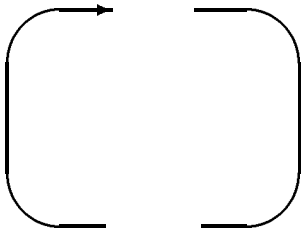
$$P(\underline{x}) \Longleftarrow t,$$

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 $\beta \in GuAct$ let $\stackrel{\beta}{\longmapsto}$ 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 $new(V)$ 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 $t \stackrel{\beta}{\longmapsto} t'$ 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 , 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 $new(y) = x$.

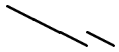


resulting terms are also equivalent so long as we update the evaluation so as to take the substitution into account:

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* $x \in fv($



$$m \overset{b,a}{\longmapsto} n,\ a \in NAct \quad \text{implies} \quad m_\sigma \overset{b\sigma,a}{\longrightarrow}_L n_\sigma$$

$$m \overset{b,c!e}{\longmapsto} n \quad \text{implies} \quad m_\sigma \overset{b\sigma,c!e\sigma}{\longrightarrow}_L n_\sigma$$

$$m \overset{b,c?x}{\longmapsto} n \quad \text{implies} \quad m_\sigma \overset{b\sigma,c?x}{\longrightarrow}_L n_\sigma$$

Figure 8: Late symbolic operational semantics

We write $t \simeq^b_L u$ 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 $\{ \simeq^b_L |\ b \in BExp \}$ is the maximal symbolic late bisimulation. We can also show that each $\simeq^b_L$ is an equivalence relation, using the same approach as with $\sim^\rho_L$.

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

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

Then the following is a symbolic bisimulation:

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

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. *$\rho \models t \overset{c?x}{\longrightarrow} t'$ if and only if $t \overset{b,c?x}{\longrightarrow}_L t'$ for some b such that $\rho \models b$*

$\overset{b,c?}{}$

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 $\rho \models b_1$

We first check that b

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 $m \stackrel{c?x}{\longmapsto} n$ 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 $b \rightarrow mgb_L(t,u)$.

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 $T: \langle \mathcal{T}, \mathcal{T} \rangle \longmapsto 2^{BExp}$ – 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 $b \rightarrow b'$ 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 notationT

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

$close(t,\, u,\, b,\, W) =$
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 $(\bigwedge_i (b_i$

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__3:R_2:
```

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

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

(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 $t \overset{b_1,\alpha}{\longrightarrow}_E t'$ 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 $u \overset{b_2,\alpha'}{\longrightarrow}_E u'$ 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. *$\rho \models t \overset{c?x}{\longrightarrow} t'$ if and only if $t \overset{b,c?x}{\longrightarrow}_E t'$ for some b such that $\rho \models b$*
2. *$\rho \models t \overset{c!v}{\longrightarrow} t'$ if and only if $t \overset{b,c!e}{\longrightarrow}_E t'$ for some b and e such that $\rho \models b$ and $\rho(e) = v$*
3. *$\rho \models t \overset{a}{\longrightarrow} t'$ if and only if $t \overset{b,a}{\longrightarrow}_E t'$ for some b and $\rho \models b$*

In analogy with Propositions 4. 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

$$R^\rho_{\mathbf{S}} = \{ (t,u)$$

Proposition 6.4

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

1.

The algorithm for computing late symbolic bisimulation presented in Figure 9 can also be modified to calculate early symbolic bisimulation.

```
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

at

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].

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 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 $\stackrel{a}{\longrightarrow}$ 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. $p \stackrel{c?x}{\longrightarrow} \lambda x.p' \Longrightarrow$ there exists $q \stackrel{c?y}{\longrightarrow} \lambda y.q'$ such that for all $v \in V, (p'[v/x], q'[v/y]) \in R$

2. for any other actions $a, p \stackrel{a}{\longrightarrow} p' \Longrightarrow$ there exists $q \stackrel{a}{\longrightarrow} q'$ 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 $p \sim q$*

$a.p \xrightarrow{a} p$ $\quad a \in NAct \cup \{\, c!v \mid c \in Chan \,\}$

$c?x.p \xrightarrow{c?x} \lambda x.p$ $\quad c \in Chan$

p

- $P(\underline{e}) \stackrel{b,c?x}{\longmapsto} t'$ because $t[\underline{e}/\underline{x}] \stackrel{b,c?x}{\longmapsto} t'$ where $P(\underline{x}) \Longleftarrow$

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

Now from Lemma A.6

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

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

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

Again applying Lemma A.6 we get

$$u \stackrel{b',c?w}{\longmapsto} u'$$

for some w

The proof follows the same pattern as before. Define

$$S^{\rho}$$

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

$$(\sigma', \eta'$$

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

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

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

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

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

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

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

$MATCH_\gamma(t,u,b,W,M,T) =_{def} H_\gamma(t,u,b,W,M,T)$. □

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.

□

Proof of Theorem 5.1:
To show $r \simeq_L^M r'$, we construct a boolean indexed family of relations **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 **S** is a late symbolic bisimulation then we are done.

Suppose $(t,u) \in S^b$ and $t \stackrel{b_1,\alpha}{\longrightarrow}_L t'$. 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 **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

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*, 1:27 –292, 1989.

[Wol86] P. Wolper. Expressing interesting properties of programs in propositional temporal logic (extended abstract). In *Proc. 13th ACM POPL*, pages 184–19 , January 1986.