Symbolic bisimulations

M. Hennessy,* H. Lin

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, \( \equiv_b \), are then defined over symbolic transition graphs. These are related to standard bisimulations by proving that \( t \equiv_b u \) 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 \( t \equiv_b u \) which can be applied to arbitrary finite symbolic trees.

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 [10] 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

\[ S \equiv a \cdot \tau \cdot b \cdot \tau \cdot S \]
written in the language CCS. The first is a simple cyclic process which performs the action \( a \) followed by a \( \tau \) action, \( ab \) action and finally another \( \tau \) action to arrive back at its original start state. Here \( a \) and \( b \) are some formal uninterpreted actions while \( \tau \) 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 \( \bar{a} \). 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 \( \alpha \). That copy now performs the external action \( b \) and then synchronises with \( Q \) using the other internal action \( \beta \) while the other copy of \( R \) is forced to idle. The operator \( \{ z, \beta \} \) indicates that the two actions \( \alpha \) and \( \beta \) 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 \( \tau \) 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

\[
S' = c?x.\tau.d![x/2].\tau.S'
\]

and

\[
P' = (Q|R|R)|\{x, \beta\},
Q' = c?x.(even(x)\rightarrow in_1!x.\beta.Q', in_2!x.\beta.Q'),
R' = in_1?x.d![x/2].\bar{\beta}.R',
T' = in_2?x.d![(x-1)/2].\bar{\beta}.T'.
\]

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 \( c?x \) while \( c!e \) denotes the output of the value of the expression \( e \) along \( c \). Communication is modelled as before with \( \tau \) 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 \([x/2] \) 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 [4–6]. 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 Fig. 1 while those for \( P' \) and \( S' \) are in Fig. 2. These graphs are convenient representations of the possible transitions which the processes can perform. The two graphs in Fig. 1 are finite and when the standard algorithm is applied to them it returns true. However, the graphs in Fig. 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

---

Fig. 1. Transition graphs for \( S \) and \( P \).

Fig. 2. Transition graphs for \( S' \) and \( P' \).
more abstract descriptions of processes in terms of symbolic actions. For example the symbolic graphs associated with $S'$ and $P'$ are given in Fig. 3. These are both finite graphs where the symbolic actions are of the form $c?x, d!x/2, d!(x-1)/2$ and $d![x/2]$ and $\tau$, 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 \[11\]. 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:

$$P_1 \equiv (c?x. \text{even}(x) \rightarrow R_1, R_2) + c?x. R_3,$$

$$P_2 \equiv (c?x. \text{even}(x) \rightarrow R_1, R_3) + (c?x. \text{odd}(x) \rightarrow R_2, R_3).$$

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 $\text{even}(x) \rightarrow R_1, R_2$ cannot 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 componentwise equivalent to $\text{even}(x) \rightarrow R_1, R_2$. For the two possible choices of $c?$ moves, to $\text{even}(x) \rightarrow R_1, R_3$ and $\text{odd}(x) \rightarrow R_2, R_3$, respectively, there are instantiations

![Fig. 3. Symbolic transition graphs for $S'$ and $P'$](image-url)
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 [10], we will have for example that

\[
\begin{align*}
\text{c?x.t} & \xrightarrow{c?x} t \\
\text{c!e.t} & \xrightarrow{c!e} t
\end{align*}
\]

for arbitrary terms. More generally, symbolic actions will have boolean guards associated with them indicating conditions under which they can be performed. However, note that this form of operational semantics must necessarily 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 \( \approx_E \) and \( \approx_L \) between open terms. For example, for arbitrary terms, \( t \approx_E u \) indicates that with respect to the boolean expression \( b \) the processes \( t \) 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 equivalence are assumed to be 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. assignments of values to variables, a boolean expressions 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 [10]. Once more there is a late and early version and, if we use \( \rho \) to range assignments of values to free variables, we obtain relations of the form

\[
\rho \models t \approx_{_E} u \quad \text{and} \quad \rho \models t \approx_{_L} u.
\]

Intuitively, these mean that with respect to the assignment \( \rho \) \( t \) is early/late bisimulation equivalent to \( u \).

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

\[
\text{\( t \approx_{_i} u \) if and only if for every assignment \( \rho \) which satisfies the boolean \( b \rho \models t \approx_{_i} u \),}
\]

where \( i \) is either \( E \) or \( L \).
This result underlines the significance of symbolic bisimulations. For example, it shows that the standard form of bisimulation equivalence between closed terms, \( \sim_t \), coincides with \( \sim_t^{true} \). 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 \( t \sim^b_t u \); for example it turns out that \( t \sim^false_t u \) for all terms \( t, u \). We are interested in calculating the weakest boolean for which \( t \sim^b_t u \). We call this \( mgb_b(t, u) \) which has the property that \( t \sim^* t \) and whenever \( t \sim^b_t u \) then \( b \) implies \( mgb_b(t, u) \). We also wish to generate a symbolic bisimulation which provides a witness to the fact that \( t \sim^b_t u \). 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 \( b \). However, we can easily find a finite representation by using the fact that if \( b \) implies \( b' \) then \( t \sim^b_t u \) implies \( t \sim^{b'}_t u \). For both the early and late case, we present algorithms which given a pair \( t, u \) returns a boolean expression logically equivalent to \( mgb_b(t, u) \) and the finite representation of a witnessing symbolic bisimulation. It is somewhat convenient to assume that the finite transition graphs are in fact trees; this makes the description of the algorithms and the proof of their correctness more straightforward. However, this is not a major restriction as every finite graph can be unwound into an equivalent tree. Moreover, with somewhat more effort the algorithms could be adopted to apply to finite graphs in general.

Our algorithms are similar to the bisimulation checking algorithm from [9] in that both follow closely the definition of bisimulations. When given two terms \( t, u \) the algorithm will return a boolean expression equivalent to \( mgb_b(t, u) \). 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 Section 2, 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. 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 countable set of variables, \( \text{Var} = \{x_0, x_1, \ldots \} \), and a set of values \( \text{V}. \) Each variable \( x \) is associated with a set of evaluations, \( \text{Eval} \), ranging over by \( \rho \), represents the set of total functions from \( \text{Var} \) to \( \text{V} \). We use the standard notation \( \rho[v/x] \) to denote the evaluation which differs from \( \rho \) only in that it maps \( x \) to \( v \). A substitution is a partial injective mapping from \( \text{Var} \) to \( \text{V} \) whose domain is finite. We use \( \text{Sub} \) to represent the set of substitutions and this set is ranged over by \( \sigma \). We use the notation \( \sigma[x\rightarrow y] \) to indicate the obvious modification to the substitution \( \sigma \) and for any finite subset \( W \) of \( \text{Var} \) we use \( \text{new}(W) \) to denote the variable \( x_i \) where \( i \) is the least index such that \( v_i \) is not in \( W \).

We also presume a set of expressions, \( \text{Exp} \), ranged over by \( e \), which includes \( \text{Var} \) and \( \text{V} \). Each \( e \) has associated with it a set of free variables, \( \text{fv}(e) \), and it is assumed that both evaluations and substitutions behave in a reasonable manner when applied to expressions; the application of \( \rho \) to \( e \), denoted \( \rho(e) \), yields a value while the application of a substitution, denoted \( e\sigma \), yields another expression with the property that \( \text{fv}(e\sigma) = \sigma(\text{fv}(e)) \) where the latter is defined in the obvious manner. It is also assumed that if \( \rho \) and \( \rho' \) agree on \( \text{fv}(e) \) then \( \rho(e) = \rho'(e) \). We also presume a set of boolean expressions, \( \text{BExp} \), ranged over by \( b \), with similar properties but we will use the more suggestive notation \( \rho \models b \) to indicate that \( \rho(b) = \text{true} \). 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, a pair 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, \( \text{Chan} \),
an output action, of the form $c!e$, or a neutral action such as $\tau$. So let $SyAct$, ranged over by $\alpha$, 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 $fv(x)$ and $bv(x)$ are empty. Then the set of guarded actions is given by

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

We use $\beta$ 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 $fu(n)$ and every branch is labelled by a guarded action such that if a branch labelled by $(b, \alpha)$ goes from node $m$ to $n$, which we write as $m \xrightarrow{b, \alpha} n$, then $fu(b) \cup fu(\alpha) \subseteq fu(m)$, and $fv(n) \subseteq fv(m) \cup bv(\alpha)$.

We will frequently write $m \xrightarrow{\text{true}, \alpha} n$ for $m \xrightarrow{\beta} n$.

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

$$t ::= \text{nil} \mid x.t \mid b\mathbin{\rightarrow}t,t$$

$$\mid t + t \mid t|t \mid t\setminus c \mid P(c)$$

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

$$P(x) \Leftarrow t,$$

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 $x$.

In this language $c?x$ binds occurrences of the variable $x$ in the subterm $t$ 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 $t \xrightarrow{\beta}$ be the least relation which satisfies the rules in Fig. 4 (the symmetric rules for $+$ and $|$ have been omitted). 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 $t \xrightarrow{\beta} t'$ if we can derive this statement from the rules in Fig. 4. One can easily check that the requirements of Definition 2.1 are satisfied.
Fig. 4. Symbolic operational semantics of CCS.

\[
\begin{align*}
\alpha.t & \xrightarrow{\nu_{y,x}^{\alpha}} t & \alpha \in NaActU \{ e!c \mid c \in Chan, e \in Exp \} \\
\nu.x.t & \xrightarrow{\nu_{u}^{\alpha}} t[y/x] & \text{where } y = new(fv(\nu.x.t)) \\
t & \xrightarrow{b_{\alpha}} t' & \text{implies } (b \rightarrow t, u) \xrightarrow{b_{\alpha}} t' \\
t & \xrightarrow{b_{\alpha}^{-1}} u' & \text{implies } (b \rightarrow t, u) \xrightarrow{-b_{\alpha}} u' \\
t & \xrightarrow{b_{\alpha}} t' & \text{implies } t + u \xrightarrow{b_{\alpha}} t' \\
t & \xrightarrow{b_{\alpha}^{-1}} t' & \text{implies } t \upharpoonright u \xrightarrow{b_{\alpha}^{-1}} t' \upharpoonright u \\
& & \alpha \in NaActU \{ e!c \mid c \in Chan, e \in Exp \} \\
t & \xrightarrow{b_{\alpha}^{t}c_{t}} t' & \text{implies } t \upharpoonright u \xrightarrow{b_{\alpha}^{t}c_{t}} t'[y/x] \upharpoonright u \\
& & y = \begin{cases} x & \text{if } x \not\in fv(u) \\ new(fv(t'[u])) & \text{otherwise} \end{cases} \\
t & \xrightarrow{b_{\alpha}^{t}c_{t}} t' \upharpoonright u \xrightarrow{b_{\alpha}^{e}} u' & \text{implies } t \upharpoonright u \xrightarrow{b_{\alpha}^{t}c_{t}} t'[e/x] \upharpoonright u' \\
t & \xrightarrow{b_{\alpha}^{-1}} t' & \text{implies } t \setminus c \xrightarrow{b_{\alpha}^{-1}} t \setminus c \\
& & \text{if } \alpha \text{ does not use the channel } c \\
t \xrightarrow{b_{\alpha}} t' & \text{implies } P(\xi) \xrightarrow{b_{\alpha}} t' \\
& & \text{if } P(\xi) \rightleftharpoons t \text{ is a declaration}
\end{align*}
\]

Fig. 5. A symbolic transition graph.

An example of a symbolic transition graph generated from the language in this way has already been seen in Fig. 3, although the sets of free variables were not shown. In Fig. 5 we give another example of a symbolic graph assuming the declaration

\[P(y) \Leftarrow c?x. x=y-d!y.P(y), c!(x+y).P(y)\]

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

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

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 kinds 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, \( m_\sigma \). Assuming some fixed symbolic transition graph let the set of terms be

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

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 \( T \). 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 \( m_\sigma \) is defined in the obvious way by \( \text{fv}(m_\sigma) = \{ \sigma(\text{fv}(m)) \} \) and \( m_\sigma \) is said to be closed if its set of free variables is empty. We will be somewhat relaxed about the condition that for \( m_\sigma \) to be a term the domain of \( \sigma \) must be contained in the set of free variables of \( m \); but whenever we construct a new term \( m_\sigma \), we assume that it is well formed in that if necessary the substitution \( \sigma \) is restricted to \( \text{fv}(m) \). Also if \( t \) is a term of the form \( m_\sigma \), 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 set of late actions, \( \text{LAct} \). As explained in Section 1 the late operational semantics uses the neutral actions such as \( r \), the output actions of the form \( c!v \) and the more symbolic form of input actions, \( c?x \). So \( \text{LAct} \) is defined to be
\[ \text{NAct} \cup \{ c!v \mid c \in \text{Chan}, v \in V \} \cup \{ c?x \mid c \in \text{Chan}, x \in \text{Var} \}. \]

The judgements \( \rho \models t \xrightarrow{a} u \) are defined to be the least ones which satisfy the rules in Fig. 6. These should be more or less self-explanatory; much of the work has already

\[
\begin{align*}
  m \xrightarrow{b_\text{act}} n, & a \in \text{NAct} \quad \text{implies} \quad \rho \models m_{\sigma} \xrightarrow{a} n_{\sigma} \\
& \text{provided} \quad \rho \models b_\sigma \\
  m \xrightarrow{c^x_\text{act}} n, & \text{implies} \quad \rho \models m_{\sigma} \xrightarrow{c^x_\sigma(\sigma)} n_{\sigma} \\
& \text{provided} \quad \rho \models b_\sigma \\
  m \xrightarrow{c^z_\text{act}} n, & \text{implies} \quad \rho \models m_{\sigma} \xrightarrow{c^z_\sigma} n_{\sigma[x \mapsto z]}, \ z = n_\sigma(\text{fv}(m_\sigma)) \\
& \text{provided} \quad \rho \models b_\sigma
\end{align*}
\]

Fig. 6. Late operational semantics.
been factored out by the symbolic transition graph and the rules merely interpret the symbolic actions using the evaluation $\rho$. In the input rule the possibly new variable $z$ is used because $x$ might be a free variable in the term $m_x$.

The standard definition of a bisimulation is a relation over closed terms and in our setting a closed term is mimicked 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, $\mathcal{L}\mathcal{B}$, over such collections of relations. Let $R = \{R^\rho | \rho \in \text{Eval}\}$ be such that each $R^\rho \subseteq \langle \mathcal{T}, \mathcal{T} \rangle$. Then $\mathcal{L}\mathcal{B}(R)$ is the Eval-indexed family of symmetric relations defined by $(t, u) \in \mathcal{E}_\mathcal{L}\mathcal{B}(R)$ if

1. $\rho \vdash t \xrightarrow{c} t'$ implies $\rho \vdash u \xrightarrow{c} u'$ for some $u'$ such that $(t'[x \mapsto z], u'[y \mapsto z]) \in R^\rho[z]$ for all $z \in V$, where $z$ is a fresh variable;
2. for any other late action $\rho \vdash t \xrightarrow{a} t'$ implies $\rho \vdash u \xrightarrow{a} u'$ for some $u'$ such that $(t', u') \in R^\rho$.

**Definition 3.1 (Late bisimulations).** $R$ is a late bisimulation if $R \subseteq \mathcal{L}\mathcal{B}(R)$, i.e. $R^\rho \subseteq \mathcal{L}\mathcal{B}(R)^\rho$, for each $\rho$.

We write $\rho \vdash t \sim_L u$ if there is a late bisimulation $R$ such that $(t, u) \in R^\rho$. It will sometimes be more convenient to denote this by $t \sim_L u$. The standard theory of bisimulations apply here; the functional $\mathcal{L}\mathcal{B}$ is (pointwise) monotonic and therefore has a maximal fixpoint and the $\rho$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 [10] 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 $\rho \vdash t \sim_L u$ then $\rho \cdot \sigma^{-1} \vdash t \sim_L u \sigma$.

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 $x \in \text{fv}(t, u)$ then $\rho \vdash t \sim_L u$ if and only if $\rho' \vdash t \sim_L u$. 
Proof. For any \( X \subseteq V \) let \( \rho =_X \rho' \) if for every \( v \in X \rho(x) = \rho'(x) \). Let \( R \) be defined by

\[
R^\rho = \{(t, u) | \exists \rho' \cdot \rho = f_v(u) \rho' \text{ and } \rho' \vdash t \sim_L u \}\).
\]

One can prove that if \( \rho = f_v(u) \rho' \) then \( \rho \vdash t \Rightarrow t' \) if and only if \( \rho' \vdash t \Rightarrow t' \) and from this it follows that \( R \) is a late bisimulation. \( \square \)

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 \( \rho \vdash t \overset{c?x}{\rightarrow} t' \) with \( \rho \vdash u \overset{c?y}{\rightarrow} u' \) we have to ensure that for each \( v \in V \rho[v/z] \vdash t'[x \mapsto z] \overset{c?z}{\sim_L} u'[y \mapsto z] \). In Section 4 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 \( \sim_L \).

Consider the graph in Fig. 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 \( f_v(p_0) = f_v(q_0) = 0 \). It is easy to check that \( \rho \vdash p_0 \sim_L q_0 \) for all evaluations \( \rho \).

Let us see how this might be deduced from the symbolic transition graph. The symbolic move \( p_0 \overset{c?x}{\rightarrow} p_1 \) can be matched by the corresponding move \( q_0 \overset{c?x}{\rightarrow} q_1 \) if we can show that \( p_1 \) and \( q_1 \) are symbolically equivalent. Here we come to a problem: we cannot always expect, for example, the move \( p_1 \overset{a}{\rightarrow} p_{11} \) to be always matched by

Fig. 7. Example graph.
$q_1 \xrightarrow{a} q_{11}$ 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 cannot be matched by $q_1 \xrightarrow{a} q_{12}$. 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, $\simeq_b$. In the present example, we will have $p_{11} \simeq_b^s q_{11}$ whenever $b$ implies $x=0$ and $p_{11} \simeq_b^s q_{12}$ whenever it implies $x \neq 0$. Let us now re-examine in what way the nodes $p_1$ and $q_1$ can be properly matched, i.e. why can we claim $p_1 \simeq_b^s q_1$ for a particular $b$. The most we can hope for in matching the typical move $p \xrightarrow{a} p_{11}$ is that the present 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 $p_1 \simeq_b^s q_1$ then we can let $B = \{x=0, x \neq 0\}$ and in the first case the move is matched by $q_1 \xrightarrow{a} q_{11}$ while in the second it is matched by $q_1 \xrightarrow{a} q_{12}$.

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 cannot use the arrows in the transition graphs directly for the same reason as in Section 3; we must work with terms. So the symbolic late operational semantics is given as a collection of relations over terms, $\mathcal{L}_b$, where $\beta$ is a guarded action. These are defined in Fig. 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 $S = \{ S_b \mid b \in BExp \}$ be a parameterised family of relations over terms. Then $\mathcal{S}_L(S)$ is the $BExp$-indexed family of symmetric relations defined by $$(t, u) \in \mathcal{S}_L(S)_b$$ if whenever $t \xrightarrow{b_1, x} t'$ there is a collection of booleans $B$ such that $b \land b_1 \to \bigvee B$ and for each $b \in B$ there exists a $u \xrightarrow{b_2, x} u'$ such that $b' \to b_2$ and
1. if $\alpha$ is of the form $a$ where $a \in \text{Act}$ then $\alpha = a$ and $(t', u') \in S_b'$
2. if $\alpha$ is of the form $c!e$ then $\alpha = c!e'$, $b' \to e = e'$ and $(t', u') \in S_b'$
3. if $\alpha$ 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). $S$ is a late symbolic bisimulation if $S \subseteq \mathcal{S}_L(S)$.

$$m \xrightarrow{b,a} n, a \in \text{Act} \quad \text{implies} \quad m_\sigma \xrightarrow{b_2, a, x} n$$
$$m \xrightarrow{b,c,e} n \quad \text{implies} \quad m_\sigma \xrightarrow{b_2, c_2, e_2} n$$
$$m \xrightarrow{b,c?x} n \quad \text{implies} \quad m_\sigma \xrightarrow{b_2, c_2, x} n \text{[}x \mapsto z\text{]}$$
$$z = n_{\text{cw}}(m_\sigma)$$

Fig. 8. Late symbolic operational semantics.
We write \( t \simeq^b u \) if there is a symbolic late bisimulation \( S \) such that \((t, u) \in S^b\). As usual the standard theory applies because \( \mathcal{L} \mathcal{B} \) is pointwise monotonic. So \( \{ t \simeq^b u \mid b \in \mathcal{B} \mathcal{E}x \} \) is the maximal symbolic late bisimulation. We can also show that each \( \simeq^b \) is an equivalence relation, using the same approach as with \( \simeq^r \).

As an example consider the graph in Fig. 7 and let \( A, B, C \) be the following pairs of sets:

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

Then the following is a symbolic bisimulation:

\[
S^{true} = A \cup A^{-1}, \quad S^{x=0} = B \cup B^{-1}, \quad S^{x\neq 0} = C \cup C^{-1}.
\]

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) \( \rho \vdash t \overset{c}{\rightarrow} t' \) if and only if \( t \overset{b,c}{\rightarrow}^L t' \) for some \( b \) such that \( \rho \vdash b \).

(2) \( \rho \vdash t \overset{e}{\rightarrow} t' \) if and only if \( t \overset{b,e}{\rightarrow}^L t' \) for some \( b \) and \( e \) such that \( \rho \vdash b \) and \( \rho(e) = v \).

(3) for each \( a \in \mathcal{N} \mathcal{A}ct, \rho \vdash t \overset{a}{\rightarrow} t' \) if and only if \( t \overset{b}{\rightarrow}^L t' \) for some \( b \) such that \( \rho \vdash b \).

**Proof.** Follows in straightforward manner from the definitions of the two arrows. \( \Box \)

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

\[
R_S = \{(t, u) \mid \exists b \cdot \rho \vdash b \text{ and } (t, u) \in S^b\}.
\]

**Proposition 4.3.** If \( S \) is a late symbolic bisimulation then \( R_S \) is a late bisimulation.

**Proof.** Let \( (t, u) \in R_S \), i.e. \( (t, u) \in S^b \) for some \( b \) such that \( \rho \vdash b \). We much show that the possible moves from \( t \) and \( u \) are properly matched.

1. Suppose \( \rho \vdash t \overset{c}{\rightarrow} t' \) where \( a \in \mathcal{N} \mathcal{A}ct. \) Then by Proposition 4.2 it follows that \( t \overset{b_1}{\rightarrow}^L t' \) for some \( b_1 \) such that \( \rho \vdash b_1 \). Therefore, there exists a set of booleans \( B \) such that \( b \land b_1 \rightarrow \mathcal{V} B \) and for each \( b' \in B \) there is a \( u \overset{b_2}{\rightarrow}^L u' \) such that \( b' \rightarrow b_2 \) and \( (t', u') \in S^{b'} \). Since \( \rho \vdash b \land b_1 \) and \( b \land b_1 \rightarrow \mathcal{V} B \), there must be \( b' \in B \) such that \( \rho \vdash b' \), and hence \( \rho \vdash b_2 \). Now apply Proposition 4.2, we have \( \rho \vdash u \overset{b}{\rightarrow} u' \) for the \( u' \) associated with this \( b' \). Since \( \rho \vdash b' \), \((t', u') \in R_S\) as required.

2. Suppose \( \rho \vdash t \overset{e}{\rightarrow} t' \). Arguing as in the previous case we get \( t \overset{b_1}{\rightarrow}^L t' \) for some \( b_1 \) and \( e \) such that \( \rho \vdash b_1 \) and \( \rho(e) = v \), and there exists a \( b' \), \( \rho \vdash b' \) and \( u \overset{b_2}{\rightarrow}^L u' \) such
that \((t', u') \in S^b'\), \(b' \rightarrow b_2 \land e = e'\), and \(\rho(e_2) = v\). This means that \(\rho \models u \xrightarrow{e_1} u'\) and 
\((t', u') \in R^b_\mathcal{S}\).

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

Conversely, starting with a late bisimulation \(R\) 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 \(S_R\) be defined by

\[ S^b_R = \{(t, u) \mid \rho \models b \text{ implies } (t, u) \in R^b\} \]

**Proposition 4.4.** If \(R\) is a late bisimulation then \(S_R\) is a symbolic late bisimulation.

**Proof.** Let \((t, u) \in S^b_R\). We show that their symbolic actions can be matched in the appropriate manner.

1. Suppose \(t \xrightarrow{b, e} t'\). We must find a set of booleans \(B\) such that \(b \land b_1 \rightarrow \bigvee B\) and for each \(b' \in B\) there must exist a move \(u \xrightarrow{b_2, e} u'\) such that \(b' \rightarrow b_2 \land e = e'\) and 
\((t', u') \in S^b_R\). For convenience let us assume that for each \(u'\) there is at most one \(\beta\) such that \(u \xrightarrow{\beta} u'\); the same argument holds even without this assumption but we would have to introduce even clumsier notation. Let \(U\) be the set \([u' \mid u \xrightarrow{b(u')} u']\).

For each \(u' \in U\) let \(b_{u'}\) be a boolean expression which satisfies

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

and let \(b_{u'} = b_{u'} \land b(u') \land e = e(u')\). Finally, let \(B = \{b_{u'} \mid u' \in U\}\). We first check that \(b \land b_1 \rightarrow \bigvee B\), i.e. \(\rho \models b \land b_1\) implies \(\rho \models b_{u'}\) for some \(u' \in U\). Assuming \(\rho \models b \land b_1\), it follows that 
\((t, u) \in R^b\) and \(\rho \models t \xrightarrow{c, z} t'\) where \(\rho(e_1) = v\) (by Proposition 4.2). So this move can be matched by some \(\rho \models u \xrightarrow{c, z} u'\) such that \((t', u') \in R^b\). This means \(\rho \models b_{u'}\). Applying the same proposition we obtain \(u \xrightarrow{b(u'), c, z} u'\) 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 \(u \xrightarrow{b(u'), c, z} u'\). It is obvious that \(b_{u'} \rightarrow b(u') \land e = e(u')\) and \((t', u') \in S^b_R\) follows from the definition of \(b_{u'}\).

2. The case \(t \xrightarrow{x} t'\) is similar.

3. Suppose \(t = m_n \xrightarrow{b, c, x} m'_n\). Again we must find a set of booleans \(B\) such that \(b \rightarrow \bigvee B\) and for every \(b'\) in \(B\) there exists a matching move \(u = n \xrightarrow{b_2, c, z} u_n\) such that \(b' \rightarrow b_2\) and 
\((m'_n[x \rightarrow z], n'_n[y \rightarrow z]) \in S^b_R\) with \(z\) a fresh variable. We proceed as in the previous
case by letting $U = \{ n'_i | n_i \xleftarrow{b(n')} c y L n'_i \}$ and defining $B$ to be the set $\{ b'_{n'} | n' \in U \}$ where $b'_{n'} = b_{n'} \land b(n')$ with $b_{n'}$ a boolean expression satisfying

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

Again we check that $b \land b_1 \rightarrow \sqrt{B}$, i.e. $\rho \vDash b \land b_1$ implies $\rho \vDash b'_{n'}$ for some $n' \in U$. Assuming $\rho \vDash b \land b_1$ we have $(m_n, n_n) \in R^v$ and $\rho \vDash m_n \xrightarrow{c x y} m'_n$ by Proposition 4.2. So there exists $\rho \vDash u_{n} \xrightarrow{c y} n'_n$ such that $(m'_{n[x \mapsto z]}, n'_{n[y \mapsto z]}) \in R^v_{(v/z)}$ for all $v$ where $z$ is a fresh variable, i.e. $\rho \vDash b'_{n'}$. Applying Proposition 4.2, there exists $b_{n'}$ such that $n_n \xrightarrow{b_{n'} c y} n'_n$ and $\rho \vDash b(n')$; the latter implies $\rho \vDash b'_{n'}$.

Now for any $b_{n'} \in B$, there is a $n_n \xrightarrow{b_{n'}, c y} n'_n$, and by construction both $b_{n'} \rightarrow b(n')$ and $(m'_{n[x \mapsto z]}, n'_{n[y \mapsto z]}) \in S^b_{\rho}$ are immediate. $\square$

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

**Theorem 4.5.** $t \simeq^b_L u$ if and only if $\rho \vDash t \sim^b_L u$ for every $\rho$ such that $\rho \vDash b$.

In particular, it follows that for closed terms $\rho \vDash t \sim^b_L u$ if and only $t \simeq^b_L u$ 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 Section 5 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 $t \simeq^b_L u$.

### 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 $\simeq^b_L$. For convenience we will only consider transition graphs which have a tree structure; this makes the algorithm easier to describe and
analyse and since in principle every finite graph can be expanded out into an equivalent finite tree this is not too severe as a restriction.

Here we describe an algorithm which given two terms $t, u$, obtained from underlying finite symbolic trees, calculates a boolean $b$ such that $t \simeq_L^b u$. This is trivial in general since $t \simeq_L^1 u$ 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 $mgb_b(t, u)$, if $t \simeq_L mgb_b(t, u)$ and whenever $t \simeq_L^b u$ then $b \rightarrow mgb_L(t, u)$.

The algorithm for computing late symbolic bisimulation is shown in Fig. 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 $mgb_b(t, u)$ and in addition exhibits a finite representation, in terms of a table, of a symbolic late bisimulation equivalence which witnesses the fact that $t \simeq_L mgb(t, u)$. 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 finite function $T: \langle F, S \rangle \rightarrow BExp$ and we use $\sqcup$ to denote

\[
bisim(t, u) = close(t, u, true, \emptyset)
\]

\[
close(t, u, b, W) =
\]

if $(t, u) \in W$ then $(true, \emptyset)$
else let $(M, T) = match(\gamma, t, u, b, W)$

for $\gamma \in \{a, c!, c? \mid a \in NAct(t, u), c \in Chan(t, u)\}$

in $(\forall M, \sqcup T, \sqcup \{ (t, u) \rightarrow \{ b \land \gamma M \} \})$

\[
match(a, t, u, b, W) =
\]

let $(M, T) = close(t, u, b \land \gamma b \land b', \{(t, u)\} \cup W)$

for $t \overset{b, \gamma}{\rightarrow_L} t, u \overset{b', \gamma}{\rightarrow_L} u$

in $(\forall (b_i \rightarrow \gamma(b_j' \land M_i)) \land (\forall (b_j' \rightarrow \gamma(b_i \land M_j)), \sqcup T_i))$

\[
match(c!, t, u, b, W) =
\]

let $(M, T) = close(t, u, b \land \gamma b \land b', e_i = e_i', \{(t, u)\} \cup W)$

for $t \overset{b, \gamma}{\rightarrow_L} t, u \overset{b', \gamma}{\rightarrow_L} u$

in $(\forall (b_i \rightarrow e_i \land e_i = e_i', \gamma M_i)) \land (\forall (b_i' \rightarrow e_i \land e_i = e_i', \gamma M_i)), \sqcup T_i))$

\[
match(c?, t, u, b, W) =
\]

let $(M, T) = let z = new(fv(t) \cup fv(u))$

$(M', T') = close(t[x \leftarrow z], u[y \leftarrow z], b \land \gamma b \land b', \{(t, u)\} \cup W)$

for $t \overset{b, \gamma}{\rightarrow_L} t, u \overset{b', \gamma}{\rightarrow_L} u$

in $(\forall z. M', T')$

in $(\forall (b_i \rightarrow \gamma(b_j' \land M_i)) \land (\forall (b_j' \rightarrow \gamma(b_i \land M_j)), \sqcup T_i))$

Fig. 9. The algorithm for computing late symbolic bisimulation.
the "disjoint union" of tables, i.e. \( T \sqcup T' \) is defined only if \( \text{dom}(T) \cap \text{dom}(T') = \emptyset \) and in this case

\[
(T \sqcup T')(t, u) = T(t, u) \text{ if } (t, u) \in \text{dom}(T)
\]

\[
T'(t, u) \text{ if } (t, u) \in \text{dom}(T').
\]

We also write \( b \in T(t, u) \) to mean \( b \rightarrow b' \) for some \( b' \in T(t, u) \).

The procedure \texttt{close} has four parameters, \( t \) and \( u \), and current terms being compared, \( b \) a boolean expression which represents the constraints accumulated by previous calls to \texttt{close} and inherited by the current call, and finally \( W \) a set of pairs of terms which have already been visited; each pair of terms will be visited at most once by the procedure and therefore if we can show that the number of terms generated by the procedure is finite then we know that it will always halt. The procedure \texttt{close} uses the procedure \texttt{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 a boolean \( M \) and a table \( T \), the values returned from the call to \texttt{close}. It should be noted that in the procedure \texttt{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 \texttt{bisim} can be considered to be of the form \( \forall z. M \) where \( M \) is quantifier free, and an evaluation \( \rho \) satisfies this expression if for all \( v \in V \)

\[
\rho[v/z] = M[v/z].
\]

We first show that the procedure \texttt{bisim} always halts. Since it calls the procedure \texttt{close} we need to establish that this always halts. Let us assume that it works on a pair of finite symbolic trees \( G_1 \) and \( G_2 \). So in a call to \texttt{close}(\( t, u, b, W \)) \( t \) is a term over \( G_1 \), i.e. \( t \) has the form \( m \), where \( m \) is a node from \( G_1 \), \( u \) is a term over \( G_2 \) and the set \( W \) consists of a set of pairs of terms \( (r_1, r_2) \) where \( r_i \) is a term over \( G_i \). As the procedure runs new terms are added to \( W \) but it turns out that because of the way new terms are generated, in calls to \texttt{match} with parameter \( c \), only a finite number of terms over \( G_i \) can ever be generated.

**Theorem 5.1.** If \( t \) and \( u \) are any terms over finite symbols transition trees of the form \( n_0, m_0 \), respectively, then \texttt{bisim}(\( t, u \)) halts.

**Proof.** Let \( V_n \) denote the set of variables \( \{x_0, x_1, \ldots, x_n\} \) and let \( k \) be such that \( V_k \) includes all the free variables which occur in the underlying trees \( G_1 \) and \( G_2 \). Let \( l \) be such that \( 2k < l \) and let \( RT \) consists of all terms of the form \( m \), where \( m \) is a node from \( G_1 \) or \( G_2 \) and \( \sigma \) is a substitution whose domain is contained in \( V_k \) and whose range is contained in \( V_l \). Then

1. \( RT \) is finite
2. if \( W \) only contains pairs of nodes from \( RT \) and \( t, u \) are in \( RT \) when \texttt{close}(\( t, u, b, W \)) is executed then all subsequent calls to \texttt{close} are of the form \texttt{close}(\( t', u', b, W' \)) where again \( t', u' \in RT \) and \( W' \subseteq RT \times RT \). This follows because for any term \( r \) in \( RT \),
\(|f_\nu(r)| \leq k\) and so any new variable generated by the match procedure is guaranteed to be \(V_1\).

These two properties guarantee that \(\text{bisim}(t, u)\) halts since with each call to the procedure \(\text{close}\) the size of the parameter \(W\) increases.  \(\Box\)

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

**Theorem 5.2** (Soundness of late algorithm). If \(\text{bisim}(t, u) = (M, T)\) then \(T(t, u) = \{M\}\) and \(t \approx^M_L u\).

**Theorem 5.3.** (Completeness of late algorithm). If \(t \approx^H_L u\) and \(\text{bisim}(t, u) = (M, T)\), then \(b \rightarrow M\).

The proofs of these two theorems use some auxiliary notions and therefore 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 some examples.

The example shown in Fig. 7 can be written in CCS syntax as

\[
P = c?x.((a.(\text{IF} x=0 \text{ THEN } f.\text{NIL ELSE } g.\text{NIL}))+\text{a.}\text{h.NIL})
\]

\[
Q = c?x.(a.\text{IF} x=0 \text{ THEN } f.\text{NIL ELSE } h.\text{NIL}) + \text{a.}(\text{IF} x=0 \text{ THEN } h.\text{NIL ELSE } g.\text{NIL})
\]

Running the algorithm on it produces:

The reduced LATE characteristic formula is

\[\text{true}\]

with the bisimulation table:

- \(L_1 R_1: \text{true}\)
- \(L_2 R_2: \text{true}\)
- \(L_3 R_3: v_1 = 0\)
- \(L_3 R_4: \text{not}(v_1 = 0)\)
- \(L_4 R_3: \text{not}(v_1 = 0)\)
- \(L_4 R_4: v_1 = 0\)

where

- \(L_1 = c?x.(a.\text{IF} x=0 \text{ THEN } f.\text{NIL ELSE } g.\text{NIL} + a.\text{h.NIL})\)
- \(L_2 = a.\text{IF} x=0 \text{ THEN } f.\text{NIL ELSE } g.\text{NIL} + a.\text{h.NIL}\)
- \(L_3 = \text{IF} x=0 \text{ THEN } f.\text{NIL ELSE } g.\text{NIL}\)
- \(L_4 = h.\text{NIL}\)
- \(R_1 = c?x.(a.\text{IF} x=0 \text{ THEN } f.\text{NIL ELSE } h.\text{NIL} + a.\text{IF} x=0 \text{ THEN } h.\text{NIL ELSE } g.\text{NIL})\)
\( R_2 = \text{a. IF } x = 0 \text{ THEN } f. \text{NIL ELSE } h. \text{NIL} + \text{a. IF } x = 0 \text{ THEN } h. \text{NIL ELSE } g. \text{NIL} \)

\( R_3 = \text{IF } x = 0 \text{ THEN } f. \text{NIL ELSE } h. \text{NIL} \)

\( R_4 = \text{IF } x = 0 \text{ THEN } h. \text{NIL ELSE } g. \text{NIL} \)

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

\( P_1 = c?x. (c! \text{ABS}(x). P_1 + c! (\neg \text{ABS}(x)). P_1) \)

\( Q_1 = c?x. (c! x. Q_1 + c! \text{ABS}(x). Q_1 + c! (\neg \text{ABS}(x)). Q_1) \)

where \( \text{ABS}(x) \) is the absolute value of \( x \), the algorithm generates a non-trivial bisimulation condition for \( P_1 \) and \( Q_1 \):

The reduced LATE characteristic formula is

\[ A_\emptyset(v_1), \text{ABS}(v_1) = v_1 \text{ or } \neg \text{ABS}(v_1) = v_1 \]

with the bisimulation table:

\[ L_1 R_1: A_\emptyset(v_1), \text{ABS}(v_1) = v_1 \text{ or } \neg \text{ABS}(v_1) = v_1 \]

\[ L_2 R_2: \text{ABS}(v_1) = v_1 \text{ or } \neg \text{ABS}(v_1) = v_1 \]

where

\[ L_1 = c?x. (c! \text{ABS}(x). P_1 + c! (\neg \text{ABS}(x)). P_1) \]

\[ L_2 = c! \text{ABS}(x). P_1 + c! (\neg \text{ABS}(x)). P_1 \]

\[ R_1 = c! x. (c! x. Q_1 + c! \text{ABS}(x). Q_1 + c! (\neg \text{ABS}(x)). Q_1) \]

\[ R_2 = c! x. Q_1 + c! \text{ABS}(x). Q_1 + c! (\neg \text{ABS}(x)). Q_1 \]

As another example consider two slightly different versions of a memory cell:

\[ M(x) = r!x. M(x) + w?y. M(y) \]

\[ N(x) = r!x. N(x) + w?y. N_1(x, y) \]

\[ N_1(x, y) = [x = x] r!y. N_1(x, y) + w?y. N(y) \]

where \([x = x]\) represents a boolean guard (which in this case is always true). The algorithm returns with

The reduced LATE characteristic formula is

\[ A_\emptyset(v_1, v_2, v_3), \text{true} \]

with bisimulation table:

\[ L_1 \{v_1/v_1\} R_1 \{v_1/v_1\} = A_\emptyset(v_1, v_2, v_3), \text{true} \]

\[ L_2 \{v_1/v_2\} R_2 \{v_1/v_2\} = A_\emptyset(v_1, v_2, v_3), \text{true} \]

\[ L_1 \{v_2/v_2\} R_1 \{v_2/v_2\} = A_\emptyset(v_1, v_3, v_1), \text{true} \]

\[ L_2 \{v_2/v_2\} R_2 \{v_2/v_2\} = (A_\emptyset(v_1, v_2, v_3, \text{true})) \text{ or } (A_\emptyset(v_1, \text{true})) \]

\[ L_1 \{v_1/v_1\} R_3 \{v_2/v_2, v_1/v_1\} = (A_\emptyset(v_1, v_2, v_3, \text{true})) \text{ or } (A_\emptyset(v_1, \text{true})) \]
where

\[ \begin{align*}
L_0\{ \} &= M(x) \\
L_1\{v_1/y\} &= M(v_1) \\
L_2\{v_2/y\} &= M(v_2) \\
R_0\{ \} &= N(x) \\
R_1\{v_1/y\} &= N_1(x, v_1) \\
R_4\{v_3/y\} &= N(v_3) \\
R_5\{v_1/y\} &= N_1(v_3, v_1) \\
R_2\{v_2/y\} &= N(v_2) \\
R_3\{v_1/y\} &= N_1(v_2, v_1)
\end{align*} \]

A slightly more artificial version of these memory cells is given by the definition:

\[ \begin{align*}
M(x) &= r!x \cdot M(x) + r!\text{ABS}(x) \cdot M(x) + r!\text{NEG}(\text{ABS}(x)) \cdot M(x) + w?y \cdot M(y) \\
N(u) &= r!\text{ABS}(u) \cdot N(u) + r!\text{NEG}(\text{ABS}(u)) \cdot N(u) + w?y \cdot N(y)
\end{align*} \]

In this case the algorithm returns with a more complicated formula which nevertheless can immediately be seen to be true:

The reduced LATE characteristic formula is

\[ \begin{align*}
(A_\_\{(v_1), (A_\_\{(v_2), (A_\_\{(v_1), true)) \\
\quad \text{and } (v_2 = -\text{NEG}(\text{ABS}(v_2)) \text{ or } v_2 = \text{ABS}(v_2)) \\
\quad \text{and } (v_1 = -\text{NEG}(\text{ABS}(v_1)) \text{ or } v_1 = \text{ABS}(v_1)) \\
\quad \text{and } (x = -\text{NEG}(\text{ABS}(x)) \text{ or } x = \text{ABS}(x))
\end{align*} \]

with bisimulation table:

\[ \begin{align*}
L_0\{ \} \quad &R_0\{ \}: (A_\_\{(v_1), (A_\_\{(v_2), (A_\_\{(v_1), true)) \\
\quad \text{and } (v_2 = -\text{NEG}(\text{ABS}(v_2)) \text{ or } v_2 = \text{ABS}(v_2)) \\
\quad \text{and } (v_1 = -\text{NEG}(\text{ABS}(v_1)) \text{ or } v_1 = \text{ABS}(v_1)) \\
\quad \text{and } (x = -\text{NEG}(\text{ABS}(x)) \text{ or } x = \text{ABS}(x))
\end{align*} \]

\[ \begin{align*}
L_1\{v_1/v_1\} \quad &R_1\{v_1/v_1\}: (A_\_\{(v_2), (A_\_\{(v_1), true)) \\
\quad \text{and } (v_2 = -\text{NEG}(\text{ABS}(v_2)) \text{ or } v_2 = \text{ABS}(v_2)) \\
\quad \text{and } (v_1 = -\text{NEG}(\text{ABS}(v_1)) \text{ or } v_1 = \text{ABS}(v_1))
\end{align*} \]

\[ \begin{align*}
L_2\{v_2/v_2\} \quad &R_2\{v_2/v_2\}: (A_\_\{(v_1), true)) \\
\quad \text{and } (v_2 = -\text{NEG}(\text{ABS}(v_2)) \text{ or } v_2 = \text{ABS}(v_2))
\end{align*} \]

where

\[ \begin{align*}
L_0\{ \} &= M(x) \\
L_1\{v_1/y\} &= M(v_1) \\
L_2\{v_2/y\} &= M(v_2) \\
R_0\{ \} &= N(x) \\
R_1\{v_1/y\} &= N(v_1) \\
R_2\{v_2/y\} &= N(v_2)
\end{align*} \]
A conventional transition graph can be viewed as a degenerated symbolic graph where all actions are neutral actions and all boolean 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 bisimulations. 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 Fig. 6 to:

\[ m \xrightarrow{b,c,x} n \implies \rho \models m \xrightarrow{c,y} n_{[x \mapsto y]} \]

provided \( \rho \models b \sigma, y \notin \text{fv}(m) \)

This rule allows changing bound variables in input actions while inferring 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 \( R = \{ R^\rho \subseteq \langle T, T \rangle \mid \rho \in \text{Eval} \} \). Then \( \mathcal{E} \mathcal{B}(R) \) is the Eval-indexed family of symmetric relations defined by

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

Definition 6.1 (Early bisimulations). \( R \) is an early bisimulation if \( R^\rho \subseteq \mathcal{E} \mathcal{B}(R)^\rho \) for each \( \rho \).

We can now adapt the notation developed for the late case to this new setting. We write \( \rho \models t \sim_{\text{E}} u \) if there is an early bisimulation \( R \) such that \( (t, u) \in R^\rho \) and as before the standard theory applies; \( \sim_{\text{E}} \), defined as \( t \sim_{\text{E}} u \) if \( \rho \models t \sim_{\text{E}} u \) is the \( \rho \)th component of the maximal fixpoint of the functional \( \mathcal{E} \mathcal{B} \).

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 [10].
Similarly, the early symbolic semantics may be obtained by changing the input rule in Fig. 8 to

\[ m \xrightarrow{b,c \cdot x} n \quad \text{implies} \quad m \xrightarrow{b \in \cdot \cdot \cdot \cdot \cdot y} n_{\pi[x \mapsto y]} \]

provided \( y \notin \text{fv} (m_e) \).

(The \( \to_L \) arrows in the other rules are changed to \( \to_E \) as well.) To define early symbolic bisimulation let \( S = \{ S^b | b \in B\text{Exp} \} \) be a parameterised family of relations over terms. Then \( \mathcal{E}B\mathcal{B} (S) \) is the \( B\text{Exp} \)-indexed family of symmetric relations defined by

\( (t, u) \in \mathcal{E}B\mathcal{B} (S)^b \) if \( t \xrightarrow{b, c \cdot x} t' \) where \( b v (x) \) is a fresh variable, then there is a collection of booleans \( B \) such that \( b \land b_1 \to \lor B \) and for each \( b' \in B \) there exists a \( u \xrightarrow{b_1, z} v' \) such that \( b' \to b_2 \) and

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

It is important to note that the set of booleans \( B \) may contain occurrences of the new variable \( b v (x) \).

**Definition 6.2** (Early symbolic bisimulations). \( S \) is an early symbolic bisimulation if

\( S^b \subseteq \mathcal{E}B\mathcal{B} (S) \).

Again adapting the notation already developed we write \( t \simeq_E^b u \) if there is a symbolic early bisimulation \( S \) such that \( (t, u) \in S^b \) and as usual the standard theory implies that \( \{ \simeq_E^b | b \in B\text{Exp} \} \) is the maximal symbolic early bisimulation and that each \( \simeq_E^b \) 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) \( \rho \models t \xrightarrow{c \cdot x} t' \) if and only if \( t \xrightarrow{b, c \cdot x} t' \) for some \( b \) such that \( \rho \models b \).

(2) \( \rho \models t \xrightarrow{c \cdot e} t' \) if and only if \( t \xrightarrow{b, c \cdot e} t' \) for some \( b \) and \( e \) such that \( \rho \models b \) and \( \rho (e) = v \).

(3) For each \( a \in N\text{Act} \), \( \rho \models t \xrightarrow{a} t' \) if and only if \( t \xrightarrow{b, a} t' \) for some \( b \) and \( \rho \models b \).

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

\[ R_S^b = \{ (t, u) | \exists b \cdot \rho \models b \land (t, u) \in S^b \} \]

Conversely, given an early bisimulation \( R \) we can construct a symbolic bisimulation \( S_R \) by letting

\[ S_R^b = \{ (t, u) | \rho \models b \implies (t, u) \in R^b \} \].

Now we have the following proposition.
Proposition 6.4. (1) If $S$ is an early symbolic bisimulation then $R_S$ is an early bisimulation.
(2) If $R$ is an early bisimulation then $S_R$ is an early symbolic bisimulation.

Proof.
1. Let $(t, u) \in R^b_S$, 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 $\rho \models t \xrightarrow{c^2_k} t'$, where $z$ is a fresh variable. By Proposition 6.3 it follows that $t \xrightarrow{b_1, c^2_k} t'$ for some $b_1$ such that $\rho \models b_1$. Therefore, there exists a set of booleans $B$ such that $b \wedge b_1 \rightarrow \bigvee B$ and for each $b' \in B$ there is a $u \xrightarrow{b_2, c^2_k} u'$ such that $b' \rightarrow b_2$ and $(t', u') \in S^b$. Let $v \in V$. Since $z$ is a fresh variable $\rho[v/z] = b \wedge b_1$. Thus there must be $b' \in B$ such that $\rho[v/z] = b'$, and hence $\rho[v/z] = b_2$ for the $u \xrightarrow{b_2, c^2_k} u'$ associated with this $b'$. As $z \notin \text{fc}(b_2)$, $\rho \models b_2$. Now applying the same Proposition we have $\rho \models u \xrightarrow{c^2_k} u'$.

Since $(t', u') \in S^b$ and $\rho[v/z] = b'$ if follows that $(t', u') \in R^b_S$, as required.

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

Let $t \xrightarrow{b_1, c^2_k} t'$, where $z$ is a fresh variable. We must find a set of booleans $B$ such that $b \wedge b_1 \rightarrow \bigvee B$ and for each $b' \in B$ there must exists a move $u \xrightarrow{b_2, c^2_k} u'$ 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 $b$ such that $u \xrightarrow{b} u'$. Let $U$ be the set $\{u' | u \xrightarrow{b(w), c^2_k} u' \}$ and for each $u' \in U$ let $b_u$ be a boolean expression which satisfies

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

and let $b_u' = b_u \wedge b(u')$. Finally let $B = \{b_u' | u' \in U \}$.

We first check that $b \wedge b_1 \rightarrow \bigvee B$, 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 $(t, u) \in R^b$ and since $\rho \models b_1$ it follows from Proposition 6.3 that $\rho \models t \xrightarrow{c^2_k} t'$. So for every $v \in V$ there exists some $u_v$ such that $\rho \models u \xrightarrow{c^2_k} u_v$ and $(t', u') \in R^b$. So the required $u'$ is $u_{\rho(z)}$ since in this case $(t', u') \in R^b$ and again by Proposition 6.3 $u \xrightarrow{b(w), c^2_k} u'$ with $\rho \models b(u')$; therefore $\rho \models b_u'$.

Also by the construction for any $b_u \in B$ there exists $u \xrightarrow{b(w), c^2_k} u'$ where $b \rightarrow b(u')$ and $(t', u') \in S^b_R$. ☐

As a direct corollary we have the following theorem.

Theorem 6.5. $\rho \models t \sim u$ if and only if $t \approx_b u$ for every boolean $b$ such that $\rho \models b$. ☐

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

\[
\text{match}(c?, t, u, b, W) = \\
\text{let } x = \text{new}(fv(t) \cup fv(u)) \\
\quad (M_{ij}, T_{ij} = \text{close}(t_i, u_j, b \land b_i \land b_j, \{(t, u)\} \cup W) \\
\quad \quad \quad \text{for } t \xrightarrow{b_i \cdot c^x} E t_i, \quad u \xrightarrow{b_j \cdot c^x} E u_j) \\
\quad \in \left( \forall x . \bigwedge_i \left( b_i \rightarrow \bigvee_j (b_j \land M_{ij}) \right) \land \bigwedge_j \left( b_j \rightarrow \bigvee_i (b_i \land M_{ij}) \right), \bigcup_{ij} T_{ij} \right).
\]

Once more we can check that the modified procedure always halts. Moreover, it is sound and complete with respect to early symbolic bisimulation:

**Theorem 6.6.** (1) If \( \text{bisim}(t, u) = (M, T) \) then \( T(t, u) = \{M\} \) and \( t \simeq^M u \).

(2) If \( t \simeq^M u \) and \( \text{bisim}(t, u) = (M, T) \) 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

\[
P_1 = c?x . (\text{IF EVEN}(x) \text{ THEN } R_1 \text{ ELSE } R_2) + c?x . R_3 \\
P_2 = c?x . (\text{IF EVEN}(x) \text{ THEN } R_1 \text{ ELSE } R_3) + c?x . \text{IF EVEN}(x) \text{ THEN } R_3 \text{ ELSE } R_2 \\
R_1 = \text{NIL} \\
R_2 = \text{tau} . \text{NIL} \\
R_3 = \text{tau} . \text{tau} . \text{NIL}
\]

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

The reduced **EARLY** characteristic formula is

\[
\text{true}
\]

with the bisimulation table:

\[
\begin{align*}
L_1 & \rightarrow R_1: \text{true} \\
L_2 & \rightarrow R_2: \text{EVEN}(v_1) \\
L_2 & \rightarrow R_3: \text{not}(\text{EVEN}(v_1)) \\
L_3 & \rightarrow R_4: \text{false} \\
L_4 & \rightarrow R_2: \text{not}(\text{EVEN}(v_1)) \\
L_4 & \rightarrow R_3: \text{EVEN}(v_1) \\
L_5 & \rightarrow R_4: \text{true} \\
L_5 & \rightarrow R_5: \text{true}
\end{align*}
\]
where

\[ \begin{align*}
L_1 &= c?x. \text{IF EVEN}(x) \text{ THEN } R_1 \text{ ELSE } R_2 + c?x. R_3 \\
L_2 &= \text{IF EVEN}(x) \text{ THEN } R_1 \text{ ELSE } R_2 \\
L_3 &= \text{NIL} \\
L_4 &= \tau . \tau . \text{NIL} \\
L_5 &= \tau . \text{NIL} \\
R_1 &= c?x. \text{IF EVEN}(x) \text{ THEN } R_1 \text{ ELSE } R_3 + c?x. \text{IF EVEN}(x) \text{ THEN } R_3 \text{ ELSE } R_2 \\
R_2 &= \text{IF EVEN}(x) \text{ THEN } R_1 \text{ ELSE } R_3 \\
R_3 &= \text{IF EVEN}(x) \text{ THEN } R_3 \text{ ELSE } R_2 \\
R_4 &= \tau . \text{NIL} \\
R_5 &= \text{NIL}
\end{align*} \]

But the boolean \( M \) returned by the late algorithm is

\[ A_\emptyset (v_1, v_2, v_3, v_4), \\
(\text{EVEN}(v_1) \text{ or } \neg(\text{EVEN}(v_2))) \text{ and } ((\neg(\text{EVEN}(v_3))) \text{ or } \text{EVEN}(v_4)) \text{ and } ((\text{EVEN}(v_1) \text{ or } \neg(\text{EVEN}(v_3))) \text{ and } ((\neg(\text{EVEN}(v_2))) \text{ or } \text{EVEN}(v_4))) \]

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 finite symbolic transition trees. The algorithms 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. 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.

The standard approach to value-passing in process algebras is to interpret the process \( c?x. p \) as the non-deterministic sum \( \sum_{v \in V} c?v. p[v/t] \). This is the approach suggested in [10] and pursued, for example, in [3, 13]. 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. Previous work which attempts to
generalise bisimulation checking to value-passing languages is reported in [8]. There
they consider a language similar to our extension of CCS which allows input and
output guards of the form $e?x$, $e!e$ 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. However, our
algorithms can also be applied to this class of data independent processes.

More recently in [12] a symbolic model for value-passing languages called *parameterised graphs*
is introduced. This is a slight generalisation of our symbolic graphs in
that the nodes can have associated with them arbitrary value expressions rather than,
as with symbolic graphs, simply variables. However, this generalised setting has not
yet been fully exploited as [12] only contains separate algorithms for calculating each
of the iterative approximations to strong bisimulation, $\sim_n$ for each $n \geq 0$.

There is considerable similarity between our use of symbolic semantics and the
approach taken in papers such as [1, 7] to handle timed processes. If the time domain
used is not discrete then such processes also give rise to infinite branching or even
uncountable labelled transition systems. Checking the equivalence of two timed processes
therefore requires some abstraction from this level of detail and one standard approach is
to introduce transitions labelled with time intervals rather than time instants. The
relationship with our symbolic semantics is still not clear but it deserves further study.

For a completely different approach to checking properties of infinite state pro-
cesses the reader is referred to [2, 14] where techniques are developed for checking
that such processes have properties expressed in temporal and modal logics.

**Acknowledgment**

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 and Zvi Schreiber for their detailed and constructive criticisms and the
financial support of the SERC grants GR/H 16537 and GR/H 13932.

**Appendix 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 Sections 3 and 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
Fig. 10. The relations $\rightarrow$ 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:

1. \( p \sim q \) implies there exists \( q \overset{c \in \text{Chan}}{\rightarrow} \lambda x.p \) such that for all \( v \in V, (p'[v/x], q'[v/y]) \in R \)

2. for any other actions \( a, p \overset{a}{\rightarrow} p' \) implies there exists \( q \overset{a}{\rightarrow} q' \) such that \( (p', q') \overset{\alpha}{\rightarrow} R \)

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 Fig. 4. For convenience, we will denote the term \( t_0 \) 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 \((t, \lambda(t))\). We then have the theorem.

**Theorem A.1.** \( p \sim q \) iff \( \rho \models p \sim_L q \) (i.e. \( \rho \models p_0 \sim_L q_0 \)) for every evaluation \( \rho \).

This follows from two more general results.

**Proposition A.2.** Let \( S^e = \{ (t_\sigma, u_\eta) | t \rho \sigma \sim u \rho \eta \} \). Then \( \{ S^e \} \) 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 \( \rho \).

**Proof.** The pair \( p_0, q_0 \) are in any \( S^e \) because \( \rho v(p) = \emptyset \) and \( p \rho \sigma = p \) for closed terms \( p \). \( \square \)
Proposition A.4. Let $R$ be the set of all pairs $(t\rho, u\eta)$ such that $\rho = t_\tau \sim_L u_\eta$. Then $R$ is a CCS-late bisimulation.

Corollary A.5. $\rho \vdash p \sim_L q$ for all $\rho$ implies $p \sim q$.

Proof. Since $\rho \vdash p \sim_L q_0$, it follows that $(p, q) \in R$. □

The proofs of the two above propositions depend on relating the arrows which underly the two different definitions of equivalence. The relationship is captured in Lemmas A.5–A.8.

Lemma A.6. (1) If $\rho \vdash b$ then $t \xrightarrow{b,c\eta} t'$ implies $t \xrightarrow{c\eta} \lambda y.r$ for some $r$ such that for all $u \xrightarrow{r[v]/y} t'[v/x] \rho$.

(2) If $t \xrightarrow{c\eta} \lambda y.r$ then there exist $b$ and $t'$ such that $\rho \vdash b$ and $t \xrightarrow{b,c\eta} t'$ for some $x$ with $r[v]/y \equiv t'[v/x] \rho$ for all $v$.

Proof. (1) By induction on why $t \xrightarrow{b,c\eta} t'$.

- $c?y.t \xrightarrow{true,c\eta} t[x/y]$ where $x = \text{new}(fv(c?y.t))$. Then $(c?y.t) \rho \xrightarrow{c\eta} (\lambda y.t) \rho$ and the requirement is satisfied.

- $t[u] \xrightarrow{b,c\eta} t'[x/z] u$ because $t \xrightarrow{b,c\eta} t'$. By induction $t \rho \xrightarrow{c\eta} \lambda y.r$ for some $r$ such that for all $v \xrightarrow{r[v]/y} t'[v/x] \rho$. Now $(t[u]) \rho \xrightarrow{c\eta} \lambda y.(r[up] \rho) \rho \equiv t[v/x] \rho \rho \rho \equiv t'[v/x] \rho \rho$. 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 $r[v/x] \rho \equiv t'[x/z][v/x] \rho \rho \equiv t'[v/x] \rho \rho$ because $x \notin fv(u)$.

- $(b \rightarrow t, u) \xrightarrow{b' \rightarrow c\eta} t'$ because $t \xrightarrow{b,c\eta} t'$.

(2) By induction on why $t \xrightarrow{c\eta} \lambda y.r$.

- $(c?y.t) \rho \xrightarrow{c\eta} (\lambda y.t) \rho$. In this case $r[v]/y$ is $t[v/x]$ and obviously $c?y.t \xrightarrow{true,c\eta} t[x/y]$ where $x = \text{new}(fv(c?y.t))$. Since $x \notin fv(t) \rightarrow \{y\}$ it follows that $t[v/x] \rho \equiv t[x/y][v/x] \rho$ for all $v$.

- $(t[u]) \rho \xrightarrow{c\eta} \lambda y.(r[up]) \rho$ because $t \rho \xrightarrow{c\eta} \lambda y.r$. By induction there exist $b$ and $t'$, $\rho \vdash b$ and $t \xrightarrow{b,c\eta} t'$ s.t. $r[v]/y \equiv t'[v/x] \rho$ for all $v$. So $t[u] \xrightarrow{b,c\eta} t'[x/z] u$ and $(r[up]) \rho \equiv r[v/x] \rho \rho \equiv t'[x/z][v/x] \rho \rho \equiv t'[x/z][v/x] \rho \rho$ because $x \notin fv(u)$.

- $(b \rightarrow t, u) \rho \xrightarrow{c\eta} \lambda y.r$ because $\rho \vdash b'$ and $t \rho \xrightarrow{c\eta} \lambda y.r$. By induction there exist $b$ and $t'$, $\rho \vdash b$ and $t \xrightarrow{b,c\eta} t'$ s.t. $r[v]/y \equiv t'[v/x] \rho$ for all $v$. So $(b \rightarrow t, u) \rho \xrightarrow{b',b,c\eta} t'$ and $\rho \vdash b \land b'$.
\[ P(e) \rho \xrightarrow{c^y} \lambda y. r \text{ because } t[e/\rho/x] \xrightarrow{c^y} \lambda y. r \text{ where } P(x) \equiv t \text{ and } f(v(t)) \equiv \{ x \}. \text{ In this case } t[e/\rho/x] \rho = t[e/\rho/x]. \text{ By induction there exist } b \text{ and } t', \rho \equiv b \text{ and } t[e/\rho/x] \xrightarrow{b.c^x} t' \text{ s.t. } r[v/y] \equiv t'[v/x] \rho \text{ for all } v. \text{ Therefore } P(e) \xrightarrow{b.c^x} t'. \]

- The remaining cases are similar. \[ \Box \]

**Lemma A.7.** (1) If \( \rho(b) = \text{true} \) and \( \rho(e) = v \) then \( t \xrightarrow{b.c^x} t' \) implies there exists \( r, t \rightarrow r' \).

(2) If \( t \rightarrow r \) then there exist \( b, t' \) and \( e \) s.t. \( \rho(b) = \text{true}, \rho(e) = v \), and \( t \xrightarrow{b.c^x} t' \) with \( r \equiv t' \).

**Proof.** By induction on the derivation of \( t \xrightarrow{b.c^x} t' \) and \( t \rightarrow r \), respectively. \[ \Box \]

**Lemma A.8.** (1) If \( \rho(b) = \text{true} \) then \( t \xrightarrow{b.t} t' \) implies there exists \( r, t \rightarrow r' \).

(2) If \( t \rightarrow r \) then there exist \( b, t' \) s.t. \( \rho(b) = \text{true} \) and \( t \xrightarrow{b.t} t' \) with \( r \equiv t' \).

**Proof.** By induction on the derivation of \( t \xrightarrow{b.t} t' \) and \( t \rightarrow r \), respectively. \[ \Box \]

**Proof of Proposition A.2.** We have to prove that \( \{ S^n \} \) is a late bisimulation.

Suppose \((s_0, u_0) \in S^n\)

1. First we examine the input rules. By the definition of the late operational semantics (Fig. 6) we have \( \rho \equiv t_\sigma \xrightarrow{c^x} t'e_{[x \mapsto x]} \), where \( x' = \text{new}(f(v(t))) \) because \( t \xrightarrow{b.c^x} t' \) for some \( b \) with \( \rho \equiv b\sigma \). Now from Lemma A.6

\[ \rho \equiv u_\rho \eta \rightarrow r \text{ for some } r \text{ s.t. for all } v, r[v/y] \equiv t'[v/x] \rho \sigma. \]

Since \( t \rho \sigma \sim u_\rho \eta \), we have

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

Again applying Lemma A.6 we get

\[ \rho \equiv u_\eta \xrightarrow{c^w} u'_\eta[w \mapsto w']. \]

where \( w' \) is \( \text{new}(f(v(u))) \) because \( u \xrightarrow{b.c^w} u' \) for some \( w \) s.t. \( \rho \eta \equiv b', \) i.e. \( \rho \equiv b' \eta \), and for all \( v, s[v/z] \equiv u'[v/w] \eta \). We need to show that for all \( v, \)

\[ (t_\sigma[x \mapsto x]', u_\rho[w \mapsto w'] \mid x' \mapsto z] \in S^n[v/z], \]

where \( z \) is a fresh variable, i.e.

\[ (t_\sigma[x \mapsto z], u'_\rho[w \mapsto z]) \in S^n[v/z]. \]

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 \( u'\rho[v/z] \eta[w \mapsto z] \equiv u'[v/w] \eta \).

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

\[
\frac{t \rho \sigma \xrightarrow{c\epsilon} r \equiv t' \rho \sigma.}
\]

Since \( t \rho \sigma \sim u \rho \eta \), we have

\[
\frac{u \rho \eta \xrightarrow{c\epsilon} r' \sim r.}
\]

Again by Lemma A.7

\[
\frac{u \xrightarrow{b',c\epsilon} u'}{s.t. \ \rho \eta \models b', \ \rho \eta(e') = v, \ and \ r' \equiv u' \rho \eta. \ So \ \rho \models u_n \xrightarrow{c\epsilon} u' \ and \ u' \rho \eta \equiv r' \sim r \equiv t' \rho \sigma, \ i.e. \ (t', u') \in S'}.\]

(3) The final case, \( \rho \models t_\sigma \xrightarrow{c} t'_\sigma \), 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 \( t \rho \sigma \xrightarrow{c\epsilon} \lambda x. r \). We must show that \( u \rho \eta \xrightarrow{c\epsilon} \lambda y. r' \) s.t. for all \( v, (r[v/x], r'[v/y]) \in R \). By Lemma A.6,

\[
\frac{t \xrightarrow{b,c\epsilon} t'}{s.t. \ \rho \models b \sigma \ and \ for \ all \ v, \ r[v/x] \equiv t'[v/x'] \rho \sigma. \ This \ in \ turn \ means}
\]

\[
\frac{\rho \models t_\sigma \xrightarrow{c\epsilon} t'_{\sigma[x' \mapsto x]},}{where \ x' \ is \ new(fv(t_\sigma)). \ Since \ \rho \models t_\sigma \sim L u_\eta \ we \ must \ have}
\]

\[
\frac{\rho \models u_\eta \xrightarrow{c\epsilon} u'_{\sigma[y' \mapsto y]}},{where \ y' \ is \ new(fv(u_\eta)), \ u \xrightarrow{b',c\epsilon} u' \ for \ some \ b' \ with \ \rho \models b' \eta \ and \ for \ all \ v,}
\]

\[
\frac{\rho[v/z] \models t'_{\sigma[x' \mapsto x'] [x' \mapsto z]} \sim L u'_{\eta[y' \mapsto y']} [y' \mapsto z]}{with \ z \ fresh. \ This \ in \ turn \ means}
\]

\[
\frac{\rho[v/z] \models t'_{\sigma[x' \mapsto z]} \sim L u'_{\eta[y' \mapsto z]}}{Applying \ Lemma \ A.6 \ we \ obtain}
\]

\[
\frac{u \rho \eta \xrightarrow{c\epsilon \ y} \lambda y. r'}{s.t. \ for \ all \ v \ r'[v/y] \equiv u'[v/y'] \rho \eta. \ Now}
\]

\[
\frac{r[v/x] \equiv t'[v/x'] \rho \sigma \equiv t' \rho[v/z] \sigma[x' \mapsto z]}
and
\[ r'[v/y] \equiv u'[v/y'] \rho \eta \equiv u'[v/z] \eta[y' \mapsto z]. \]

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

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 Fig. 11 where again the symmetric rules for + and | 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: if \((p, q) \in R\) then

\[ p \xrightarrow{a} p' \text{ implies there exists } q \xrightarrow{a} q' \text{ and } (p', q') \in R, \]

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 Fig. 4, we show that early concrete bisimulation (as defined in Section 6) coincides with \(\sim\):

\[ p \sim q \text{ iff } \rho \models p \overset{\sim_E}{=} q \text{ (i.e. } \rho \models p_0 \overset{\sim_E}{=} q_0) \text{ for every evaluation } \rho. \]

The proof follows the same pattern as before. Define

\[ S^\rho = \{ (t, u) \mid t \rho \sigma \sim u \rho \eta \} \]

and

\[ R = \{ (t \rho \sigma, u \rho \eta) \mid \rho \models t \sim_E u \}. \]

Then one can show that \(\{S^\rho\}\) is an early bisimulation and \(R\) is a CCS-early bisimulation, from which the result follows immediately.

\[ a.p \xrightarrow{a} p \]
\[ c?x.p \xrightarrow{c!x} p[v/x] \]
\[ p \xrightarrow{a} p' \]
\[ p \xrightarrow{a} p' \]
\[ p \xrightarrow{c!v} p', q \xrightarrow{c?v} q' \]
\[ p \xrightarrow{\alpha} p' \]
\[ l[l/x] \xrightarrow{\alpha} p' \]
\[ p \xrightarrow{\alpha} p', b \equiv true \]
\[ p \xrightarrow{\alpha} p', b \equiv false \]

\[ a \in \{ \tau \} \cup \{ c.v \mid c \in \text{Chan} \} \]
\[ c \in \text{Chan}, v \in V \]
\[ p + q \xrightarrow{a} p' \]
\[ p \mid q \xrightarrow{a} p' \mid q \]
\[ p \mid c \xrightarrow{a} p \mid c \]
\[ P(\tau) \xrightarrow{\alpha} p' \]
\[ P(\tau) \xrightarrow{\alpha} p' \text{ if } \alpha \text{ does not use the channel } c \]
\[ P(\tau) \xrightarrow{\alpha} p' \text{ if } \alpha \text{ is a declaration} \]
\[ b \rightarrow p, q \xrightarrow{\alpha} p' \]
\[ b \rightarrow q, p \xrightarrow{\alpha} p' \]

Fig. 11. Early operational semantics of CCS-closed terms.
The proof of these two results depends on relating the two different arrow relations:
1. $tp \xrightarrow{e \rho} r$ if and only if there exist $b$ and $t'$ such that $\rho \models b$ and $t \xrightarrow{b,c,x} t'$ for some $x$ with $r \equiv t' \rho[v/x]$.
2. $tp \xrightarrow{e \rho} r$ if and only if there exist $b, t'$ and $e$ s.t. $\rho(b) = true$, $\rho(e) = v$ and $t \xrightarrow{b,c,e} t'$ with $r \equiv t' \rho$.
3. $tp \xrightarrow{e \rho} r$ if and only if there exist $b, t'$ s.t. $\rho(b) = true$ and $t \xrightarrow{b,c,e} t'$ with $r \equiv t' \rho$.

Appendix 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 $t, u$ are terms over two finite symbolic transition trees and $bisim(t, u)$ returns the pair $(M, T)$ then $M = mb(t, u)$ and $T$ can be used to construct a symbolic bisimulation $S$ such that $(t, u) \in S^M$. So for the remainder of this section let us fix such a pair of trees, $G, H$.

Let $T_G$ and $T_H$ be the sets of terms over $G$ and $H$, respectively. We use $t, t', t''$, ... to range over $T_G$ and $u, u', u''$, ... to range over $T_H$. 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 terms in $W$ that have already been visited. The first series of definitions will formalise this idea.

We say a pair of guarded symbolic transitions, $a, a'$ are of the same type $\gamma \in \{a, c!, c? | a \in NAct, c \in Chan\}$, if
1. $a = a$ and $a' = a$ with $a \in NAct$,
2. $a = c?x$ and $a' = c?y$ with $c \in Chan$,
3. $a = c!e$ and $a' = c!e'$ with $c \in Chan$.

When $a, a'$ are of the same type we write $a = a'$ for $e = e'$ if $a = c!e$ and $a' = c!e'$, and true otherwise.

We use $BB(t_0, u_0, W, T)$ to mean the following condition is satisfied:
if $T(t, u) = b$ then whenever $t \xrightarrow{b_1, z} t'$ there is a set of booleans $B$ such that $b \land b_1 \rightarrow \bigvee B$ and for each $b' \in B$ there exists a $u \xrightarrow{b_2, z} u'$ such that
- $b' \rightarrow b_2$,
- $(t', u') \notin \{(t_0, u_0)\} \cup W$ implies $(t', u') \in dom(T)$, and
- (i) if $a = a$ then $a' = a$ and $(t', u') \in dom(T) \& (t', u') \notin W$ implies $b' \rightarrow T(t', u')$.
- (ii) if $a = c!e$ then $a' = c!e'$ with $b' \cdot c = c'$, and $(t', u') \in dom(T) \& (t', u') \notin W$ implies $b' \rightarrow T(t', u')$.
- (iii) if $a = c?x$ then $a' \equiv c?y$, and $(t'[x \mapsto z], u'[y \mapsto z]) \in dom(T)$ & $(t', u') \notin W$ implies $b' \rightarrow T(t'[x \mapsto z], u'[y \mapsto z])$ for some $z \notin fin(t, u)$, and similarly for $u$.  

Intuitively, $\mathcal{B}(t_0, u_0, W, T)$ means "$T$ is an approximation of a symbolic bisimulation for $t_0$ and $u_0$ with respect to $W". It will be a symbolic bisimulation if $W = \emptyset$ and $(t_0, u_0) \in \text{dom}(T)$. In that case $t_0 \sim_L^{(t_0, u_0)} u_0$.

Lemma B.1. If $\text{dom}(T_1) \cap \text{dom}(T_2) = \emptyset$ then $\mathcal{B}(t, u, W, T_1) \& \mathcal{B}(t, u, W, T_2)$ implies $\mathcal{B}(t, u, W, T_1 \cup T_2)$.

Proof. Straightforward from the definition of $\mathcal{B}$. \(\square\)

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

Definition B.2. Let $\text{CLOSE}(t, u, b, W, M, T)$ be true if the following conditions are satisfied:

(C1) $W \cap \text{dom}(T) = \emptyset$ and either $(t, u) \in W \& M = \text{true}$ or $(t, u) \in \text{dom}(T) \& T(t, u) = b \& M$.

(C2) $\mathcal{B}(t, u, W, T)$.

Let $\text{BISIM}(t, u, M, T) = \text{CLOSE}(t, u, \text{true}, \emptyset, M, T)$. \(\square\)

Definition B.3. For each $\gamma$ let $\text{MATCH}_{,\gamma}(t, u, b, W, M, T)$ be true if

(\(M, 1\)) $W \cap \text{dom}(T) = \emptyset$, $(t, u) \notin W$ and $(t, u) \notin \text{dom}(T)$.

(\(M, 2\)) $\mathcal{B}(t, u, W, T)$.

(\(M, 3\)) For any type $\gamma$ derivation $t \xrightarrow{b_1, \gamma} t'$ there is a set of booleans $B$ such that $b \& M \& b_1 \rightarrow \forall B$ and for each $b' \in B$ there exists a $u \xrightarrow{b_2, \gamma} u'$ such that $b' \rightarrow b_2$, $(t', u') \notin \{ (t, u) \} \cup W \Rightarrow (t', u') \in \text{dom}(T)$, and

- if $\alpha = a$ then $\alpha' = a$ and $(t', u') \in \text{dom}(T) \& (t', u') \notin W \Rightarrow b' \rightarrow T(t', u')$.
- if $\alpha \equiv c \! \& e$ then $\alpha' \equiv c \! \& e'$ with $b' \rightarrow e = e'$, and $(t', u') \in \text{dom}(T) \& (t', u') \notin W \Rightarrow b' \rightarrow T(t', u')$.
- if $\alpha \equiv c?x$ then $\alpha' \equiv c?y$, and $(t'[x \mapsto z], u'[y \mapsto z]) \in \text{dom}(T) \& (t', u') \notin W \Rightarrow b' \rightarrow T(t'[x \mapsto z], u'[y \mapsto z])$ for some $z \notin f(t, u)$. \(\square\)

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, implies that of close.

Proposition B.4. If $\text{MATCH}_{,\gamma}(t, u, b, W, M, T)$ for each type $\gamma$ then $\text{CLOSE}(t, u, b, W, M, T)$, where $M = \land, M$, and $T = \land, T, \cup \{ (t, u) \mapsto b \& M \}$. \(\square\)

Proof. The only non-trivial condition is C2, i.e. $\mathcal{B}(t, u, W, T)$. Since the graphs are recursive trees, by Lemma B.1 we only need to check $T(t, u)$ which is $b \& M$, and this case can established using $M, 3$. \(\square\)
We now show that appropriate instances of the verification condition of close imply those of match.

**Proposition B.5.** Suppose \((t, u) \notin W\). If

\[
\text{CLOSE}(t_i, u_j, b \land b_i \land b_j' \land z = x', \{(t, u)\} \cup W, M_{ij}, T_{ij})
\]

for all type \(\gamma\) derivations \((t \xleftarrow{b_i, a_i, L} t_i, u \xrightarrow{b_j, a_j, L} u_j)\) then

\[
\text{MATCH}((t, u, b, W, M, T)
\]

where \(M = \bigwedge_i (b_i \rightarrow \bigvee_j (b_j' \land M_{ij})) \land \bigwedge_j (b_j' \rightarrow \bigvee_i (b_i \land M_{ij}))\), \(T = \bigcup_{ij} T_{ij}\), and \(t_i, u_j, M_{ij}\) are as follows

\[
M_{ij}, t_i, u_j = \left\{ \begin{array}{ll}
\forall z M_{ij}, t_i[x \mapsto z], u_j[y \mapsto z] & \text{if } x \equiv c?x, \ x' \equiv c?y, \\
M_{ij}, t_i, u_j & \text{otherwise},
\end{array} \right.
\]

with \(z \notin \text{fv}(t, u)\).

**Proof.** \(M_{\cdot 1}\) is straightforward. To show \(M_{\cdot 2}\), i.e. \(\mathcal{B}(t, u, W, T)\), assume \(T(t', u') = b'\). As an example we consider the case \(\gamma \equiv a \in N.\text{Act}\). Since \(T = \bigcup_{ij} T_{ij}, T_{ij}(t', u') = b'\) for some \(i, j\). By (1) and C2 we have \(\mathcal{B}(t_i, u_j, \{(t, u)\} \cup W, T)\). So let \(t' \xleftarrow{b_i, a_i, L} t''\). Then there exists a set of booleans \(B\) such that \(b \land b_i \rightarrow \bigvee B\) and for each \(b'' \in B\) there exists an \(u' \xrightarrow{b_2, a_2, L} u''\) such that \(b'' \rightarrow b_{2^*}, (t'', u'') \notin \{(t_i, u_j)\} \cup \{(t, u)\} \cup W\) implies \((t'', u'') \notin \text{dom}(T_{ij})\), and \((t'', u'') \notin \text{dom}(T_{ij})\) \& \((t'', u'') \notin \{(t, u)\} \cup W\) implies \(b'' \rightarrow T(t'', u'')\).

We want to show

\[
(t'', u'') \notin \{(t, u)\} \cup W \implies (t'', u'') \notin \text{dom}(T)
\]

\[
(t'', u'') \in \text{dom}(T) \& (t'', u'') \notin W \implies b'' \rightarrow T(t'', u'').
\]

For the first statement, assume \((t'', u'') \notin \{(t, u)\} \cup W\). We only need to consider the case \((t'', u'') = (t_i, u_j)\) and \((t_i, u_j) \notin \{(t, u)\} \cup W\). In this case by C1 we have \((t_i, u_j) \notin \text{dom}(T_{ij}) \subseteq \text{dom}(T)\).

For the second statement, assume \((t'', u'') \notin \text{dom}(T) \& (t'', u'') \notin W\). Since \((t'', u'') \notin \text{dom}(T_{ij})\) and the graphs are recursive trees, \((t'', u'') \notin \text{dom}(T_{ij})\). By C1 we have \((\{(t, u)\} \cup W) \cup \text{dom}(T_{ij}) = \emptyset\), so \((t'', u'') \notin (t, u)\). Hence \((t'', u'') \notin \{(t, u)\} \cup W\). Therefore, \(b'' \rightarrow T(t'', u'') = T(t'', u'')\).

The final condition we must establish is \(M_{\cdot 3}\). As an example consider the move \(t \xleftarrow{b_i, a_i, L} t_i\). Set \(B = \{b \land b_i \land b_j' \land M_{ij} | u \xleftarrow{b_j, a_j, L} u_j\}\). Then \(b \land b \land M \rightarrow \bigvee\) and each \(b \land b_i \land b_j' \land M_{ij}\) in \(B\) has the move \(u \xrightarrow{b_j, a_j, L} u_j\) associated with it which satisfies \((t_i, u_j) \notin \{(t, u)\} \cup W\) implies \((t_i, u_j) \notin \text{dom}(T_{ij}) \subseteq \text{dom}(T)\). Moreover, if \((t_i, u_j) \notin \text{dom}(T) \& (t_i, u_j) \notin W\) then \((t_i, u_j) \notin (t, u)\) and therefore \((t_i, u_j) \notin \{(t, u)\} \cup W\). So by (1) and C1 \(T_{ij}(t_i, u_j) = b \land b_i \land b_j' \land M_{ij}\), therefore \(b \land b_i \land b_j' \land M_{ij}\) implies \(T_{ij}(t_i, u_j) = T(t_i, u_j)\). \(\square\)
Proof of Theorem 5.2. From $\text{bisim}(t,u) = (M,T)$ we have $\text{BISIM}(t,u,M,T)$, i.e. $\text{CLOSE}(t,u,\text{true}, M, T)$. $C1$ gives $(t,u) \subset \text{dom}(T)$ and $(T(t,u) = M$. Together with $C2$, namely $\mathcal{B}(t,u,\emptyset, T)$, this guarantees $T$ is a symbolic bisimulation for $t \sim_M u$. □

We now turn to the completeness of the algorithm.

Proposition B.6. Suppose $t \sim b' u$ and $\text{BISIM}(t,u,M,T)$. Then $b \rightarrow M$.

Proof. Let $S = \{S^b\}$ be a symbolic bisimulation such that $(t,u) \in S^b$. We show if $(t,u) \in S^b$, $b \rightarrow b'$ and $\text{CLOSE}(t,u,b',W,M,T)$ then $b \rightarrow M$, by induction on the size of $\mathcal{T}_G \times \mathcal{T}_H = W$.

The base case, when $W = \mathcal{T}_G \times \mathcal{T}_H$, is trivial since $M = \text{true}$ by $C1$ and hence $b \rightarrow M$.

Otherwise, we have $M = \bigwedge_i M_i$ and for all pairs of type $\gamma$ derivations $(t,i) \rightarrow M_i$ and $\text{CLOSE}(t_i,u_j,b \land b_i \land b_j \land \alpha = \alpha', \{t,u\} \cup W, M_{ij}, T_{ij})$ where $M_i = \bigwedge_i (b_i \rightarrow \bigvee_j (b_j \land M_{ij})) \land \bigwedge_j (b_j \rightarrow \bigvee_i (b_i \land M_{ij}))$, $T = \bigcup_{ij} T_{ij}$, and $t_i, u_j, M_{ij}$ are as follows

$$M_{ij}, t_i, u_j = \begin{cases} 
\forall z M_{ij}, t_i[x \mapsto z], u_j[y \mapsto z] & \text{if } \alpha = c?x, \ x' = c?y, \\
M_{ij}, t_i, u_j & \text{otherwise,}
\end{cases}$$

with $z \notin f v(t,u)$.

We need to establish $b \rightarrow M_i$. for each type $\gamma$ and as an example we consider the case when it is $c!$. Since $(t,u) \in S^b$ we can use the definition of symbolic bisimulations to associate with each derivation of this type, $t \rightarrow^{b_i \rightarrow \gamma, t_i}$ a fixed set of booleans $B_i$, one for each corresponding type $c!$ derivation from $u, u \rightarrow^{b_j \rightarrow \gamma, u_j}$. Then for each $b_{ij} \in B_i$ we know $b_{ij} \rightarrow b_i \land b_i \land b_j \land \alpha = \alpha'$ and $(t_i, u_j) \in S^{b_{ij}}$. Since $b \rightarrow b'$, $b_{ij} \rightarrow b_i \land b_i \land b_j \land \alpha = \alpha'$, we can apply induction to get $b_{ij} \rightarrow M_{ij}$. Hence $b_{ij} \rightarrow b_i \land M_{ij}$. Since $b_{ij}$ is an arbitrary element of $B_i$ and $\bigvee b_i \rightarrow \bigvee_j (b_j \land M_{ij})$.  

Now $b \land b_i \rightarrow \bigvee_j b_i \land b_j \land \alpha = \alpha'$ and so $b \land b_i \rightarrow \bigvee_j b_i \land M_{ij}$ which in turn implies $b \rightarrow (b_i \rightarrow \bigvee_j b_i \land M_{ij})$. This is true for any $i$ and hence $b \rightarrow (b_i \rightarrow \bigvee_j b_i \land M_{ij})$.

In a completely symmetric manner we can also establish $b \rightarrow (b_i \rightarrow \bigvee_i b_i \land M_{ij})$.

Combining these two we get the required $b \rightarrow M$.

Proof of Theorem 5.3. An immediate corollary to the above proposition. □

References