Symbolic Transition Graph with Assignment*

Huimin Lin
Laboratory for Computer Science
Institute of Software, Chinese Academy of Sciences
P.O Box 8718, Beijing 100080
E-mail: lhm@ios.ac.cn

Abstract

A new model for message–passing processes is proposed which generalizes the notion of symbolic transition graph as introduced in [HL95], by allowing *assignments* to be carried in transitions. The main advantage of this generalization is that a wider class of processes can be represented as finite state graphs. Two kinds of operational semantics, ground and symbolic, are given to such graphs. On top of them both ground and symbolic bisimulations are defined and are shown to agree with each other. An algorithm is also presented which computes bisimulation formulae for finite state symbolic transition graphs with assignments, in terms of the greatest solutions of *predicate equation systems*.

1 Introduction

Transition graphs are a popular semantic model for concurrency. The theory of bisimulation for such graphs has been well developed [Mil89]. Efficient bisimulation checking algorithms for finite state graphs have been proposed and are at the hearts of many existing verification tools for concurrency ([CPS89, GLZ89, SV89], among others). Traditional transition graphs use unstructured atomic actions as labels and, as a consequence, when non-trivial data domains are involved even very simple processes would generate infinite state graphs, thus beyond the scope of the existing tools. For example consider the process C defined by

$$C \Leftarrow c?x.d!x.C$$

where c, d are channel names and x ranges over integers. The standard transition graph for such a simple process is in fact infinite branching, because C has a transition $c?n$ for each integer n (Figure 1, left). This is somehow counter-intuitive, since all C can do is repeatedly read (input) an integer from c then send (output) it along d, so conceptually only *two* actions are needed in the description of its behaviour. Of course these actions are more complex in nature

*Supported by grants from the President Fund of Chinese Academy of Sciences, the National Science Foundation of China, and EU KIT 119 project SYMSEM.

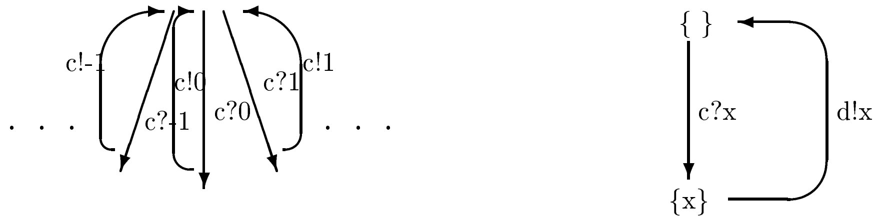

Figure 1: Traditional and symbolic graphs for C

as they bear internal structures: an input action should indicate the channel to read from as well the variable to store the received value, and similarly for an output action.

With such intuition in mind the notion of *symbolic transition graph* (STG for short) was advocated in [HL95]. In a STG each node is associated with a set of free (data) variables and each transition is labeled by an abstract action guarded by a boolean "trigger". Many processes with infinite state transition graphs are indeed symbolically finite state. For example the STG for the process C has only two states, as pictured in the right of Figure 1 where the trivial boolean guard *true* has been omitted. Based on such symbolic transition systems a theory of *symbolic bisimulation* has been developed which precisely captures the standard notion of bisimulation. An algorithm is also presented which, when given two finite state symbolic graphs, computes a characteristic boolean formula for them, namely they are bisimilar if and only if the formula evaluates to *true*.

But still many intuitively simple processes can not be pictured as finite STGs. For example given the following definition

$$P(x) \Leftarrow c!x.P(x+1)$$

the process $P(0)$ can first output, along channel c, the integer 0, then 1, 2, ,3 .... The STG for $P(x)$ has countably many states representing $P(x+1)$, $P(x+2)$, $P(x+3)$, ..., with edges $P(x+n) \overset{true,c!x+n}{\longmapsto} P(x+n+1)$ for $n \geq 0$.

The main purpose of this paper is to generalize the notion of symbolic transition graph so that processes like $P(x)$ above can be associated with finite state graphs. This is achieved by introducing *assignments* into labels. An edge now takes the form $n \overset{b,\overline{x}:=\overline{e},\alpha}{\longmapsto} n'$, where, besides a boolean condition b and an abstract action α, there is also an assignment $\overline{x} := \overline{e}$. Roughly it means if b is evaluated to *true* at node n then the action α can be fired, and, after the transition, the free variables $\overline{x}$ at node n' will have the values of $\overline{e}$ *evaluated at* n. With such extension the graph for $P(0)$ has only two nodes, as depicted in the left of Figure 2 (where the trivial boolean condition *true* has been omitted from the edges). We call such a graph *symbolic transition graph with assignment* (STGA for short). Note that STG is a special case of STGA with trivial assignment (identity mapping).

In a traditional transition graph each node denotes a unique process (closed

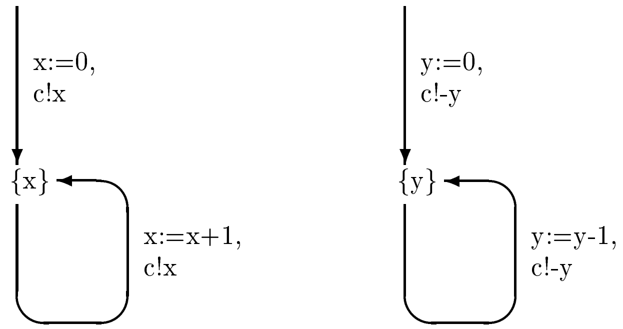

Figure 2: Symbolic transition graphs with assignments

term). But a node in a STGA has a set of free variables, leaving room for different interpretations. Given a STGA a *state* consists of a node n together with an evaluation (or *environment*) ρ mapping variables to values. At a state n_ρ one can use ρ to evaluate the outgoing edges at n, resulting a ground transition. As an example let us call m the node with free variable $\{x\}$ in the left graph of Figure 2 and let $\rho_i(x) = i$ for integer i. Then we have the following ground transitions:

$$m_{\rho_0} \xrightarrow{c!1} m_{\rho_1} \xrightarrow{c!2} m_{\rho_2} \xrightarrow{c!3} m_{\rho_3} \xrightarrow{c!4} \ldots$$

Note that here the changes of states are caused not only by performing actions $c!1$ and $c!2$ ..., but also by *modifying the value of the state variable* x: in the first transition the value of x changes from 0 to 1, and in the second from 1 to 2, *etc*. In this way a STGA is interpreted as a traditional transition graph whose nodes are the states of the STGA and edges the ground transitions, so the notion of (ground) bisimulation can be defined over states in the standard way.

A STGA can also be given a more abstract interpretation using *terms*. A *term* is a pair n_σ where n is a node in the graph and σ is a substitution. Accordingly, an edge can be interpreted as symbolic transitions between terms. For example, the edge $m \overset{true,x:=x+1,c!x}{\longmapsto} m$ gives rise to the symbolic transitions $m_{[x/x]} \overset{true,c!x+1}{\longrightarrow} m_{[x+1/x]}$, $m_{[x+1/x]} \overset{true,c!x+1+1}{\longrightarrow} m_{[x+1+1/x]}$, *etc*. In this way a STGA can be interpreted as a symbolic transition system between open terms. In the sequel we will often write (n, σ) more succinctly as n_σ. On top of the symbolic interpretation the notion of symbolic bisimulation can be defined in the spirit of [HL95]. One of the main results of this paper shows that these two notions of bisimulation in fact coincide.

By introducing assignments into edges many (even symbolically) infinite state transition systems can be represented by finite state STGAs, which opens the possibility of computing (symbolic) bisimulations for such systems. In this paper we present an algorithm which, when given a pair of STGAs, returns a predicate equation system whose greatest solution characterizes the most general condition for the two STGAs to be bisimilar. For instance the equation system for the two STGAs in Figure 2 is

$$X_{0,0} = (0 = -0) \wedge X_{1,1}[0/x, 0/y]$$
$$X_{1,1} = (x+1 = -(y-1)) \wedge X_{1,1}[x+1/x, y-1/y]$$

Due to the complexity induced by assignments it is not generally possible to compute the solution as a finite boolean expression, however one can reason about bisimilarity based on such equation system using knowledge about data domain. For example, The greatest solution of the above equations amounts to: for any $n \geq 0$

$$(x \underbrace{+1 + \ldots + 1}_{n} = -(y \underbrace{-1 - \ldots - 1}_{n}))[0, 0/x, y],$$

which can be easily verified using elementary knowledge about integers.

The rest of this section is devoted to the discussion of related work. STGA is formally defined in the next section. Section 3 gives operational semantics and defines both ground and symbolic bisimulations. In section 4 the notion of predicate equation systems is introduced, followed by an algorithm for generating such equation systems from STGAs. Finally, concluding remarks are drawn in Section 5 where further research directions are also outlined.

Related Work Symbolic transition graph with assignment is a direct generalization of symbolic transition graph, first advocated in [HL95]. The notion of symbolic bisimulation is inherited from that paper. Ground bisimulation is defined on terms there, while it is defined over *states* in this paper which reflects more closely the intuition of computation.

Data independent processes are studies in [JP92] where tests on data are not allowed and data expressions are restricted to single data values or variables. With the help of "schematic variables" checking (early) bisimulation for finite state data independent processes can be reduced to that for finite state pure processes. In [CR94] a theory of *abstract interpretations* is developed for value–passing processes, the idea being to map (large or infinite) value spaces down to (small) "abstract value" spaces and instantiate data variables over the abstract value space. This approach relies on the existence of such abstraction functions. [DHG95] proposes techniques for model checking a class of problems with (small) control parts and (large) data parts, by instantiating the control variables to obtain an intermediate description of the systems and perform model checking at such intermediate level. [HI93] considers a process language with assignment and builds a denotational model for testing equivalence. [Sch94] proposes *parameterised graphs* and contains separate algorithms for calculating each of the iterative approximations to strong bisimulation.

2 Symbolic Transition Graph with Assignment

Let ι range over a set of *base types*. We presuppose the following syntactic categories:

$v, \ldots \in Val_\iota$: a set of data values
$x, \ldots \in DVar_\iota$: a countable set of data variables
$e, \ldots \in DExp_\iota$: a set of data expressions
$b, \ldots \in BExp$: a set of boolean expressions
$c, \ldots \in Chan$: a set of channel names

It is assumed that both Val_ι and $DVar_\iota$ are included in $DExp_\iota$, and that $e = e' \in BExp$ for any $e, e' \in DExp_\iota$. $BExp$ is equipped with the usual operators $\wedge, \vee, \neg, \Rightarrow$ and $\forall$.

An *evaluation* $\rho \in Eval$ is a type-respecting mapping from Var to Val and we use the standard notation $\rho\{x \mapsto v\}$ to denote the evaluation which differs from ρ only in that it maps x to v. An application of ρ to a data expression e, denoted $\rho(e)$, always yields a value from Val and similarly for boolean expressions; $\rho(b)$ is either true or false. We use $\rho \models b$ to indicate that $\rho(b) = true$, and $b \models b'$ to mean for any ρ if $\rho(b)$ is true then so is $\rho(b')$. We will also write $b = b'$ for $b \models b'$ and $b' \models b$. A finite set of boolean expressions B is called a b-partition if $\bigvee B = b$.

A *substitution* σ is a type-respecting mapping from data variables to expressions and we use $e\sigma$ to denote the result of applying σ to the expression e. We will write $[\overline{e}/\overline{x}]$ for the substitution sending $\overline{x}$ to $\overline{e}$. If $\sigma \equiv [\overline{e}/\overline{x}]$ then $dom(\sigma) = \{\overline{x}\}$ and $cod(\sigma) = fv(\overline{e})$. Composition of substitutions is denoted by juxtaposition, and $\emptyset$ is the (polymorphic) empty substitution. $\sigma[x \mapsto e]$ denotes the substitution which differs from σ only in that it sends x to e. If $\sigma \equiv [\overline{e}/\overline{x}]$ then the application of σ to ρ is defined by $\sigma\rho = \rho\{\overline{x} \mapsto \rho(\overline{e})\}$. It is easy to see that $(\sigma\rho)(e) = \rho(e\sigma)$ when $fv(e) \subseteq dom(\sigma)$.

An *assignment* θ has the form $\overline{x} := \overline{e}$, with $\overline{x}$ and $\overline{e}$ having the same length and type. We will often take the liberty to identify an assignment $\overline{x} := \overline{e}$ with the substitution $[\overline{e}/\overline{x}]$.

An *action* is either a silent action τ, an input action $c?x$, or an output action $c!e$, where $c \in Chan$. Actions are ranged over by α. The sets of free and bound variables of actions are given by $fv(c!e) = fv(e)$, $bv(c?x) = \{x\}$, and empty otherwise. The set of channel names used in an action is defined by $chan(c?x) = chan(c!e) = \{c\}$ and $chan(\tau) = \emptyset$. Two actions are *equal over* b, written $\alpha =^b \alpha'$, if when α has the form $c!e$ then α' has the form $c!e'$ and $b \models e = e'$, otherwise $\alpha \equiv \alpha'$. A *guarded action with assignment* is a triple (b, θ, α) where b is a boolean expression, θ an assignment, and α an action.

Definition 2.1 A symbolic transition graph with assignments (STGA for short) is a rooted directed graph where each node n has an associated finite set of free variables $fv(n)$ and each edge is labeled by a guarded action with assignment. A STGA is *well-formed* if whenever $(b, \overline{x} := \overline{e}, \alpha)$ is the label of an edge from n to m, written $n \xmapsto{b,\overline{x}:=\overline{e},\alpha} m$, then $fv(b, \overline{e}) \subseteq fv(n)$, $fv(\alpha) \subseteq \{\overline{x}\}$, and $fv(m) \subseteq \{\overline{x}\} \cup bv(\alpha)$. □

We will often simply write $n \xmapsto{\theta,\alpha} m$ for $n \xmapsto{true,\theta,\alpha} m$, and will also omit θ when it is the identity assignment on $fv(n)$.

Two STGAs can be composed using parallel composition operator $\|$ and restriction operator $\backslash$. When composing two STGAs it is required that they use disjoint name spaces for data variables (so the only way for them to cooperate is through communication). The nodes of the parallel composition $(\mathcal{G}\|\mathcal{H})\backslash R$, with R a set of channel names, are pairs of those of $\mathcal{G}$ and $\mathcal{H}$ with $fv(<n,m>) = fv(n) \cup fv(m)$, and the edges are created by the following rules (where the symmetric rule of par has been omitted):

$$\text{par} \quad \frac{n \overset{b,\overline{x}:=\overline{e},\alpha}{\longmapsto} n'}{<n,m> \overset{b,\overline{x},\overline{y}:=\overline{e},\overline{y},\alpha}{\longmapsto} <n',m>} \quad \begin{array}{l} chan(\alpha) \cap R = \emptyset \\ fv(m) = \{\overline{y}\} \end{array}$$

$$\text{com} \quad \frac{n \overset{b_1,\overline{x}:=\overline{e}_1,c?z}{\longmapsto} n', \quad m \overset{b_2,\overline{y}:=\overline{e}_2,c!\overline{e}}{\longmapsto} m'}{<n,m> \overset{b_1 \wedge b_2,\overline{x},\overline{y},z:=\overline{e}_1,\overline{e}_2,e[\overline{e}_2/\overline{y}],\tau}{\longmapsto} <n',m'>}$$

Proposition 2.2 *1. The STGA for $(\mathcal{G}\|\mathcal{H})\backslash R$ as defined above is well-formed.*

2. $(\mathcal{G}\|\mathcal{H})\backslash R = (\mathcal{H}\|\mathcal{G})\backslash R$ and $((\mathcal{G}\|\mathcal{H})\|\mathcal{F})\backslash R = (\mathcal{G}\|(\mathcal{H}\|\mathcal{F}))\backslash R$.

As an example of how to generate symbolic graphs from a process description language let us consider regular value–passing *CCS* given by the following BNF grammar:

$$t \quad ::= \quad \mathbf{0} \mid \alpha.t \mid b \rightarrow t \mid t+t \mid P(\overline{e})$$

where for each process identifier P there is a definition $P(\overline{x}) \Leftarrow t$ which satisfies $fv(t) \subseteq \{\overline{x}\}$ and is *guarded*, *i.e.* every identifier in t is within the scope of an action prefixing $\alpha.\_$. Given a term t in this language a STGA can be generated using the following rules:

$$\frac{}{\alpha.t \overset{true,\emptyset,\alpha}{\longmapsto} t} \qquad \frac{t \overset{b,\theta,\alpha}{\longmapsto} t'}{t+u \overset{b,\theta,\alpha}{\longmapsto} t'} \qquad \frac{t \overset{b,\theta,\alpha}{\longmapsto} t'}{u+t \overset{b,\theta,\alpha}{\longmapsto} t'}$$

$$\frac{t \overset{b,\theta,\alpha}{\longmapsto} t'}{b' \rightarrow t \overset{b \wedge b',\theta,\alpha}{\longmapsto} t'} \qquad \frac{t \overset{b,\theta,\alpha}{\longmapsto} t'}{P(\overline{e}) \overset{b[\overline{e}/\overline{x}],\theta[\overline{x}:=\overline{e}],\alpha}{\longmapsto} t'} \quad P(\overline{x}) \Leftarrow t \text{ is a definition}$$

The conventional way to infer moves from a recursively defined process term $P(\overline{e})$ with definition clause $P(\overline{x}) \Leftarrow t$ is to substitute $\overline{e}$ for $\overline{x}$ in t and look for moves from the resulted term:

$$\frac{t[\overline{e}/\overline{x}] \overset{b,\alpha}{\longmapsto} t'}{P(\overline{e}) \overset{b,\alpha}{\longmapsto} t'}$$

This is an *eager* approach: substitutions are performed when moves are inferred. A disadvantage with this approach is that very often it results in infinite state graphs. An example is the process $P(0)$ where $P(x) \Leftarrow c!x.P(x+1)$. Though simple, using this rule the symbolic graph for $P(0)$ will have infinite number of nodes representing $P(0)$, $P(0+1)$, $P(0+1+1)$, $\ldots$.

$$\frac{m \overset{b,\theta,\tau}{\longmapsto} n}{m_\rho \xrightarrow{\tau} n_{\theta\rho}} \; \rho \models b \qquad \frac{m \overset{b,\theta,c!e}{\longmapsto} n}{m_\rho \xrightarrow{c!\rho(e\theta)} n_{\theta\rho}} \; \rho \models b \qquad \frac{m \overset{b,\theta,c?y}{\longmapsto} n}{m_\rho \xrightarrow{c?y} n_{\theta\rho}} \; \rho \models b$$

Figure 3: Late Operational Semantics

Here we adopt a rather *lazy* approach: the substitutions necessary in inferring moves from recursively defined terms are postponed. They are carried in the transitions and will be performed later when processes are compared for bisimulation, as we shall see below. For regular value–passing *CCS* it is not difficult to see that graphs generated with the new rules are always *finite*, because any process term involves only finite many recursive definition clauses and each such clause gives rise to a finite subgraph. In the Introduction we have already seen how the process $P(0)$ can be pictured as a finite graph with assignments.

3 Operational Semantics and Bisimulations

Given a STGA a *state* n_ρ is a pair consisting of a node n together with an evaluation ρ supplying values for free variables of n (it is understood that ρ is restricted to $fv(n)$). The late (ground) operational semantics is defined as the least relation over states generated by the rules in Figure 3. Late (ground) bisimulation can then be defined in the standard way:

Definition 3.1 A late (ground) bisimulation is a symmetric relation R over states such that: if $(m_\rho, n_\varrho) \in R$ then

1. $m_\rho \xrightarrow{c?y} m'_{\rho'}$ implies there exists $n_\varrho \xrightarrow{c?z} n'_{\varrho'}$ and $(m'_{\rho'\{y\mapsto v\}}, n'_{\varrho'\{z\mapsto v\}}) \in R$ for all $v \in \mathit{Val}$.
2. for any other actions $m_\rho \xrightarrow{a} m'_{\rho'}$ implies there exists $n_\varrho \xrightarrow{a} n'_{\varrho'}$ and $(m'_{\rho'}, n'_{\varrho'}) \in R$.

We write $m_\rho \sim n_\varrho$ if there is a late bisimulation R such that $(m_\rho, n_\varrho) \in R$. Two graphs $\mathcal{G}$ and $\mathcal{G}'$ are bisimilar with respect to ρ and ρ' if $r_\rho \sim r'_{\rho'}$ where r and r' are the roots of the two graphs, respectively. $\square$

We can also give a more abstract operational semantics to STGAs without referring to evaluations. This semantics is defined on *terms*. Given a symbolic transition graph $\mathcal{G}$, a term n_σ consists of a node and a substitution with $dom(\sigma) = fv(n)$. The set of free variables of the term $t \equiv n_\sigma$ is $fv(t) = fv(fv(n)\sigma)$. t is called *closed* when $fv(t) = \emptyset$.

The *symbolic* operational semantics is defined as the least relation generated by the rules in Figure 4. It is called "symbolic" because boolean and data expressions do not get evaluated, instead they are symbolically carried in transitions.

$$\frac{m \overset{b,\theta,\tau}{\longmapsto} n}{m_\sigma \overset{b\sigma,\tau}{\longrightarrow} n_{\theta\sigma}} \qquad \frac{m \overset{b,\theta,c!e}{\longmapsto} n}{m_\sigma \overset{b\sigma,c!(e\theta\sigma)}{\longrightarrow} n_{\theta\sigma}} \qquad \frac{m \overset{b,\theta,c?x}{\longmapsto} n}{m_\sigma \overset{b\sigma,c?z}{\longrightarrow} n_{\theta\sigma[x\mapsto z]}} \; z \notin \begin{array}{l} dom(\theta)\cup \\ fv(cod(\theta)\sigma) \end{array}$$

Figure 4: Symbolic Operational Semantics

The semantic rules only perform the assignments (as substitutions) and rename input variables if necessary, when passing through edges.

With symbolic operational semantics (late) *symbolic bisimulation* can then be defined as in [HL95]:

Definition 3.2 Let $\mathbf{S} = \{ S^b \mid b \in \mathit{BExp} \}$ be a *BExp*-indexed family of symmetric relations over terms. Define the functional $\mathcal{B}$ by letting $\mathcal{B}(\mathbf{S})$ be the *BExp*-indexed family of symmetric relations such that $(t, u) \in \mathcal{B}(S)^b$ if and only if

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

$\mathbf{S}$ is a late symbolic bisimulation if $\mathbf{S} \subseteq \mathcal{B}(\mathbf{S})$. We write $t \simeq^b u$ if there is a late symbolic bisimulation $\mathbf{S}$ such that $(t, u) \in S^b$. Two graphs $\mathcal{G}$ and $\mathcal{G}'$ are symbolic bisimilar over b if $r_\emptyset \simeq^b r'_\emptyset$ where r and r' are the roots of the two graphs. □

The following theorem states that symbolic bisimulation captures ground bisimulation. It's proof follows the line of Theorem **4.5** in [HL95].

Theorem 3.3 *$n_\sigma \simeq^b m_{\sigma'}$ if and only if $n_{\sigma\rho} \sim m_{\sigma'\rho}$ for every ρ such that $\rho \models b$.*

4 Computing Bisimulation for STGA

If a graph is finite, i.e. without loops, the presence of substitutions in the edges is superficial: they can be removed by travelling from the root to the leaves, starting from the empty substitution, applying symbolic interpretation (Figure 4) when passing through an edge and computing new free variables when encountering a node, resulting in a symbolic transition graph without assignments. For such graphs algorithms for computing bisimulation formulae already exist ([HL95]).

Substitutions make real differences for graphs with loops. For example, suppose we are trying to match symbolic transitions for the two graphs in Figure 2, starting at the pair of loop entrances with empty substitutions. The first time we pass the loop body the matching condition is $x+1 = -(y-1)$, but the second time it becomes $x+1+1 = -(y-1-1)$, and the third time $x+1+1+1 = -(y-1-1-1)$ ..., due to the substitutions $x := x + 1$ and $y := y - 1$ in the edges. Instead of attempting to compute a single bisimulation formula for a given pair of STGAs, here we present an algorithm which returns a *predicate equation system* over a set of *predicate variables*. The meaning of these predicate variables are taken as the greatest solution of the equation system.

4.1 Predicate Equation Systems

Formally let the sets of *predicate types* be given by

$$\pi ::= prop \mid \iota_1 \dots \iota_n \to prop$$

For each predicate type π we assume a set of *predicate variables* PV_π, ranged over by $X, Y, \dots$. Then the set of well-formed propositions and predicates are generated by the following rules:

$$\frac{b \in BExp}{b : prop} \qquad \frac{b \in BExp,\ \Phi : prop}{b \Rightarrow \Phi : prop}$$

$$\frac{x \in DVar,\ \Phi : prop}{\forall x \Phi : prop} \qquad \frac{\Phi, \Psi : prop}{\Phi \wedge \Psi : prop}$$

$$\frac{\Phi, \Psi : prop}{\Phi \vee \Psi : prop} \qquad \frac{\overline{e} \in DExp_{\iota_1} \times \dots \times DExp_{\iota_n},\ \Lambda : \iota_1 \dots \iota_n \to prop}{\Lambda \overline{e} : prop}$$

$$\frac{X \in PV_{\iota_1 \dots \iota_n \to prop}}{X :\ \iota_1 \dots \iota_n \to prop} \qquad \frac{\overline{x} \in DVar_{\iota_1} \times \dots \times DVar_{\iota_n},\ \Phi : prop}{(\overline{x})\Phi : \iota_1 \dots \iota_n \to prop}$$

Note that predicate variables always appear in positive positions. As usual the universal qualification $\forall x \Phi$ binds x in Φ. Functional abstraction $(\overline{x})\Phi$ is another binder: here each $x_i \in \overline{x}$ is a bound variable with scope Φ. The set of free data variables of Φ is denoted $fv(\Phi)$. Φ is *data closed* if $fv(\Phi) = \emptyset$.

Let **Bool** be the set of truth values $\{false, true\}$ with the natural ordering $false \prec true$. A (groud) *valuation* $\mathcal{V}$ maps each $X \in PV_{\iota_1 \dots \iota_n \to prop}$ to a function from $Val_{\iota_1} \times \dots \times Val_{\iota_n}$ to **Bool**. The **Bool** ordering $\prec$ can be extended pointwise to truth valued functions by letting $f \prec g$ *iff* $f(\overline{v}) \prec g(\overline{v})$ for all vectors of values $\overline{v}$ with appropriate type, and to valuations by letting $\mathcal{V} \prec \mathcal{V}'$ *iff* $\mathcal{V}(X) \prec \mathcal{V}'(X)$ for all X. Propositions and predicates are interpreted with respect to an evaluation ρ and a valuation $\mathcal{V}$ inductively thus

$$\begin{aligned}
&\|b\|_{\rho,\mathcal{V}} = \rho(b)\\
&\|b \Rightarrow \Phi\|_{\rho,\mathcal{V}} = \rho(b) \Rightarrow \|\Phi\|_{\rho,\mathcal{V}}\\
&\|\forall x \Phi\|_{\rho,\mathcal{V}} = \forall v \in Val\ \|\Phi\|_{\rho\{x \mapsto v\},\mathcal{V}}\\
&\|\Phi_1 \wedge \Phi_2\|_{\rho,\mathcal{V}} = \|\Phi_1\|_{\rho,\mathcal{V}} \wedge \|\Phi_2\|_{\rho,\mathcal{V}}\\
&\|\Phi_1 \vee \Phi_2\|_{\rho,\mathcal{V}} = \|\Phi_1\|_{\rho,\mathcal{V}} \vee \|\Phi_2\|_{\rho,\mathcal{V}}\\
&\|\Lambda \overline{e}\|_{\rho,\mathcal{V}} = \|\Lambda\|_{\rho,\mathcal{V}}(\rho(\overline{e}))\\
&\|X\|_{\rho,\mathcal{V}} = \mathcal{V}(X)\\
&\|(\overline{x})\Phi\|_{\rho,\mathcal{V}} = \lambda \overline{v} :\ Val\ \|\Phi\|_{\rho\{\overline{x} \mapsto \overline{v}\},\mathcal{V}}
\end{aligned}$$

The operators $\Rightarrow$, $\wedge$, $\vee$ and $\forall$ on the right hand sides are semantic.

A *predicate equation system* E is of the form

$$\{ X_i = \Lambda_i \mid i \in I \}$$

where X_i and Λ_i are of the same type and $X_i \neq X_j$ for $i \neq j$. E is *predicate closed* if all predicate variables in its right hand sides are included in $\{ X_i \mid i \in I \}$. E is *closed* if it is predicate closed and all its right hand sides are data closed. In an equation $X = (\overline{x})\Phi$, $\overline{x}$ is the *formal parameter* of X. In the algorithm below we will write the formal and actual parameters of predicate variables together, in the form of substitutions. For instance, the equation system

$$\begin{aligned} X_{0,0} &= (0 = -0) \wedge X_{1,1}(0,0) \\ X_{1,1} &= (x,y)((x+1 = -(y-1)) \wedge X_{1,1}(x+1, y-1)) \end{aligned}$$

will be written as

$$\begin{aligned} X_{0,0} &= (0 = -0) \wedge X_{1,1}[0/x, 0/y] \\ X_{1,1} &= (x+1 = -(y-1)) \wedge X_{1,1}[x+1/x, y-1/y]. \end{aligned}$$

Given a valuation $\mathcal{V}$ and an evaluation ρ, a *solution* of E is a fix-point of the right hand sides of E, *i.e.* a function ζ mapping $X_i : \iota_1 \ldots \iota_n \to prop$ to $Val_{\iota_1} \times \ldots \times Val_{\iota_n} \to \mathbf{Bool}$ such that

$$\zeta(X_i) = \|\Lambda_i\|_{\rho,\mathcal{V}\{\zeta\}} \qquad\qquad i \in I$$

where $\mathcal{V}\{\zeta\}$ denotes the obvious modification to $\mathcal{V}$ by ζ.

It is straightforward to check that, with respect to a valuation $\mathcal{V}$ and an evaluation ρ, the meaning of right hand sides of E defines a monotonic functional $\lambda\zeta\|\overline{\Lambda}\|_{\rho,\mathcal{V}\{\zeta\}}$. Therefore by a standard result due to Tarski [Tar55] it has both the least and the greatest solutions, and it is the latter that we are interested in (since we are dealing with strong bisimulation here). If E is closed then its solutions do not depend on any valuation and evaluation, and we shall simply say "ζ is a solution of E".

Propositions and predicates can also be interpreted symbolically. A *symbolic valuation* $\mathcal{U}$ is a function mapping each $X \in PV_{\iota_1\ldots\iota_n \to prop}$ to a function $f : DExp_{\iota_1} \times \ldots \times DExp_{\iota_n} \to BExp$ which is consistent with respect to evaluation in the sense that $\rho(\overline{e}) = \rho(\overline{e}')$ implies $\rho(f(\overline{e})) = \rho(f(\overline{e}'))$ for any ρ. Then a symbolic meaning can be assigned to propositions and predicates with respect to a symbolic valuation by the following rules:

$$\begin{aligned}
&[\![b]\!]_{\mathcal{U}} = b \\
&[\![b \Rightarrow \Phi]\!]_{\mathcal{U}} = b \Rightarrow [\![\Phi]\!]_{\mathcal{U}} \\
&[\![\forall x \Phi]\!]_{\mathcal{U}} = \forall x [\![\Phi]\!]_{\mathcal{U}} \\
&[\![\Phi_1 \wedge \Phi_2]\!]_{\mathcal{U}} = [\![\Phi_1]\!]_{\mathcal{U}} \wedge [\![\Phi_2]\!]_{\mathcal{U}} \\
&[\![\Phi_1 \vee \Phi_2]\!]_{\mathcal{U}} = [\![\Phi_1]\!]_{\mathcal{U}} \vee [\![\Phi_2]\!]_{\mathcal{U}} \\
&[\![\Lambda\overline{e}]\!]_{\mathcal{U}} = [\![\Lambda]\!]_{\mathcal{U}}(\overline{e}) \\
&[\![X]\!]_{\mathcal{U}} = \mathcal{U}(X) \\
&[\![(\overline{x})\Phi]\!]_{\mathcal{U}} = \lambda\overline{x}[\![\Phi]\!]_{\mathcal{U}}
\end{aligned}$$

where $\Rightarrow$, $\wedge$, $\vee$ and $\forall$ on the right hand sides are operators on *BExp*.

The set of all boolean expressions *BExp* can be ordered by letting $b \sqsubset b'$ *iff* $b \models b'$. With this ordering *BExp* forms a complete lattice. $\sqsubset$ agrees with $\prec$ on **Bool** in the sense that $b \sqsubset b'$ *iff* $b\rho \prec b'\rho$ for any ρ. This ordering can be extended pointwise to functions from $DExp_{\iota_1} \times \ldots \times DExp_{\iota_n}$ to *BExp* in a standard way.

Given an evaluation ρ a symbolic valuation $\mathcal{U}$ can be "evaluated" to a (ground) valuation $\mathcal{U}_\rho$ by letting, for each X,

$$\mathcal{U}_\rho(X)(\overline{v}) = \rho(\mathcal{U}(X)(\overline{v})) \qquad \text{for all } \overline{v}$$

Then we have the following lemma:

Lemma 4.1 *For any proposition* Φ *and abstraction* Λ, $\rho([\![\Phi]\!]_{\mathcal{U}}) = \|\Phi\|_{\rho,\mathcal{U}_\rho}$ *and* $\rho([\![\Lambda]\!]_{\mathcal{U}}) = \|\Lambda\|_{\rho,\mathcal{U}_\rho}$ *for any evaluation* ρ.

Proof: By mutual induction on the structures of Φ and Λ. $\Box$

A *symbolic solution* of a predicate equation system $E : \ \{ X_i = \Lambda_i \mid i \in I \}$, with respect to $\mathcal{U}$, is a function η mapping $X_i : \iota_1 \ldots \iota_n \rightarrow prop$ to $DExp_{\iota_1} \times \ldots \times DExp_{\iota_n} \rightarrow BExp$ for each $i \in I$ such that

$$\eta(X_i) = [\![\Lambda_i]\!]_{\mathcal{U}\{\eta\}} \qquad\qquad i \in I$$

where $\mathcal{U}\{\eta\}$ denotes the obvious modification to $\mathcal{U}$ by η. When E is closed then its symbolic solutions do not depend on any $\mathcal{U}$. Again as the symbolic meaning of the right hand sides of E is a monotonic functional with respect to $\sqsubset$, E has both the least and the greatest symbolic solutions.

Proposition 4.2 *For a closed equation system* E, η *is a symbolic solution of* E *iff* η_ρ *is a solution of* E *for any* ρ. *In particular* η *is the greatest symbolic solution of* E *iff* η_ρ *is the greatest solution of* E *for any* ρ.

4.2 The Algorithm

A *path* in a graph is a sequence of connected edges. Given two graphs $\mathcal{G}$ and $\mathcal{H}$, a *matching path* is a pair consisting of a path p in $\mathcal{G}$ and a path q in $\mathcal{H}$:

$$\begin{array}{l} p = n_0 \overset{b_0,\theta_0,\alpha_0}{\longmapsto} n_1 \overset{b_1,\theta_1,\alpha_1}{\longmapsto} \ldots \overset{b_{k-1},\theta_{k-1},\alpha_{k-1}}{\longmapsto} n_k \\ q = m_0 \overset{b'_0,\theta'_0,\alpha'_0}{\longmapsto} m_1 \overset{b'_1,\theta'_1,\alpha'_1}{\longmapsto} \ldots \overset{b'_{k-1},\theta'_{k-1},\alpha'_{k-1}}{\longmapsto} m_k \end{array}$$

such that α and α' are of the same type, i.e., either $\alpha \equiv \tau \equiv \alpha'$, or $\alpha \equiv c!e$ and $\alpha' \equiv c!e'$, or $\alpha \equiv c?x$ and $\alpha \equiv c?y$. A *matching loop* is a matching path such that $n_0 = n_k$, $m_0 = m_k$ and (n_0, m_0), called the *entry* of the matching loop, is in a matching path from the roots of $\mathcal{G}$ and $\mathcal{H}$. We will simply write "loop" for "matching loop" and "loop entry" for "matching loop entry".

$$bisim(\mathcal{G},\mathcal{H}) = \text{for each } (n,n') \in loopEntries \ return\ X_{n,n'} = match(n,\emptyset,n',\emptyset)$$

$$match(n,\sigma,n',\sigma') = \bigwedge_{\gamma} match_{\gamma}(n,\sigma,n',\sigma')$$

$$\begin{array}{l}
match_{\tau}(n,\sigma,n',\sigma') = \\
\text{let } B_{ij} = close(n_i, \theta_i\sigma, n'_j, \theta'_j\sigma') \\
\qquad\qquad \text{for } n \overset{b_i,\theta_i,\tau}{\longmapsto} n_i, n' \overset{b'_j,\theta'_j,\tau}{\longmapsto} n'_j \\
\text{in } \bigwedge_i (b_i\sigma \Rightarrow \bigvee_j (b'_j\sigma' \wedge B_{ij})) \wedge \bigwedge_j (b'_j\sigma' \Rightarrow \bigvee_i (b_i\sigma \wedge B_{ij}))
\end{array}$$

$$\begin{array}{l}
match_{c!}(n,\sigma,n',\sigma') = \\
\text{let } B_{ij} = close(n_i, \theta_i\sigma, n'_j, \theta'_j\sigma') \\
\qquad\qquad \text{for } n \overset{b_i,\theta_i,c!e_i}{\longmapsto} n_i, n' \overset{b'_j,\theta'_j,c!e'_j}{\longmapsto} n'_j \\
\text{in } \bigwedge_i (b_i\sigma \Rightarrow \bigvee_j (b'_j\sigma' \wedge e_i\theta_i\sigma = e'_j\theta'_j\sigma' \wedge B_{ij})) \wedge \\
\quad \bigwedge_j (b'_j\sigma' \Rightarrow \bigvee_i (b_i\sigma \wedge e_i\theta_i\sigma = e'_j\theta'_j\sigma' \wedge B_{ij}))
\end{array}$$

$$\begin{array}{l}
match_{c?}(n,\sigma,n',\sigma') = \\
\text{let } B_{ij} = \text{let } z = newVar() \\
\qquad\qquad\qquad B'_{ij} = close(n_i, \theta_i\sigma[x \mapsto z], n'_j, \theta'_j\sigma'[y \mapsto z]) \\
\qquad\qquad\qquad\quad \text{for } n \overset{b_i,\theta_i,c?x}{\longmapsto} n_i, n' \overset{b'_j,\theta'_j,c?y}{\longmapsto} n'_j \\
\qquad\qquad \text{in } \forall z B'_{ij} \\
\text{in } \bigwedge_i (b_i\sigma \Rightarrow \bigvee_j (b'_j\sigma' \wedge B_{ij})) \wedge \bigwedge_j (b'_j\sigma' \Rightarrow \bigvee_i (b_i\sigma \wedge B_{ij}))
\end{array}$$

$$\begin{array}{l}
close(n,\sigma,n',\sigma') = \text{if } (n,n') \in loopEntries \text{ then } return\ X_{n,n'}(\sigma,\sigma') \\
\qquad\qquad\qquad\qquad\quad \text{else } match(n,\sigma,n',\sigma'))
\end{array}$$

Figure 5: The Algorithm for Late Symbolic Bisimulation

The algorithm for late strong bisimulation is presented in Figure 5, where γ ranges over $\{\tau, c!, c?\}$. It takes as input a pair of finite symbolic transition graphs, $\mathcal{G}$ and $\mathcal{H}$, with disjoint name spaces for data variables and uses the set *loopEntries* of their matching loop entries which also includes the pair of the root nodes of $\mathcal{G}$ and $\mathcal{H}$. The output is a predicate equation system. The function *bisim* introduces a predicate variable for each matching loop entry and creates an equation for each matching loop. At each pair of matching nodes it calls function *match* to generate a proposition which embodies a partition as required by the definition of symbolic bisimulation.

The correctness of the algorithm is a direct consequence of the following theorem:

Theorem 4.3 *Let $E = \{X_{n,n'} = \Lambda_{n,n'}\}$ be the equation system returned by the algorithm on $\mathcal{G}$ and $\mathcal{H}$.*

1. *If* η *is a symbolic solution of* E *and* $\eta(X_{n,n'})(\overline{e},\overline{e}') = b$ *then* $n_{[\overline{e}/\overline{x}]} \simeq^b n'_{[\overline{e}'/\overline{x}']}$.
2. *If* η *is the greatest symbolic solution of* E *and* $n_{[\overline{e}/\overline{x}]} \simeq^b n'_{[\overline{e}'/\overline{x}']}$ *then* $b \models \eta(X_{n,n'})(\overline{e},\overline{e}')$.

where $\overline{x}$ *and* $\overline{x}'$ *are the free variables of* n *and* n'*, respectively.*

Proof: *1.* Let $S^b = \{(n_{[\overline{e}/\overline{x}]}, n'_{[\overline{e}'/\overline{x}']})\}$ where $b = \eta(X_{n,n'})(\overline{e},\overline{e}')$. Then it can be shown that $\mathbf{S} = \{S^b\}$ is a symbolic bisimulation.

2. Suppose $\mathbf{S} = \{S^b\}$ is a symbolic bisimulation between terms of $\mathcal{G}$ and $\mathcal{H}$ such that $(n_{[\overline{e}/\overline{x}]}, n'_{[\overline{e}'/\overline{x}']}) \in S^b$ for some $S^b \in \mathbf{S}$. For each $X_{n,n'}$ set $\eta'(X_{n,n'})(\overline{e},\overline{e}') = \begin{cases} b & \text{if } (n_{[\overline{e}/\overline{x}]}, n'_{[\overline{e}'/\overline{x}']}) \in S^b \text{ for some } S^b \in \mathbf{S}. \\ \mathit{false} & \text{otherwise.} \end{cases}$ Then it can be shown that η' is a symbolic solution of E. Hence $\eta'(X_{n,n'})(\overline{e},\overline{e}') \models \eta(X_{n,n'})(\overline{e},\overline{e}')$. $\square$

The algorithm reduces the problem of deciding bisimulation of two STGAs to the problem of checking validity for the greatest solution of a closed equation system. Unlike boolean equation systems which can be effectively solved [Lar92, Liu92], a predicate equation system may involve arbitrary data and boolean expressions and it is in general not possible to automatically compute its greatest solution, except for some special cases (see **Remark** below). However one can reason with the greatest solution using data domain knowledge. For instance the equation system returned by the algorithm for the two STGAs in Figure 2 is

$$\begin{aligned} X_{0,0} &= (0 = -0) \wedge X_{1,1}[0/x, 0/y] \\ X_{1,1} &= (x+1 = -(y-1)) \wedge X_{1,1}[x+1/x, y-1/y]. \end{aligned}$$

It spells out, for any $n \geq 0$

$$(x \underbrace{+1 + \ldots + 1}_{n} = -(y \underbrace{-1 - \ldots - 1}_{n}))[0, 0/x, y],$$

which can be easily proved by mathematical induction using elementary knowledge on integers.

As another example let us consider the two STGAs in Figure 6 which can be written in *CCS* syntax as

$$\begin{aligned} &M \Leftarrow r?x.c!x.M_1(x) \\ &M_1(x) \Leftarrow r?u.\mathit{if}\ u > 0\ \mathit{then}\ M_1(x+u)\ \mathit{else}\ M_1(x-u) \\ &N \Leftarrow r?y.c!y.N_1(y) \\ &N_1(y) \Leftarrow r?v.N_1(y + |v|) \end{aligned}$$

where $|v|$ denotes the absolute value of v. Both processes accumulate the inputs received via r and output the sums along c. Running the algorithm generates the following result:

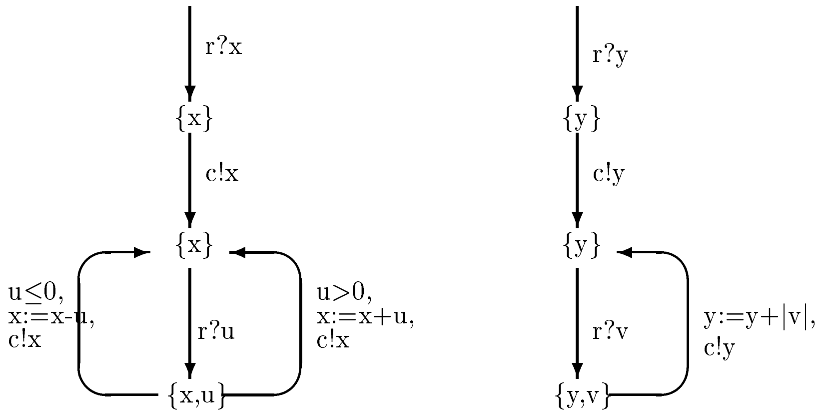

Figure 6: Processes which accumulate inputs

$$
\begin{aligned}
X_{0,0} &= \forall z_1 X_{2,2}[z_1/x, z_1/y] \\
X_{2,2} &= \forall z_2\ (\neg z_2 > 0 \Rightarrow x - z_2 = y + |z_2| \wedge X_{2,2}[x - z_2/x, y + |z_2|/y]) \wedge \\
&\qquad (z_2 > 0 \Rightarrow x + z_2 = y + |z_2| \wedge X_{2,2}[x + z_2/x, y + |z_2|/y]) \wedge \\
&\qquad (\neg z_2 > 0 \wedge x - z_2 = y + |z_2| \wedge X_{2,2}[x - z_2/x, y + |z_2|/y] \vee \\
&\qquad z_2 > 0 \wedge x + z_2 = y + |z_2| \wedge X_{2,2}[x + z_2/x, y + |z_2|/y])
\end{aligned}
$$

Using laws of propositional calculus the second equation can be simplified to

$$
\begin{aligned}
X_{2,2} = \forall z_2\ &(\neg z_2 > 0 \Rightarrow x - z_2 = y + |z_2| \wedge X_{2,2}[x - z_2/x, y + |z_2|/y]) \wedge \\
&(z_2 > 0 \Rightarrow x + z_2 = y + |z_2| \wedge X_{2,2}[x + z_2/x, y + |z_2|/y])
\end{aligned}
$$

So $\forall z_1 X_{2,2}[z_1/x, z_1/y]$ means, for any $n > 1$,

$$
\begin{aligned}
\forall z_1 \forall z_2\ (z_2 > 0 \Rightarrow \ldots \forall z_n\ &(z_n > 0 \Rightarrow z_1 + \ldots + z_{n-1} + z_n = z_1 + \ldots + |z_n|) \wedge \\
&\neg z_n > 0 \Rightarrow z_1 + \ldots + z_{n-1} - z_n = z_1 + \ldots + |z_n|) \wedge \\
&\vdots \\
\neg z_2 > 0 \Rightarrow \ldots \forall z_n\ &(z_n > 0 \Rightarrow z_1 - \ldots - z_{n-1} + z_n = z_1 + \ldots + |z_n|) \wedge \\
&\neg z_n > 0 \Rightarrow z_1 - \ldots - z_{n-1} - z_n = z_1 + \ldots + |z_n|).
\end{aligned}
$$

These booleans can be easily shown to be *true* by mathematical induction using simple facts about integers.

Note that the greatest solutions of the above two equation systems amount to infinite conjunctions, and mathematical induction is essential in establishing their truth.

Remark An important special case is simple STGAs where edges do not carry assignments (more accurately all assignments are identity mappings). For such graphs the only substitutions in the resulting equation systems are simply renamings generated when matching input actions. So if we try to compute the greatest fixpoint of such an equation system by iteration, starting from the constant function mapping every data expression to *true*, then this process will converge after at most n iterations where n is the maximum number of times that data variables are bound. Hence the greatest solutions of the equation systems for simple STGAs *can* be computed automatically. [HL95] presents an algorithm which computes bisimulation formulae for such graphs, without referring to predicate equations. This is achieved by ordering data variables and always using the smallest unused variable when matching an input action, so that the algorithm will visit each matching loop n times if the active variables in the loop are bound by n input actions.

5 Conclusions and Future Work

We have introduced the notion of symbolic transition graphs in which edges are labeled by boolean conditions, actions, as well as assignments. The presence of assignments as part of labels makes it possible to represent state transitions not only caused by communication actions, but also by changing values of state variables. As a consequence many traditional transition graphs which are infinite state can be represented as finite STGAs. A theory of symbolic bisimulation for such STGAs has been developed, and it is shown that this notion of symbolic bisimulation precisely captures the standard notion of bisimulation when STGAs are interpreted as traditional transition systems over states. Thus STGAs are mathematically tractable. We have also presented an algorithm which, when given two STGAs, computes a predicate equation system whose greatest solution characterizes the condition under which the two STGAs are bisimilar. The algorithm has been implemented (in Standard ML) and tested on some small problems, including all of the examples used in this paper.

We have only developed the theory of symbolic bisimulation and the algorithm for the late version of bisimulation equivalence. The same can be done for early bisimulation as well, but due to space limitation these have to be omitted from this extended abstract.

The algorithm reduces the problem of checking bisimulation for STGAs to the problem of reasoning about the greatest solutions of predicate equation systems over data domain. As future work we would like to investigate techniques for verifying properties concerning such solutions. The STGA model is very general and it would be interesting to identify some subclasses for which the greatest solutions are computable (one such case is STGs as discussed in the remark of the previous section). Another topic for further research is to extend the current framework to deal with weak equivalences.

Acknowledgments: Thanks to Matthew Hennessy and Julian Rathke for discussions on this subject. Comments from two anonymous referees have led to several improvements. The main body of this work was carried out during a visit to Sussex University in 1995, and I am grateful to the EU KIT project for financial support.

References

[CPS89] R. Cleaveland, J. Parrow, and B. Steffen. A semantics based verification tool for finite state systems. In *Proceedings of the* 9^{th} *International Symposium on Protocol Specification, Testing and Verification*, North Holland, 1989.

[CR94] R. Cleaveland and J. Riely. Testing-based abstractions for value-passing systems. In *CONCUR'94*, number 836 in Lecture Notes in Computer Science, pages 417 – 432. Springer–Verlag, 1994.

[DHG95] W. Damn, H. Hungar, and O. Grumberg. What if model checking must be truly symbolic? In *Proceedings of the Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, 1995. Aarhus, Denmark.

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

[HI93] M. Hennessy and A. Ingolfsdottir. Communicating processes with value-passing and assignment. *Formal Aspects of Computing*, 3:346 – 366, 1993.

[HL95] M. Hennessy and H. Lin. Symbolic bisimulations. *Theoretical Computer Science*, 138:353 – 389, 1995.

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

[Lar92] K.G. Larsen. Efficient local correctness checking. In *Computer Aided Verification*, Lecture Notes in Computer Science. Springer–Verlag, 1992.

[Liu92] X. Liu. *Specification and Decomposition in Concurrency.* Ph.d. thesis, Aalborg University, 1992.

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

[Sch94] M.Z. Schreiber. *Value-passing Process Calculi as a Formal Method.* Ph.D. thesis, Imperial College, 1994.

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

[Tar55] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5, 1955.