Computing Strong/Weak Bisimulation Equivalences and Observation Congruence for Value-Passing Processes *

Zhoujun Li and Huowang Chen

Department of Computer Science, Changsha Institute of Technology
Changsha, Hunan, 410073, P.R.China
zjli@csit.edu.cn

Abstract. We introduce an improved version of the symbolic transition graph with assignment (STGA) of Lin. The distinction of our model is that the assignment of a transition is performed after rather than before the action. Consequently, it has two advantages over the original one: on one hand, most regular value-passing processes can be represented more intuitively and compactly as such graphs; on the other hand, the natural definitions of symbolic double transitions can be given. The rules which generate the improved STGAs from regular value-passing processes are presented. The various versions (late/early, ground/symbolic) of strong operational semantics and strong bisimulation are given to such graphs, respectively. Our strong bisimulation algorithms are based on the late strong bisimulation algorithm of Lin, however, ours are more concise and practical. Finally, the improved STGAs are generalized to both symbolic observation graphs with assignments and symbolic congruence graphs with assignments, and therefore weak bisimulation equivalence and observation congruence can be checked, respectively.

1 Introduction

Process description languages are useful for specifying, designing and verifying concurrent distributed systems [1,12]. Bisimulation provides an excellent semantic theory for them[12]. Hence bisimulation checking, especially weak bisimulation and observation congruence checking, is the central issue and critical step for the application of such process description languages in practice. As we know, the transition graph is a standard semantic model for finite-state processes, any regular process expression of pure-CCS can be represented as a finite transition graph. Based on finite transition graphs and their variants—observation graphs and congruence graphs, efficient algorithms for checking strong/weak bisimulation equivalences and observation congruence have been proposed and used to build verification tools [2,3].

As for value-passing processes with infinite data domain, infinite transition graphs must be generated and compared in order to check bisimulation between

* This work is partially supported by 863 Hi Tech Project and NNSF of China.

two processes, hence the existing tools are no longer applicable except for the so called "data independent processes" [6]. To overcome the disadvantage, symbolic transition graphs (STGs for short) were advocated in [4] as intuitive and finite representations for a large number of value-passing processes, by using symbolic actions with boolean guards.

But even very simple processes such as $P(x) \leftarrow c!x^2.P(2x+1)$ can not be described by finite STGs. By introducing assignments into labels, the notion of symbolic transition graph with assignment (STGA for short) and an algorithm for late strong bisimulation were proposed in [10]. An edge there takes the form

$m \xrightarrow{b, \bar{x} := \bar{e}, \alpha} n$. The intuition is that if the boolean condition b is satisfied at node m then the assignment $\bar{x} := \bar{e}$ is performed and the result is used to evaluate the action α (if it is an output action) and to update the data values of $\bar{x}$ at node n . Thus the process term $P(x)$ can be depicted as the left STGA in Figure 1.

Figure 1: Two versions of STGAs for $P(x)$

Such STGA is indeed a finite representation of $P(x)$, although it looks somewhat strange at first sight. In order to check weak bisimulation and observation congruence along the lines of [8,9], we need to give a reasonable and natural definition of symbolic double transitions between the nodes of STGAs.

Intuitively, we should infer $m \xrightarrow{b \wedge b' \theta, \theta' \theta, c!e'} n$ from $m \xrightarrow{b, \theta, c!e} n$ and $m \xrightarrow{b', \theta', \tau} n$, where e' is a data expression satisfying the equation $e' \theta' \theta = e \theta$. For example, let θ, θ' and e be $x := 2x + 1, x := x^3 + 1$ and $x + 1$ respectively, then $e'[(2x + 1)^3 + 1/x] = (2x + 1) + 1$, so $e' = \sqrt[3]{x - 1} + 1$. In general, it is difficult, if not impossible, to figure out e' . Hence we can not give a reasonable definition of symbolic double transitions for the original STGAs.

However, if the assignment of a transition is performed after rather than before the action, we can obtain an improved version of STGA. In our model, an edge now takes the form $m \xrightarrow{b, \alpha, \bar{x} := \bar{e}} n$. It means that if b is satisfied at node m then the action α is performed and the assignment $\bar{x} := \bar{e}$ is used to update the data values of $\bar{x}$ at node n . Here the assignment no longer applies to α , so $P(x)$ can be expressed as the improved STGA in the right side of Figure 1.

In fact it is the key to swap the order of the assignment and the action. Based on our model, we can define symbolic double transitions over nodes, which are shown in Definition 4.1 and 6.1 respectively. It is rather surprising that most regular value-passing processes can be represented more intuitively and compactly as the improved STGAs. For example given the following definitions

$$Q(x) \Leftarrow \alpha_1.R(e_1), \quad R(x) \Leftarrow \alpha_2.Q(e_2)$$

where α_i is τ or $cl!e'_i$, e_i and e'_i are data expressions containing x , $Q(x)$ can be pictured as the original and improved STGAs respectively in Figure 2.

Figure 2: Two versions of STGAs for $Q(x)$

In the above example, the two nodes $n_0\{x\}$ and $n_1\{x\}$ of the right STGA in Figure 2 correspond directly (one-to-one) to processes $Q(x)$ and $R(x)$, namely, there is no semantic gap between the syntactic and the graphic representations of the processes. But it is not the case for the left one, since the two nodes $m_1\{x\}$ and $m_2\{x\}$ represent neither $Q(x)$ nor $R(x)$.

As a more illustrative example let us consider the process $S(x)$ defined by

$$S(x) \Leftarrow \text{even}(x) \rightarrow \alpha_1.S(e_1) + \text{odd}(x) \rightarrow \alpha_2.S(e_2)$$

where α_i is τ or $cl!e'_i$. $S(x)$ can be represented as an improved STGA with one node $n\{x\}$ and two edges $n \xrightarrow{\text{even}(x), \alpha_1, x:=e_1} n$ and $n \xrightarrow{\text{odd}(x), \alpha_2, x:=e_2} n$. The original STGA for $S(x)$ contains at least three nodes and six edges. It is somehow counter-intuitive and too complex.

STGA is a very general semantic model for value-passing processes. It can be seen as a flowchart for concurrent computation. The STGAs in Figure 3 represent two processes which output through channel d the integral part of the square root of the corresponding integer input through channel c . The left one is an implementation of the right one.

Figure 3

The rest of the paper is organised as follows: in the next section the improved STGA is formally defined and the generation rules of such graphs are presented.

In section 3 two versions of late operational semantics and late strong bisimulations are given to such graphs. In section 4 two versions of late weak bisimulation and late observation congruence are defined, and moreover, equivalent characterizations of their symbolic versions are established. The notions of SOGA and SCGA are also given in this section. Algorithms for checking late strong/weak bisimulation equivalences and observation congruence are presented in section 5. Section 6 outlines the changes necessary to handle the early case. Finally some conclusions and future research directions are included in section 7.

2 An Improved STGA Model

For brevity, we assume that all data values allowed to pass through channels are of the same type. Val is the set of such data values, ranged over by v . We also presuppose the following syntactic categories: $Var = \{x_0, x_1, x_2, \dots\}$ is a countably infinite set of data variables, ranged over by x, y, z . $PVar$ is a set of predicate variables, ranged over by X . $DExp$ is a set of data expressions, ranged over by e . $BExp$ is a set of boolean expressions, ranged over by b . $Chan$ is a set of channel names, ranged over by c . Moreover, we assume that $Var \cup Val \subseteq DExp$, and $e = e' \in BExp$ for any $e, e' \in DExp$. $BExp$ is closed under the usual operators $\wedge, \vee, \neg, \rightarrow$ and $\forall$. If $V \subseteq Var$ is a finite set, we use $new(V)$ to denote the least variable $x_i \in Var$ such that x_i is not in V .

An *evaluation* $\rho \in Eval$ is a mapping from Var to Val and we use the notation $\rho\{\bar{x} \mapsto \bar{v}\}$ to denote the evaluation which differs from ρ only in that it maps $\bar{x}$ to $\bar{v}$. Obviously $\rho(e) \in Val$ and $\rho(b) \in \{true, false\}$. We use $\rho \models b$ to indicate $\rho(b) = true$, and $b \models b'$ to mean $\rho \models b$ implies $\rho \models b'$ for any ρ . 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 $\vee B = b$.

A *substitution* $\sigma \in Sub$ is a mapping from Var to $DExp$. We write $[\bar{e}/\bar{x}]$ for the substitution sending $\bar{x}$ to $\bar{e}$, with $\bar{x}$ and $\bar{e}$ having the same length. If $\sigma \equiv [\bar{e}/\bar{x}]$ then $dom(\sigma) = \{\bar{x}\}$, $cod(\sigma) = fv(\bar{e})$ and $n(\sigma) = dom(\sigma) \cup cod(\sigma)$. $\emptyset$ denotes the empty substitution, $\sigma|V$ the restriction of σ on V and $e\sigma$ the result of applying σ to e . If $dom(\sigma_1) \cap dom(\sigma_2) = \emptyset$, then $\sigma_1 \cup \sigma_2$ is also a substitution. Composition of substitutions σ and σ' is denoted by $\sigma\sigma'$ such that $(e)\sigma\sigma' = (e\sigma)\sigma'$, and $\sigma[x \mapsto e]$ denotes the substitution which differs from σ only in that it sends x to e . If $\sigma \equiv [\bar{e}/\bar{x}]$ then the application of σ to ρ is defined by $\sigma\rho = \rho\{\bar{x} \mapsto \rho(\bar{e})\}$. It is easy to see that $(\sigma\rho)(e) = \rho(e\sigma)$ and $\sigma\rho \models b$ iff $\rho \models b\sigma$.

An *assignment* $\theta \in Assign$ has the form $\bar{x} := \bar{e}$, which can be identified with the substitution $[\bar{e}/\bar{x}]$. Thus the above notations for substitution can be also applied to assignment. $\theta \setminus x = \theta|(dom(\theta) - \{x\})$.

An *action* $\alpha \in Act$ is either a silent action τ , an input action $c?x$, or an output action $c!e$. We use $chan(\alpha)$ to denote the set of channel names occurring in α , $fv(\alpha)$ and $bv(\alpha)$ the sets of free and bound variables of α . $\alpha\sigma$ is defined by $(c!e)\sigma \equiv c!(e\sigma)$, and $\alpha\sigma \equiv \alpha$ otherwise. We write $\alpha =^b \alpha'$ to mean that if $\alpha \equiv c!e$ then $\alpha' \equiv c!e'$ and $b \models e = e'$, otherwise $\alpha \equiv \alpha'$. We also assume that $\hat{\tau} = \varepsilon$ and $\hat{\alpha} = \alpha$ if $\alpha \neq \tau$. A *guarded action with assignment* is a triple (b, α, θ) .

Definition 2.1 A symbolic transition graph with assignment (STGA for short) is a rooted directed graph where each node n is associated with a finite set of free variables $fv(n)$ and each edge is labelled by a guarded action with assignment such that if $m \xrightarrow{b, \alpha, \bar{x} := \bar{e}} n$ is a labelled edge from m to n , then $fv(b, \alpha, \bar{e}) \subseteq fv(m)$, $fv(n) \subseteq fv(m) \cup bv(\alpha) \cup \{\bar{x}\}$ and $bv(\alpha) \cap (\{\bar{x}\} \cup fv(\bar{e})) = \emptyset$.

In the above definition, neither $\{\bar{x}\} \subseteq fv(m)$ nor $fv(\alpha) \subseteq \{\bar{x}\}$ is required. Thus identity assignment is unnecessary in the improved STGA, while it is indispensable in the original STGA. The trivial boolean guard *true* or the empty assignment $\emptyset$ will be omitted from the edges. The constraint $bv(\alpha) \cap (\{\bar{x}\} \cup fv(\bar{e})) = \emptyset$ is imposed just to simplify the parallel composition rules and the operational semantics of the improved STGAs.

Like [10], the parallel composition $(\mathcal{G} \parallel \mathcal{H}) \setminus R$ of two improved STGAs, $\mathcal{G}$ and $\mathcal{H}$, with disjoint data variable spaces, are generated by the following simple rules:

$$\begin{array}{c} \text{Par} \quad \frac{m \xrightarrow{b, \alpha, \bar{x} := \bar{e}} m'}{b, \alpha, \bar{x} := \bar{e}} \quad \text{chan}(\alpha) \cap R = \emptyset \\ \hline \langle m, n \rangle \xrightarrow{\quad} \langle m', n \rangle \\ \\ \text{Com} \quad \frac{\begin{array}{c} b_1, c?z, \bar{x} := \bar{e}_1 \quad b_2, c!e, \bar{y} := \bar{e}_2 \\ m \xrightarrow{\quad} m' \quad n \xrightarrow{\quad} n' \end{array}}{b_1 \wedge b_2, \tau, \bar{x}, \bar{y}, z := \bar{e}_1, \bar{e}_2, e} \\ \hline \langle m, n \rangle \xrightarrow{\quad} \langle m', n' \rangle \end{array}$$

For any term t in *regular* value-passing CCS, which is given by the following BNF grammar:

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

we can generate an improved STGA by means of the following rules:

$$\begin{array}{ll} \alpha.P(\bar{e}) \xrightarrow{true, \alpha, \bar{x} := \bar{e}} P(\bar{x}) & \alpha \in \{\tau, c!e'\} \\ \alpha.u \xrightarrow{true, \alpha, \emptyset} u & \alpha \in \{\tau, c!e'\}, u \text{ is not of the form } P(\bar{e}) \\ c?y.u \xrightarrow{true, c?z, \emptyset} u[z/y] & \text{where } z = new(fv(c?y.u)) \\ u \xrightarrow{b, \alpha, \theta} u' & \text{implies } b' \rightarrow u \xrightarrow{b \wedge b', \alpha, \theta} u' \\ u \xrightarrow{b, \alpha, \theta} u' & \text{implies } u + w \xrightarrow{b, \alpha, \theta} u' \\ u \xrightarrow{b, \alpha, \theta} u' & \text{implies } w + u \xrightarrow{b, \alpha, \theta} u' \\ t \xrightarrow{b, \alpha, \theta} t' & \text{implies } P(\bar{e}) \xrightarrow{b[\bar{e}/\bar{x}], \alpha[\bar{e}/\bar{x}], \theta[\bar{x} := \bar{e}]} t' \quad \alpha \in \{\tau, c!e'\} \\ t \xrightarrow{b, c?y, \theta} t' & \text{implies } P(\bar{e}) \xrightarrow{b[\bar{e}/\bar{x}], c?z, \theta[(\bar{x} := \bar{e}) \setminus y]} t'[z/y] \\ & \text{where } z = new(n(\theta[(\bar{x} := \bar{e}) \setminus y]) \cup fv(c?y.t')) \end{array}$$

where $P(\bar{x}) \Leftarrow t$ is a definition.

Here we use $\bar{\theta}$ to denote the result obtained by eliminating the identity assignments from θ . Like [10], we also adopt a *lazy* approach (the last two rules above) to infer moves from a recursively defined process term $P(\bar{e})$. However, there are two main distinctions between our generation rules and those of [10]:

(1) A case analysis of the action prefix $\alpha.u$ is performed. The first rule is the key, which transforms the applied occurrence $P(\bar{e})$ of P in $\alpha.P(\bar{e})$ into the stan-

dard template $P(\bar{x})$ directly, by introducing the assignment $\bar{x} := \bar{e}$ immediately. In this way the graphic representation is true of the syntactic representation of the process and thus there is no semantic gap between the two kinds of representations. Besides, the standard template $P(\bar{x})$ is easier to be reused and hence the generated STGA may be more compact. The examples in the Introduction are clear proofs, from which more extreme examples can be derived. Of course, there exist also counter-examples. For instance consider the process $P(e_3)$, where $P(x) \Leftarrow c!e_1.P(e_2)$ and $e_3 = e_2$. $P(e_3)$ can be represented as an original STGA with one node $m\{x\}$ and one edge $m \xrightarrow{true, x := e_2, c!e_1} m$. The improved STGA for $P(e_3)$ contains two nodes and two edges.

(2) A mechanism of renaming bound names by the function $new()$ is introduced when dealing with input action $d?y$. The significance of α -conversion can be seen from the following example. Consider the process M defined by

$$M \Leftarrow c?x.N(x) \\ N(x) \Leftarrow a!x.N(x+1) + b?y.N(y) + c?z.N(z) + d?v.N(v)$$

It is unimaginable what will happen without α -conversion when generating the STGA of M . It is the very cause that α -conversion with $new()$ is also proposed in [4]. To reduce the side-effect of α -conversion, we introduce $[\bar{x} := \bar{e}] \setminus y$ in the last rule to avoid unnecessary renaming of the bound name y .

Based on the above analysis and comparison, we think, in most cases, the improved STGA representation of a process may be more intuitive and concise than the original one. Hence when applying any bisimulation checking algorithm to the improved STGAs, an obvious benefit is that its execution time is shorter, the returned predicate equation system is simpler and easier to be solved.

3 Late Strong Operational Semantics and Bisimulation

Since each node of a STGA $\mathcal{G}$ may be associated with a set of free variables, we can provide two natural interpretations for it as [10]. A *state* n_ρ consists of a node n and an evaluation ρ , where ρ is restricted to $fv(n)$. One can use ρ to evaluate each outgoing edge from n , resulting in a ground transition. We use p, q to range over states. If p is a state n_ρ , we use $p\{x \mapsto v\}$ to denote the state $n_{\rho\{x \mapsto v\}}$.

Definition 3.1 The late ground operational semantics is defined as the least relation over states satisfying the following rules:

$$\frac{m \xrightarrow{b, \tau, \theta} n}{m_\rho \xrightarrow{\tau} n_{\theta\rho}} \quad \rho \models b \qquad \frac{m \xrightarrow{b, c!e, \theta} n}{m_\rho \xrightarrow{c!e(\rho)} n_{\theta\rho}} \quad \rho \models b \qquad \frac{m \xrightarrow{b, c?x, \theta} n}{m_\rho \xrightarrow{c?x} n_{\theta\rho}} \quad \rho \models b$$

Definition 3.2 A late strong ground bisimulation is a symmetric relation R over states such that if $(m_\rho, n_\delta) \in R$ then

1. $m_\rho \xrightarrow{c?x} m'_{\rho'}$ implies there exists $n_\delta \xrightarrow{c?y} n'_{\delta'}$ and $(m'_{\rho'\{x \mapsto v\}}, n'_{\delta'\{y \mapsto v\}}) \in R$ for all $v \in Val$.

2. for any other action $\alpha \in \{\tau, c!v\}$, $m_\rho \xrightarrow{\alpha} m'_{\rho'}$ implies there exists $n_\delta \xrightarrow{\alpha} n'_{\delta'}$ and $(m'_{\rho'}, n'_{\delta'}) \in R$.

We write $m_\rho \sim_l n_\delta$ if there is a late strong ground bisimulation R such that $(m_\rho, n_\delta) \in R$. Two STGAs $\mathcal{G}$ and $\mathcal{G}'$ are late strong ground bisimilar with respect to ρ if $r_\rho \sim_l r'_{\rho'}$, where r and r' are the root nodes of the two graphs.

We can also give an abstract operational semantics to $\mathcal{G}$. A term n_σ , also written as (n, σ) , consists of a node n and a substitution σ . n_σ means $n_\sigma|_{fv(n)}$. We use t, u to range over terms. If t is a term n_σ , we use $t[x \mapsto z]$ to denote the term $n_{\sigma[x \mapsto z]}$, $fv(t)$ the set $fv(fv(n)\sigma)$ of free variables and $t\rho$ the state $n_{\sigma\rho}$.

Definition 3.3 The late symbolic operational semantics is defined as the least relation over terms satisfying the following rules:

$$\frac{m \xrightarrow{b, \tau, \theta} n}{m_\sigma \xrightarrow{b\sigma, \tau} n_{\theta\sigma}} \quad \frac{m \xrightarrow{b, c!e, \theta} n}{m_\sigma \xrightarrow{b\sigma, c!e\sigma} n_{\theta\sigma}} \quad \frac{m \xrightarrow{b, c?x, \theta} n}{m_\sigma \xrightarrow{b\sigma, c?z} n_{\theta\sigma[x \mapsto z]}} \quad z \notin fv(m_\sigma) \cup fv((fv(n) - \{x\})\theta\sigma)$$

Definition 3.4 Let $S = \{S^b | b \in BExp\}$ be a boolean expression indexed family of symmetric relations over terms. S is a late strong symbolic bisimulation if $(t, u) \in S^b$ implies

Whenever $t \xrightarrow{b_1, \alpha} t'$ with $bv(\alpha) \cap fv(t, u, b) = \emptyset$, then there is a $b \wedge b_1$ -partition B with $fv(B) \subseteq fv(t, u, b)$ such that for each $b' \in B$ there exists a $u \xrightarrow{b_2, \alpha'} u'$ such that $b' \models b_2, \alpha =^{b'} \alpha'$ and $(t', u') \in S^{b'}$. We write $t \sim_L^b u$ if there is a late strong symbolic bisimulation S such that $(t, u) \in S^b$. Two STGAs $\mathcal{G}$ and $\mathcal{G}'$ are late strong symbolic bisimilar over b if $r_\theta \sim_L^b r'_{\theta'}$, where r and r' are the root nodes of the two graphs.

The following theorem underlies the significance of symbolic bisimulation. Its proof follows the lines of Theorem 4.5 in [4].

Theorem 3.5 $m_\sigma \sim_L^b n_{\sigma'}$ if and only if $m_{\sigma\rho} \sim_l n_{\sigma'\rho}$ for every ρ such that $\rho \models b$.

4 Late Weak Bisimulation and Observation Congruence

To define the late ground/symbolic double arrow relations, we first define the late symbolic double transitions over nodes.

Definition 4.1 The late symbolic double transitions are defined as the least relations over nodes satisfying the following rules:

- $m \xrightarrow{true, \varepsilon, \emptyset}_L m$.
- $m \xrightarrow{b, \alpha, \theta} n$ implies $m \xrightarrow{b, \alpha, \theta}_L n$.
- $m \xrightarrow{b, \tau, \theta} n$ implies $m \xrightarrow{b \wedge b', \theta, \alpha\theta, \theta'\theta}_L n$.
- If α is not of the form $c?x$ then $m \xrightarrow{b, \alpha, \theta} n$ implies $m \xrightarrow{b', \tau, \theta'} n$.

Definition 4.2 The late ground double arrows are defined as the least relations over states satisfying the following rules:

$$\begin{array}{c}
\frac{b, \varepsilon, \theta}{m \xRightarrow{\varepsilon} n} \quad \rho \models b \qquad \frac{b, \tau, \theta}{m \xRightarrow{\tau} n} \quad \rho \models b \\
\frac{b, c!e, \theta}{m \xRightarrow{c!e} n} \quad \rho \models b \qquad \frac{b, c?x, \theta}{m \xRightarrow{c?x} n} \quad \rho \models b \\
\frac{b, c!e, \theta}{m \xRightarrow{c!e} n} \quad \rho \models b \qquad \frac{b, c?x, \theta}{m \xRightarrow{c?x} n} \quad \rho \models b
\end{array}$$

Definition 4.3 A late weak ground bisimulation is a symmetric relation R over states such that if $(m_\rho, n_\delta) \in R$ implies

1. Whenever $m_\rho \xrightarrow{c?x}_l m'_{\rho'}$ then there exists $n_\delta \xrightarrow{c?y}_l n'_{\delta'}$ and for all $v \in Val$, there exists $n''_{\delta''}$ such that $n'_{\delta'\{y \mapsto v\}} \xRightarrow{\varepsilon}_l n''_{\delta''}$ and $(m'_{\rho'\{x \mapsto v\}}, n''_{\delta''}) \in R$.
2. Whenever $m_\rho \xrightarrow{\alpha}_l m'_{\rho'}$ for any other action $\alpha \in \{\tau, c!v\}$ then $n_\delta \xRightarrow{\hat{\alpha}}_l n'_{\delta'}$ for some $n'_{\delta'}$ and $(m'_{\rho'}, n'_{\delta'}) \in R$.

We write $m_\rho \approx_l n_\delta$ if there is a late weak ground bisimulation R such that $(m_\rho, n_\delta) \in R$. Two STGAs $\mathcal{G}$ and $\mathcal{G}'$ are late weak ground bisimilar with respect to ρ if $r_\rho \approx_l r'_{\rho'}$, where r and r' are the root nodes of $\mathcal{G}$ and $\mathcal{G}'$, respectively.

The late ground observation congruence $\simeq_l$ is then defined in terms of $\approx_l$ as usual. To define the symbolic version of late weak bisimulation and late observation congruence, we first introduce the late symbolic double arrows.

Definition 4.4 The late symbolic double arrows are defined as the least relations over terms satisfying the following rules:

$$\begin{array}{c}
\frac{b, \varepsilon, \theta}{m \xRightarrow{\varepsilon} n} \quad \frac{b, \tau, \theta}{m \xRightarrow{\tau} n} \quad \frac{b, c!e, \theta}{m \xRightarrow{c!e} n} \\
\frac{b\sigma, \varepsilon}{m_\sigma \xRightarrow{\varepsilon} n_{\theta\sigma}} \quad \frac{b\sigma, \tau}{m_\sigma \xRightarrow{\tau} n_{\theta\sigma}} \quad \frac{b\sigma, c!e\sigma}{m_\sigma \xRightarrow{c!e\sigma} n_{\theta\sigma}} \\
\frac{b, c?x, \theta}{m \xRightarrow{c?x} n} \quad \text{where } z \notin fv(m_\sigma) \cup fv((fv(n) - \{x\})\theta\sigma) \\
\frac{b\sigma, c?z}{m_\sigma \xRightarrow{c?z} n_{\theta\sigma[x \mapsto z]}}
\end{array}$$

Lemma 4.5 If $t \xRightarrow{b, \alpha}_L t'$, then one of the following cases holds:

- $b = true, \alpha = \varepsilon$ and $t' = t$.
- If $\alpha \neq \varepsilon$ then $t \xrightarrow{b, \alpha} t'$; otherwise $t \xrightarrow{b, \tau} t'$.
- There exist t'', b_1, b_2 such that $t \xrightarrow{b_1, \tau} t'' \xRightarrow{b_2, \alpha}_L t'$ and $b = b_1 \wedge b_2$.
- If α is not of the form $c?x$ then there exist t'', b_1, b_2 such that $t \xRightarrow{b_1, \alpha}_L t'' \xrightarrow{b_2, \tau} t'$ and $b = b_1 \wedge b_2$.

Definition 4.6 Let $S = \{S^b \mid b \in BExp\}$ be a boolean expression indexed family of symmetric relations over terms. S is a late weak symbolic bisimulation if $(t, u) \in S^b$ implies

Whenever $t \xrightarrow{b_1, \alpha} t'$ with $bv(\alpha) \cap fv(t, u, b) = \emptyset$, then there is a $b \wedge b_1$ -partition

B with $fv(B) \subseteq fv(t, u, b)$ such that for each $b' \in B$ there exists a $u \xRightarrow{b_2, \alpha'}_L u'$ such that $b' \models b_2 \alpha = b' \alpha'$ and

- if $\alpha \equiv c?x$ then there is a b' -partition B' such that for each $b'' \in B'$ there exists a $u' \xrightarrow{b_3, \varepsilon}_L u''$ such that $b'' \models b_3$ and $(t', u'') \in S^{b''}$.
- otherwise $(t', u') \in S^{b'}$.

We write $t \approx_L^b u$ if there is a late weak symbolic bisimulation S such that $(t, u) \in S^b$. Two STGAs $\mathcal{G}$ and $\mathcal{G}'$ are late weak symbolic bisimilar over b if $r_\emptyset \approx_L^b r'_\emptyset$, where r and r' the root nodes of the two graphs.

The late symbolic observation congruence $\simeq_L^b$ is defined in terms of $\approx_L^b$ as usual. The two versions of late weak bisimulation equivalence and late observation congruence can be related as in the case of late strong bisimulation.

Theorem 4.7 (Soundness and Completeness of $\approx_L$ and $\simeq_L$)

1. $m_\sigma \approx_L^b n_{\sigma'}$ if and only if $m_{\sigma\rho} \approx_l n_{\sigma'\rho}$ for every $\rho \models b$.
2. $m_\sigma \simeq_L^b n_{\sigma'}$ if and only if $m_{\sigma\rho} \simeq_l n_{\sigma'\rho}$ for every $\rho \models b$.

We can give an equivalent characterization of late weak symbolic bisimulation only in terms of late symbolic double arrows. It underlies our algorithm for checking late weak bisimulation.

Theorem 4.8 Let $S = \{S^b \mid b \in BExp\}$ be a boolean expression indexed family of symmetric relations over terms. S is a late weak symbolic bisimulation, if and only if for any $(t, u) \in S^b$ and $\alpha \in \{\varepsilon, c!e, c?x\}$

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

- if $\alpha \equiv c?x$ then there is a b' -partition B' such that for each $b'' \in B'$ there exists a $u' \xrightarrow{b_3, \varepsilon}_L u''$ such that $b'' \models b_3$ and $(t', u'') \in S^{b''}$.
- otherwise $(t', u') \in S^{b'}$.

To establish the equivalent definition of late symbolic observation congruence, we must introduce the positive $\xrightarrow{b, \alpha, \theta}_{+L}$ (resp. $\xrightarrow{b, \alpha}_{+L}$) which differs from $\xrightarrow{b, \alpha, \theta}_L$ (resp. $\xrightarrow{b, \alpha}_L$) only in that it excludes the reflexive case.

Theorem 4.9 Two term t, u are late symbolic observation congruent with respect to b , i.e. $t \simeq_L^b u$, if and only if for any $\alpha \in \{\varepsilon, c!e, c?x\}$

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

- if $\alpha \equiv c?x$ then there is a b' -partition B' such that for each $b'' \in B'$ there exists a $u' \xrightarrow{b_3, \varepsilon}_L u''$ such that $b'' \models b_3$ and $t' \approx_L^{b''} u''$.
- otherwise $t' \approx_L^{b'} u'$.

And similarly for u .

Based on Definition 4.1, Theorem 4.8 and Theorem 4.9, we can generalize the notions of observation graph and congruence graph of [2] to the symbolic level to check weak bisimulation and observation congruence for finite STGAs. We use $ND(\mathcal{G})$ and $E(\mathcal{G})$ to denote the sets of nodes and edges of $\mathcal{G}$ respectively.

Given a STGA $\mathcal{G}$, its late SOGA $\mathcal{G}^O$ is defined by $ND(\mathcal{G}^O) = ND(\mathcal{G})$ and $E(\mathcal{G}^O) = \{m \xrightarrow{b, \alpha, \theta}_L n \mid m, n \in ND(\mathcal{G}) \text{ and } \alpha \in \{\varepsilon, c!e, c?x\}\}$. Since $m \xrightarrow{true, \varepsilon, \phi}_L m$, there will be one such self ε -looping edge for each node in $\mathcal{G}^O$.

The late SCGA $\mathcal{G}^C$ is a variant of $\mathcal{G}^O$ that records the possibility of initial τ -actions. Let r be the root node of $\mathcal{G}$, then $\mathcal{G}^C$ can be constructed as follows:

- i) if r has no incoming edges, $\mathcal{G}^C$ is obtained simply by excising the self ε -looping edge of r from $\mathcal{G}^O$.
- ii) otherwise we first construct a new STGA $\mathcal{G}'$ with a new root node r' such that $fv(r') = fv(r)$, $ND(\mathcal{G}') = ND(\mathcal{G}) \cup \{r'\}$ and $E(\mathcal{G}') = E(\mathcal{G}) \cup \{r' \xrightarrow{b, \alpha, \theta}_L n \mid r \xrightarrow{b, \alpha, \theta}_L n \in E(\mathcal{G})\}$. Obviously $r_\emptyset \sim_L^{true} r'_\emptyset$ and r' has no incoming edges. Then $\mathcal{G}^C$ is obtained by excising the self ε -looping edge of r' from $\mathcal{G}'^O$.

The symbolic approach used in this paper can avoid completely the state explosion due to the instantiation of input variables. However, $\mathcal{G}^O$ and $\mathcal{G}^C$ may be infinite graphs if $\mathcal{G}$ contains τ -cycles with assignments. Since our algorithm for weak bisimulation and observation congruence is based on finite SOGAs and SCGAs, let us consider which kinds of $\mathcal{G}$ can produce finite $\mathcal{G}^O$ and $\mathcal{G}^C$:

- (1) If there is no τ -cycle in $\mathcal{G}$, then $\mathcal{G}^O$ and $\mathcal{G}^C$ are finite.
- (2) If each edge of every τ -cycle of $\mathcal{G}$ contains no assignments, we can also generate finite $\mathcal{G}^O$ and $\mathcal{G}^C$. Moreover, under certain conditions, we can even eliminate such τ -cycles in advance.
- (3) As shown in Figure 3, the τ -cycle may contain edges with assignments but its execution will terminate because of the changes of values of data variables. In this way, the manipulation of data expressions will be carried out by means of assignments and controlled τ -cycles. If the upper bound of the execution times of such a τ -cycle can be calculated and this upper bound is independent of the assigned variables, then we can generate finite $\mathcal{G}^O$ and $\mathcal{G}^C$ as well. For instance the upper bound is x or $\lfloor \sqrt{x} \rfloor$ for the left STGA in Figure 3, and moreover, it is independent of the assigned variables y_1, y_2 and y_3 .

Fortunately, by investigating a large number of examples, we find out that the above three kinds of STGAs are sufficient to describe many practical concurrent communicating systems such as network protocols. How to check weak bisimulation and observation congruence directly on top of STGAs rather than on SOGAs and SCGAs deserves further research.

5 The Algorithms for Bisimulations

A late strong bisimulation algorithm for STGAs was presented in [10]. Based on the improved STGA model, we propose a more concise and practical algorithm for late strong bisimulation, and further extend the framework to deal with late

weak bisimulation and late observation congruence. we refer the readers to [10] for a detailed account on such notions as predicate equation system.

The algorithm for late strong bisimulation is in Figure 4. Its input is a pair of finite STGAs, $\mathcal{G}$ and $\mathcal{H}$, with disjoint name spaces for data variables. The output is a predicate equation system. Starting from the pair of root nodes (r, r') of $\mathcal{G}$ and $\mathcal{H}$, the function *close* introduces a predicate variable $X_{m,n}$ and the corresponding equation for each pair of matching nodes (m, n) . Thus the formulation and correction proof of the algorithm become clearer and simpler than the algorithm of [10], while the additional predicate variables can be eliminated just by substitutions. It must be pointed out that each $X_{m,n}$ always has formal parameters $fv(m) \cup fv(n)$, which are listed in order as in $\mathcal{G}$ and $\mathcal{H}$. Since $fv(m) \cap fv(n) = \emptyset$, we will write $X_{m,n}(\sigma \cup \sigma')$ for $X_{m,n}(\sigma, \sigma')$. Hence each applied occurrence of $X_{m,n}$ will be written as $X_{m,n}(\sigma)$, which means $X_{m,n}(fv(m)\sigma, fv(n)\sigma)$.

$$\begin{aligned}
bisim(\mathcal{G}, \mathcal{H}) &= \{ \text{ PES } := \emptyset \} \\
&\quad \text{close}(r, r', \emptyset, \emptyset) \\
&\quad \text{return}(\text{ PES }) \\
\\
close(m, n, \sigma, W) &= \\
&\quad \text{if } (m, n) \notin W \\
&\quad \text{then } \{ \text{ PES } := \text{ PES } \cup \{ X_{m,n} = \text{match}(m, n, W) \} \} \\
&\quad \text{if } W \neq \emptyset \\
&\quad \text{then return}(X_{m,n}(\sigma)) \\
\\
match(m, n, W) &= \bigwedge_{\gamma} \{ \text{match}_{\gamma}(m, n, W) \mid \gamma \in \text{ NAType}(m, n) \} \\
\\
match_{\tau}(m, n, W) &= \\
&\quad \text{let } B_{ij} = \text{close}(m_i, n_j, \theta_i \cup \theta'_j, W \cup \{(m, n)\}) \\
&\quad \quad \text{for } m \xrightarrow{b_i, \tau, \theta_i} m_i, \quad n \xrightarrow{b'_j, \tau, \theta'_j} n_j \\
&\quad \text{in } \bigwedge_i (b_i \rightarrow \bigvee_j (b'_j \wedge B_{ij})) \wedge \bigwedge_j (b'_j \rightarrow \bigvee_i (b_i \wedge B_{ij})) \\
\\
match_{cl}(m, n, W) &= \\
&\quad \text{let } B_{ij} = \text{close}(m_i, n_j, \theta_i \cup \theta'_j, W \cup \{(m, n)\}) \\
&\quad \quad \text{for } m \xrightarrow{b_i, cl, e_i, \theta_i} m_i, \quad n \xrightarrow{b'_j, cl, e'_j, \theta'_j} n_j \\
&\quad \text{in } \bigwedge_i (b_i \rightarrow \bigvee_j (b'_j \wedge e_i = e'_j \wedge B_{ij})) \wedge \bigwedge_j (b'_j \rightarrow \bigvee_i (b_i \wedge e_i = e'_j \wedge B_{ij})) \\
\\
match_{c?}(m, n, W) &= \\
&\quad \text{let } z = \text{newVar}() \\
&\quad \quad B_{ij} = \text{close}(m_i, n_j, \theta_i[x \mapsto z] \cup \theta'_j[y \mapsto z], W \cup \{(m, n)\}) \\
&\quad \quad \text{for } m \xrightarrow{b_i, c?x, \theta_i} m_i, \quad n \xrightarrow{b'_j, c?y, \theta'_j} n_j \\
&\quad \text{in } \bigwedge_i (b_i \rightarrow \bigvee_j (b'_j \wedge \forall z B_{ij})) \wedge \bigwedge_j (b'_j \rightarrow \bigvee_i (b_i \wedge \forall z B_{ij}))
\end{aligned}$$

Figure 4: The Algorithm for Late Strong Bisimulation

Compared to the algorithm of [10], another important advantage of our algorithm is that we avoid finding out all matching loop entries in advance by extending

the functions *close* and *match* with parameter W to record the pairs of matching nodes searched. Thus our algorithm is more practical when applied to complex STGAs. $NAType(m, n)$ in Figure 4 is the set of types of actions that appear in the next transitions from m and n . The types of actions τ, cle and $c?x$ are $\tau, c!$ and $c?$, respectively.

The correctness of the algorithm is guaranteed by the following theorem.

Theorem 5.1 Let $PES = \{X_{m,n} = \Lambda_{m,n}\}$ be the predicate equation system returned by the algorithm on $\mathcal{G}$ and $\mathcal{H}$.

- (1) If η is a symbolic solution of PES and $\eta(X_{m,n})(\sigma) = b$ then $m_\sigma \sim_L^b n_\sigma$.
- (2) If η is the greatest symbolic solution of PES and $m_\sigma \sim_L^b n_{\sigma'}$ then $b \models \eta(X_{m,n})(\sigma \cup \sigma')$.

For instance the simplified equation system PES returned by the algorithm for the two STGAs in Figure 3 is:

$$\begin{aligned} X_{0,0} &= \forall z_1 X_{1,1}(z_1, z_1) \\ X_{1,1} &= (x < 0 \wedge u < 0) \vee (x \geq 0 \wedge u \geq 0 \wedge X_{2,2}(x, 0, 1, 1, u, 0)) \\ X_{2,2} &= (y_3 \leq x \wedge (v+1)^2 \leq u \wedge X_{2,2}(x, y_1+1, y_2+2, y_3+y_2+2, u, v+1)) \vee \\ &\quad (y_3 > x \wedge (v+1)^2 > u \wedge y_1 = v \wedge X_{3,3}) \\ X_{3,3} &= true \end{aligned}$$

Where the formal parameter sets of $X_{0,0}$, $X_{1,1}$, $X_{2,2}$ and $X_{3,3}$ are $\{\}$, $\{x, u\}$, $\{x, y_1, y_2, y_3, u, v\}$ and $\{\}$, respectively. By mathematical induction and the fact $1 + 3 + 5 + \dots + (2n-1) = n^2$, we can conclude that $X_{0,0} = true$, i.e. the two STGAs in Figure 3 are late strong bisimilar.

Given two STGAs $\mathcal{G}$ and $\mathcal{H}$, the algorithm for computing late weak symbolic bisimulation (resp. late symbolic observation congruence) is similar to the one presented in Figure 4 but working on $\mathcal{G}^O$ and $\mathcal{H}^O$ (resp. $\mathcal{G}^C$ and $\mathcal{H}^C$). As Definition 4.4, Theorem 4.8 and Theorem 4.9 indicate, we need not only to replace $match_\tau$ with $match_\varepsilon$ but also to modify $match_{c?}$ to

$$\begin{aligned} match_{c?}(m, n, W) &= \\ \text{let } z &= newVar() \\ B_{i,jk} &= close(m_i, n_{jk}, \theta_i[x \mapsto z] \cup \theta'_{jk}(\theta'_j[y \mapsto z]), W \cup \{(m, n)\}) \\ \text{for } m &\xrightarrow{b_i, c?x, \theta_i}_L m_i, \quad n \xrightarrow{b'_j, c?y, \theta'_j}_L n_j \xrightarrow{b'_{jk}, \varepsilon, \theta'_{jk}}_L n_{jk} \\ B_{il,j} &= close(m_{il}, n_j, \theta_{il}(\theta_i[x \mapsto z]) \cup \theta'_j[y \mapsto z], W \cup \{(m, n)\}) \\ \text{for } m &\xrightarrow{b_i, c?x, \theta_i}_L m_i \xrightarrow{b_{il}, \varepsilon, \theta_{il}}_L m_{il}, \quad n \xrightarrow{b'_j, c?y, \theta'_j}_L n_j \\ \text{in } \bigwedge_i (b_i &\rightarrow \bigvee_j (b'_j \wedge \forall z (\bigvee_k (b'_{jk} \theta'_j[y \mapsto z] \wedge B_{i,jk})))) \wedge \\ &\bigwedge_j (b'_j \rightarrow \bigvee_i (b_i \wedge \forall z (\bigvee_l (b_{il} \theta_i[x \mapsto z] \wedge B_{il,j})))) \end{aligned}$$

The correctness of the modified algorithm can be guaranteed by a similar theorem as Theorem 5.1. These algorithms reduce the problem of checking bisimulations to the problem of computing the greatest solution of predicate equation systems. Unlike boolean equation systems, which can be effectively solved [7,11], in general, the greatest solution of a predicate equation system can not be computed automatically. However, one can reason about the greatest solution using data domain knowledge, as in the above example.

6 The early case

In this section we briefly outline how to deal with the early case by modifying the previous semantic theory and checking algorithms. It turns out that only those parts concerning input actions need changing.

For the early ground operational semantics, we only need to change the rule concerning guarded input with assignment in Definition 3.1 to

$$\frac{m \xrightarrow{b, c?x, \theta} n}{m_\rho \xrightarrow{c?v} n_{\theta\rho\{x \mapsto v\}}} \quad \rho \models b, \quad v \in Val$$

Hence the definition of early strong ground bisimulation can be obtained from Definition 3.2 by merging the first clause into the second one.

The early symbolic operational semantics is the same as the late case, while the definition of early strong symbolic bisimulation is obtained from Definition 3.4 by replacing the condition $fv(B) \subseteq fv(t, u, b)$ with $fv(B) \subseteq fv(t, u, b) \cup bv(\alpha)$. It means that the early version allows to partition over the value space for an input variable, while in the late version this is prohibited.

Because the early bisimulation allows a single input move from one process to be matched by several such moves from the other, so there is no danger for input moves to absorb τ moves after them. Hence the early symbolic double transitions are defined as follows.

Definition 6.1 The early symbolic double transitions are defined as the least relations over nodes satisfying the following rules:

- $m \xrightarrow{true, \varepsilon, \emptyset} m.$
- If $m \xrightarrow{b, \alpha, \theta} n$ and $\alpha \in \{\tau, c!e\}$, then $m \xrightarrow{b, \alpha, \theta} n.$
- If $m \xrightarrow{b, c?x, \theta} n$, then $m \xrightarrow{b, true, c?x, \theta} n.$
- If $m \xrightarrow{b, \tau, \theta} n$ and $\alpha \in \{\varepsilon, \tau, c!e\}$, then $m \xrightarrow{b \wedge b' \theta, \alpha \theta, \theta' \theta} n.$
- If $m \xrightarrow{b, \tau, \theta} n$ and $\alpha \in \{\varepsilon, \tau, c!e\}$, then $m \xrightarrow{b \wedge b_1 \theta, b_2 \theta \setminus x, c?x, \theta'(\theta \setminus x)} n.$
- If $m \xrightarrow{b, \alpha, \theta} n$ and $\alpha \in \{\varepsilon, \tau, c!e\}$, then $m \xrightarrow{b \wedge b' \theta, \alpha, \theta' \theta} n.$
- If $m \xrightarrow{b_1, b_2, c?x, \theta} n$, then $m \xrightarrow{b_1, b_2 \wedge b' \theta, c?x, \theta' \theta} n.$

For the path: $m_0\{x\} \xrightarrow{x>0, \tau, x:=x+1} m_1\{x\} \xrightarrow{true, c?x, \emptyset} m_2\{x\} \xrightarrow{x<0, \tau, x:=x+4} m_3\{x\},$

we must introduce an early symbolic double transition $m_0\{x\} \xrightarrow{x>0, x<0, c?x, x:=x+4} m_3\{x\}$ to distinguish the free and bound occurrences of x .

The early ground double arrow is defined similarly as Definition 4.2, except that it is in terms of early symbolic double transitions rather than late symbolic double transitions and the rule for input action must be modified to

$$\frac{m \xrightarrow{b_1, b_2, c?x, \theta} E n}{m \xrightarrow{c?v} E n_{\theta(\rho\{x \mapsto v\})}} \quad \rho \models b_1 \wedge b_2[v/x], \quad v \in Val$$

Hence the definition of early weak ground bisimulation can be obtained from Definition 4.3 by merging the first clause into the second one.

For the same reason, to define the early symbolic double arrow, the rule for input action is

$$\frac{m \xrightarrow{b_1, b_2, c?x, \theta} E n}{m_{\sigma} \xrightarrow{b_1 \sigma \wedge b_2 \sigma[x \mapsto z], c?z} E n_{\theta(\sigma[x \mapsto z])}} \quad \text{where } z \notin fv(m_{\sigma}) \cup fv((fv(n) - \{x\})\theta\sigma)$$

The early weak symbolic bisimulation differs from the late one only for input moves: now input moves can absorb τ moves after them. As a consequence, the two partitions used in Definition 4.6 can be combined into one, hence the special clause for input moves can be merged into the other one. The same for early symbolic observation congruence.

Fortunately, we can also establish equivalent characterisations of early weak symbolic bisimulation (resp. early symbolic observation congruence) only in terms of early symbolic double arrows $\Longrightarrow_E$ (resp. $\Longrightarrow_{+E}$). Thus the only difference between early strong and early weak symbolic bisimulation is that the strong version is defined in terms of symbolic single arrows $\xrightarrow{b, \alpha}$, while the weak version is defined in terms of early symbolic double arrows $\Longrightarrow_{E}^{b, \alpha}$.

The algorithm for late strong bisimulation can also be modified to check early strong bisimulation. As may be expected, we only need to change $match_{c?}$ to:

$$\begin{aligned} match_{c?}(m, n, W) = \\ \text{let } z = newVar() \\ B_{ij} = close(m_i, n_j, \theta_i[x \mapsto z] \cup \theta'_j[y \mapsto z], W \cup \{(m, n)\}) \\ \text{for } m \xrightarrow{b_i, c?x, \theta_i} m_i, \quad n \xrightarrow{b'_j, c?y, \theta'_j} n_j \\ \text{in } \forall z (\bigwedge_i (b_i \rightarrow \bigvee_j (b'_j \wedge B_{ij})) \wedge \bigwedge_j (b'_j \rightarrow \bigvee_i (b_i \wedge B_{ij}))) \end{aligned}$$

Given two STGAs $\mathcal{G}$ and $\mathcal{H}$, we first construct the corresponding early SOGAs (resp. early SCGAs) $\mathcal{G}'$ and $\mathcal{H}'$. The checking algorithm for early weak symbolic bisimulation (resp. early symbolic observation congruence) is similar to the one presented in Figure 4 but working on $\mathcal{G}'$ and $\mathcal{H}'$. As indicated above, we need not only to replace $match_{\tau}$ with $match_{\varepsilon}$ but also to modify $match_{c?}$ to:

$$\begin{aligned} match_{c?}(m, n, W) = \\ \text{let } z = newVar() \\ B_{ij} = close(m_i, n_j, \theta_i([x \mapsto z]) \cup \theta'_j([y \mapsto z]), W \cup \{(m, n)\}) \\ \text{for } m \xrightarrow{b_{i1}, b_{i2}, c?x, \theta_i} E m_i, \quad n \xrightarrow{b'_{j1}, b'_{j2}, c?y, \theta'_j} E n_j \\ \text{in } \forall z (\bigwedge_i (b_{i1} \wedge b_{i2}[x \mapsto z] \rightarrow \bigvee_j (b'_{j1} \wedge b'_{j2}[y \mapsto z] \wedge B_{ij})) \wedge \\ \bigwedge_j (b'_{j1} \wedge b'_{j2}[y \mapsto z] \rightarrow \bigvee_i (b_{i1} \wedge b_{i2}[x \mapsto z] \wedge B_{ij}))) \end{aligned}$$

7. Conclusion

Based on the improved STGA model, We have presented algorithms to check strong/weak bisimulation equivalences and observation congruence for value-passing processes. These algorithms reduce the problems of checking bisimulations for STGAs to the problems of reasoning about the greatest solutions of predicate equation systems over data domain. We would like to investigate techniques for verifying properties concerning such solutions in the future. Currently we are implementing all these algorithms in standard ML, and hope to establish an automatic verification tool of practical usefulness for value-passing processes and the π -calculus.

References

1. G.Bruns. Distributed System Analysis with CCS. Prentice Hall,1996.
2. R.Cleaveland, J.Parrow and B.Steffen. The concurrency workbench: A semantics based verification tool for the verification of concurrent systems. ACM Transactions on Programming Language and Systems, Vol.15, No.1, pages 36-72, 1993.
3. J.Godskesen, K.Larsen and M.Zeeberg. Tav user manual. Report R89-19, Aalborg University, 1989.
4. M.Hennessy and H.Lin. Symbolic bisimulations. Theoretical Computer Science, 138:353-389, 1995.
5. M.Hennessy and H.Lin. Proof systems for message-passing process algebras. Formal Aspects of Computing, No 3, 1996.
6. B.Jonsson and J.Parrow. Deciding bisimulation equivalences for a class of non-finite-state programs. Information and computation, 107: 272-302, 1993.
7. K.G.Larsen.Efficient local correctness checking. In CAV'92, LNCS. Springer-Verlag, 1992.
8. Z.J.Li. Checking weak bisimulation equivalence and observation congruence for value-passing processes with finite symbolic transition graphs. Technical report 6/97, Changsha Institute of Technology, 1997.
9. Z.J.Li and H.W.Chen. Checking strong/weak bisimulation equivalences and observation congruence for the π -calculus. In ICALP'98, LNCS 1443. Springer-Verlag, 1998.
10. H.Lin. Symbolic transition graph with assignment. In CONCUR'96, LNCS 1119. Springer-Verlag, 1996.
11. X.Liu. Specification and decomposition in concurrency. Ph.D.thesis, Aalborg University, 1992.
12. R.Milner. Communication and Concurrency. Prentice-Hall,1989.