Towards A Modal Logic For π-Calculus *

Taolue Chen† Tingting Han Jian Lu
State Key Laboratory of Novel Software Technology,
Nanjing University, Nanjing, Jiangsu, P.R.China 210093

Abstract

The π-calculus is one of the most important mobile process calculi and has been well studied in literatures. Temporal logic is thought as a good compromise between description convenience and abstraction and can support useful computational applications, such as model-checking. In this paper, we use symbolic transition graph inherited from π-calculus to model concurrent systems. A wide class of processes, that is, the finite-control processes can be represented as finite symbolic transition graph. A new version modal logic for π-calculus, an extension of the modal μ-calculus with boolean expressions over names, and primitives for name input and output are introduced as an appropriate temporal logic for the π-calculus. Since we make a distinction between proposition and predicate, the possible interactions between recursion and first-order quantification can be solved. A concise semantics interpretation for our modal logic is given. Based on the above work, we provide a model checking algorithm for the logic. This algorithm follows the well-known Winskel's tag set method to deal with fixpoint operator. As for the problem of name instantiating, our algorithm follows the 'on-the-fly' style, and systematically employs schematic names. The correctness of the algorithm is shown.

Key Words: *π-calculus, Symbolic Transition Graph, Modal Logic, Model Checking Algorithm.*

1 Introduction

Over the last decades, various calculi of mobile processes, notably the π-calculus [7], have been the focus of research in concurrency theory. Because of the deficiency of using algebra method to model and describe related properties of systems, e.g. mobility, safety, a lot of research has focused on modal logic of calculus. Modal logic (temporal logic especially) is thought as a good compromise between description convenience and abstraction. In addition, many modal logics support useful computational applications, such as model-checking. As a powerful language to describe mobile and dynamic process networks, the problem of verifying general temporal and functional properties, cast in terms of the π-calculus, has been investigated in-depth. Some modal logic systems for π-calculus have been provided in the literatures, such as [8][1][3]. In [1][3] and more recently [5], Amadio and Dam introduced recursion into the modal logic via fixpoints, as in the propositional μ-calculus, thus has the ability to express properties for processes with infinite behaviors. These logic systems may be referred as π-μ-calculus. The main concern of these two papers is to formulate proof systems for deriving statements asserting whether a process satisfies a formula. Moreover, from our points of view, although the composition proof systems in the two papers are subtle, they are a little tedious, especially the completeness proof. In our opinions, this might due to the lack of adequate 'symbolic' information. The start point is to remedy this deficiency in some sense. It is well-known that symbolic technique has been widely used for name-passing calculi, especially for providing the complete proof system for bisimulation equivalence and devising efficient bisimulation checking algorithm, [6]. In this paper, we borrow the ideas from this technique and adapt it to devising model checking algorithms.

We present our main idea in brief. In this paper, first, we use symbolic transition graph to model concurrent systems. A wide class of processes, that is, the finite-control processes can be represented as a finite symbolic transition graph. Second, we introduce a new version modal logic for π-calculs, which is an extension of the modal μ-calculus with boolean expressions over names, and primitives for name input and output as an appropriate temporal logic for the π-calculus. Note

*Supported by NNSFC (60273034, 60233010), 863 Program (2001AA113110, 2002AA116010), 973 Program of China (2002CB312002), JSFC (BK2002203, BK2002409)

†Email: ctl@ics.nju.edu.cn

that in our logic, the 'bound output' modality is worth paying attention to. The fresh name quantification due to Pitts which is used in spatial logic [2] is subsumed implicitly, thus we must face the problem of possible interactions between recursion and first-order quantification. To solve this problem, we make a distinction between proposition and predicate in the syntax of logic system, thus a concise semantics interpretation for our modal logic can be given while the notion of 'property sets' is not needed. We defer more details to Section 2. As an essential application of our logic, in 3 we present a model checking algorithm. We follow the well-known Winskel's tag set method to deal with fixpoint operator since we prefer the local algorithm. As for the problem of name instantiating, our algorithm follows the 'on-the-fly' style, and systematically employs schematic names, that is, the fresh name set of current node and logical formula with one new name. The correctness of the algorithm is shown. Note that due to space restriction, the content on π-calculus and symbolic transition graph (STG) is omitted in this extended abstract, since they have little to do with our own contribution and can be found easily in common literatures. We refer the reader to [7][8][6] for more details. All of the notations of this paper follows them. Moreover, all proofs in this paper are omitted.

The rest of the paper is organized as follows: In Section 2, the modal logic is introduced and the semantics is given, some useful properties are also discussed in this section. The model checking algorithm is presented and its correctness is shown in Section 3. The paper is concluded with Section 4 where related work is also discussed.

2 Modal Logic System

2.1 Syntax

We assume a countably infinite set $\mathcal{V}$ of name variables ranged over by $x, y, z \ldots$, such that $\mathcal{V} \cap \mathcal{N} = \emptyset$. And we assume a countably infinite set $\mathcal{X}$ of *predicate variables*, ranged over by $X, Y, Z, \ldots$. Each predicate variable has been assigned an *arity* $n \in \omega$, written $X : n$. The syntax of the formula is defined by BNF as follows:

$$\begin{array}{rcl} \alpha & ::= & \tau \mid u?(x) \mid u?v \mid u!v \mid u!(x) \\ \phi & ::= & true \mid u = v \\ A, B & ::= & \phi \mid \Lambda(\tilde{u}) \mid \neg A \mid A \wedge B \mid \forall x.A \mid \langle \alpha \rangle A \\ \Lambda & ::= & X \mid (\tilde{x})A \mid \nu X.\Lambda \end{array}$$

where, $u, v \in \mathcal{N} \cup \mathcal{V}$.

The syntax is divided into two categories: *propositions* and *predicates*. Semantically, propositions denote sets of nodes in a STG (i.e. process terms), while predicates denote functions from sets of names to sets of nodes. For propositions, the operators are rather standard since it is adapted from well-known Hennessy-Milner Logic. A predicate is either a predicate variable X, or an abstraction $(\tilde{x})A$, or a greatest fixpoint $\nu X.\Lambda$. When forming an abstraction $(\tilde{x})A$, as our notation indicates, it is required that $\tilde{x}$ be a vector of distinct **name variables**. Then the arity of a predicate Λ is defined as: the arity of X if Λ has the form X or $\nu X.\Lambda$, or the length of $\tilde{x}$ if Λ has the form $(\tilde{x})A$. In abstractions and applications we always require arities to be matched properly.

In formulas of the form $\forall x.A$, $\langle u?(x) \rangle.A$, $\langle u!(x) \rangle.A$, $(\tilde{x})A$ and $\nu X.\Lambda$, the distinguished occurrences of x and X are binding, with the scope of the propositions A or predicate Λ. These introduce the notions of bound and free name variables as well as bound and free predicate variables in the usual way. The set of free names , free name variables and free predicate variables of a formula A are denoted by $fn(A)$, $fnv(A)$ and $fpv(A)$ respectively. Formulas that do not have free name variables are **name-closed**. Formulas that do not have free predicate variables are **predicate-closed**. A formula is **closed** if it is both name-closed and predicate-closed.

We defined on formulas the relation $\equiv_\alpha$ of α-congruence in the standard way, that is, as the least congruence identifying formulas modulo renaming of bound (name and predicate) variables. We will consider formulas always modulo α-congruence. Note that for formula, the notion of name substitution is extended to function from $\mathcal{N} \cup \mathcal{V}$ to $\mathcal{N}$, i.e. we allow the name variables to be replaced by names. Note that for convenience, we identify β-equivalence formulas, that is, $((\tilde{x})A)(\tilde{u})$ and $A[\tilde{u}/\tilde{x}]$.

The unary operator $\neg$ is negative. An occurrence of a predicate variable is positive if it is under an even number of negative operators. X occurs positively in a formula A if every occurrence of X in A is positive. Otherwise we say X occurs negatively in A. A fixpoint predicate $\nu X.\Lambda$ is **well-formed** if $fn(\Lambda) = fnv(\Lambda) = \emptyset$ and X occurs positively in Λ. Note that we require that predicate Λ has no **free name**, thus $n(\Lambda(\tilde{u}))$ and $fv(\Lambda(\tilde{u}))$ are totally determined by the actual parameter $\tilde{u}$, which is very important to the soundness of semantics. A formula is well-formed if every fixpoint subformula in it is well-formed. In the sequel, we only consider well-formed formulas.

As usual, in our modal logic system, we can de-

fine some standard derived connectives, e.g. $\neg$, $\vee$, $[\alpha]$, $\exists x.A$, $\mu X.\Lambda$. Since they are routine, we omit the details.

2.2 Semantics

Given STG G, by the concrete operational semantics rules, we can get a concrete graph denoted by $\mathcal{G}$. The semantics of formula is defined by assigning to each formula A a node set of $\mathcal{G}$, i.e. $[\![A]\!]$, namely all the nodes of $\mathcal{G}$ that satisfy the property denoted by A. For convenience, we denote n_σ by s and for any $s \equiv n_\sigma$, $s[c/b] \equiv n_{\sigma[b\mapsto c]}$. Since formulas may contain free name variables and free predicate variables, to interpret them we need name valuations and predicate valuations. A name valuation ρ is an extended version of substitution, which is a total mapping from $\mathcal{N} \cup \mathcal{V} \to \mathcal{N}$ with identity on $\mathcal{N}$. A predicate valuation ξ assigns to every predicate variable X of arity k a function $\xi(X) : \mathcal{N}^k \to \wp(\mathcal{G})$. As usual, the relation $\subseteq$ can be extended point-wise to functional space as follows: for each k, two functions $f^{(k)}, g^{(k)} : \mathcal{N}^k \to \wp(\mathcal{G})$, define $f^{(k)} \sqsubseteq g^{(k)}$ *iff* $f(\tilde{n}) \subseteq g(\tilde{n})$ for any $\tilde{n} \in \mathcal{N}^k$. Thus, the functional space $\mathcal{N}^k \to \wp(\mathcal{G})$ forms a complete lattice w.r.t. $\sqsubseteq$. The denotation of formulas is defined inductively in Figure.1.

If A is name-closed then $[\![A]\!]_{\rho;\xi}$ does not depend on ρ and will be written $[\![A]\!]_\xi$. Furthermore, if A is name-closed and predicate-closed, then $[\![A]\!]_{\rho;\xi}$ depends on neither ρ nor ξ and it will be written as $[\![A]\!]$. We will write $s \models A$ to denote $s \in [\![A]\!]$.

As in the case of first-order logic, the following lemma which relates substitutions and valuations is common, and will be used implicitly.

Lemma 1 *The following properties hold:*

(i) $[\![A[b/x]]\!]_{\rho;\xi} = [\![A]\!]_{\rho[x\mapsto b];\xi}$.

(ii) $[\![\Lambda[F/X]]\!]_{\rho;\xi} = [\![\Lambda]\!]_{\rho;\xi[X\mapsto\xi(F)]}$.

It is routine to show that for any formula A and B with $A \equiv_\alpha B$, $[\![A]\!]_{\rho,\xi} = [\![B]\!]_{\rho,\xi}$ for any ρ and ξ, which justifies our decision to identify α-equivalent formulas. Also, we can easily show the monotonicity of the semantics function Λ, since it is required that X occur positively in Λ. Thus, $\lambda f.[\![A]\!]_{\rho,\xi[X\mapsto f]}$ is a monotone functional over the complete lattice $(\{f : \mathcal{N}^k \to \wp(\mathcal{G})\}, \sqsubseteq)$. By Knaster-Tarski Theorem, we can draw the conclusion that $\nu X.\Lambda$ is the greatest fixpoint of $\lambda f.[\![A]\!]_{\rho,\xi[X\mapsto f]}$. The soundness of semantics can be obtained.

Now, we make some remarks on the choice of modality for the logical system. Generally speaking, there are two styles of syntax for the 'Hennessy-Milner logic' like systems for π-calculus. One is used by [8], the other is used by [3][5]. We follow the style of the former since from our point of view, it is clearer. However, the semantics is dramatically different. First, [8] lacks a modality for bound output, although the syntax $\langle\bar{a}(x)\rangle A$ exists in the logic system. It is not difficult to see that the semantics for $\langle\bar{a}(x)\rangle A$ does not coincide with the intuition very much. Second, the input modality in this paper coincides with the $\langle\bar{a}(x)\rangle^L$ in [8]. We don't introduce the corresponding modality for the other two 'input' modality because they can be rendered in our framework as follows:

$$\langle a(b)\rangle A \stackrel{def}{=} \exists x.\langle a?x\rangle A \quad \langle a(b)\rangle^E A \stackrel{def}{=} \forall x.\langle a?x\rangle A$$

The bound output modality needs more remarks. Note that our semantics for this modality coincides with Dam's though the syntax is different. To make this modality clearer, we consult to the fresh name quantification N and Dam's syntax a little. In fact,

$$\langle a!(x)\rangle A \stackrel{def}{=} \langle a\rangle \mathit{N}x.x \leftarrow A$$

We think reader who is familiar with N can easily understand this. We refer the reader to [2] for details. It is worth pointing out that as we mentioned in Section 1, such a quantification conveys difficulties when giving an interpretation though it is only implicit in our logic. The similar problems have been considered in [2]. As a remedy, [2] introduces the notion of *PSets* (Property sets). However, such a semantic device makes the semantics definition rather complex. Our solution is to make distinction between proposition and predicate, thus the possible interactions between recursion and first-order quantification can be solved. The advantage of our system lies in that the semantics of our logic is clearer and more concise. What's more, it is more favorable for model checking purpose. Also, it is worth pointing out that we need not introduce $\langle a!(b)\rangle$-like modality, since by the semantics, the choice of concrete name as the content of output action is immaterial, therefore, we use a variable instead of name.

Now, we set to establish some important results concerning the properties of logical formula, which is important for the model checking algorithm. Following [2], we use transposition as a useful tool to give some concise proof of properties concerning fresh names. The following definition extends the notion of transposition to predicate.

Definition 1 *Let θ be a transition. A function $f : \mathcal{N} \to \wp(\mathcal{G})$ is θ-preserving if $(f(n))\theta = f(n\theta)$ for any n. A valuation ξ is θ-preserving if $\xi(X)$ is θ-preserving for any X.*

$$
\begin{array}{rcl}
[\![\phi]\!]_{\rho;\xi} & = & \begin{cases} \mathcal{G} & \text{If } \rho \models \phi \\ \emptyset & \text{o.w.} \end{cases} \\
[\![A \wedge B]\!]_{\rho;\xi} & = & [\![A]\!]_{\rho;\xi} \cap [\![B]\!]_{\rho;\xi} \\
[\![\neg A]\!]_{\rho;\xi} & = & \mathcal{G} \backslash [\![A]\!]_{\rho;\xi} \\
[\![\langle \tau \rangle A]\!]_{\rho;\xi} & = & \{s \mid \exists s', s.t.\ s \xrightarrow{\tau} s' \wedge s' \in [\![A]\!]_{\rho;\xi}\} \\
[\![\langle u!v \rangle A]\!]_{\rho;\xi} & = & \{s \mid \exists s', s.t.\ s \xrightarrow{u\bar{\rho}v\rho} s' \wedge s' \in [\![A]\!]_{\rho;\xi}\} \\
[\![\langle u?(x) \rangle A]\!]_{\rho;\xi} & = & \{s \mid \exists s', s.t.\ s \xrightarrow{u\rho(b)} s' \wedge s'[c/b] \in [\![A]\!]_{\rho[x \mapsto c];\xi} \text{ for all } c \in \mathcal{N}\} \\
[\![\langle u!(x) \rangle A]\!]_{\rho;\xi} & = & \{s \mid \exists s', s.t.\ s \xrightarrow{u\bar{\rho}(b)} s' \wedge s'[c/b] \in [\![A]\!]_{\rho[x \mapsto c];\xi} \text{ for some } c \notin fn(s) \cup fn(A)\} \\
[\![\langle u?v \rangle A]\!]_{\rho;\xi} & = & \{s \mid \exists s', s.t.\ s \xrightarrow{u\rho(b)} s' \wedge s'[v\rho/b] \in [\![A]\!]_{\rho;\xi}\} \\
[\![\Lambda(\tilde{u})]\!]_{\rho;\xi} & = & [\![\Lambda]\!]_{\rho;\xi}(\rho(\tilde{u})) \\
[\![X]\!]_{\rho;\xi} & = & \xi(X) \\
[\![(\tilde{x})A]\!]_{\rho;\xi} & = & \lambda \tilde{y}.[\![A]\!]_{\rho[\tilde{x} \mapsto \tilde{y}];\xi} \\
[\![\nu X.\Lambda]\!]_{\rho;\xi} & = & \sqcup\{F : \mathcal{N}^k \to \wp(\mathcal{G}) \mid F \sqsubseteq [\![\Lambda]\!]_{\rho;\xi[F/X]}\}
\end{array}
$$

Figure 1. Interpretation of Formula

Lemma 2 *Given a transposition θ and a function $f : \mathcal{N} \to \wp(\mathcal{G})$, define $f^\theta : \mathcal{N} \to \wp(\mathcal{G})$, then the following properties hold:*

(i) f^θ is θ-preserving.

(ii) If $f \sqsubseteq g$ and g is θ-preserving, then $f^\theta \sqsubseteq g$.

Lemma 3 *Suppose ξ is θ-preserving, then the following properties hold:*

(i) $([\![A]\!]_{\rho;\xi})\theta = [\![A\theta]\!]_{\rho;\xi}$.

(ii) $[\![\Lambda]\!]_{\rho;\xi}$ is θ-preserving.

According to the semantics of $\forall x.A$ and $\langle a?(x) \rangle A$, to check if $P \in [\![\forall x.A]\!]$ requires to instantiate x with every name. However, as the following lemma demonstrates, it is sufficient to consider only the free names of A plus one fresh name. This finite characterization will be exploited in the model checking algorithm.

Lemma 4 *Suppose $c \notin fn(s, A)$, then the following properties hold:*

(i) $s \in [\![\forall x.A]\!]_{\rho;\xi}$ iff $s \in \bigcap_{k \in fn(A) \cup \{c\}} [\![A]\!]_{\rho[x \mapsto k];\xi}$.

(ii) $s \in [\![\langle u?(x) \rangle A]\!]_{\rho;\xi}$ iff there exists s' s.t. $s \xrightarrow{\rho(u)(b)} s'$ and $s'[k/b] \in [\![A]\!]_{\rho[x \mapsto k];\xi}$ for $k \in fn(A) \cup fn(s) \cup \{c\}$.

The semantics definition of the $\langle a!(x) \rangle A$ is stated in 'existential' style, i.e. $s \models \langle a!(x) \rangle A$ if there is **some** fresh name c and s', such that $s \xrightarrow{a(b)} s'$ and $s'[c/b] \models A[c/x]$. We give such a definition because from our point of view, it may coincide with our intuition of bound output and restriction operator better. However, since c is not free in either s or A, this particular choice of c should not matter. That is, any other name d with $d \notin fn(s, A)$ should equally do. Thus indeed the semantics can also be characterized 'universally'.

Lemma 5 *$s \in [\![\langle a!(x) \rangle A]\!]_{\rho;\xi}$ iff there exists s' s.t. $s \xrightarrow{\bar{a}(b)} s'$ and $s'[c/b] \in [\![A]\!]_{\rho[x \mapsto c];\xi}$ for **every** $c \notin fn(s, A)$.*

3 Model Checking Algorithm

In this section, we devote to providing a model checking algorithm for our modal logic. Based on the results in the last section, now the most challenging problem is to deal with the fixpoint operator. For propositional μ-calculus, many researchers have provided a lot of methods to solve this problem. We choose the so called local model checking algorithm since the global algorithm requires a prior construction of state space, which is impossible in our setting.

One of the notable features of such an algorithm is the mechanism used to keep track of unfolding fixpoint

formula. We follow the framework due to Winskel [9], sometimes referred as tag set method, i.e. we introduce *tags* into fixpoint formula, which remembers exactly which points of the model have been visited before. However, differing from [9], we lift it to the predicate case and in our algorithm, the tag sets will contain pairs $(\tilde{n}, s)$ of name vector and the node of STG. Formally, let $T = \{(\tilde{b_1}, s_1), \ldots, (\tilde{b_n}, s_n)\}$, where, $\tilde{n}_i$ $(1 \leq i \leq n)$ are vectors of the same length, say k and for $\forall i, j, i \neq j$, we have $\tilde{b}_i \neq \tilde{b}_j$. For any tag set T, we use λT to denote a function $\mathcal{N}^k \to \wp(\mathcal{G})$ defined as follows:

$$(\lambda T)(\tilde{b}) = \begin{cases} \bigcup_i \{s_i\} & \text{if } (\tilde{b}, s_i) \in T \\ \emptyset & \text{if o.w.} \end{cases}$$

Now, the fixpoint predicate $\nu X.\Lambda$ can be generalized to $\nu X.[T]\Lambda$, note that the X must have the same arity as T and the usage of T lies in recording what points of the model have been visited before thus is only a bookkeeping device. The definitions of $n(\nu X.[T]\Lambda)$, $fv(\nu X.[T]\Lambda)$ and $fpv(\nu X.[T]\Lambda)$ are the same as the corresponding definition for $\nu X.\Lambda$. Obviously, $\nu X.\Lambda$ can be covered as $\nu X.[]\Lambda$.

The denotation of $\nu X.[T]\Lambda$ is a simple extension for $[\![\nu X.\Lambda]\!]_{\rho;\xi}$ as follows:

$$[\![\nu X.[T]\Lambda]\!]_{\rho;\xi} = \sqcup\{F : \mathcal{N}^k \to \wp(\mathcal{G}) | F \sqsubseteq [\![\Lambda]\!]_{\rho;\xi[X \mapsto F]} \sqcup \lambda T\}$$

There now follows a technical lemma which is a generalization of the so called Reduction Lemma of [9], the essence of the tag set method.

Lemma 6 *(Reduction Lemma) Let $L = \mathcal{N}^k \to \wp(\mathcal{G})$ be a complete lattice and let ϕ be a monotone functional. Then for any $f \in L$,*

$$f \sqsubseteq \nu g.\phi(g) \quad \textit{iff} \quad f \sqsubseteq \phi(\nu g.(\phi(g) \sqcup f))$$

Since X occurs positively in Λ, $\lambda f.[\![\Lambda]\!]_{\rho;\xi[X \mapsto f]} \sqcup \lambda T$ is a monotonic functional over $\mathcal{N}^k \to \wp(\mathcal{G})$, and $[\![\nu X.[T]\Lambda]\!]_{\rho;\xi}$ is its greatest fixpoint. So, using Lemma 6, the following lemma can be easily proved.

Lemma 7 *If $s \notin \lambda T(\tilde{b})$, then*

$$s \in [\![\nu X.[T]\Lambda]\!]_{\rho;\xi}(\tilde{b}) \quad \textit{iff} \quad s \in [\![\Lambda[\nu X.[T \cup \{(\tilde{b}, s)\}]\Lambda/X]]\!]_{\rho;\xi}(\tilde{b})$$

Now, we present our algorithm as follows. The algorithm inputs a STG with the root n and substitution σ and a closed formula A. As usual, we denote m_σ as s. The algorithm returns *true* if $s = m_\sigma$ satisfies A, otherwise, it returns false. The pseudo-codes of the algorithm is presented in Figure.2. Recall that for a set of names $V \subset_{fin} \mathcal{N}$, function new$(V)$ returns the least name in $\mathcal{N} \backslash V$.

Now, we devote to proving the correctness of our algorithm. To establish the termination property of the algorithm, we need to bound on the number of names for model checking processFirst, we should point out that according to the rule for transferring process terms to STG, all bound names in a STG are different. And since we adopt the α-equivalence for formula, we can assume the bound name variables in a formula are also different. Now, we write $N_{\mathcal{G}}$ for the number of names (including free and bound names) contained in the nodes of $\mathcal{G}$ and N_A for the number of names and name variables contained in A. Note that names in tag set of the formula does not be included, since it only contributes as a bookkeeping. The following lemma is important, by which we can conclude that provided that each term only appears once in each tag set (just as in our algorithm), the size of tag set is bounded since the STG we consider is finite.

Lemma 8 *For each recursive call of check, with caller parameter (s, A) and the callee parameter (s',A'), $N_{s'} + N_{A'} \leq N_s + N_A$*

We now use this fact to give a well-founded ordering to formula. We write $A \ll_{\mathcal{G}} A'$ iff A' is not a fixpoint formula and A is a proper sub-formula of A', otherwise A is the form $(\Lambda[\nu X.[T \cup \{\tilde{b}, s\}]\Lambda/X])(\tilde{b})$ and A' is $(\nu X.[T]\Lambda)(\tilde{b})$ where $(\tilde{b}, s) \notin T$ and T contains only nodes from $\mathcal{G}$. We aim to show that the transitive closure $\ll^+_{\mathcal{G}}$ of this relation is a well-founded order whenever $\mathcal{G}$ is finite.

Lemma 9 *If $\mathcal{G}$ is finite, then $\ll^+_{\mathcal{G}}$ is a well-founded order.*

Theorem 1 *For any STG with root r and closed formula A, the following properties hold:*

(i) check(r, A) terminates;

(ii) check$(r, A) = true$ iff $r \in [\![A]\!]$.

4 Conclusion

In this section, we conclude our work and discuss the future work. This paper deals with temporal logic for mobile concurrent system and related model-checking algorithm. A new version modal logic system, an extension of the modal μ-calculus with boolean expressions over names, and primitives for name input and output is introduced as an appropriate temporal logic for the π-calculus. Also our logical system can be seen as the extension for the modal logic in [8]. We give a concise semantics interpretation for our modal logic

$$
\begin{array}{l}
\mathrm{check}(m_\sigma, \phi) \stackrel{def}{=} Ev(\phi\sigma); \\
\mathrm{check}(m_\sigma, \neg A) \stackrel{def}{=} \neg\mathrm{check}(m_\sigma, A); \\
\mathrm{check}(m_\sigma, A \wedge B) \stackrel{def}{=} \mathrm{check}(m_\sigma, A) \wedge \mathrm{check}(m_\sigma, B); \\
\mathrm{check}(m_\sigma, \forall x.A) \stackrel{def}{=} \bigwedge_{b \in fn(m_\sigma, A)} \mathrm{check}(m_\sigma, A[b/x]) \wedge \\
\qquad\qquad\qquad\qquad \mathrm{check}(m_\sigma, A[\mathrm{new}(fn(m_\sigma, A))/x]); \\
\mathrm{check}(m_\sigma, \langle\tau\rangle A) \stackrel{def}{=} \bigvee_{m \stackrel{\phi,\tau}{\mapsto} n} Ev(\phi\sigma) \wedge \mathrm{check}(n_\sigma, A); \\
\mathrm{check}(m_\sigma, \langle a?(x)\rangle A) \stackrel{def}{=} \bigvee_{m \stackrel{\phi, b(c)}{\mapsto} n} Ev(\phi\sigma \wedge [b\sigma = a]) \\
\qquad\qquad\qquad\qquad \wedge \bigwedge_{d \in fn(m_\sigma, A) \cup \mathrm{new}(fn(m_\sigma, A))} \mathrm{check}(n_{\sigma[c \mapsto d] \uparrow fn(n)}, A[d/x]); \\
\mathrm{check}(m_\sigma, \langle a!c\rangle A) \stackrel{def}{=} \bigvee_{m \stackrel{\phi, \tilde{b}d}{\mapsto} n} Ev(\phi\sigma \wedge [a = b\sigma] \wedge [c = d\sigma]) \wedge \mathrm{check}(n_\sigma, A); \\
\mathrm{check}(m_\sigma, \langle a?b\rangle A) \stackrel{def}{=} \bigvee_{m \stackrel{\phi, c(d)}{\mapsto} n} Ev(\phi\sigma \wedge [a = c\sigma]) \wedge \mathrm{check}(n_{\sigma[d \mapsto b] \uparrow fn(n)}, A); \\
\mathrm{check}(m_\sigma, \langle a!(x)\rangle A) \stackrel{def}{=} \bigvee_{m \stackrel{\phi, \tilde{b}(c)}{\mapsto} n} Ev(\phi\sigma \wedge [b = a\sigma]) \\
\qquad\qquad\qquad\qquad \wedge \mathrm{check}(n_{\sigma[c \mapsto \mathrm{new}(fn(m_\sigma, A))]}, A[\mathrm{new}(fn(m_\sigma, A))/x]); \\
\mathrm{check}(m_\sigma, (\nu X.[T]\Lambda)(\tilde{b})) = \begin{cases} true & \text{if } (\tilde{b}, m_\sigma) \in T \\ \mathrm{check}(m_\sigma, (\Lambda[\nu X.[T \cup \{(\tilde{b}, m_\sigma)\}]\Lambda/X])(\tilde{b})) & \text{o.w.} \end{cases};
\end{array}
$$

Figure 2. Local Model Checking Algorithm

by making a distinction between proposition and predicate, thus the possible interactions between recursion and first-order quantification can be solved. Based on the above work, a local model checking algorithm for the logic is presented in this paper. We follow the well-known Winskel's tag set method to deal with fixpoint operator. As for the problem of name instantiating, our algorithm follows the 'on-the-fly' style, and systematically employs schematic names. The correctness of the algorithm is shown.

There are several directions for further research. How to improve efficiency of our algorithm, is an interesting problem. Also, we are investigating how to generate information diagnosis messages which will be useful in debugging a system when the answer returned by the algorithm is 'no'. Maybe we should apply symbolic technique in more depth.

References

[1] R.M.Amadio, M.Dam. Toward a Modal Theory of Types for the π Calculus. Proc. Formal Techniques in Real Time and Fault Tolerant Systems 96, Uppsala. SLNCS 1135, 1996.

[2] L.Caires, L.Cardelli. A Spatical Logic for Concurrency (Part I). TACS'2001, Lecture Notes in Computer Science 2215, pp.1-30, Springer, 2001.

[3] M.Dam. Model Checking Mobile Processes. Information and Computation 129: 25-51, 1996.

[4] M.Dam. On the Decidability of Process Equivalences for the pi-Calculus. Theoretical Computer Science 183, pp. 215-228, 1997.

[5] M.Dam. Proof systems for π-Calculus Logics. To appear de Queiroz (ed.), "Logic for Concurrency and Synchronisation", Studies in Logic and Computation, Oxford Univ Press, 2003.

[6] Z.Li, H.Chen. Checking Strong/weak Bisimulation Equivalence and Observation Congruence for the π-calculus. In ICALP'98, Lecture Notes in Computer Science 1443, pp.707-718, Springer, 1998.

[7] R.Milner, J.Parrow, D.Walker. A Calculus of Mobile Process, part I/II. Journal of Information and Computation, 100:1-77, Sept.1992.

[8] R.Milner, J.Parrow, D.Walker. Modal Logics for Mobile Process. Theoretical Computer Science, 114:149-171, 1993.

[9] G.Winskel. A Note on Model Checking the Modal μ-calculus. Theoretical Computer Science 83:157-167, 1991.