Journal of Computers Vol. 29 No. 5, 2018, pp. 142-159
doi:10.3966/199115992018102905012

# Partial-order Algorithm of Model Checking in $\mu$-Predicate Ambient Logic

Hua Jiang[1], Rong-De Lin[2], Fu-Min Zou[3], Ling-Xiang Li[4]*

[1] Key Lab of Granular Computing, Minnan Normal University, ZhangZhou, China
sg_jh@126.com

[2] School of Mathematical Science, Huaqiao university, QuanZhou, China
linrd@hqu.edu.cn

[3] Fujian Provincial Key Laboratory of Big Data Mining and Applications, Fuzhou, Fujian, China, 350008
fmzou@fjut.edu.cn

[4] School of Electronics and Information Engineering, Hunan University of Science and Engineering, Yongzhou, China, 425199
lilingxiang2013@hotmail.com

Received 7 June 2017; Revised 7 November 2017; Accepted 7 December 2017

**Abstract.** This paper introduces a kind of partial-order algorithm of model-checking in finite-control mobile ambients against μ-predicate ambient logic(Ambient logic based on first-order μ-calculus). Based on Tarski's fixpoint theorem, nested predicate equations and Block Dependency Graph, a group of partial ordering relation between inter-mediate results in the process of model checking is given, and thus a kind of local model-checking algorithm is proposed. To our knowledge, this is the first algorithm which exponent of time complexity is d/2+2, the exponent of space complexity is d/2 (d is the alternate nesting depth of fixpoint operator in the formula) and this is the third model-checking algorithm for predicate ambient logic with recursion. This paper's contributions are: (1) getting a group of partial ordering relation between intermediate results in the computing process of model checking for μ-calculus first-order predicate ambient logic based on Tarski's fixpoint theorem; (2) using the partial ordering relation to design an algorithm for model checking in mobile ambients; (3) analyzing the complexity of the algorithm.

**Keywords:** algorithms and complexity, mobile ambients, model checking, predicate ambient logic

## 1 Background

Model checking technique is widely applied in the design and verification of a finite-control concurrent system. Model checking algorithm can be divided into two categories: global checking algorithm and local checking algorithm. The former, by given a logic expression, gets all the states that satisfy the logic expression in the system, and the latter, by given a logic expression and a state in the system, judges whether the state satisfy the logic expression or not.

The ambient calculus [1] is a formal model for distributed mobile computing system, which tree-like structure induced by ambients and nested sub-ambients may fundamentally characterize the spacial properties of a mobile computation environment. As the rapid growth of mobile computing environments in recent years, security [2] and privacy [3] in such environments are the hot spots in today's research.

Ambient logic is firstly proposed in [4], which can specify both temporal and spacial properties in the evolution of a process. Bound and fresh name quantifiers are introduced to ambient logic in [5]. Fixpoint

* Corresponding Author

operator is introduced to some sub-fragments of ambient logic in [6-7], but these sub-logics do not contain bound quantifier. Ambient logic based on first-order μ-calculus (μ-predicate ambient logic) is defined in [8-9], which contains fresh name quantifier and fixpoint operator, and uses predicate variables to form fixpoint formulas, thus simplifies the semantics of logic formulas.

On the model-checking of mobile ambients, several kinds of fragments for finite ambient calculus are discussed in reference [4, 10]. The complexity of checking algorithm is analyzed in [11]. The finite-control ambient calculus is proposed in [12] which provides the model checking algorithm against the finite-control ambient calculus. But ambient-based logics used in [4, 10, 12] are finite and has no recursion.

Actually, the model checking algorithm for the finite-control mobile ambients against μ-predicate ambient logic belongs to local model checking algorithm. The first algorithm is provided in reference [8], which points out that the algorithm is decidable if the logic formula excludes the operator "$\triangleright$", but does not consider the efficiency of the algorithm too much. The second algorithm following reference [8] is studied in reference [13] which exponent of time complexity is relation to the formula alternate nesting depth of fixpoint operator.

On the other hand, in recent years, Fabio and Giacoma [14], Brodo [15], Aman and Ciobanu [16], Ali et al. [17], Unal and Caglayan [18], Bodei et al. [19], Bogdan and Gabriel [20] and Siewe [21] extends mobile ambients from different aspects, improve their expressing capability and mobile computing capability, but do not study the algorithm of model checking too much.

In this paper the author, based on Tarski's fixpoint theorem, studies the partial ordering relation among the intermediate results in the computation process of model checking of μ-predicate ambient logic, and designs a kind of high performance algorithm, and further algorithm complexity analyzing presents that it is close to the demand of practical application.

## 2 Ambient Calculus

In this session, concepts and terminologies of Ambient Calculus [1] closely related to this paper are given. Let $N$ be the countable set of names, using *n*, *m*, ... to express the elements, $W$ be the countable set of process variables, using $X, Y, Z, \ldots$ to express, BNF of ambient calculus grammar is defined as follows:

$M ::=$ capabilities

$in\ n$ can enter $n$ | $out\ n$ can exit $n$ | $open\ n$ can open $n$ | $\varepsilon$ null path | $M.M$ composite path

$P, Q ::=$ processes

$0$ inactivity | $P \mid Q$ composition | $(\nu n)P$ name restriction | $n[P]$ ambient |

$M.P$ capability action | $(n).P$ input | $\langle n \rangle$ output | $X$ variable | $\mathrm{fix}X.P$ recursion

Free names, bound names, free variables and bound variables in process *P* are labelled as *fn*(*P*), *bn*(*P*), *fv*(*P*) and *bv*(*P*), the process that *fv*(*P*) is null is called closed process, the set of all closed processes is labelled as $\mathcal{P}$. If only the bound names and bound variables of both processes are different, then both processes are called $\alpha$-equivalent, the semantics of process with $\alpha$-equivalent are the same, *P*[*m/n*] is labelled as the result of substitution of *m* for free occurrence of *n* in process *P*. This paper only considers model checking of the finite-control process.

The semantics of ambient calculus is defined by the following structural congruence relation ($\equiv$) and reduction relation ($\rightarrow$).

Structural congruence relation ($\equiv$) of ambient calculus is defined as follows:

$StrRefl \quad P \equiv P$ $\qquad$ $StrSymm \quad P \equiv Q \Rightarrow Q \equiv P$

$StrParZero \quad P \mid 0 \equiv P$ $\qquad$ $StrTrans \quad P \equiv Q, Q \equiv R \Rightarrow P \equiv R$

$StrparComm \quad P \mid Q \equiv Q \mid P$ $\qquad$ $StrParAss \quad P \mid (Q \mid R) \equiv (P \mid Q) \mid R$

$StrResZero \quad (\nu n)0 \equiv 0$ $\qquad$ $StrResPar \quad (\nu n)(P \mid Q) \equiv P \mid (\nu n)Q \quad if\ \ n \notin fn(P)$

$StrResRes \quad (\nu n)(\nu m)P \equiv (\nu m)(\nu n)P$ $\qquad$ $StrResAmb \quad (\nu n)(m[P]) \equiv m[(\nu n)P] \quad if\ \ n \neq m$

$StrFixSelf \quad fixX.X \equiv 0$ $\qquad$ $StrRec \quad fixX.P \equiv P[fixX.P / X]$

Reduction relation ($\rightarrow$) of ambient calculus is defined as follows:

$Red\ In \quad n[in\ m.P \mid Q] \mid m[R] \rightarrow m[n[P \mid Q] \mid R]$ $\qquad$ $Red\ Out \quad m[n[out\ m.P \mid Q] \mid R] \rightarrow n[P \mid Q] \mid m[R]$

$Red\ Open \quad open\ n.P \mid n[Q] \rightarrow P \mid Q$ $\qquad$ $Red\ I/O \quad (n).P \mid \langle q \rangle \rightarrow P\{q/n\}$

$Red\ Res \quad P \rightarrow Q \Rightarrow (vn)P \rightarrow (vn)Q$ $\qquad$ $Red\ Par \quad P \rightarrow Q \Rightarrow P \mid R \rightarrow Q \mid R$

$Red\ Amb \quad P \rightarrow Q \Rightarrow n[P] \rightarrow n[Q]$ $\qquad$ $Red\ \equiv \quad P \equiv P', p' \rightarrow Q', Q' \equiv Q \Rightarrow P \rightarrow Q$

Considering the decidability of model checking algorithm, this paper only considers model checking of the finite-control process.

## 3 Ambient Logic

Ambient logic considered in this paper is first-order μ-calculus logic for mobile ambients put forward in reference [9]. Let $V$ as countable infinite set of name variables, using *x*, *y*,... to express the elements, then $V \cap N = \phi$, $x$ is countable infinite set of predicate variables, using *X*, *Y*,... to express the elements.

The BNF of ambient logic is defined as follows ($n \in V \cup N$):

$$A, B ::= \qquad propositions$$

$$\mathrm{T} \mid \perp \mid 0 \mid A \wedge B \mid A \vee B \mid A \mid B \mid A \triangleright B \mid n[A] \mid A@n \mid$$

$$n®A \mid A \oslash n \mid \boxdot A \mid \diamond A \mid \forall x.A \mid \exists x.A \mid Иx.A \mid F(\overline{n})$$

$$F ::= \qquad predicates$$

$$X \mid vX.F \mid \mu X.F \mid (\overline{x})A$$

$F(\overline{n})$ is predicate application, $(\overline{x})A$ is proposition abstraction.

Ambient logic formula can be divided into two categories: proposition and predicate. Each predicate variable has a certain arity to express the numbers of formal parameter for predicate variable. The arity of predicate is defined as: if the form of $F$ is $X$ or $\sigma X.F'$, then the arity of $F$ equals to that of $X$; if the form of $F$ is $(\overline{x})A$, then the arity of $F$ equals to the length of $\overline{x}$. For the abstraction of proposition or the application of predicate, matching arities are needed. In ambient logic, proposition is explained as the element in $2^P$, the predicate whose arity is *k* is explained as the function of $N^k \rightarrow 2^P$.

The value space of predicate variable of arity k is $\Omega = \{f \mid f : N^k \rightarrow 2^P, k \in \mathbb{N}\}$, inclusion relation $\subseteq$ in sets of $2^P$ extends to the space of function *k* point by point.

The definition of partial ordering relation $\subseteq$ of $\Omega$ is $f^{(k)} \subseteq g^{(k)}\ if\ f\ \forall \overline{u} \in N^k \Rightarrow f^{(k)}(\overline{u}) \subseteq g^{(k)}(\overline{u})$. The definition of $\bigcup$ or $\bigcap$ is $\forall \overline{u} \in N^k \Rightarrow (f^{(k)} \bigcup g^{(k)})(\overline{u}) = f^{(k)}(\overline{u}) \bigcup g^{(k)}(\overline{u})$ or $\forall \overline{u} \in N^k \Rightarrow (f^{(k)} \bigcap g^{(k)})(\overline{u}) = f^{(k)}(\overline{u}) \bigcap g^{(k)}(\overline{u})$ respectively, thus $<\Omega, \subseteq>$ forms a complete lattice. Label $\rho$ as the valuation environment of names ($v \bigcup N$ is full map on $N$ and is reflexive on $N$), label $\rho[u/x]$ as the value of $\rho$ at *x* is corrected into *u* and others are unchanged. Label $\xi$ as the valuation environment of predicate (to assign a function to any predicate variable of arity *k*, $(X) : N^k \rightarrow 2^P$), $\xi[f/X]$ shows modifying the value of $\xi$ at *X* into *f*, and others are unchanged. If $x \subseteq x'$ and $\xi'$ is the environment of $x'$, then $\xi[\xi']$ shows changing the value of $\xi$ at $x'$ into the corresponding value of $\xi'$ and others are unchanged. The definition of the partial ordering relation $\subseteq$ between ambients is $\xi_1 \subseteq \xi_2\ if\ f\ \forall\ X \in x, \xi_1(X) \subseteq \xi_2(X)$ all environments form a complete lattice under $\subseteq$.

The semantic of ambient logic formula is defined as below:

$[\![\mathrm{T}]\!]\xi\rho = P$

$[\![\perp]\!]\xi\rho = \phi$

$[\![0]\!]\xi\rho = \{P \mid P \equiv 0\}$

$[\![A \wedge B]\!]\xi\rho = [\![A]\!]\xi\rho \bigcap [\![B]\!]\xi\rho$

$[\![A \vee B]\!]\xi\rho = [\![A]\!]\xi\rho \bigcup [\![B]\!]\xi\rho$

$[\![A \mid B]\!]\xi\rho = \{P \mid P \equiv Q \mid R\ and\ Q \in [\![A]\!]\xi\rho, R \in [\![B]\!]\xi\rho\}$

$[\![A \triangleright B]\!]\xi\rho = \{P \mid if\ Q \in [\![A]\!]\xi\rho\ then\ P \mid Q \in [\![B]\!]\xi\rho\}$

$\llbracket n[A]\rrbracket\xi\rho = \{P \mid P = \rho(n)[Q]\ \mathit{for\ some}\ Q \in \llbracket A\rrbracket\xi\rho\}$
$\llbracket A@n\rrbracket\xi\rho = \{P \mid \rho(n)[P]\ f \in \llbracket A\rrbracket\xi\rho\}$
$\llbracket n®A\rrbracket\xi\rho = \{P \mid P \equiv (\nu\rho(n))Q\ \mathit{for\ some}\ Q \in \llbracket A\rrbracket\xi\rho\}$
$\llbracket A \oslash n\rrbracket\xi\rho = \{P \mid (\nu\rho(n))P \in \llbracket A\rrbracket\xi\rho\}$
$\llbracket \boxdot A\rrbracket\xi\rho = \{P \mid (\forall Q, P \to Q \Rightarrow Q \in \llbracket A\rrbracket\xi\rho\}$
$\llbracket \diamond A\rrbracket\xi\rho = \{P \mid \exists Q, P \to Q \wedge Q \in \llbracket A\rrbracket\xi\rho\}$
$\llbracket \forall_x.A\rrbracket\xi\rho = \bigcap_n \llbracket A\rrbracket\xi\rho(n/x)$
$\llbracket \exists_x.A\rrbracket\xi\rho = \bigcup_n \llbracket A\rrbracket\xi\rho(n/x)$
$\llbracket Иx.A\rrbracket\xi\rho = \bigcup_{n\notin fn(A)} \{P \mid P \in \llbracket A\rrbracket\xi\rho(n/x)\ \mathit{and}\ n \notin fn(P)\}$
$\llbracket F(\overline{n})\rrbracket\xi\rho = \llbracket F\rrbracket\xi\rho(\overline{n})$
$\llbracket X\rrbracket\xi\rho = \xi(X)$
$\llbracket \nu X.F\rrbracket\xi\rho = \bigcup\{f \mid f \subseteq \llbracket F\rrbracket\xi[f/X]\rho\}$
$\llbracket \mu X.F\rrbracket\xi\rho = \bigcap\{f \mid f \supseteq \llbracket F\rrbracket\xi[f/X]\rho\}$
$\llbracket (\overline{x})A\rrbracket\xi\rho = \lambda\overline{n}.\llbracket A\rrbracket\xi\rho[\overline{n}/\overline{x}]$

Considering the decidability of model checking algorithm, formula $A$ discussed in this paper satisfies (1) $A$ excludes operator "$\rhd$"; (2) $A$ excludes free name variable and free predicate variable.

## 4 Nested Predicate Equation Systems

The Nested Predicate Equation Systems and the equivalent of denotation semantics between Ambient logic formula and nested predicate equation systems are studied in [13], which shows that nested predicate equation system is more general expressive than ambient logic formula [13]. In this session, after introducing the grammar and semantics of Nested Predicate Equation System closely related to this paper, the algorithm translating Ambient logic formula into Nested Predicate Equation Systems is given. And then a kind of Automata Composed with Nested Predicate Equation Systems and mobile ambients process, namely Block Dependency Graph (BDG) is proposed, and BDG constructing algorithm and BDG computing algorithm are presented as well.

### 4.1 Grammar of Nested Predicate Equation System

Predicate equation with fixpoint is $X(\overline{x}) =_\sigma \phi$, $X$ is predicate variable, $\overline{x}$ is vector of name variables and $fnv(\phi) = \{\overline{x}\}$, $\sigma = \{\mu, \nu\}$, $\phi$ is the proposition without fixpoint.

Nested predicate equation system BNF is defined as below:

$$\varepsilon ::= \in \mid X(\overline{x}) =_\sigma \phi)\varepsilon \mid$$

$\in$ is empty sequence. In nested predicate equation system $\varepsilon$¸ predicate variables appeared from the left of each equation are different from each other. Predicate variable set appeared from the left of each equation in $\varepsilon$ is labelled as $lhs(\varepsilon)$, the predicate variable is bounded in $\varepsilon$. Predicate variable set appeared from the right of each equation in $\varepsilon$ is labelled as $rhs(\varepsilon)$, the variable in $rhs(\varepsilon) - lhs(\varepsilon)$ is free predicate variable in $\varepsilon$. For equation system $\varepsilon, \varepsilon'$, if $lhs(\varepsilon) \bigcap lhs(\varepsilon') = \varnothing, \varepsilon :: \varepsilon'$ is a new equation system formed by connecting two equation systems.

### 4.2 Semantics of Nested Predicate Equation System

Let $\varepsilon$ be a nested predicate equation system, $\xi$ is environment, $fnv(\phi) = \{\overline{x}\}$, the semantics $\|\varepsilon\|\xi$ of $\varepsilon$ in $\xi$ is defined as below:

If $\varepsilon \equiv \in$, then $\|\varepsilon\|\xi = \xi$;

If $\varepsilon \equiv (X(\overline{x}) =_{\nu} \phi)\varepsilon'$, then $\| \varepsilon \| \xi = \bigcup \{ f \mid f \subseteq [\![ (\overline{x})\phi ]\!] \| \varepsilon' \| \xi[f / X] \}$

If $\varepsilon \equiv (X(\overline{x}) =_{\sigma} \phi)\varepsilon'$, then $\| \varepsilon \| \xi = \bigcap \{ f \mid f \supseteq [\![ (\overline{x})\phi ]\!] \| \varepsilon' \| \xi[f / X] \}$

According to semantics of nested predicate equation system, it's easy to prove that the following lemma 1, lemma 2 and lemma 3 are correct.

**Lemma 1:** Let $\varepsilon_1$, $\varepsilon_2$ be nested predicate equation system, satisfy $lhs(\varepsilon_1) \bigcap lhs(\varepsilon_2) = \phi$, $lhs(\varepsilon_1) \bigcap rhs(\varepsilon_2) = \phi$, and $rhs(\varepsilon_1) \bigcap lhs(\varepsilon_2) = \phi$, then $\| \varepsilon_1 \| \| \| \varepsilon_2 \| \xi = \| \varepsilon_1 :: \varepsilon_2 \| \xi$, for any $X \in lhs(\varepsilon_1)$, there is $(\| \varepsilon_1 :: \varepsilon_2 \| \xi)(X) = (\| \varepsilon_1 \| \xi)(X)$, and for any $X \in lhs(\varepsilon_2)$, there is $(\| \varepsilon_1 :: \varepsilon_2 \| \xi)(X) = (\| \varepsilon_2 \| \xi)(X)$.

**Lemma 2:** If $\xi_1 = \| \varepsilon_1 :: (X_1(\overline{x_1}) =_{\sigma} \phi_1) :: (X_2(\overline{x_2}) =_{\sigma} \phi_2) :: \varepsilon_2 \| \xi$

$\xi_2 = \| \varepsilon_1 :: (X_2(\overline{x_2}) =_{\sigma} \phi_2) :: (X_1(\overline{x_1}) =_{\sigma} \phi_1) :: \varepsilon_2 \| \xi$

then $\xi_1 = \xi_2$

**Lemma 3:** If $X_1 \notin Dep(X_2)$ or $X_2 \notin Dep(X_1)$,

$\xi_1 = \| \varepsilon_1 :: (X_1(\overline{x_1}) =_{\sigma 1} \phi_1) :: (X_2(\overline{x_2}) =_{\sigma 2} \phi_2) :: \varepsilon_2 \| \xi$

$\xi_2 = \| \varepsilon_1 :: (X_2(\overline{x_2}) =_{\sigma 2} \phi_2) :: (X_1(\overline{x_1}) =_{\sigma 1} \phi_1) :: \varepsilon_2 \| \xi$

then $\xi_1 = \xi_2$

## 4.3 Translating Ambient Logic Formula into Nested Predicate Equation System

For given an ambient logic formula $\sigma X.A$, the fixpoint type of predicate variable $X$ is labelled as $X.\sigma$, and the definition of dependency relationship between predicates is: if $X' \in fpv(A)$, then $X$ depends on $X'$, labelled as $\langle X, X' \rangle$.

The nesting depth $ad(X)$ of predicate variable $X$ is defined as:

$$ad(X) = 1 + \max\{ad(X') \mid \langle X, X' \rangle \wedge X.\sigma \neq X'.\sigma\}, \ \max\{\varnothing\} = 0.$$

if $fpv(A) = \{X\}$, then $ad(X) = 1$.

Label $d$ as the alternative nesting depth of fixpoint operators in ambient logic formula:

$$d = \max\{ ad(X) \mid X \in \mathcal{X} \}$$

Let $p \in \{\mathrm{T}, \perp, 0\}$, $\circledast \in \{\wedge, \vee, |\}$, $\ominus \in \{ \boxdot, \diamond, \forall x, \exists x, Иx\}$, $\circledcirc \in \{n[], @n, n®, \oslash n\}$, then $\circledcirc(B) \in \{n[B], B@n, n®B, B \oslash n\}$

In the following algorithm 1, function *formulaToequation*($A$) translates proposition $A$ into nested predicate equation system $\varepsilon$. This translation is carried out by nesting function *ftoe*(), which has four parameters : $\phi$ is the proposition or sub-proposition, $\sigma$ is the fixpoint type of current predicate variable, $i$ is the usable subscript for next predicate variable, and $k$ is the nesting depth of current predicate variable. The return value of *ftoe*() is a triple $(X(\overline{x}), \varepsilon, j)$ explained as below: $X(\overline{x})$ is left part of the equation in equation system corresponding with proposition $\phi$, $\varepsilon$ is the equation system (each equation has four parameters: left part of the equation, type of equation's fixpoint, right part of the equation and nesting layer of the equation), and $j$ is the available subscript for next predicate variable. $X_1$ is the first predicate variable in the equation system.

**Algorithm 1.** Translating proposition into nested predicate equation system

```
$formula\ Toequatio(A)$    $// A \to \varepsilon$
  { By $\alpha$-conversion, let all bound variable's name of $A$ are different from each other;
    Calculate nesting depth of each predicate variable in $A$.
    $(X(\bar{x}), \varepsilon, j) = ftoe(A, \sigma, 1, 0)$
    $return\ (X(\bar{x}), \varepsilon)$
  }
$ftoe\ (\phi, \sigma, i, k)$
  { case $\phi$ of
    $p$:          $return\ (X_i(\bar{x_i}), \{X_i(\bar{x_i}), \sigma, p, k\}, i+1)$
    $| \phi_1 \circledast \phi_2 : (Y(\bar{y}), \varepsilon_1, i_1) = ftoe(\phi_1, \sigma, i+1, k), (Z(\bar{z}), \varepsilon_2, i_2) = ftoe(\phi_2, \sigma, i_1, k)$
                          $return\ (X_i(\bar{x_i}), \{X_i(\bar{x_i}), \sigma, Y(\bar{y}) \circledast Z(\bar{z}), k\} :: \varepsilon_1 :: \varepsilon_2, i_1)$
    $| \ominus \phi'$:       $(Y(\bar{y}), \varepsilon_1, i_1) = ftoe(\phi', \sigma, i+1, k)$
                          $return\ (X_i(\bar{x_i}), \{X_i(\bar{x_i}), \sigma, \ominus Y(\bar{y}), k\} :: \varepsilon_1, i_1)$
    $| \circledcirc \phi'$:       $(Y(\bar{y}), \varepsilon_1, i_1) = ftoe(\phi', \sigma, i+1, k)$
                          $return\ (X_i(\bar{x_i}), \{X_i(\bar{x_i}), \sigma, \circledcirc Y(\bar{y}), k\} :: \varepsilon_1, i_1)$
    $| Y(\bar{e})$:     $if\ Y\ is\ \alpha\ X_j\ (j < i), let \{\bar{x_j}\} = fnv(X_j)$
                    $return\ (X_i(\bar{x_i}), \{X_i(\bar{x_i}), \sigma, ((\bar{x_j})(X_j(\bar{x_j})))(\bar{e}), k\}, i+1)$
              $else$      $return\ (X_i(\bar{x_i}), \{X_i(\bar{x_i}), \sigma, ((\bar{x_j})(X_j(\bar{x_j})))(\bar{e}), k\}, i+1)$
    $| ((\bar{x})\phi'(\bar{e})$:          $(Y(\bar{y}), \varepsilon_1, i_1) = ftoe(\phi', \sigma, i+1, k)$
                    $return\ (X_i(\bar{x_i}), \{X_i(\bar{x_i}), \sigma, ((\bar{y})(Y(\bar{y})))(\bar{e}), k\}, i+1)$
    $| (\sigma' X.((\bar{x})\phi'))(\bar{e})$          $(Y(\bar{y}), \varepsilon_1, i_1) = ftoe(\phi'[X_{i+1}/X], \sigma', i+2, ad(X))$
  $return\ (X_i(\bar{x_i}), \{X_i(\bar{x_i}), \sigma, ((\bar{y})(X_{i+1}(\bar{y})))(\bar{e}), k\} :: \{X_{i+1}(\bar{y}), \sigma', Y(\bar{y}), ad(X)\} :: \varepsilon_1, i_1)$
  }
```

**Theorem 1**: Let $\phi$ be a proposition, $\xi$ be an environment, $\varepsilon$ is the nested predicate equation system of $\phi$ translated by algorithm 1, $1 \le i \le k, (X_k) \cap fpv(\phi) = \phi$, if $(X(\bar{x}), \varepsilon, j) = ftoe(A, \sigma, 1, 0)$, then $[\![(\bar{x})\phi]\!]\xi = \| \varepsilon \| \xi(X)$.

**Proof**: refers to Appendix.

Theorem 1 shows that proposition $\phi$ and its nested predicate equation systems $\varepsilon$ are equal with semantics.

## 4.4 Block Dependency Graph and Computation Algorithm

Block Dependency Graph (BDG) is an Automata Composed with nested predicate equation system and mobile ambients process. The data structure of node in BDG is described as $(P, X(\bar{x}, \phi', \sigma, k), val)$, which $P$ is process, $X(\bar{x}) =_\sigma \phi'$ is nested predicate equation, $k$ is nesting layer of the fixpoint operator, and $val$ is the current node's state value. If $\sigma = \nu$ the initial value of $val$ is $val = true$, and if $\sigma = \mu$ the initial value is $val = false$.

Function $new(V)$ is producing a fresh name that does not exist in name set $V$.

Algorithm 2 is the algorithm that constructs BDG.

**Algorithm 2.**

```
$Install\_Set((P, X(\overline{x}, \phi, \sigma, k)))$//constructs BDG, $S_{k0}$ saves the node
//whose nesting layer of fixpoint is $k$ , node value is initial value
{
$if\ \{(P, X(\overline{x}, \phi, \sigma, k), false), (P, X(\overline{x}, \phi, \sigma, k), true)\} \cap S_{k0} = \varnothing\ \ then$
        //does not deal with repeated nodes
$if(\sigma = \mu)\ add(S_{k0}, (P, X(\overline{x}, \phi, \sigma, k), false));$
$else\ add(S_{k0}, (P, X(\overline{x}, \phi, \sigma, k), true));$
//the set formed by the following produced nodes is the direct dependecy set
//$DDS$ of node$(P, X(\overline{x}, \phi, \sigma, k), val)$, node $(P, X(\overline{x}, \phi, \sigma, k), val)$ belongs to
//the direct active set $DAS$ of the following produced nodes
case $\phi$ of
$Y(\overline{y}) \wedge Z(\overline{z}) : Install\_Set((P, Y(\overline{y}, \phi_1, \sigma, k)));\ \ Install\_Set((P, Z(\overline{z}, \phi_2, \sigma, k)));$
$| Y(\overline{y}) \vee Z(\overline{z}) : Install\_Set((P, Y(\overline{y}, \phi_1, \sigma, k)));\ \ Install\_Set((P, Z(\overline{z}, \phi_2, \sigma, k)));$
$| Y(\overline{y}) \mid Z(\overline{z}) : \forall P_1 \mid P_2 \equiv P, Install\_Set((P_1, Y(\overline{y}, \phi_1, \sigma, k)));\ Install\_Set((P_2, Z(\overline{z}, \phi_2, \sigma, k)));$
$| n[Y(\overline{y})] : if\ \ P \equiv n[P']\ then\ \ Install\_Set((P', Y(\overline{y}, \phi', \sigma, k)));$
$| n®Y(\overline{y}) : if\ \ n \in fn(P)\ \ then\ return;$// on the basis of lemma1
                $Install\_Set((P, Y(\overline{y}, \phi', \sigma, k)));$
        $if\ \ bn(P) \neq \varnothing\ then\ \forall m \in bn(P) \wedge P \equiv (vm)Q\ \ Install\_Set((Q[n/m], Y(\overline{y}, \phi', \sigma, k)));$
$| Y(\overline{y})@n : Install\_Set((n[P], Y(\overline{y}, \phi', \sigma, k)));$
$| Y(\overline{y}) \oslash n : Install\_Set(((vn)P, Y(\overline{y}, \phi', \sigma, k)));$
$|\boxdot Y(\overline{y}) : \forall Q\ if : P \to Q\ then\ \ Install\_Set((Q, Y(\overline{y}, \phi', \sigma, k)));$
$|\Diamonddot Y(\overline{y}) : \forall Q\ if : P \to Q\ then\ \ Install\_Set((Q, Y(\overline{y}, \phi', \sigma, k)));$
$| \forall x.Y(\overline{y}) : \forall n \in fn(P, \phi) \cup \{new(fn(P, \phi)\}\ Install\_Set((P, Y(\overline{y}, \phi'[n/x], \sigma, k)));$
$| \exists x.Y(\overline{y}) : \forall n \in fn(P, \phi) \cup \{new(fn(P, \phi)\}\ Install\_Set((P, Y(\overline{y}, \phi'[n/x], \sigma, k)));$
$| \text{И}x.Y(\overline{y}) :\ \ Install\_Set((P, Y(\overline{y}, \phi'[new(fn(P, \phi))/x], \sigma, k)));$
$| ((\overline{y})(Y(\overline{y})))(\overline{e}) :\ \ Install\_Set((P, Y(\overline{y}, \phi'[\overline{e}/\overline{y}], \sigma, k)));$
  }}
```

The function *Install_Set*() produces BDG is structurally complied with mobile ambients semantics and ambient logic semantics, thus the correctness of Algorithm 2is obvious.

The node set corresponding with the *i*th nesting layer of fixpoint operator in BDG is labelled as $R_i$, $R_i = \{(P, X(\overline{x}, \phi', \sigma, i), val) \mid 1 \leq i \leq d\}$.

The number of all operators in formula *A* is labelled as *op*(*A*), the definition of *op*(*A*) is as below:

$op(p) ::= 0 \qquad op(A_1 \circledast A_2) ::= 1 + op(A_1) + op(A_2)$

$op(\ominus A') ::= 1 + +op(A') \qquad op(\circledcirc A') ::= 1 + op(A')$

$op(Y(\overline{n})) ::= 0 \qquad op(((\overline{x})A'))(\overline{n})) ::= op(A')$

$op((\sigma X.((\overline{x})A'))(\overline{n})) ::= 1 + +op(A')$

The degree of parallelism of process *P* is labelled as *par*(*P*), *par*(*P*)is defined as:

| | | |
|---|---|---|
| $P \equiv 0$ | : | $par(P) ::= 0$ |
| $P \equiv a_1[P_1] \mid ... \mid a_k[P_k]$ | : | $par(P) ::= \max\{k,\ par(P_1), \ldots,\ par(P_K)\}$ |
| $P \equiv M.P'$ | : | $par(P) ::= \max\{k,\ par(P')\}$ |
| $P \to P'$ | : | $par(P) ::= \max\{par(P),\ par(P')\}$ |

The number of operators '®' in formula *A* is labelled as $op_{®}(A)$, the number of quantifiers $'\forall, \exists'$ is labelled as $op_{\forall,\exists}(A)$, the number of operators $'|'$ is labelled as $op_{|}(A)$, the number of operators $'\boxdot, \Diamonddot'$ is labelled as $op_{\boxdot,\diamond}(A)$, the number of operators $'\wedge,\vee,|'$ is labelled as $op_{\wedge,\vee,|}(A)$. The size of bound names set in process *P* is labelled as |*bn*(*P*)|; the size of names set and free names set in process *P* and formula *A* is labelled as |*N*(*P*, *A*)| and |*fn*(*P*, *A*)| respectively; label $V_{LTS}(P)$ as the total number of nodes in the label transition system corresponded with process *P*. Label the size of nodes produced by function *Install_Set*() in algorithm 2 as $\boldsymbol{M}$, then:

$$\boldsymbol{M} = V_{LTS}(P)\times[op(A)+op_{\wedge,\vee,|}(A)]\times 2^{par(P)\times op_{|}(A)}\times[|bn(P)|+1]^{op_{®}(A)}\times|N(P,\phi)|^{op_{\forall,\exists}(A)}$$

**Lemma 4:** The number of nodes produced by function *Install_Set*()in algorithm 2 is less than $\boldsymbol{M}$.

The proof of lemma 4 refers to Appendix.

In BDG, a node's value can be computed from its direct dependency set's nodes, see algorithm 3.

**Algorithm 3.**

```
$Value\_Record((P, X(\overline{x},\phi,\sigma,i), val))$// computes node's value
{
case $\phi$ of
$T$:  $return(true)$
$|\bot$:  $return(false)$;
$|\,0$:  $\{if(P\equiv 0)\ return(true); else\ return(false)$;
$|\,Y(\overline{y})\wedge Z(\overline{z})$ and $DDS((P,X(\overline{x},\phi,\sigma,i),val))=\{(P,Y(\overline{y},\phi_1,\sigma_1,i_1),val_1),(P,Z(\overline{z},\phi_2,\sigma_2,i_2),val_2)\}$:
            $return(val_1\wedge val_2)$;
$|\,Y(\overline{y})\vee Z(\overline{z})$ and $DDS((P,X(\overline{x},\phi,\sigma,i),val))=\{(P,Y(\overline{y},\phi_1,\sigma_1,i_1),val_1),(P,Z(\overline{z},\phi_2,\sigma_2,i_2),val_2)\}$:
            $return(val_1\vee val_2)$;
$|\,Y(\overline{y})\,|\,Z(\overline{z})$ and $DDS((P,X(\overline{x},\phi,\sigma,i),val))=\bigcup_{P_1|P_2\equiv P}\{(P_1,Y(\overline{y},\phi_1,\sigma_1,i_1),val_1),(P_2,Z(\overline{z},\phi_2,\sigma_2,i_2),val_2)\}$:
            $return(\bigvee_{P_1|P_2\equiv P} val_1\wedge val_2)$;
$|\,n[Y(\overline{y})]$: $if\ \ DDS((P,X(\overline{x},\phi,\sigma,i),val))=\{(P_1,Y(\overline{y},\phi_1,\sigma_1,i_1),val_1)\}: return(val_1)$;
        $else\ \ if\ \ DDS(P,X(\overline{x},\phi,\sigma))=\varnothing: return(false)$;
$|\,n®Y(\overline{y}): if\ \ DDS((P,X(\overline{x},\phi,\sigma,i),val))=\bigcup_j\{(P_j,Y_j(\overline{y}_j,\phi_j,\sigma_j,i_j),val_j)\}$:  $return(\bigvee_j val_j)$;
        $else\ \ if\ \ DDS((P,X(\overline{x},\phi,\sigma,i),val))=\varnothing: return(false)$;
$|\,Y(\overline{y})@n$:  and $DDS((P,X(\overline{x},\phi,\sigma,i),val))=\{(P_1,Y(\overline{y},\phi_1,\sigma_1,i_1),val_1)\}: return(val_1)$;
$|\,Y(\overline{y})\oslash n$:  and $DDS((P,X(\overline{x},\phi,\sigma,i),val))=\{(P_1,Y(\overline{y},\phi_1,\sigma_1,i_1),val_1)\}: return(val_1)$;
$|\boxdot Y(\overline{y})$  $if(DDS((P,X(\overline{x},\phi,\sigma,i),val))=\varnothing)\ then\ \ return(true)$;
        $else\ if(DDS((P,X(\overline{x},\phi,\sigma,i),val))=\bigcup_j\{(P_j,Y_j(\overline{y}_j,\phi_j,\sigma_j,i_j),val_j)\})\ return(\bigwedge_j val_j)$;
$|\diamond Y(\overline{y}): if(DDS((P,X(\overline{x},\phi,\sigma,i),val))=\varnothing)\ then\ \ return(false)$;
        $else\ if(DDS((P,X(\overline{x},\phi,\sigma,i),val))=\bigcup_j\{(P_j,Y_j(\overline{y}_j,\phi_j,\sigma_j,i_j),val_j)\})\ return(\bigvee_j val_j)$;
$|\,\forall x.Y(\overline{y})$ and $DDS((P,X(\overline{x},\phi,\sigma,i),val))=\bigcup_j\{(P_j,Y_j(\overline{y}_j,\phi_j,\sigma_j,i_j),val_j)\}$:  $return(\bigwedge_j val_j)$;
$|\,\exists x.Y(\overline{y})$ and $DDS((P,X(\overline{x},\phi,\sigma,i),val))=\bigcup_j\{(P_j,Y_j(\overline{y}_j,\phi_j,\sigma_j,i_j),val_j)\}$:  $return(\bigvee_j val_j)$;
$|\,\text{И}x.Y(\overline{y})$ and $DDS((P,X(\overline{x},\phi,\sigma,i),val))=\{(P_1,Y(\overline{y},\phi_1,\sigma_1,i_1),val_1)\}: return(val_1)$;
$|\,((\overline{y})(Y(\overline{y})))(\overline{n})$ and $DDS((P,X(\overline{x},\phi,\sigma,i),val))=\{(P_1,Y(\overline{y},\phi_1,\sigma_1,i_1),val_1)\}: return(val_1)$;
  }
```

Function *Value_Record*() complies with mobile ambients' semantics and ambient logic's semantics, the correctness is obvious.

Once the BDG is made, the model checking is translated into the computation of BDG. Ifa logic formula contains fixpoint operator and its alternate nesting depth is *d*, according to Tarski's fixpoint theorem [22], the computation of BDG firstly start from the node set which the alternate nesting depth is*d* (hereinafter referred to as $R_d$). After all nodes in $R_d$ reach the current fixpoint, start to compute $R_{d-1}$. if the value of any node in $R_{d-1}$ is changed, then $R_d$ needs to recompute the current fixpoint from the initial value. After all nodes in $R_{d-1}$ reach the current fixpoint, compute $R_{d-2}$..., and so on. Furthemore, if any node's value in $R_i$ is changed, then the current fixpoint values for all $R_k (i < k \leq d)$ need to be recomputed from the initial value.

Let $R_{i0}$ denotesthe node subset waiting for computation, the initial value is $S_{i0}$, $R_{i1}$ denotesthe node subsetwith determinate values under current environment $\xi$, $R_{i2}$ denotes the nodesetwithundetermined values, *i* indicates nesting layer, $R_i = R_{i0} \cup R_{i1} \cup R_{i2}$.

To compute values of nodes in $R_i$, Algorithm 3 may begin from any node in $R_{i0}$, which lemma 2 can ensure the correctness of entire results. During the process of computing, if the *val* of a node is changed under current environment $\xi$, according to the principle of monotonicity function, this change is non-retrievable, so the algorithm may move this node into set $R_{i1}$ which needs not to be recomputedunder current environment $\xi$. On the other hand, if the *val* of a node is not changed, move this node into set $R_{i2}$, and it needsreturn $R_{i0}$ to be recomputed afterthe valueof its direct dependency set is changed. Finally when $R_{i0} = \varnothing$, all nodes of $R_i$ are in $R_{i1}$ (the *val*under current environment $\xi$ is impossible to change any more), or in $R_{i2}$ (the *val* keeps unchanged in the process of computing), $R_i$ reaches the current fixpoint state.

## 5 Partial-order Relation in BDG and New Algorithm of Model Checking

In this section, it firstly presents a kind of partial-order relation between inter-media results during BDG computation. Next, a new BDG computation algorithm, model checking algorithm, and as well as, its compliexity analyzing are given.

First of all, some symbols are defined as below:

Let *J* and *K* are the node sets in BDG, and satisfy: (1) |*J*|=|*K*|; (2) the value *val*of each node in the set is unique, meaning $(P, X(\overline{x}, \phi, \sigma, i), \mathit{false})$ and $(P, X(\overline{x}, \phi, \sigma, i), \mathit{true})$ cannot appear in the same set at the same time.

**Definition 1:** Define operator $\Subset$ in *J* and *K*: if $J \Subset K$, then

$$\forall (P, X(\overline{x}, \phi, \sigma, i), \mathit{false}) \in J \Rightarrow (P, X(\overline{x}, \phi, \sigma, i), \mathit{false}) \in K \vee (P, X(\overline{x}, \phi, \sigma, i), \mathit{true}) \in K;$$
$$\forall (P, X(\overline{x}, \phi, \sigma, i), \mathit{true}) \in J \Rightarrow (P, X(\overline{x}, \phi, \sigma, i), \mathit{true}) \in K$$

Obviously, $\Subset$ is reflexive, antisymmetrical and transitive. It is a partial ordering relation in computing node set.

Algorithm 3 shows that, during the computing of $R_1$, there exists a computation sequence denoted as $R_1^0, R_1^1, R_1^2, \cdots, R_1^\omega$ (when $R_1$ reaches fixpoint state, it is labelled as $R_1^\omega$).

Let $R_1^{j_1}$ denotes the value of $R_1$, the computation sequence of computing $R_2$ is denoted as $R_2^{j_1 0}, R_2^{j_1 1}, R_2^{j_1 2}, \cdots, R_2^{j_1 \omega}$. Similarly, let $R_1^{j_1}$ and $R_2^{j_1 j_2}$ denote the value of $R_1$ and $R_2$ respectively, the computation sequence of computing $R_3$ is denoted as $R_3^{j_1 j_2 0}, R_3^{j_1 j_2 1}, R_3^{j_1 j_2 2}, \cdots, R_3^{j_1 j_2 \omega}$.

Summarily, let $R_1^{j_1}$ 、$R_2^{j_1 j_2}$ ..., and $R_{d-1}^{j_1 j_2 \cdots j_{d-1}}$ denote the value of $R_1$ 、$R_2$ ..., and $R_{d-1}$ respectively, the computation sequence of computing $R_d$ is denoted as $R_d^{j_1 j_2 \cdots j_{d-1} 0}, R_d^{j_1 j_2 \cdots j_{d-1} 1}, R_d^{j_1 j_2 \cdots j_{d-1} 2}, \cdots, R_d^{j_1 j_2 \cdots j_{d-1} \omega}$.

Among it, $R_i^{j_1 j_2 \cdots j_{i-1} k} ::= R_{i0}^{j_1 j_2 \cdots j_{i-1} k} \cup R_{i1}^{j_1 j_2 \cdots j_{i-1} k} \cup R_{i2}^{j_1 j_2 \cdots j_{i-1} k}$

According to Tarski's fixpoint theorem, the computation sequence of nodes in BDG is:

$$R_d^{00...00}, R_d^{00...01}, R_d^{00...02}, \cdots, R_d^{00...0\omega}, R_{d-1}^{00...1}, R_d^{00...10}, R_d^{00...11}, R_d^{00...12}, \cdots, R_d^{00...1\omega},$$
$$R_{d-1}^{00...2}, ..., R_{d-1}^{00...\omega}, ..., R_2^{01}, R_d^{01...00}, R_d^{01...01}, R_d^{01...02}, \cdots, R_d^{01...0\omega}, R_{d-1}^{01...1}, R_d^{01...10},$$
$$R_d^{01...11}, R_d^{01...12}, \cdots, R_d^{01...1\omega}, R_{d-1}^{01...1}, ..., R_{d-1}^{01...\omega}, ..., R_2^{02}, ..., R_2^{0\omega}, R_1^{1}, R_d^{10...00},$$
$$R_d^{10...01}, R_d^{10...02}, \cdots, R_d^{10...0\omega}, R_{d-1}^{10...1}, R_d^{10...10}, R_d^{10...11}, R_d^{10...12}, \cdots, R_d^{10...1\omega}, R_{d-1}^{10...2},$$
$$..., R_{d-1}^{10...\omega}, ..., R_2^{11}, R_d^{11...00}, R_d^{11...01}, R_d^{11...02}, \cdots, R_d^{11...0\omega}, R_{d-1}^{11...1}, R_d^{11...10}, R_d^{11...11},$$
$$R_d^{11...12}, \cdots, R_d^{11...1\omega}, R_{d-1}^{11...2}, ..., R_{d-1}^{11...\omega}, ..., R_2^{12}, ..., R_2^{1\omega}, R_1^{2}, ...R_1^{\omega}$$

The initial value of $R_i$ comes from $Install\_Set((P, X(\overline{x}, \phi, \sigma, k)))$ in algorithm 2, $R_i^{j_1 j_2 \cdots j_{i-1} 0} = S_{i0}$. If $\sigma_i = \nu$, the initial value of each node *val* in $R_i^{j_1 j_2 \cdots j_{i-1} 0}$ is *true*, and if $\sigma_i = \mu$, the initial value of each node *val* in $R_i^{j_1 j_2 \cdots j_{i-1} 0}$ is *false*.

Assume the corresponding functions of computing nodes in $R_1, R_2, ..., R_{d-1}, R_d$ are $f_1, f_2, ..., f_{d-1}, f_d$, then:

$$\begin{aligned}
&R_1^{j_1+1} = f_1(R_1^{j_1}, R_2^{j_1\omega}, ..., R_{d-1}^{j_1\omega...\omega}, R_d^{j_1\omega...\omega\omega})\\
&R_2^{j_1(j_2+1)} = f_2(R_1^{j_1}, R_2^{j_1 j_2}, ..., R_{d-1}^{j_1 j_2...\omega}, R_d^{j_1 j_2...\omega\omega})\\
&...\\
&R_{d-1}^{j_1 j_2 \cdots (j_{d-1}+1)} = f_{d-1}(R_1^{j_1}, R_2^{j_1 j_2}, ..., R_{d-1}^{j_1 j_2 \cdots j_{d-1}}, R_d^{j_1 j_2 \cdots j_{d-1}\omega})\\
&R_d^{j_1 j_2 \cdots j_{d-1}(j_d+1)} = f_d(R_1^{j_1}, R_2^{j_1 j_2}, ..., R_{d-1}^{j_1 j_2 \cdots j_{d-1}}, R_d^{j_1 j_2 \cdots j_{d-1} j_d})
\end{aligned}$$

$f_1, f_2, ..., f_{d-1}, f_d$ are monotonic functions.

Let $R_1^{j_1}$ , $R_2^{j_1 j_2}$ …, and $R_{k-1}^{j_1 j_2 \cdots j_{k-1}}$ denote the value of $R_1$ , $R_2$ …, and $R_{k-1}$ respectively, the computation sequence of computing $R_k$ is denoted as $R_k^{j_1 j_2 \cdots j_{k-1} 0}, R_k^{j_1 j_2 \cdots j_{k-1} 1}, R_k^{j_1 j_2 \cdots j_{k-1} 2}, \cdots, R_k^{j_1 j_2 \cdots j_{k-1} \omega}$. If $\sigma_k = \nu$, in the process of computing, the value *val* of each node in $R_k$ either keeps initial value *true* or changes into *false*. Once the value of node *val* is changed into *false*, the node is moved into set $R_{k1}$ , the node in sequence $R_k^{j_1 j_2 \cdots j_{k-1} 0}, R_k^{j_1 j_2 \cdots j_{k-1} 1}, R_k^{j_1 j_2 \cdots j_{k-1} 2}, \cdots, R_k^{j_1 j_2 \cdots j_{k-1} \omega}$ does not change any more, and each intermediate result satisfies $R_k^{j_1 j_2 \cdots j_{k-1} \omega} \Subset \cdots \Subset R_k^{j_1 j_2 \cdots j_{k-1} 2} \Subset R_k^{j_1 j_2 \cdots j_{k-1} 1} \Subset R_k^{j_1 j_2 \cdots j_{k-1} 0}$. If $\sigma_k = \mu$, each intermediate result satisfies $R_k^{j_1 j_2 \cdots j_{k-1} 0} \Subset R_k^{j_1 j_2 \cdots j_{k-1} 1} \Subset R_k^{j_1 j_2 \cdots j_{k-1} 2} \Subset \cdots \Subset R_k^{j_1 j_2 \cdots j_{k-1} \omega}$.

**Definition 2:**

Assume digital sequence $j_1 j_2 j_3 ... j_k$ and $l_1 l_2 l_3 ... l_k$ have the same length *k*, the value of each digit is non-negative integer, and satisfies

$$\exists i(1 \le i \le k \wedge (k-i)\%2 == 0 \wedge j_i > l_i \wedge \forall x(1 < x \le k \wedge x \ne i \wedge j_x = l_x))$$

Then it is labelled as $P(j_1 j_2 j_3 ... j_k) = l_1 l_2 l_3 ... l_k$ . That's to say, only one digit is different in both sequences, and other digits are the same. In the process of computing, $R_k^{j_1 j_2 j_3 \cdots j_k}$ appears after $R_k^{l_1 l_2 l_3 ... l_k}$ . Let label $Even(j_1 j_2 j_3 ... j_k)$ denotes the consequence extracted from all even digits in sequence $j_1 j_2 j_3 ... j_k$, and label $Odd(j_1 j_2 j_3 ... j_k)$ denotes the consequence extracted from all odd digits in sequence $j_1 j_2 j_3 ... j_k$ , and $h(j_1 j_2 j_3 ... j_k, i)$ denotes the consequence $j_1 j_2 ... j_i$ extracted from the first *i*digits in $j_1 j_2 j_3 ... j_k$. If all digits in the sequence are zero, label it as $\overline{0}$.

**Lemma 5:** If $\sigma_k = \mu$ , then $R_k^{P(j_1 j_2 \cdots j_k)} \Subset R_k^{(j_1 j_2 \cdots j_k)}$ .

The proof of lemma 5 refers to Appendix.

**Lemma 6:** If $\sigma_k = \nu$ , then $R_k^{j_1 j_2 \cdots j_k} \Subset R_k^{j_1 j_2 \cdots j_k}$ .

The proof of lemma 6 is similar to lemma 5.

From lemma 5, it is known that if $\sigma_k = \mu$ , then $R_k^{P(j_1 j_2 \cdots j_{k-1}\omega)} \Subset R_k^{j_1 j_2 \cdots j_{k-1}\omega}$ . When computing $R_k^{j_1 j_2 j_3 \cdots j_{k-1}\omega}$ , we can start to compute from $R_k^{P(j_1 j_2 j_3 \cdots j_{k-1}\omega)}$ directly, not from the node's initial value $S_{k0}$ , that means the computing times is diminished. From lemma 6, similarly, if $\sigma_k = v$, when computing $R_k^{j_1 j_2 j_3 \cdots j_{k-1}\omega}$, we can start computation from $R_k^{P(j_1 j_2 j_3 \cdots j_{k-1}\omega)}$ directly, need not start from the node's initial value $S_{k0}$.

Summarily, if the current calculation sequence is $\overline{l}$ and *d*%2==0(*d* is a even number), according to lemma 5 and lemma 6, the solution to the BDG is algorithm 4 as below.

**Algorithm 4.**

```
$ValueSet()$//computes BDG {
    $for(i_2 = 0; i_2 < m_2; i_2 ++)$ //saves the variable initialization of intermediate results
    {
       $G_{30}^{i_2} = S_{30}, G_{31}^{i_2} = \varnothing;$
          $for(i_4 = 0; i_4 < m_4; i_4 ++)$ {
          $G_{50}^{i_2 i_4} = S_{50}, G_{51}^{i_2 i_4} = \varnothing;$
          ......
          $for(i_{d-2} = 0; i_{d-2} < m_{d-2}; i_{d-2} ++)\{G_{(d-1)0}^{i_2 i_4 \cdots i_{d-2}} = S_{(d-1)0}, G_{(d-1)1}^{i_2 i_4 \cdots i_{d-2}} = \varnothing;\}$
          }
}
$i = d;$ //computing from $R_d$
$while(i > 0)$
   { $while(R_{i0} \neq \varnothing)$
      { $(P, X(\overline{x}, \phi, \sigma, i), val) = readrecord(R_{i0});$
      $R_{i0} = R_{i0} - \{(P, X(\overline{x}, \phi, \sigma, i), val)\}$
      } ; //extracts a node from $R_{i0}$
       $val' = (P, X(\overline{x}, \phi, \sigma, i), val).val;$ //saves the ancient value
      $(P, X(\overline{x}, \phi, \sigma, i), val).val = Value\_Record((P, X(\overline{x}, \phi, \sigma, i), val));$
      //computes and updates $val$ of the node
       $if\ (val' \neq (P, X(\overline{x}, \phi, \sigma, i), val).val)\ break;$
      $R_{i2} = R_{i2} + \{(P, X(\overline{x}, \phi, \sigma, i), val)\}$ }
   }
 $if(R_{i0} = \varnothing)\{R_{i0} = R_{i2};$          //reaches the current fixpoint
   $if(i\%2 == 1 \wedge i \geq 3) G_{i1}^{Even(h(\overline{l}, i))} = R_{i1}, G_{i0}^{Even(h(\overline{l}, i))} = R_{i0};$ //saves $R_{i1}$ and $R_{i0}$
   $i--;$ }
$else\ \{R_{i1} = R_{i1} + \{(P, X(\overline{x}, \phi, \sigma, i), val)\}\}; R_{i0} = R_{i0} + R_{i2};$
   $if(i < d)\ \{for(j = i+1; j \leq d; j++)$
      //recovers the initial state of the inner nodes  in workspace
         $\{if(j\%2 == 1 \wedge Odd(h(\overline{l}, j)) \neq \overline{0})$
          $R_{j0} = G_{j0}^{Even(P(h(\overline{l}, j)))}, R_{j1} = G_{j1}^{Even(P(h(\overline{l}, j)))}, R_{i2} = \varnothing;$
        $else\ R_{j0} = S_{j0}, R_{j1} = \varnothing, R_{i2} = \varnothing;$
        }
 $i = d;$ //recomputes from $R_d$
   }}}}
```

**Example 1:** According to algorithm 4, if *d*=3, the computation sequences for the node of each layer in BDG are:

$$
\begin{aligned}
&R_3^{000}=S_{30},R_3^{001},R_3^{002},\cdots,R_3^{00\omega},R_2^{01},R_3^{010}=S_{30},R_3^{011},R_3^{012},\cdots,R_3^{01\omega},\\
&R_2^{02},R_3^{020}=S_{30},R_3^{021},R_3^{022},\cdots,R_3^{02\omega},R_2^{03},...,R_2^{0\omega},R_1^{1},R_3^{100}=R_3^{00\omega},\\
&R_3^{101},R_3^{102},\cdots,R_3^{10\omega},R_2^{11},R_3^{110}=R_3^{01\omega},R_3^{111},R_3^{112},\cdots,R_3^{11\omega},R_2^{12},\\
&R_3^{120}=R_3^{02\omega},R_3^{121}\cdots,R_3^{12\omega},R_2^{13},...,R_2^{1\omega},...,R_1^{2},...,R_1^{\omega}
\end{aligned}
$$

**Example 2:** According to algorithm 4, initial value is $G_{50}^{i_2i_4}=S_{50},G_{51}^{i_2i_4}=\varnothing$, the data accessed through $G_{50}^{i_2i_4}$ and $G_{51}^{i_2i_4}$ is as below:

$$
\begin{aligned}
&\{G_{50}^{i_2i_4}=R_{50}^{0i_20i_4\omega},G_{51}^{i_2i_4}=R_{51}^{0i_20i_4\omega}\}\rightarrow\{R_{50}^{0i_21i_40}=G_{50}^{i_2i_4},R_{51}^{0i_21i_40}=G_{51}^{i_2i_4}\}\\
&\rightarrow\{G_{50}^{i_2i_4}=R_{50}^{0i_21i_4\omega},G_{51}^{i_2i_4}=R_{51}^{0i_21i_4\omega}\}\rightarrow\{R_{50}^{0i_22i_40}=G_{50}^{i_2i_4},R_{51}^{0i_22i_40}=G_{51}^{i_2i_4}\}\\
&\rightarrow\{G_{50}^{i_2i_4}=R_{50}^{0i_22i_4\omega},G_{51}^{i_2i_4}=R_{51}^{0i_22i_4\omega}\}\rightarrow\{R_{50}^{0i_23i_40}=G_{50}^{i_2i_4},R_{51}^{0i_23i_40}=G_{51}^{i_2i_4}\}\\
&\rightarrow...\rightarrow\{G_{50}^{i_2i_4}=R_{50}^{0i_2\omega i_4\omega},G_{51}^{i_2i_4}=R_{51}^{0i_2\omega i_4\omega}\}\rightarrow\{R_{50}^{1i_20i_40}=G_{50}^{i_2i_4},R_{51}^{1i_20i_40}=G_{51}^{i_2i_4}\}\\
&\rightarrow\{G_{50}^{i_2i_4}=R_{50}^{1i_20i_4\omega},G_{51}^{i_2i_4}=R_{51}^{1i_20i_4\omega}\}\rightarrow\{R_{50}^{1i_21i_40}=G_{50}^{i_2i_4},R_{51}^{1i_21i_40}=G_{51}^{i_2i_4}\}\\
&\rightarrow\{G_{50}^{i_2i_4}=R_{50}^{1i_21i_4\omega},G_{51}^{i_2i_4}=R_{51}^{1i_21i_4\omega}\}\rightarrow\{R_{50}^{1i_22i_40}=G_{50}^{i_2i_4},R_{51}^{1i_22i_40}=G_{51}^{i_2i_4}\}\\
&\rightarrow\{G_{50}^{i_2i_4}=R_{50}^{1i_22i_4\omega},G_{51}^{i_2i_4}=R_{51}^{1i_22i_4\omega}\}\rightarrow\{R_{50}^{1i_23i_40}=G_{50}^{i_2i_4},R_{51}^{1i_23i_40}=G_{51}^{i_2i_4}\}\\
&\rightarrow...\rightarrow\{G_{50}^{i_2i_4}=R_{50}^{1i_2\omega i_4\omega},G_{51}^{i_2i_4}=R_{51}^{1i_2\omega i_4\omega}\}\rightarrow\{R_{50}^{2i_20i_40}=G_{50}^{i_2i_4},R_{51}^{2i_20i_40}=G_{51}^{i_2i_4}\}\\
&\rightarrow\{G_{50}^{i_2i_4}=R_{50}^{2i_20i_4\omega},G_{51}^{i_2i_4}=R_{51}^{2i_20i_4\omega}\}\rightarrow\{R_{50}^{2i_21i_40}=G_{50}^{i_2i_4},R_{51}^{2i_21i_40}=G_{51}^{i_2i_4}\}\\
&\rightarrow\{G_{50}^{i_2i_4}=R_{50}^{2i_21i_4\omega},G_{51}^{i_2i_4}=R_{51}^{2i_21i_4\omega}\}\rightarrow\{R_{50}^{2i_22i_40}=G_{50}^{i_2i_4},R_{51}^{2i_22i_40}=G_{51}^{i_2i_4}\}\\
&\rightarrow\{G_{50}^{i_2i_4}=R_{50}^{2i_22i_4\omega},G_{51}^{i_2i_4}=R_{51}^{2i_22i_4\omega}\}\rightarrow\{R_{50}^{2i_23i_40}=G_{50}^{i_2i_4},R_{51}^{2i_23i_40}=G_{51}^{i_2i_4}\}\\
&\rightarrow...\rightarrow\{G_{50}^{i_2i_4}=R_{50}^{2i_2\omega i_4\omega},G_{51}^{i_2i_4}=R_{51}^{2i_2\omega i_4\omega}\}\rightarrow\{R_{50}^{2i_20i_40}=G_{50}^{i_2i_4},R_{51}^{2i_20i_40}=G_{51}^{i_2i_4}\}\\
&\rightarrow...\rightarrow\{G_{50}^{i_2i_4}=R_{50}^{\omega i_2\omega i_4\omega},G_{51}^{i_2i_4}=R_{51}^{\omega i_2\omega i_4\omega}\}
\end{aligned}
$$

Algorithm 4 can significantly improve the efficiency of BDG computing, thus a high performance model checking algorithm is presented as Algorithm 5.

**Algorithm 5.** The Model Checking Algorithm for Mobile Ambients

```
$Model\_Check(P,A)$
{
$(X(\overline{x}),\mathcal{E})=formulaToequation(A)$;
$for(i=1;i\le d;i++)\ S_{i0}=\varnothing,S_{i1}=\varnothing,S_{i2}=\varnothing$;
    //$S_{i0}$ saves node waiting for computing
$Install\_Set((P,X(\overline{x},\phi,\sigma,k)))$; //produces all computing nodes
$for(i=1;i\le d;i++)\ R_{i0}=S_{i0},R_{i1}=S_{i1},R_{i2}=S_{i2}$; //copies nodes to workspace
$ValueSet()$; //computes BDG
$return((P,X(\overline{x},\phi,\sigma,k),val).val)$; //returns the $val$ of node $(P,X(\overline{x},\phi,\sigma,k),val)$
}
```

In rest of this session, Theorem 2 proves the correctness of Algorithm 5, Theorem 3 and Theorem 4 present the time and space complexity of Algorithm 5 respectively.

**Theorem 2:** According to Algorithm 5, if the last computing result of $Model\_Check(P,A)$ is $(P,X(\overline{x},\phi,\sigma,k),val).val=true$, then $P\vDash A$, otherwise $P\nvDash A$.

**Proof:**

For the correctness of function $formulaToequation()$, see the proof of theorem 1. The computing of $Install\_Set((P,X(\overline{n},\phi,\sigma)))$ and $Value\_Record((P,X(\overline{x},\phi,\sigma)))$ completely comply with mobile ambients semantics and ambient logic semantics, the correctness is obvious.

Function $ValueSet()$ in algorithm 4 starts to compute from the node in $R_d$, the innermost layer of BDG.

Use variable $i$ to denote the current computing number of layer. There are two scenarios some steps may be skip during BDG computing. One is when computation meets one node's value is changed, because each nodes of its nesting layer depth is deeper than the current layer should be recomputed, so the computing can skip to node set $R_d$ directly, which is the deepest layer. Two is when node sets in the current layer reach their fixpoint states, the computing can skip to ($i$-1)th layer. These two kind skips of computing steps are carried out by the computing order of the nested fixpoint block, it is accordant with Tarski's fixpoint theorem and the correctness is obvious.

As Algorithm 4 shows, once the value of a node in $R_{i0}$ is changed, then it is moved into $R_{i1}$. In case when the node's value in the outer layer is unchanged, according to monotonicity, the node's value cannot be changed again, obviously this processing is correct. After the node's value is changed in the outer layer, and $i \ge 3 \wedge i\%2 == 1 \wedge Odd(h(\overline{l},i)) \ne \overline{0}$, then $R_{i0} = G_{i0}^{Even(P(h(\overline{l},i)))}, R_{i1} = G_{i1}^{Even(P(h(\overline{l},i)))}, R_{i2} = \varnothing$, not $R_{i0} = S_{i0}, R_{i1} = \varnothing, R_{i2} = \varnothing$. From algorithm 4, when $i\%2 == 1 \wedge i \ge 3$ and $R_i$ reaches the current fixpoint, then $G_{i1}^{Even(h(\overline{l},i))} = R_{i1}$, $G_{i0}^{Even(h(\overline{l},i))} = R_{i0}$, according to lemma 5 and lemma 6, this processing is correct and can reduce the computing times of the algorithm to improve the performance of the algorithm. When the nodes in $R_1$ reach fixpoint state, it shows that the whole computation is over. If $X_1.\sigma = true$ in $(P, X_1(\overline{x}, \phi, \sigma))$, then $P \vDash \phi$, otherwise $P \nvDash \phi$. □

**Theorem 3:** The time complexity of algorithm 5 is $O(d^2 \cdot (2M/d)^{d/2+2})$.

**Proof** (brief)**:**

The key operating in algorithm 5 is *ValueSet*() which comes from algorithm 4. Assume $m_i = |R_i|$ $(1 \le i \le d)$ is the size of corresponding node set in $R_i$, then $M = \sum_{i=1}^{d} m_i$. The calling times of function $f_i$ are computing times of the corresponding nodes in $R_i$, label it as $|f_i|$.

According to algorithm 4, the computation sequence of the corresponding nodes in $R_1$ is $R_1^0, R_1^1, R_1^2, \cdots, R_1^{\omega}$, the total number of computing nodes is $m_1$. If $\sigma_1 = \nu$, the initial value of each node's *val* in $R_{10}$ is *true*. In the process of computing, once the value of *val* turns into *false*, the node enters into $R_{11}$. The value of *val* in the node does not change again in the process of the whole computing from the monotonicity of $f_1$. At worst, the node is found out whose *val*ischanged into *false* from *true* after computing all nodes in $R_{10}$. If $\sigma_1 = \mu$, the analysis process is similar. So at worst, the computing times of the corresponding nodes in $R_1$ are $|f_1| = 1 + 2 + ... + m_1 = m_1(m_1 + 1)/2 \le m_1^2$.

When the value of $R_1$ is $R_1^{i_1}$, the computation sequence of the corresponding nodes in $R_2$ is $R_2^{i_1 0}, R_2^{i_1 1}, R_2^{i_1 2}, \cdots, R_2^{i_1 \omega}$, the computing times of the corresponding nodes in $R_2$ are $1 + 2 + ... + m_2$, the numbers of different values of $R_1$ are $m_1$ at most, so $|f_2| = m_1(1 + 2 + ... + m_2) \le m_1 \cdot m_2^2$.

According to algorithm 4, only when $i_1 i_3 = 00$, $R_3$ starts to compute from $S_{30}$. The times of this case are $m_2$ at most. The times of changing corresponding node's value in block $R_3$ are $m_2 \cdot m_3$ and the computing times are not greater than $m_2 \cdot m_3^2$. When a node's value in $R_2$ has be changed, $R_3$ needs to be recomputed the fixpoint once, the times of this case are $m_1 \cdot m_2$ at most, the computing times are not greater than $m_1 \cdot m_2 \cdot m_3$ at most, so $|f_3| \le m_2 \cdot m_3^2 + m_1 \cdot m_2 \cdot m_3$.

According to algorithm 4, only when $R_3$ is changed, $R_4$ needs to compute from $S_{40}$. The times of changing corresponding node's value in $R_3$ are not greater than $m_2 \cdot m_3$, so $|f_4| \le m_2 \cdot m_3 \cdot m_4^2$

When $i_1 i_3 i_5 = 000$, $R_5$ starts to compute from $S_{50}$. The times of this case are $m_2 \cdot m_4$ at most. The times of changing corresponding node's value in $R_5$ are $m_2 \cdot m_4 \cdot m_5$ and the computing times are not greater than $m_2 \cdot m_4 \cdot m_5^2$. When a node's value in $R_4$ has be changed, $R_5$ needs to be recomputed the fixpoint once, the times of this case are $m_2 \cdot m_3 \cdot m_4$ at most, the computing times are $m_2 \cdot m_3 \cdot m_4 \cdot m_5$ at most, so $|f_5| \le m_2 \cdot m_4 \cdot m_5^2 + m_2 \cdot m_3 \cdot m_4 \cdot m_5$.

After changing any corresponding node's value of $R_5$, the node in $R_6$ needs to compute from $S_{60}$. At worst, the times of this case are not greater than $m_2 \cdot m_4 \cdot m_5$, so the computing times of the corresponding node in equality block $R_6$ are $|f_6| \le m_2 \cdot m_4 \cdot m_5 \cdot m_6^2$.

……

When $i \ge 3$, $|f_{2i}| \le m_2 \cdot m_4 \cdot \ldots \cdot m_{2i-2} \cdot m_{2i-1} \cdot m_{2i}^2$

$$\begin{aligned}|f_{2i+1}| &\le m_2 \cdot m_4 \cdot \ldots \cdot m_{2i} \cdot m_{2i+1}^2 + m_2 \cdot m_4 \cdot \ldots \cdot m_{2i-2} \cdot m_{2i-1} \cdot m_{2i} \cdot m_{2i+1} \\ &= (m_2 \cdot m_4 \cdot \ldots \cdot m_{2i} \cdot m_{2i+1}) \cdot (m_{2i-1} + m_{2i+1})\end{aligned}$$

When the alternate nesting depth is $d$ and $d \bmod 2 = 0$, the total computing times of algorithm 4 are:

$$\begin{aligned}\sum_{i=1}^{d}|f_i| &\le m_1^2 + m_1 \cdot m_2^2 + m_2 \cdot m_3^2 + m_1 \cdot m_2 \cdot m_3 + m_2 \cdot m_3 \cdot m_4^2 + m_2 \cdot m_4 \cdot m_5^2 \\ &\quad + m_2 \cdot m_3 \cdot m_4 \cdot m_5 + m_2 \cdot m_4 \cdot m_5 \cdot m_6^2 + \ldots + (m_2 \cdot m_4 \cdot \ldots \cdot m_{d-2} \cdot m_{d-1}) \cdot (m_{d-3} + m_{d-1}) \\ &\quad + m_2 \cdot m_4 \cdot \ldots \cdot m_{d-2} \cdot m_{d-1} \cdot m_d^2 \\ &< (m_2 \cdot m_4 \cdot \ldots \cdot m_{d-2} \cdot m_d) \cdot (m_1^2 + m_3^2 + \ldots + m_{d-1}^2 + m_1 \cdot m_2 + m_1 \cdot m_3 + m_3 \cdot m_4 + m_3 \cdot m_5 \\ &\quad + m_5 \cdot m_6 + \ldots + m_{d-3} \cdot m_{d-1} + m_{d-1} \cdot m_d) \\ &< (m_2 \cdot m_4 \cdot \ldots \cdot m_{d-2} \cdot m_d) \cdot M^2 \\ &= O(d^2 \cdot (2M/d)^{d/2+2})\end{aligned}$$

When $d \bmod 2 = 1$, the total computing times of algorithm 4 are:

$$\begin{aligned}\sum_{i=1}^{d}|f_i| &\le m_1^2 + m_1 \cdot m_2^2 + m_2 \cdot m_3^2 + m_1 \cdot m_2 \cdot m_3 + m_2 \cdot m_3 \cdot m_4^2 + m_2 \cdot m_4 \cdot m_5^2 \\ &\quad + m_2 \cdot m_3 \cdot m_4 \cdot m_5 + m_2 \cdot m_4 \cdot m_5 \cdot m_6^2 + \ldots + (m_2 \cdot m_4 \cdot \ldots \cdot m_{d-1} \cdot m_d) \cdot (m_{d-2} + m_d) \\ &< (m_2 \cdot m_4 \cdot \ldots \cdot m_{d-1}) \cdot (m_1^2 + m_3^2 + \ldots + m_d^2 + m_1 \cdot m_2 + m_1 \cdot m_3 + m_3 \cdot m_4 + m_3 \cdot m_5 \\ &\quad + m_5 \cdot m_6 + \ldots + m_{d-2} \cdot m_d) \\ &< (m_2 \cdot m_4 \cdot \ldots \cdot m_{d-1}) \cdot M^2 \\ &= O(d^2 \cdot (2M/d)^{\lfloor d/2 \rfloor + 2})\end{aligned} \qquad \square$$

**Theorem 4:** The space complexity of algorithm 5 is $O(2M/d)^{d/2}$

**Proof** (brief):

The main quantity demanded of memory is *ValueSet*() which comes from algorithm 4. In the process of computing of algorithm 4, $R_{i0}$ and $R_{i1} (1 \le i \le d)$ needs a storage unit respectively, the total numbers of storage units are 2*d*, execute statement $if(i\%2 == 1 \wedge i \ge 3) G_{i1}^{Even(h(\bar{l},i))} = R_{i1}, G_{i0}^{Even(h(\bar{l},i))} = R_{i0}$; when $i = 3$, it needs $2m_2$ storage units to save intermediate results. When $i = 5$, it needs $2m_2 \cdot m_4$ storage units, …. When $i = d - 1$, it needs $2m_2 \cdot m_4 \cdot m_6 \cdot \ldots \cdot m_{d-2}$ storage units. So the total numbers of space in algorithm 4 are:

$$2d + 2m_2 + 2m_2 \cdot m_4 + \ldots + 2m_2 \cdot m_4 \cdot m_6 \cdot \ldots \cdot m_{d-2} < O(2M/d)^{d/2} \qquad \square$$

## 6 Conclusion

This paper deeply analyzes and studies the model checking algorithm of finite-control mobile ambients for μ- calculus first-order predicate ambient logic. Based on Tarski's fixpoint theorem, a group of partial ordering relation in the process of computing is found, and comes up with an algorithm whose time complexity has exponent relation to *d*/2+2 (*d* is the alternating nesting depth of the formula). To the author's knowledge, this is the third model checking algorithm of finite-control mobile ambients for μ-calculus predicate ambient logic based on first-order, and has the best performance at present. The improvement of the algorithm performance makes it possible to solve ambient logic formula with more alternating nesting depth of fixpoint operators. The research findings in this paper are significant in both theory study and practical application in model checking of mobile ambients. At the same time, the research method in this paper also applies to the design of local model checking algorithm in π-calculus and CCS calculus. The best results of the current model checking of μ-calculus is that: time complexity has exponent relation to half of the alternative nesting depth of the formula while space complexity has linear relation to the alternative nesting depth of the formula, next work is considering whether we can further reduce space complexity of the model checking algorithm of finite-control mobile ambients for μ-calculus predicate ambient logic based on first-order, and develop the tool of model checking for finite-control mobile ambients.

## Acknowledgements

This work is supported by National Nature Science Foundation of China (No. 61472406), Nature Science Foundation of Fujian (No. 2015J01269 and No. 2016J01304), the Talent Introduction Foundation of Minnan Normal University, and Yongzhou Municipal Science and Technology Bureau project [(2015)7]

## References

[1] L. Cardelli, A.D. Gordon, Mobile ambients, Theoretical Computer Science 240(1)(2000) 77-213.

[2] D.Y. Kao, Exploring privacy requirements and their online managements, Journal of computers 26(2)(2015) 34-45.

[3] C.C. Chang, Y.H. Huang, Sharing large secret messages using two AMBTC shadows, Journal of Computers 26(2)(2015) 56-63.

[4] L. Cardelli, A.D. Gordon, Anytime, anywhere: modal logics for mobile ambients, in: Proc. the POPL'2000, 2000.

[5] L. Cardelli, A.D. Gordon, Logical properties of name restriction, in: Proc. the CONCUR 2001, 2001.

[6] L. Cardelli, G. Ghelli, A query language based on the ambient logic, in: Proc. the European Symposium on Programming (ESOP'01), 2001.

[7] S.D. Zilio, Fixed points in ambient logic, in: Proc of the 3rd Workshop on Fixed Points in Computer Science, 2001.

[8] H.M. Lin, Space logic of mobile ambients, Science in China Series E: Technological Sciences 34(2)(2004) 139-150.

[9] H.M. Lin, A predicate mu-calculus for mobile ambients, Journal of Computer Science and Technology 20(1)(2005) 95-104.

[10] W. Charatonik, J.M. Talbot, The decidability of model checking mobile ambients, in: Proc. the 15th Annual Conference of European Association for Computer Science Logic, 2001.

[11] W. Charatonik, S. Dal Zilio, A.D. Gordon, S. Mukhopadhyay, The complexity of model checking mobile ambients, in: Proc. the 4th International Conf on Foundations of Software Science and Computation Structures, 2001.

[12] W. Charatonik, A. Gordon, J.M. Talbot, Finite-control mobile ambients, in: Proc. the European Symposium on Programming (ESOP'02), 2002.

[13] H. Jiang, X. Li, Model checking for mobile ambients, Journal of Computer Research and Development 46(10)(2009) 1750-1757.

[14] G. Fabio, V.M. Giacoma, A decentralised graphical implementation of mobile ambients, The Journal of Logic and Algebraic Programming 80(2)(2011) 113-136.

[15] L. Brodo, On the expressiveness of the π-calculus and the mobile ambients, in: T. Rus (Ed.), Algebraic Methodology and Software Technology, Springer Berlin Heidelberg, Germany, 2011, pp. 44-59.

[16] B. Aman, G. Ciobanu, Coordinating parallel mobile ambients to solve SAT problem in polynomial number of steps, in: M. Sirjani (Ed.), Coordination Models and Languages, Springer Berlin Heidelberg, Germany, 2012, pp. 122-136.

[17] N. Ali, F. Chen, C. Solis, Modeling support for mobile ambients in service oriented architecture, in: Proc. the 2012 IEEE First International Conference on Mobile Services (MS), 2012.

[18] D. Unal, M.U. Caglayan, XFPM-RBAC: XML-based specification language for security policies in multidomain mobile networks, Security and Communication Networks 6(12)(2013) 1420-1444.

[19] C. Bodei, L. Brodo, R. Bruni, Open multiparty interaction, in N. Marti-Oliet, M. Palomino (Eds.), Recent Trends in Algebraic Development Techniques, Springer Berlin Heidelberg, Germany, 2013, pp. 1-23.

[20] A. Bogdan, C. Gabriel, Expressing mobile ambients in temporal logic of actions, Proceedings of the Romanian Academy, series A 15(1)(2014) 95-104.

[21] F. Siewe, A privacy type system for context-aware mobile ambients, Procedia Computer Science 52(2015) 98-105.

[22] A. Tarski, A lattice-theoretical fixpoint theorem and its applications, Pacific Journal of Mathematics 5(2)(1955) 285-309.

## Appendix

**Theorem 1**: Let $\phi$ be a proposition, $\xi$ be an environment, $\phi$ is translated into nested predicate equation system $\varepsilon$ by algorithm 1, $1 \le i \le k, \{X_k\} \cap fpv(\phi) = \varnothing$, if $(X(\overline{x}), \varepsilon, j) = ftoe(A, \sigma, 1, 0)$, then $[\![(\overline{x})\phi]\!]\xi = \| \varepsilon \| \xi(X)$

**Proof**: Without loss of generality, let $\sigma = \nu$ (for the case of $\sigma = \mu$, the proof is similar).

(1) $\phi \equiv p$

$\| \varepsilon \| \xi(X) \equiv \| X_i(\overline{x_i}) =_\sigma p \| \xi(X_i)$

$= \bigcup \{f \mid f \subseteq [\![p]\!]\xi[f / X_i]\}$ // the semantics of nested predicate equation system

$= [\![\nu X_i.p]\!]\xi$ // the semantics of logic formula

$= [\![p]\!]\xi$ // $X_i$ does not appear in $p$

Other cases for $\phi$ are structural induction as below. Induction hypothesis: let $\phi'$ be any proposition whose length is less than $\phi$, $\xi$ be an environment, $1 \le i \le k$, $X_k$ does not appear freely in $\phi'$, if $(X(\overline{x}), \varepsilon', j) = ftoe(\phi', \sigma, i, k)$, there is $[\![(\overline{x})\phi']\!]\xi = \| \varepsilon' \| \xi(X)$.

(2) $\phi \equiv Y(\overline{e})$, if there is $X_j (j < i)$, and $Y \equiv X_j$, label it as $fnv(X_j) = \{\overline{x_j}\}$

$\| \varepsilon \| \xi(X) \equiv \| X_i(\overline{x_i}) =_\nu ((\overline{x_i})(X_j(\overline{x_j})))(\overline{e}) \| \xi(X_i)$

$= \| X_i(\overline{x_i}) =_\nu ((\overline{x_i})(X_j(\overline{e})) \| \xi(X_i)$

$= \bigcup \{f \mid f \subseteq [\![(\overline{x_i})(X_j)(\overline{e})]\!]\xi[f / X_i]\}$

$= [\![\nu X_i.(\overline{x_i})(X_j(\overline{e}))]\!]\xi$

$= [\![(\overline{x_i})(X_j(\overline{e}))]\!]\xi$ // $X_i \ne X_j$

$= [\![(\overline{x_i})\phi]\!]\xi$

If there is not $X_j$, there is $\| \varepsilon \| \xi(X) = [\![(\overline{x_i})(Y\overline{n}))]\!]\xi = [\![(\overline{x_i})\phi]\!]\xi$ in a similar way.

(3) $\phi \equiv \phi \circledast \phi$

Let $(Y(\overline{y}), \varepsilon_1, i_1) = fote(\phi_1, \sigma, i+1, k), (Z(\overline{z}), \varepsilon_2, i_2) = fote(\phi_2, \sigma, i_1, k)$

From induction hypothesis there is $[\![(\overline{y})\phi_1]\!]\xi = \| \varepsilon_1 \| \xi(Y), [\![(\overline{z})\phi_2]\!]\xi = \| \varepsilon_2 \| \xi(Z)$

Because $lhs(\varepsilon_1) \bigcap lhs(\varepsilon_2) = \phi, lhs(\varepsilon_1) \bigcap rhs(\varepsilon_2) = \phi$ and $rhs(\varepsilon_1) \bigcap lhs(\varepsilon_2) = \phi,$ , according to lemma 1, there is $[\![(\overline{y})\phi_1]\!]\xi = \| \varepsilon_1 :: \varepsilon_2 \| \xi(\overline{y}), [\![(\overline{z})\phi_2]\!]\xi = \| \varepsilon_1 :: \varepsilon_2 \| \xi(\overline{z})$, and $[\![(\overline{x_i})\phi]\!]\xi = [\![(\overline{x_i})(Y(\overline{y})) \circledast (Z(\overline{z}))]\!] \| \varepsilon_1 :: \varepsilon_2 \| \xi$ is easily to get from the semantics of ambient logic.

$\| \varepsilon \| \xi(X) \equiv \| X_i(\overline{x_i}) = Y(\overline{y}) \circledast Z(\overline{z})\} :: \varepsilon_1 :: \varepsilon_2 \| \xi(X_i)$

$= \bigcup \{f \mid f \subseteq [\![(\overline{x_i})(Y(\overline{y}) \circledast Z(\overline{z}))]\!] \| \varepsilon_1 :: \varepsilon_2 \| \xi [f / X_i]$

$= \bigcup \{f \mid f \subseteq [\![(\overline{x_i})(\phi_1 \circledast \phi_2]\!]\xi [f / X_i]\}$

$= [\![\nu X_i.(\overline{x_i})(\phi_1 \circledast \phi_2]\!]\xi$

$= [\![(\overline{x_i})(\phi_1 \circledast \phi_2]\!]\xi$

(4) The proof of $\equiv \ominus \phi'$, $\phi \equiv \odot \phi'$ and $\phi \equiv ((\overline{x})\,\phi')(\overline{n})$ resembles the above case and are omitted here.

(5) $\phi \equiv (\sigma' X.((\overline{x}')\phi'))(\overline{e})$

Without loss of generality, let $\sigma' = \mu$ (for the case of $\sigma' = v$, the proof is similar). Let $(Y(\overline{y}), \varepsilon_1, i_1) =$ $ftoe(\phi_1'[X_{i+1}/X], \sigma', i+2, ad(X))$, $\varepsilon' = \{X_{i+1}(\overline{y}) =_{\sigma'} Y(\overline{y})\} :: \varepsilon_1$, $\varepsilon = \{X_i(\overline{x_i}) =_\sigma ((\overline{y})X_{i+1}(\overline{y})))(\overline{e})\} :: \varepsilon'$, there is $\{\overline{x}'\} = \{\overline{y}\}$.

Induction hypothesis: $[\![(\overline{y})\phi'[X_{i+1}/X]]\!]\xi = \|\varepsilon_1\|\xi(Y)$

$$\begin{aligned}
\|\varepsilon'\|\xi(X_{i+1}) &\equiv \|X_{i+1}(\overline{y}) =_{\sigma'} Y(\overline{y})\} :: \varepsilon_1\|\xi(X_{i+1}) \\
&= \bigcap\{f \mid f \supseteq [\![Y]\!]\|\varepsilon_1\|\xi[f/X_{i+1}]\} \\
&= \bigcap\{f \mid f \supseteq [\![(\overline{y}\phi')[X_{i+1}/X]]\!]\xi[f/X_{i+1}]\} \\
&= [\![\mu X_{i+1}\cdot(\overline{y})\phi'[X_{i+1}/X]]\!]\xi
\end{aligned}$$

$$\begin{aligned}
\|\varepsilon\|\xi(X) &\equiv \|X_i(\overline{x_i}) =_\sigma ((\overline{y})(X_{i+1}(\overline{y})))(\overline{e})\} :: \varepsilon'\|\xi(X_i) \\
&= [\![vX_i.(\overline{x_i})(X_{i+1}(\overline{e}))]\!]\|\varepsilon'\|\xi \\
&= [\![(\overline{x_i})((vX_{i+1}.(\overline{y})\phi')[X_{i+1}/X](\overline{e})]\!]\xi \\
&= [\![(\overline{x_i})((vX.(\overline{x'})\phi')(\overline{e})]\!]\xi \\
&= [\![(\overline{x_i})\phi]\!]\xi
\end{aligned}$$

**Lemma 4:** The number of nodes produced by function *Install_Set*()in algorithm 2 is less than $M$.

**Proof** (brief):

First of all, because the nested predicate equation systems $\mathcal{E}$ translated by function *formulaToequation*() is semantically correspond with the formula *A*, then the size of $\mathcal{E}$ is correspond with the size of predicate variable set exists in the left of each equation, that is $|lhs(\mathcal{E})| = op(A) + op_{\wedge,\vee,|}(A)$. And if the effectiveness of operator $'|,®,\forall,\exists'$, is not taken into account, the number of computing nodes produced by function $Install\_Set()$ in algorithm 2 is less than $V_{LTS}(P) \times [op(A) + op_{\wedge,\vee,|}(A)]$.

A single $'|'$, $op_|(A)$, '®' and $op_®(A)$ operator in the formula may increase the number of computing nodes to $2^{par(P)}$, $2^{par(P)\times op_|(A)}$, $|bn(P)|+1$ and $[|bn(P)|+1]^{op_®(A)}$ times at most respectively.

For any process *P* and any formula *A*, there are $|fn(P,A)| \le |N(P,A)|$.

After some steps of transition, *P* is reduced into *P'*, *A'* is sub-formula of *A*, and there is $|fn(P',A')| \le |N(P,A)|$. And given a formula *A*, one $'\forall'$ or $'\exists'$ can only increase the number of computing nodes to $|N(P,A)|$ times at most. So if the number of $'\forall'$ and $'\exists'$ is $op_{\forall,\exists}(A)$, then it can cause the number of computing nodes increase to $|N(P,A)|^{op_{\forall,\exists}(A)}$ times at most.

In conclusion, the number of computing nodes produced by function $Install\_Set()$ in algorithm 2 is less than:

$$V_{LTS}(P) \times [op(A) + op_{\wedge,\vee,|}(A)] \times 2^{par(P)\times op_|(A)} \times [|bn(P)|+1]^{op_®(A)} \times |N(P,\phi)|^{op_{\forall,\exists}(A)} \qquad \square$$

**Lemma 5:** If $\sigma_k = \mu$, then $R_k^{P(j_1 j_2 \cdots j_k)} \Subset R_k^{j_1 j_2 \cdots j_k}$.

**Proof:**

(1) case: $k = d$

Let $P(j_1 j_2 j_3 \ldots j_k) = l_1 l_2 l_3 \ldots l_k$ and $j_i = l_i + 1$.

(a) Only the *d*th bit are different in both sequences, then $P(j_1 j_2 j_3 \ldots (j_d + 1)) = j_1 j_2 j_3 \ldots j_d$ and

$$R_d^{j_1 j_2 j_3 \ldots (j_d+1)} = f_d(R_1^{j_1}, R_2^{j_1 j_2}, \ldots, R_{d-1}^{j_1 j_2 \cdots j_{d-1}}, R_d^{j_1 j_2 \cdots j_{d-1} j_d}),\ \sigma_d = \mu$$

$R_d^{j_1 j_2 \cdots j_{d-1} j_d} \Subset R_d^{j_1 j_2 \cdots j_{d+1}}$ is correct obviously.

(b) Only the (*d*-2)th bit is different in both sequences, then $P(j_1 j_2 j_3 \ldots (j_{d-2}+1) j_{d-1} j_d) = j_1 j_2 j_3 \ldots j_{d-2} j_{d-1} j_d$. According to Tarski's fixpoint theorem, there are

$$R_d^{0\ldots0001} = f_d(R_1^{0}, R_2^{00}, \ldots, R_{d-3}^{0\ldots0}, R_{d-2}^{0\ldots00}, R_{d-1}^{0\ldots000}, R_d^{0\ldots0000})$$
$$R_d^{0\ldots0101} = f_d(R_1^{0}, R_2^{00}, \ldots, R_{d-3}^{0\ldots0}, R_{d-2}^{0\ldots01}, R_{d-1}^{0\ldots010}, R_d^{0\ldots0100})$$

Because $R_{d-1}^{0\ldots000} = R_{d-1}^{0\ldots010}$ , $R_d^{0\ldots0000} = R_d^{0\ldots0100}$ , $\sigma_{d-2} = \mu \Rightarrow$ $R_{d-2}^{0\ldots00} \Subset R_{d-2}^{0\ldots01}$, $f_d$ is a monotonic function, there is $R_d^{0\ldots0001} \Subset R_d^{0\ldots0101}$. Analogously, there are $R_d^{0\ldots000k} \Subset R_d^{0\ldots010k}$, $R_d^{0\ldots000\omega} \Subset R_d^{0\ldots010\omega}$. And according to Tarski's fixpoint theorem, there are

$$R_{d-1}^{0\ldots001} = f_{d-1}(R_1^{0}, R_2^{00}, \ldots, R_{d-3}^{0\ldots0}, R_{d-2}^{0\ldots00}, R_{d-1}^{0\ldots000}, R_d^{0\ldots000\omega})$$
$$R_{d-1}^{0\ldots011} = f_{d-1}(R_1^{0}, R_2^{00}, \ldots, R_{d-3}^{0\ldots0}, R_{d-2}^{0\ldots01}, R_{d-1}^{0\ldots010}, R_d^{0\ldots010\omega})$$

Similarily, because $R_{d-2}^{0\ldots00} \Subset R_{d-2}^{0\ldots01}$, $R_{d-1}^{0\ldots000} \Subset R_{d-1}^{0\ldots010}$, $R_d^{0\ldots000\omega} \Subset R_d^{0\ldots010\omega}$, $f_{d-1}$ is a monotonic function, there is $R_{d-1}^{0\ldots000} \Subset R_{d-1}^{0\ldots011}$. So there are $R_{d-1}^{0\ldots00\,j_{d-1}} \Subset R_{d-1}^{0\ldots01\,j_{d-1}}$, $R_{d-1}^{0\ldots00\omega} \Subset R_{d-1}^{0\ldots01\omega}$, and there are

$$R_d^{0\ldots00\,j_{d-1}1} = f_d(R_1^{0}, R_2^{00}, \ldots, R_{d-3}^{0\ldots0}, R_{d-2}^{0\ldots00}, R_{d-1}^{0\ldots00\,j_{d-1}}, R_d^{0\ldots00\,j_{d-1}0})$$
$$R_d^{0\ldots01\,j_{d-1}1} = f_d(R_1^{0}, R_2^{00}, \ldots, R_{d-3}^{0\ldots0}, R_{d-2}^{0\ldots01}, R_{d-1}^{0\ldots01\,j_{d-1}}, R_d^{0\ldots01\,j_{d-1}0})$$

Finally, because $R_{d-2}^{0\ldots00} \Subset R_{d-2}^{0\ldots01}$,, $R_{d-1}^{0\ldots00\,j_{d-1}} \Subset R_{d-1}^{0\ldots01\,j_{d-1}}$, $R_d^{0\ldots00\,j_{d-1}0} \Subset R_d^{0\ldots01\,j_{d-1}0}$, there is $R_d^{0\ldots00\,j_{d-1}1} \Subset R_d^{0\ldots01\,j_{d-1}1}$. On the analogy of this, there are $R_d^{0\ldots00\,j_{d-1}j_d} \Subset R_d^{0\ldots01\,j_{d-1}j_d}$, $R_d^{0\ldots00\,j_{d-1}\omega} \Subset R_d^{0\ldots01\,j_{d-1}\omega}$.

Repeat above analysis, there is $R_d^{0\ldots0\,j_{d-2}j_{d-1}j_d} \Subset R_d^{0\ldots0(j_{d-2}+1)j_{d-1}j_d} \Subset R_d^{0\ldots0(j_{d-2}+2)j_{d-1}j_d} \Subset\cdots\Subset R_d^{0\ldots0(j_{d-2}+k)j_{d-1}j_d}$, so $R_k^{P(j_1 j_2 \cdots j_k)} \Subset R_k^{j_1 j_2 \cdots j_k}$

(c) Only the $d-2i(2<i<d/2)$ th bit are different in sequence $P(j_1 j_2 j_3 \cdots j_k)$ and sequence $j_1 j_2 j_3 \cdots j_k$. By similar analyzing to (b), lemma 5 is correct.

(2) When $1 \le k < d$, the analysis is similar to the case of $k = d$. □