基于偏序规律的 μ-演算一阶谓词界程逻辑模型检测

江 华

(闽南师范大学粒计算重点实验室 福建 漳州 363000)
(韶关学院信息科学与工程学院 广东 韶关 512005)

摘 要 基于 μ-演算的一阶谓词界程逻辑,用谓词变量构造不动点公式,方便描述闭环系统的性质,公式语义简洁.该逻辑在有限控制移动界程上的模型检测目前性能最好的算法的时间复杂度与公式中不动点算子交错嵌套深度 d 呈指数关系,空间复杂度与 d 呈线性关系.文中设计了一个基于 μ-演算的一阶谓词界程逻辑在有限控制移动界程上的模型检测高效算法,这也是目前已知的第 3 个同类算法,算法的时间复杂度与 $d/2+1$ 呈指数关系,空间复杂度与 d 呈线性关系.文中所做的工作有:(1)找到了基于 μ-演算的一阶谓词界程逻辑模型检测计算过程中的中间结果满足的两组偏序关系;(2)利用找到的偏序关系设计了一个快速模型检测算法;(3)分析了算法的复杂度.

关键词 模型检测;移动界程;μ-演算;嵌套谓词等式系
中图法分类号 TP316 **DOI 号** 10.11897/SP.J.1016.2016.02547

Model Checking for First-Order Predicate Ambient Logic Based on μ-Calculus with Partial Orders

JIANG Hua

(*Key Laboratory of Granular Computing, Minnan Normal University, Zhangzhou, Fujian* 363000)
(*School of Information Science and Engineering, Shaoguan University, Shaoguan, Guangdong* 512005)

Abstract First-order predicate ambient logic based on μ-calculus uses predicate variable to construct fixpoint formula which is convenient to describe the properties of closed-loop system. The formula's semantics is brief. For the best model checking algorithm in finite control mobile ambient based on the logic, its time complexity has exponent relation to d and space complexity has linear relation to d, and d is the alternating nesting depth of the formula in fixpoint. This paper comes up with an efficient model checking algorithm for first-order predicate ambient logic in finite control mobile ambient based on μ-calculus, which is the third known kindred algorithm at present, whose time complexity has exponent relation to $d/2+1$, space complexity has linear relation to d. This paper's contributions are: (1) getting two groups of partial ordering relation among intermediate results in the process of computing for first-order predicate ambient logic model checking based on μ-calculus; (2) using the partial ordering relation to design a quick algorithm for model checking; (3) analyzing the complexity of this algorithm.

Keywords model checking; mobile ambient; μ-calculus; nested predicate equations

1 引 言

移动界程演算(Mobile Ambients)[1]最先由 Cardelli 等人于 2000 年提出,用于分布式移动计算系统的形式化建模与验证,空间描述能力强.2000 年 Cardelli 等人[2]首次提出界程逻辑,具备同时表达进程在时间和空间上演变的能力.2001 年 Cardelli

收稿日期:2015-01-08;在线出版日期:2015-07-15.本课题得到国家自然科学基金(61472406)、福建省自然科学基金(2015J01269,2016J01304)、韶关市哲学社会科学规划课题(G2012001)、广东省自然科学基金(S2013010015944)和闽南师范大学人才引进项目资助.
江 华,男,1970 年生,博士,副教授,主要研究方向为模型检测、算法设计与分析.E-mail: sg_jh@126.com.

等人[3]将受囿名量词和新名量词引入界程逻辑；Cardelli 等人[4]和 Zilio① 分别在一些界程逻辑片断中引入不动点算子，但是这些逻辑片断没有受囿名量词. 中国科学院林惠民院士[5-6]于 2004 至 2005 年提出了基于 μ-演算的一阶谓词界程逻辑，引入新名字量词和不动点算子，用谓词变量构造不动点公式，简化了公式语义. 在移动界程模型检测方面，Cardelli 等人[2]和 Charatonik 等人[7]研究了几种有穷界程和有限界程的模型检测问题，Charatonik 等人[8]对上述模型检测算法的复杂度进行了分析，2002 年 Charatonik 等人[9]提出了有限控制移动界程(Finite-control Mobile Ambients，FMA)，并提出了针对 FMA 的模型检测算法，但上述这些研究工作采用的逻辑都没有递归，是有穷的. 林惠民院士[5]于 2004 年提出了第一个针对有限控制移动界程上基于 μ-演算的一阶谓词界程逻辑模型检测算法，指出算法是可判断的；江华和李祥[10]于 2009 年研究了有限控制移动界程上基于 μ-演算的一阶谓词界程逻辑模型检测问题，设计了一个局部模型检测算法，算法时间复杂度的指数部分是 $O(d)$，d 是界程逻辑公式中不动点算子交错嵌套深度，这是自林惠民院士的研究工作之后的又一个基于 μ-演算的一阶谓词界程逻辑模型检测算法.

μ-演算模型检测技术[11-14]自提出以后，在有限控制并发系统的设计与验证中得到了广泛应用. 设 n 为变迁系统状态规模，k 为 μ-演算公式中不动点算子嵌套深度，d 为 μ-演算公式中不动点算子交错嵌套深度. 根据 Tarski 不动点定理[15]，在完全 μ-演算全局模型检测计算时，可以用逐次迭代来计算包含不动点算子嵌套的 μ-演算公式，μ-演算全局模型检测算法的时间复杂度为 $O(n^{k+1})$. 1986 年 Emerson 和 Lei[16]证明了 μ-演算公式中相同类型的不动点算子嵌套时可以不改变模型检测算法的复杂度，据此设计的 μ-演算全局模型检测算法的时间复杂度为 $O(n^{d+1})$. 随后 Andersen[17]、Cleaveland 和 Steffen 等人[18-19]对文献[16]的算法进行了一些改进，但算法的时间复杂度不变. 1994 年 Long 等人[20]对用 Tarski 不动点定理计算 μ-演算公式的计算过程进行了细致的分析，找到了计算过程中的中间结果间存在的一组偏序关系，据此设计了一个时间复杂度和空间复杂度同为 $O(n^{\lceil d/2\rceil+1})$ 的 μ-演算全局模型检测算法. 2010 年江华[21]对 $d=4$ 的 μ-演算全局模型检测的计算过程进行了深入分析，找到了其计算过程中的中间结果间存在的两组偏序关系，据此设计的 μ-演算全局模型检测算法的时间复杂度与 $d/2$ 呈指数关系，空间复杂度为 $O(dn)$. 相关的工作还有：Liu 等人[22]于 1998 年提出了一个高性能的命题 μ-演算局部模型检测算法；刘剑和林惠民院士[23]于 2003 年证明了模态图与谓词 μ-演算公式在语义上的一致性.

近几年在移动界程演算模型检测方面的研究还有：2011 年 Fabio 和 Giacoma[24]提出了一个有限进程移动界程的图形表示方法，实现了对递归语义的图形化建模；Brodo[25]对 π-演算和移动界程的描述能力进行了研究，通过提出轨迹模拟和轨迹中止的概念从而针对模型中的闭环提出了一个解决方案. 2012 年 Aman 和 Ciobanu[26]提出了一个移动界程的改良版本——parMA，给出布尔可满足问题基于 parMA 的求解方案；Ali 等人[27]结合面向服务的体系结构(Service-Orierited Architecture，SOA)建模语言和移动界程，提出了界程-SOA 模型并开发了相应的建模工具，允许图形化建模移动的 SOA 模型. 2013 年 Unal 等人[28]提出了 XFPM-RBAC(XML-based Formal Policy language for Mobility with Role-Based Access Control)，用移动界程和界程逻辑描述角色的移动性和角色的安全规则；Bodei 等人[29]结合 CCS 演算、π-演算和移动界程提出了 LINK 演算，能很好地对开放的多进程系统进行描述和建模. 2014 年 Aman 等人[30]提出了行为时序逻辑移动界程演算，通过实例展示了该演算的描述和验证能力. 2015 年 Siewe[31]提出了基于移动界程的隐私类型系统，确保隐私信息在计算过程中的安全性.

本文通过挖掘基于 μ-演算的一阶谓词界程逻辑模型检测算法在计算过程中中间结果间存在的偏序关系，设计出高性能的移动界程模型检测算法，大大降低了基于 μ-演算的一阶谓词界程逻辑模型检测算法的复杂度，使之更接近实际应用的需要. 本文的研究方法与文献[21]所用的研究方法类似，均是挖掘按照 Tarski 不动点定理计算不动点时中间结果间存在的偏序关系来优化算法设计，且本文算法与文献[21]中算法具有相同的时间复杂度和空间复杂度. 但是本文与文献[21]有两点重要的区别：

① Zilio S D. Fixed points in the ambient logic[C/OL]. Proceedings of the 3rd Workshop on Fixed Points in Computer Science (FICS 2001). Florence, Italy, 2001. http://www.researchgate.net/publication/2378344_Fixed_ Points_in_the_Ambient_Logic

(1)两者研究的对象不同，本文研究的是基于 μ-演算的一阶谓词界程逻辑模型检测算法，文献[21]研究的是命题 μ-演算模型检测算法；(2)算法类型不同，本文算法属于局部模型检测算法，文献[21]中算法属于全局模型检测算法.

2　移动界程演算

令 $\mathcal{N}$ 是名字可数集，$n,m,\cdots\in\mathcal{N}$，$\mathcal{W}$ 是进程变量可数集，$X,Y,\cdots\in\mathcal{W}$，移动界程演算语法的 BNF 定义如下[1]：

$M::=$ capabilities

$in\ n$ can enter $n\mid out\ n$ can exit $n\mid open\ n$ can open $n\mid$ ε null path$\mid M.M$ composite path

$P,Q::=$ processes

0 inactivity$\mid P\mid Q$ composition$\mid(\nu n)P$ name restriction$\mid$ $n[P]$ ambient$\mid M.P$ capability action$\mid(n).P$ input$\mid$ $\langle n\rangle$ output$\mid X$ variable$\mid \mathrm{fix}X.P$ recursion

进程 P 中的自由名字、受囿名字、自由变量和受囿变量分别记为 $fn(P)$、$bn(P)$、$fv(P)$ 和 $bv(P)$，如果 $bv(P)=\varnothing\wedge fv(P)=\varnothing$，则称 P 为闭进程，闭进程集合记为 $\mathcal{P}$，如果两个进程只是受囿名字和受囿变量不同，称这两个进程 α-等价，α-等价的进程相互之间语义完全一致，$P[m/n]$ 表示用 m 替换 P 中的自由名字 n. 基于模型检测算法的可判断性，本文只考虑有限控制界程[9]的模型检测. 下面的结构同余关系($\equiv$)和归约关系($\rightarrow$)构成了移动界程的语义.

移动界程的结构同余关系($\equiv$)定义[1]如下：

StrRefl	$P\equiv P$
StrSymm	$P\equiv Q\Rightarrow Q\equiv P$
StrParZero	$P\mid 0\equiv P$
StrTrans	$P\equiv Q,Q\equiv R\Rightarrow P\equiv R$
StrparComm	$P\mid Q\equiv Q\mid P$
StrParAss	$P\mid(Q\mid R)\equiv(P\mid Q)\mid R$
StrResZero	$(\nu n)0\equiv 0$
StrResPar	$(\nu n)(P\mid Q)\equiv P\mid(\nu n)Q$ if $n\notin fn(P)$
StrResRes	$(\nu n)(\nu m)P\equiv(\nu m)(\nu n)P$
StrResAmb	$(\nu n)(m[P])\equiv m[(\nu n)P]$ if $n\neq m$
StrFixSelf	$\mathrm{fix}X.X\equiv 0$
StrRec	$\mathrm{fix}X.P\equiv P[\mathrm{fix}X.P/X]$

移动界程的归约关系($\rightarrow$)定义[1]如下：

Red In	$n[in\ m.P\mid Q]\mid m[R]\rightarrow m[n[P\mid Q]\mid R]$
Red Out	$m[n[out\ m.P\mid Q]\mid R]\rightarrow n[P\mid Q]\mid m[R]$
Red Open	$open\ n.P\mid n[Q]\rightarrow P\mid Q$
Red I/O	$(n).P\mid\langle q\rangle\rightarrow P[q/n]$
Red Res	$P\rightarrow Q\Rightarrow(\nu n)P\rightarrow(\nu n)Q$
Red Par	$P\rightarrow Q\Rightarrow P\mid R\rightarrow Q\mid R$
Red Amb	$P\rightarrow Q\Rightarrow n[P]\rightarrow n[Q]$
Red$\equiv$	$P\equiv P',\ p'\rightarrow Q',\ Q'\equiv Q\Rightarrow P\rightarrow Q$

3　基于 μ-演算的一阶谓词界程逻辑

本文采用的界程逻辑是由文献[6]提出的基于 μ-演算的一阶谓词界程逻辑(以下简称界程逻辑). 记 $\mathcal{V}$ 是名字变量可数无穷集，$x,y,\cdots\in\mathcal{V}$，$\mathcal{V}\cap\mathcal{N}=\varnothing$，$\mathcal{X}$ 是谓词变量可数无限集，$X,Y,\cdots\in\mathcal{X}$. 界程逻辑语法的 BNF 定义如下[6]，满足 $n\in\mathcal{V}\cup\mathcal{N}$：

$A,B::=$ propositions

$\top\mid\bot\mid 0\mid A\wedge B\mid A\vee B\mid A\mid B\mid A\triangleright B\mid$
$n[A]\mid A@n\mid n\circledR A\mid A\oslash n\mid\boxdot A\mid\diamondsuit A\mid$
$\forall x.A\mid\exists x.A\mid\curlywedge x.A\mid F(\bar{n})$

$F::=$ predicates

$X\mid\nu X.F\mid\mu X.F\mid(\bar{x})A$

界程逻辑公式有命题和谓词两类. 谓词都有确定的目数，命题抽象和谓词应用时，要求目数一致.

公式 A 中的自由名字变量和自由谓词变量分别用 $fnv(A)$ 和 $fpv(A)$ 表示. 名字计值环境用 ρ 表示，$\rho[u/x]$ 表示将 ρ 在 x 处的值修改为 u. 谓词计值环境用 ξ 表示，$\xi[f/X]$ 表示将 ξ 在 X 处的取值置换为 f，$\xi[\xi']$ 表示将 ξ 在 $\mathcal{X}'$ 上的取值置换为 ξ' 的相应取值.

界程逻辑公式的语义归纳定义[6]如下：

$[\![\top]\!]\xi\rho=\mathcal{P}$

$[\![\bot]\!]\xi\rho=\varnothing$

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

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

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

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

$[\![n[A]]\!]\xi\rho=\{P\mid P\equiv\rho(n)[Q]$ for some $Q\in[\![A]\!]\xi\rho\}$

$[\![A@n]\!]\xi\rho=\{P\mid\rho(n)[P]\in[\![A]\!]\xi\rho\}$

$[\![n\circledR A]\!]\xi\rho=\{P\mid P\equiv(\nu\rho(n))Q$ for some $Q\in[\![A]\!]\xi\rho\}$

$[\![A\oslash n]\!]\xi\rho=\{P\mid(\nu\rho(n))P\in[\![A]\!]\xi\rho\}$

$[\![\boxdot A]\!]\xi\rho=\{P\mid\forall Q,P\to Q\Rightarrow Q\in[\![A]\!]\xi\rho\}$

$[\![\diamondsuit A]\!]\xi\rho=\{P\mid\exists Q,P\to Q\wedge Q\in[\![A]\!]\xi\rho\}$

$[\![\forall x.A]\!]\xi\rho=\bigcap_n[\![A]\!]\xi\rho[n/x]$

$[\![\exists x.A]\!]\xi\rho=\bigcup_n[\![A]\!]\xi\rho[n/x]$

$[\![\mathit{И}x.A]\!]\xi\rho=\bigcup_{n\notin fn(A)}\{P\mid P\in[\![A]\!]\xi\rho[n/x]\text{ and }n\notin fn(P)\}$

$[\![F(\bar n)]\!]\xi\rho=[\![F]\!]\xi\rho(\bar n)$

$[\![X]\!]\xi\rho=\xi(X)$

$[\![\nu X.F]\!]\xi\rho=\sqcup\{f\mid f\sqsubseteq[\![F]\!]\xi[f/X]\rho\}$

$[\![\mu X.F]\!]\xi\rho=\sqcap\{f\mid f\sqsupseteq[\![F]\!]\xi[f/X]\rho\}$

$[\![(\bar x)A]\!]\xi\rho=\backslash\bar n.[\![A]\!]\xi\rho[\bar n/\bar x]$

基于模型检测算法的可判断性，本文只考虑不含并发伴随算子“▷”、受囿谓词变量闭且名字闭的公式在有限控制界程上的模型检测.

4 嵌套谓词等式系及模型检测算法[10]

嵌套谓词等式系与界程逻辑公式从语义的角度看是等价的，嵌套谓词等式系是界程逻辑公式更一般意义的表示形式[10].

4.1 嵌套谓词等式系的语法

包含不动点类型的谓词等式形如 $X(\bar x)=_\sigma\phi$，等式中 X 为谓词变量，$\bar x$ 为互不相同的一列名字变量且 $fnv(\phi)=\{\bar x\}$，$\sigma=\{\mu,\nu\}$，ϕ 为命题，ϕ 中没有不动点算子.

嵌套谓词等式系定义为

$$\mathcal{E}::=\varepsilon\mid(X(\bar x)=_\sigma\phi)\mathcal{E}.$$

嵌套谓词等式系中 ε 代表空序列. $\mathcal{E}$ 中等式左边出现的变量为左谓词变量，且互不相同，左谓词变量集合记为 $lhs(\mathcal{E})$，$lhs(\mathcal{E})$ 即 $\mathcal{E}$ 的受囿谓词变量集；各等式右边出现的谓词变量为右谓词变量，右谓词变量集合记为 $rhs(\mathcal{E})$，$rhs(\mathcal{E})-lhs(\mathcal{E})$ 为 $\mathcal{E}$ 的自由谓词变量集. 对于等式系 $\mathcal{E},\mathcal{E}'$，若 $lhs(\mathcal{E})\cap lhs(\mathcal{E}')=\varnothing$，用 $\mathcal{E}::\mathcal{E}'$ 表示将两个等式系连接在一起而得到的新等式系.

4.2 嵌套谓词等式系的语义

设 $\mathcal{E}$ 是嵌套谓词等式系，ξ 是环境，$fnv(\phi)=\{\bar x\}$，$\mathcal{E}$ 在 ξ 上的语义(解) $\|\mathcal{E}\|\xi$ 定义[10]如下：

若 $\mathcal{E}\equiv\varepsilon$，$\|\mathcal{E}\|\xi=\xi$；

若 $\mathcal{E}\equiv(X(\bar x)=_\nu\phi)\mathcal{E}'$，$\|\mathcal{E}\|\xi=\sqcup\{f\mid f\sqsubseteq[\![(\bar x)\phi]\!]\|\mathcal{E}'\|\xi[f/X]\}$；

若 $\mathcal{E}\equiv(X(\bar x)=_\mu\phi)\mathcal{E}'$，$\|\mathcal{E}\|\xi=\sqcap\{f\mid f\sqsupseteq[\![(\bar x)\phi]\!]\|\mathcal{E}'\|\xi[f/X]\}$.

4.3 界程逻辑公式转换为嵌套谓词等式系

界程逻辑公式 $\sigma X.A$ 中，谓词变量 X 的不动点类型记为 $X.\sigma$，如果 $X'\in fpv(A)$，则称 X 依赖 X'，记为 $\langle X,X'\rangle$. 谓词变量 X 的交错嵌套深度 $ad(X)$ 定义如下：

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

根据定义，若 $fpv(A)=\{X\}$，则 $ad(X)=1$.

令：$p\in\{\top,\bot,0\}$，$\circledast\in\{\wedge,\vee,\mid\}$，$\ominus\in\{\boxdot,\diamondsuit,\forall x,\exists x,\mathit{И}x\}$，$\circledcirc\in\{n[\,],@n,n\textcircled{R},\oslash n\}$，则 $\circledcirc(B)\in\{n[B],B@n,n\textcircled{R}B,B\oslash n\}$.

算法 1[10]. 命题转换为嵌套谓词等式系(详见附录 1).

4.4 模型检测算法及其复杂度

算法 2[10]. 有限控制移动界程上的基于 μ-演算的一阶谓词界程逻辑局部模型检测算法(详见附录 2).

算法 2 的核心思想是依据 Tarski 不动点定理，采用目标驱动策略，尽量减少重复计算量，以提高算法性能. 算法 2 中计算结点的数据结构为 $(P,X(\bar x),\phi',\sigma,id),val)$，其中，$P$ 为进程，$X(\bar x)=_\sigma\phi'$ 为等式，id 为等式所处的交错嵌套深度，val 为结点值，当 $\sigma=\nu$，计算初值 $val=$true，当 $\sigma=\mu$，计算初值 $val=$false.

算法 2 运行时，各结点均从初值开始计算，最先计算不动点交错嵌套层次最深($id=d$)的结点，当 $id=d$ 层的所有结点的值到达当前不动点值时，开始计算 $id=d-1$ 层的结点. $id=d-1$ 层只要有结点值发生改变，$id=d$ 层的所有结点均需从初值重新计算. 当 $id=d-1$ 层结点值到达当前不动点状态后，计算 $id=d-2$ 层的结点. $id=d-2$ 层只要有结点值发生改变，$id=d$ 和 $id=d-1$ 层的所有结点均需从初值重新计算. 整个计算过程中，只要有结点值发生改变，那么比该结点交错嵌套层次深的所有结点均需从初值重新计算，直至最外层($id=1$)的所有结点到达不动点，计算停止.

公式 A 中包含的所有算子的数量记为 $op(A)$，$op(A)$ 定义如下：

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

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

$op(Y(\bar n))=0$，$op(((\bar x)A')(\bar n))=op(A')$，

$op((\sigma X.((\bar x)A'))(\bar n))=1+op(A')$.

进程 P 在运行过程中子界程最大并行度 $par(P)$ 定义为

(1) $P\equiv 0: par(P)=0$

(2) $P\equiv M.P': par(P)=\max(1, par(P'))$

(3) $P\equiv a_1[P_1]|a_2[P_2]|\cdots|a_k[P_k]: par(P)=\max(k, par(P_1), par(P_2), \cdots, par(P_k))$

(4) $P\rightarrow P': par(P)=\max(par(P), par(P'))$

记公式 A 中含算子‘®’的数量为 $op_{®}(A)$，含量词‘∀,∃’的数量为 $op_{\forall,\exists}(A)$，含算子‘|’的数量为 $op_{|}(A)$，含算子‘⊡,◈’的数量为 $op_{\boxdot,\diamond}(A)$，含算子‘∧,∨,|’的数量为 $op_{\wedge,\vee,|}(A)$；$bn(P)$ 的规模记为 $|bn(P)|$；进程 P 和公式 A 中自由名字集规模和名字集规模分别记为 $|fn(P,A)|$ 和 $|N(P,A)|$；记 $V_{\mathrm{LTS}}(P)$ 为进程 P 对应的标号转移系统(LTS)的结点规模. 若算法 2 中 $InstallSet()$ 函数产生的计算结点集规模为 M，则有[10]

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

从文献[10]可知，如果公式的交错嵌套深度为 d，利用算法 2 进行模型检测的时间复杂度为 $O(((M+d)/d)^d)$，空间复杂度为 $O(M)$.

5　计算过程规律提取及算法改进

若 J,K 为计算结点集合，且满足(1) $|J|=|K|$；(2)集合内各结点的 val 取值唯一，即 $(P,X(\bar{x},\phi,\sigma,i),\mathrm{false})$ 和 $(P,X(\bar{x},\phi,\sigma,i),\mathrm{true})$ 不可能同时出现在同一集合中.

定义 1.　在 J,K 上定义⋐算子：若 $J\Subset K$，则

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

$\forall(P,X(\bar{x},\phi,\sigma,i),\mathrm{true})\in J\Rightarrow(P,X(\bar{x},\phi,\sigma,i),\mathrm{true})\in K$.

显然，⋐关系是自反、反对称和传递的，是计算结点集上的一个偏序关系.

设交错嵌套深度为 4 且最内层不动点算子为最大不动点的嵌套谓词等式系：

$$\mathcal{E}_1::\mathcal{E}_2::\mathcal{E}_3::\mathcal{E}_4 \qquad (1)$$

$\mathcal{E}_4$ 表示最内层等式块，$\mathcal{E}_1$，$\mathcal{E}_3$ 对应的不动点类型 $\sigma_1=\sigma_3=\mu$，$\mathcal{E}_2$，$\mathcal{E}_4$ 对应的不动点类型 $\sigma_2=\sigma_4=\nu$，$\mathcal{E}_1$，$\mathcal{E}_2$，$\mathcal{E}_3$，$\mathcal{E}_4$ 等式块对应的结点集合记为 R_1，R_2，R_3，R_4，其中 $R_i=R_{i0}\cup R_{i1}\cup R_{i2}$ $(1\leq i\leq 4)$ (R_{i0}，R_{i1}，R_{i2} 在算法 2 中定义和赋值). 在界程 P 上，依据算法 2 计算 R_1 时，有计算序列 $R_1^0, R_1^1, R_1^2, \cdots, R_1^{\omega}$，其中 R_1^0 为初始状态，由算法 2 中函数 $InstallSet()$ 赋值，R_1^{ω} 为到达不动点状态.

当 R_1 取值为 $R_1^{j_1}$ 时，计算 R_2 的计算序列记为 $R_2^{j_1 0}, R_2^{j_1 1}, R_2^{j_1 2}, \cdots, R_2^{j_1\omega}$，其中，$R_2^{j_1 0}$ 为初始状态，$R_2^{j_1\omega}$ 为当前不动点状态.

当 R_1 取值为 $R_1^{j_1}$，R_2 取值为 $R_2^{j_1 j_2}$ 时，计算 R_3 的计算序列为 $R_3^{j_1 j_2 0}, R_3^{j_1 j_2 1}, R_3^{j_1 j_2 2}, \cdots, R_3^{j_1 j_2\omega}$.

当 R_1 取值为 $R_1^{j_1}$，R_2 取值为 $R_2^{j_1 j_2}$，R_3 取值为 $R_3^{j_1 j_2 j_3}$ 时，计算 R_4 的计算序列为 $R_4^{j_1 j_2 j_3 0}, R_4^{j_1 j_2 j_3 1}, R_4^{j_1 j_2 j_3 2}, \cdots, R_4^{j_1 j_2 j_3\omega}$.

依据算法 2 计算，各层结点的计算顺序为

$R_4^{0001}, R_4^{0002}, \cdots, R_4^{000\omega}, R_3^{001}, R_4^{0011}, R_4^{0012}, \cdots, R_4^{001\omega}, R_3^{002}, \cdots, R_3^{00\omega}, R_2^{01}, R_4^{0101}, R_4^{0102}, \cdots, R_4^{010\omega}, R_3^{011}, R_4^{0111}, \cdots, R_4^{011\omega}, R_3^{012}, \cdots, R_3^{01\omega}, R_2^{02}, \cdots, R_2^{0\omega}, R_1^1, R_4^{1001}, R_4^{1002}, \cdots, R_4^{100\omega}, R_3^{101}, R_4^{1011}, \cdots, R_4^{101\omega}, R_3^{102}, \cdots, R_3^{10\omega}, R_2^{11}, \cdots, R_2^{1\omega}, R_1^2, \cdots, R_1^{\omega}$.

依据算法 2 对等式系(1)对应结点进行计算，记 $R_i^{j_1 j_2\cdots j_{i-1}k}=R_{i0}^{j_1 j_2\cdots j_{i-1}k}\cup R_{i1}^{j_1 j_2\cdots j_{i-1}k}\cup R_{i2}^{j_1 j_2\cdots j_{i-1}k}$ $(1\leq i\leq 4)$. 由于 $\sigma_1=\sigma_3=\mu$，R_1^0，$R_3^{j_1 j_2 0}$ 中各结点 val 初值为 false；$\sigma_2=\sigma_4=\nu$，$R_2^{j_1 0}$，$R_4^{j_1 j_2 j_3 0}$ 中各结点 val 初值为 true.

设等式块 $\mathcal{E}_1::\mathcal{E}_2::\mathcal{E}_3::\mathcal{E}_4$ 对应结点的计算函数分别为 f_1, f_2, f_3, f_4，则有

$$R_1^{j_1+1}=f_1(R_1^{j_1}, R_2^{j_1\omega}, R_3^{j_1\omega\omega}, R_4^{j_1\omega\omega\omega}),$$

$$R_2^{j_1(j_2+1)}=f_2(R_1^{j_1}, R_2^{j_1 j_2}, R_3^{j_1 j_2\omega}, R_4^{j_1 j_2\omega\omega}),$$

$$R_3^{j_1 j_2(j_3+1)}=f_3(R_1^{j_1}, R_2^{j_1 j_2}, R_3^{j_1 j_2 j_3}, R_4^{j_1 j_2 j_3\omega}),$$

$$R_4^{j_1 j_2 j_3(j_4+1)}=f_4(R_1^{j_1}, R_2^{j_1 j_2}, R_3^{j_1 j_2 j_3}, R_4^{j_1 j_2 j_3 j_4}).$$

具体 f_i 的定义如定义 2.

定义 2.　f_i 的定义.

```
f_i(R_1, R_2, ..., R_d)
{
  while(R_i0 ≠ ∅)
  {
    (P, X(x̄, φ, σ, i), val) = readrecord(R_i0);
        //从 R_i0 中提取结点
    del(R_i0, (P, X(x̄, φ, σ, i), val))
    val' = val; //保留原来的值
    (P, X(x̄, φ, σ, i), val).val =
    ValueRecord((P, X(x̄, φ, σ, i), val));
        //计算并更新结点的 val 值
    if(val' = val) add(R_i2, (P, X(x̄, φ, σ, i), val));
```

```
        //结点值不变
    else{add(R_{i1},(P,X(x̄,ϕ,σ,i),val));
        for (j=1;j≤i;j++)
        {r=R_{j2}∩DAS((P,X(x̄,ϕ,σ,i),val));
    //计算 R_{j2} 中属于结点(P,X(x̄,ϕ,σ,i),val)的
    //直接作用集的结点
        R_{j2}=R_{j2}-r; R_{j0}=R_{j0}+r;
        }
        break; //退出 while 循环
        }
  }
  return (R_i);
}
```

从 f_i 定义可知，f_i 为单调函数.

计算 R_4 时，在计算序列 $R_4^{j_1j_2j_3{}^0}, R_4^{j_1j_2j_3{}^1}, R_4^{j_1j_2j_3{}^2}, \cdots, R_4^{j_1j_2j_3{}^\omega}$ 中，由于 $\sigma_4=\nu$，R_4 中各结点的 val 值或保持初值 true 不变，或为 false，且一旦结点 val 值改变为 false 后，在 $R_4^{j_1j_2j_3{}^0}, R_4^{j_1j_2j_3{}^1}, R_4^{j_1j_2j_3{}^2}, \cdots, R_4^{j_1j_2j_3{}^\omega}$ 序列中不再发生改变，各中间结果间满足 $R_4^{j_1j_2j_3{}^\omega}\Subset\cdots\Subset R_4^{j_1j_2j_3{}^2}\Subset R_4^{j_1j_2j_3{}^1}\Subset R_4^{j_1j_2j_3{}^0}$.

计算 R_3 时，$\sigma_3=\mu$，各中间结果间满足 $R_3^{j_1j_2{}^0}\Subset R_3^{j_1j_2{}^1}\Subset R_3^{j_1j_2{}^2}\Subset\cdots\Subset R_3^{j_1j_2{}^\omega}$.

计算 R_2 时，$\sigma_2=\nu$，各中间结果间满足 $R_2^{j_1{}^\omega}\Subset\cdots\Subset R_2^{j_1{}^2}\Subset R_2^{j_1{}^1}\Subset R_2^{j_1{}^0}$.

计算 R_1 时，$\sigma_1=\mu$，各中间结果间满足 $R_1^0\Subset R_1^1\Subset R_1^2\Subset\cdots\Subset R_1^\omega$.

引理 1. 用算法 2 对式(1)进行迭代计算，则有

(1) $R_4^{j_1(j_2+1)j_3{}^\omega}\Subset R_4^{j_1j_2j_3{}^\omega}$；

(2) $R_3^{j_1j_2{}^\omega}\Subset R_3^{(j_1+1)j_2{}^\omega}$.

证明参见附录 3.

当 R_1 取值为 $R_1^{j_1}$，R_2 取值为 $R_2^{j_1(j_2+1)}$，R_3 取值为 $R_3^{j_1(j_2+1)j_3}$ 时，计算 R_4 的计算序列为 $R_4^{j_1(j_2+1)j_3{}^0}, R_4^{j_1(j_2+1)j_3{}^1}, R_4^{j_1(j_2+1)j_3{}^2}, \cdots, R_4^{j_1(j_2+1)j_3{}^\omega}$. 由引理 1 可知 $R_4^{j_1(j_2+1)j_3{}^\omega}\Subset R_4^{j_1j_2j_3{}^\omega}\Subset\cdots\Subset R_4^{j_1{}^0j_3{}^\omega}\Subset R_4^{j_1{}^0j_3{}^0}=\cdots=R_4^{j_1j_2j_3{}^0}=R_4^{j_1(j_2+1)j_3{}^0}$. 因 $\sigma_4=\nu$，计算 $R_4^{j_1(j_2+1)j_3{}^\omega}$ 时，可以不必从 $R_4^{j_1(j_2+1)j_3{}^0}$ 开始计算，而是从序列 $R_4^{j_1j_2j_3{}^\omega}, R_4^{j_1(j_2-1)j_3{}^\omega}, \cdots, R_4^{j_1{}^0j_3{}^\omega}$ 中任选一个值开始计算，都能保证计算结果的正确性. 通过选取序列 $R_4^{j_1j_2j_3{}^\omega}, R_4^{j_1(j_2-1)j_3{}^\omega}, \cdots, R_4^{j_1{}^0j_3{}^\omega}$ 中最近计算的值作为计算 $R_4^{j_1(j_2+1)j_3{}^\omega}$ 的计算初值，可以有效的利用之前的计算结果，减少迭代计算的次数，提高模型检测算法的性能.

因 $R_3^{(j_1+1)j_2{}^0}=R_3^{j_1j_2{}^0}=\cdots=R_3^{0j_2{}^0}\Subset R_3^{0j_2{}^\omega}\Subset\cdots\Subset R_3^{j_1j_2{}^\omega}\Subset R_3^{(j_1+1)j_2{}^\omega}$，$\sigma_3=\mu$，计算 $R_3^{(j_1+1)j_2{}^\omega}$ 时，可以不从 $R_3^{j_1(j_2+1)0}$ 开始计算，而是从序列 $R_3^{0j_2{}^\omega}, \cdots, R_3^{(j_1-1)j_2{}^\omega}, R_3^{j_1j_2{}^\omega}$ 中任选一个值开始计算，同样可以保证计算的正确性，减少迭代计算次数，提高模型检测算法的性能.

根据以上分析，对谓词等式系(1)对应结点进行计算时，可在算法 2 的基础上，对计算过程进行优化，改进后的算法为算法 3.

在算法 3 中，$R_i=R_{i0}\cup R_{i1}\cup R_{i2}$，$a$，$b$ 数组分别存放 R_3，R_4 迭代计算过程中的迭代序号和最新计算数据. $a[j]$ 保存序列 $R_3^{0j0}, R_3^{0j\omega}, R_3^{1j\omega}, \cdots, R_3^{\omega j\omega}$ 中最新计算数据，$b[k]$ 保存序列 $R_4^{j_10k0}, R_4^{j_10k\omega}, R_4^{j_11k\omega}, \cdots, R_4^{j_1\omega k\omega}$ 中最新计算数据，当 j_1 改变时，b 数组重新初始化.

算法 3.

函数 *formulaChangToequation*() 见算法 1，*ModelCheck*(P, F)，*InstallSet*((P, $X(\bar{n},\phi,\sigma)$)) 和 *ValueRecord*((P, $X(\bar{x},\phi,\sigma)$)) 见算法 2. *ValueSet*() 函数重定义如下：

```
ValueSet()//嵌套谓词等式系的计算
{
R_1=R_1^0;
a[0:|R_2^{00}|]={(R_3^{000},0),⋯,(R_3^{000},0)};//初始化 a 数组
for(i=0;i<=|R_1^0|;i++){
  R_2=R_2^{00};
  b[0:|R_3^{000}|]={(R_4^{0000},0),⋯,(R_4^{0000},0)};//初始化b数组
  for(j=0;j<=|R_2^{00}|;j++){
    (R_3,k)=a[j];//取序列R_3^{0j0},R_3^{0jω},R_3^{1jω},⋯,R_3^{(i-1)jω}
                 //中最新计算的值作为 R_3^{ijω} 的初值
for(;k<=|R_3^{000}|;k++){
    (R_4,l)=b[k];//取序列R_4^{i0k0},R_4^{i0kω},R_4^{i1kω},⋯,R_4^{i(j-1)kω}
                 //中最新计算的值作为 R_4^{ijkω} 的初值
    for(;l<=|R_4^{0000}|;l++){
      r_4=f_4(R_1,R_2,R_3,R_4); //计算 R_4^{ijk(l+1)}
      if(r_4==R_4) //到达当前不动点
        b[k]=(R_4,l), break;
      else R_4=r_4;
    }
    r_3=f_3(R_1,R_2,R_3,R_4); //计算 R_3^{ij(k+1)}
    if(r_3==R_3) //到达当前不动点
      a[j]=(R_3,k), break;
    else R_3=r_3;
```

```
    }
    r2=f2(R1,R2,R3,R4); //计算 R2^{i(j+1)}
    if(r2==R2) break; //到达当前不动点
    else R2=r2;
    }
  r1=f1(R1,R2,R3,R4); //计算 R1^{i+1}
  if(r1==R1) break; //到达当前不动点
  else R1=r1;
}}
```

例 1.　根据算法 3，嵌套谓词等式系 $\mathcal{E}_1::\mathcal{E}_2::\mathcal{E}_3::\mathcal{E}_4$，各层计算结点的计算序列为

$(R_1^0=S_{10},R_2^{00}=S_{20},R_3^{000}=S_{30},R_4^{0000}=S_{40}),R_4^{0001},R_4^{0002},\cdots,R_4^{000\omega},R_3^{001},(R_4^{0010}=S_{40}),R_4^{0011},R_4^{0012},\cdots,R_4^{001\omega},R_3^{002},(R_4^{0020}=S_{40}),R_4^{0021},R_4^{0022},\cdots,R_4^{002\omega},R_3^{003},\cdots,R_3^{00\omega},R_2^{01},(R_3^{010}=S_{30},(R_4^{0100},l)=(R_4^{000\omega},\omega)),R_4^{010(l+1)},R_4^{010(l+2)},\cdots,R_4^{010\omega},R_3^{011},((R_4^{0110},l)=(R_4^{001\omega},\omega)),R_4^{011(l+1)},R_4^{011(l+2)},\cdots,R_4^{011\omega},R_3^{012},((R_4^{0120},l)=(R_4^{002\omega},\omega)),R_4^{012(l+1)}\cdots,R_4^{012\omega},R_3^{013},\cdots,R_3^{01\omega},R_2^{02},\cdots,R_2^{0\omega},R_1^1,(R_2^{10}=S_{20},(R_3^{100},k)=(R_3^{00\omega},\omega),R_4^{10k0}=S_{40}),R_4^{10k1},R_4^{10k2},\cdots,R_4^{10k\omega},R_3^{10(k+1)},(R_4^{10(k+1)0}=S_{40}),R_4^{10(k+1)1},R_4^{10(k+1)2},\cdots,R_4^{10(k+1)\omega},R_3^{10(k+2)},\cdots,R_3^{10\omega},R_2^{11},((R_3^{110},k)=(R_3^{01\omega},\omega),(R_4^{11k0},l)=(R_4^{10k\omega},\omega)),R_4^{11k(l+1)},R_4^{11k(l+2)},\cdots,R_4^{11k\omega},R_3^{11(k+1)},((R_4^{11(k+1)0},l)=(R_4^{10(k+1)\omega},\omega)),R_4^{11(k+1)(l+1)},R_4^{11(k+1)(l+2)},\cdots,R_4^{11(k+1)\omega},R_3^{11(k+2)},\cdots,R_3^{11\omega},\cdots,R_2^{12},\cdots,R_2^{1\omega},R_1^2,\cdots,R_1^{\omega}$

在计算序列中，赋值语句只是读取已有数据，S_{i0} 保存 $\mathcal{E}_i$ 对应的计算结点，其中结点值均取初值.

定理 1.　用算法 3 计算，如果 $ModelCheck(P,F)$ 最后计算结果 $(P,X(\bar{x},\phi,\sigma,id),val).val=$true，则 $P\models F$，否则 $P\not\models F$.

证明参见附录 4.

定理 2.　利用算法 3 对嵌套谓词等式系(1)进行计算，时间复杂度为 $O((M+d)^2)$.

证明参见附录 5.

定理 3.　算法 3 的空间复杂度为 $O(M)$.

证明参见附录 6.

6　算法应用范围扩展

6.1　$\sigma_4=\mu$ 的情形

设交错嵌套深度为 4 且最内层不动点算子为最小不动点的嵌套谓词等式系：

$$\mathcal{E}_1::\mathcal{E}_2::\mathcal{E}_3::\mathcal{E}_4 \tag{2}$$

$\mathcal{E}_4$ 为最内层等式块，不动点类型分别为 $\sigma_1=\sigma_3=\nu$，$\sigma_2=\sigma_4=\mu$，$\mathcal{E}_1,\mathcal{E}_2,\mathcal{E}_3,\mathcal{E}_4$ 等式块对应的结点集为 R_1,R_2,R_3,R_4.

由于 $\sigma_1=\sigma_3=\nu$，$R_1^0,R_3^{j_1j_20}$ 中各结点 val 初值为 true；$\sigma_2=\sigma_4=\mu$，$R_2^{j_10},R_4^{j_1j_2j_30}$ 中各结点 val 初值为 false.

等式块 $\mathcal{E}_1::\mathcal{E}_2::\mathcal{E}_3::\mathcal{E}_4$ 对应结点的计算函数分别为 f_1,f_2,f_3,f_4，则有

$R_1^{j_1+1}=f_1(R_1^{j_1},R_2^{j_1\omega},R_3^{j_1\omega\omega},R_4^{j_1\omega\omega\omega})$，

$R_2^{j_1(j_2+1)}=f_2(R_1^{j_1},R_2^{j_1j_2},R_3^{j_1j_2\omega},R_4^{j_1j_2\omega\omega})$，

$R_3^{j_1j_2(j_3+1)}=f_3(R_1^{j_1},R_2^{j_1j_2},R_3^{j_1j_2j_3},R_4^{j_1j_2j_3\omega})$，

$R_4^{j_1j_2j_3(j_4+1)}=f_4(R_1^{j_1},R_2^{j_1j_2},R_3^{j_1j_2j_3},R_4^{j_1j_2j_3j_4})$.

具体 f_i 的定义见定义 2.

当 R_1 取值为 $R_1^{j_1}$，R_2 取值为 $R_2^{j_1j_2}$，R_3 取值为 $R_3^{j_1j_2j_3}$ 时计算 R_4，有计算序列 $R_4^{j_1j_2j_30},R_4^{j_1j_2j_31},R_4^{j_1j_2j_32},\cdots,R_4^{j_1j_2j_3\omega}$. 由于 $\sigma_4=\mu$，在计算过程中，R_4 中各结点的 val 值或保持初值 false 不变，或为 true，且一旦结点 val 值改变为 true 之后，在 $R_4^{j_1j_2j_30},R_4^{j_1j_2j_31},R_4^{j_1j_2j_32},\cdots,R_4^{j_1j_2j_3\omega}$ 序列中不再发生改变，各中间结果间满足 $R_4^{j_1j_2j_30}\Subset R_4^{j_1j_2j_31}\Subset R_4^{j_1j_2j_32}\Subset\cdots\Subset R_4^{j_1j_2j_3\omega}$.

$\sigma_3=\nu$，各中间结果间满足 $R_3^{j_1j_2\omega}\Subset\cdots\Subset R_3^{j_1j_22}\Subset R_3^{j_1j_21}\Subset R_3^{j_1j_20}$.

$\sigma_2=\mu$，各中间结果间满足 $R_2^{j_10}\Subset R_2^{j_11}\Subset R_2^{j_12}\Subset\cdots\Subset R_2^{j_1\omega}$.

$\sigma_1=\nu$，各中间结果间满足 $R_1^{\omega}\Subset\cdots\Subset R_1^2\Subset R_1^1\Subset R_1^0$.

引理 2.　用算法 2 对式(2)进行迭代计算，则有

(1) $R_4^{j_1j_2j_3\omega}\Subset R_4^{j_1(j_2+1)j_3\omega}$；

(2) $R_3^{(j_1+1)j_2\omega}\Subset R_3^{j_1j_2\omega}$.

证明与引理 1 的证明类似，证明略.

根据引理 2，可用算法 3 来计算嵌套谓词等式系(2)对应结点的不动点值.

6.2　嵌套深度为 d 的情形

算法 3 讨论的是交替深度 $d=4$ 的算法. 下面将算法 3 推广到 $d>4$ 的情形. 对于嵌套谓词等式系(3)：

$$\mathcal{E}_1::\mathcal{E}_2::\cdots::\mathcal{E}_d \tag{3}$$

令 $n=\lfloor(d+3)/4\rfloor$，将 $\mathcal{E}_1,\mathcal{E}_2,\cdots,\mathcal{E}_d$ 按从左到右的顺序进行分块，4 个一组，如果 $d \bmod 4=0$，则可以分为 $n=d/4$ 块：$A_0=(\mathcal{E}_1,\cdots,\mathcal{E}_4)$，$A_1=(\mathcal{E}_5,\cdots,$

$\mathcal{E}_8),\cdots,A_{n-1}=(\mathcal{E}_{d-3},\cdots,\mathcal{E}_d)$.

如果 $d \bmod 4\neq 0$,则分为 $n=d/4+1$ 块:

$A_0=(\mathcal{E}_1,\cdots,\mathcal{E}_4),A_1=(\mathcal{E}_5,\cdots,\mathcal{E}_8),\cdots,$

$A_n=(\mathcal{E}_{4n+1},\cdots,\mathcal{E}_d)$.

嵌套谓词等式系(3)对应结点模型检测算法如算法 4.

算法 4. 对编号为 x 的等式块对应的计算结点进行计算.

```
ValueSet_x()
//计算块号为 x 的嵌套谓词等式系对应结点
{
R_{x,1}=R^0_{x,1};a_x[0:|R^{00}_{x,2}|]={(R^{000}_{x,3},0),…,(R^{000}_{x,3},0)};
 //初始化 a_x 数组
for(i_x=0;i_x<=|R^0_{x,1}|;i_x++){
 R_{x,2}=R^{00}_{x,2};b_x[0:|R^{000}_{x,3}|]={(R^{0000}_{x,4},0),…,(R^{0000}_{x,4},0)};
 //初始化 b_x 数组
  for(j_x=0;j_x<=|R^{00}_{x,2}|;j_x++){
   (R_{x,3},k_x)=a_x[j_x];//取序列 R^{0j0}_{x,3},R^{0jω}_{x,3},R^{1jω}_{x,3},…,
   //R^{(i-1)jω}_{x,3} 中最近计算的值作为计算 R^{ijω}_{x,3} 的初值
   for(;k_x<=|R^{000}_{x,3}|;k_x++){
    (R_{x,4},l_x)=b_x[k_x];//取序列 R^{i0k0}_{x,4},R^{i0kω}_{x,4},R^{i1kω}_{x,4},…,
    //R^{i(j-1)kω}_{x,4} 中最近计算的值作为计算 R^{ijkω}_{x,4} 的初值
    for(;l_x<=|R^{0000}_{x,4}|;l_x++){
     ValueSet_{x+1}();
     r_{x,4}=f_{x,4}(R_1,R_2,…,R_d);//计算 R^{ijk(l+1)}_{x,4}
     if(r_{x,4}==R_{x,4}) //R_{x,4} 到达当前不动点
      b_x[k_x]=(R_{x,4},l_x),break;
     R_{x,4}=r_{x,4};
    }
    r_{x,3}=f_{x,3}(R_1,R_2,…,R_d); //计算 R^{ij(k+1)}_{x,3}
    if(r_{x,3}==R_{x,3}) //R_{x,3} 到达当前不动点
     a_x[j_x]=(R_{x,3},k_x),break;
    R_{x,3}=r_{x,3};
   }
   r_{x,2}=f_{x,2}(R_1,R_2,…,R_d); //计算 R^{i(j+1)}_{x,2}
   if(r_{x,2}==R_{x,2})break; //R_{x,2} 到达当前不动点
   R_{x,2}=r_{x,2};
  }
 r_{x,1}=f_{x,1}(R_1,R_2,…,R_d); //计算 R^{i+1}_{x,1}
 if(r_{x,1}==R_{x,1})break; //R_{x,1} 到达当前不动点
 R_{x,1}=r_{x,1};
}}
```

在算法 4 中,$ValueSet_x()$ 计算编号为 x 的等式系模块对应的结点,$ValueSet_x()$ 嵌套调用了 $ValueSet_{x+1}()$,计算交错嵌套层次更深的等式系对应结点的当前不动点值. $f_{x,i}$ 对应 f_{4x+i},$R_{x,i}$ 对应 R_{4x+i},计算编号为 n 的等式系模块时,循环次数为模块中不动点算子交错嵌套深度. 用算法 4 计算时,最先调用 $ValueSet_1()$,$ValueSet_1()$ 调用 $ValueSet_2()$,$ValueSet_2()$ 调用 $ValueSet_3()$,…,$ValueSet_{n-1}()$ 调用 $ValueSet_n()$. 当编号为 n 的等式块对应的结点到达当前不动点时,计算编号为 $n-1$ 等式块对应的结点. 当编号为 $n-1$ 等式块对应的结点集中有结点值发生改变,此时继续调用 $ValueSet_n()$,直到编号为 n 的等式块对应的结点再一次到达当前不动点,流程返回 $ValueSet_{n-1}()$. 如此反复,直到编号为 $n-1$ 的等式块对应的结点集到达当前不动点,流程转移到 $ValueSet_{n-2}()$ 进行计算,… . 这样一直计算下去,直到编号为 1 的等式块对应的结点集到达当前不动点,整个计算流程结束.

定理 4. 用算法 4 对嵌套谓词等式系(3)对应的结点集进行计算,算法时间复杂度为 $O((M+d)^{\lfloor d/2\rfloor+1})$.

证明参见附录 7.

定理 5. 用算法 4 对嵌套谓词等式系(3)进行计算,算法空间复杂度为 $O(M)$.

证明参见附录 8.

例 2. 对嵌套谓词等式系(3)对应的结点进行计算,设各等式块对应的计算结点数均为 40,利用本文算法与文献[10]中的算法计算,当不动点算子交错嵌套深度 d 取不同值时,在最坏的情况下(各计算结点 val 值最终均由初值翻转后才到达最终不动点),各函数的计算次数和 $|f|$ 的值见表 1.

表 1 本文算法与文献[10]中算法计算时各函数计算量比较

$m_i=40$	本文算法	文献[10]中算法
$d=1$	41	41
$d=2$	1.7×10^3	1.7×10^3
$d=3$	5.1×10^3	7.1×10^4
$d=4$	8.4×10^3	2.8×10^6
$d=5$	1.5×10^5	1.2×10^8
$d=6$	5.8×10^6	4.9×10^9
$d=7$	1.7×10^7	2.0×10^{11}
$d=8$	2.8×10^7	8.2×10^{12}

7 结束语

本文对基于有限控制移动界程的 μ-演算一阶谓词界程逻辑局部模型检测算法进行了深入的分析和研究,指出文献[10]中模型检测算法在计算过程

中，中间结果间存在两组偏序关系，并据此提出了一个局部模型检测快速算法，算法的时间复杂度与 $d/2+1$ 呈指数关系，空间复杂度与 M 呈线性关系，d 为公式中不动点算子交错嵌套深度，M 为计算结点集规模. 据本文作者所知，这是继文献[5,10]后，目前公开的第三个基于有限控制移动界程的 μ-演算一阶谓词界程逻辑局部模型检测算法. 相比之前出现的同类算法，其性能得到了大大提高. 本文的研究成果在促进界程逻辑模型检测技术的推广应用方面以及在推进界程逻辑模型检测理论的深入研究方面均意义重大. 同时，本文的研究方法同样适应于 π-演算和 CCS 演算局部模型检测算法的设计.

致　谢　本文在研究过程中，得到了中国科学院软件研究所林惠民院士、李广元研究员和福建省粒计算重点实验室祝峰教授的大力指导和支持，在此深表感谢！

参　考　文　献

[1] Cardelli L, Gordon A D. Mobile ambients. Theoretical Computer Science, 2000, 240(1): 177-213

[2] Cardelli L, Gordon A D. Anytime, anywhere: Modal logics for mobile ambients//Proceedings of the POPL'2000. Boston, USA, 2000: 365-377

[3] Cardelli L, Gordon A D. Logical properties of name restriction //Proceedings of the 5th International Conference on Typed Lambda, Calculi and Applications. Kraków, Poland, 2001: 46-60

[4] Cardelli L, Ghelli G. A query language based on the ambient logic//Proceedings of the 2001 European Symposium on Programming. Genova, Ital, 2001: 1-22

[5] Lin Hui-Min. Space logic of mobile ambients. Science in China Series E: Technological Sciences, 2004, 34(2): 139-150(in Chinese)
(林惠民. 移动进程的空间逻辑. 中国科学 E 辑, 2004, 34(2): 139-150)

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

[7] Charatonik W, Talbot J M. The decidability of model checking mobile ambients//Proceedings of the 15th Annual Conference of European Association for Computer Science Logic. Pairs, France, 2001: 339-354

[8] Charatonik W, Dal Zilio S, Gordon A D, et al. The complexity of model checking mobile ambients//Proceedings of the 4th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2001). Genova, Italy, 2001: 152-167

[9] Charatonik W, Gordon A, Talbot J M. Finite-control mobile ambients//Proceedings of the ESOP 2002: the 11th European Symposium on Programming. LNCS 2305. Grenoble, France, 2002: 295-313

[10] Jiang Hua, Li Xiang. Model checking for mobile ambients. Journal of Computer Research and Development, 2009, 46(10): 1750-1757(in Chinese)
(江华，李祥. 移动界程模型检测. 计算机研究与发展, 2009, 46(10): 1750-1757)

[11] Kozen D. Results on the propositional μ-calculus//Proceedings of the 9th Colloquium on Automata, Languages and Programming. LNCS 140. Reykjavik, Iceland, 1982: 348-359

[12] de Bakker J W. Mathematical Theory of Program Correctness. NJ, USA: Prentice-Hall, Inc. Upper Saddle River, 1980

[13] Park D. Fixpoint Induction and Proof of Program Semantics. Machine Intelligence 5. Edinburgh: Edinburgh University Press, 1970: 59-78

[14] Stirling C. Modal and temporal logics for processes// Proceedings of the 8th Banff Higher Order Workshop. Banff, Canada, 1996: 149-237

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

[16] Emerson E A, Lei C L. Efficient model checking in fragments of the prepositional mu-calculus//Proceedings of the 1st Annual Symposium on Logic in Computer Science. New Orleans, USA, 1985: 84-96

[17] Andersen H R. Model checking and Boolean graphs// Proceedings of the 4th European Symposium on Programming. LNCS 582. Rennes, France, 1992: 1-19

[18] Cleaveland R, Klein M, Steffen B. Faster model checking for the modal mu-calculus//Proceedings of the 4th International Workshop on Computer Aided Verification. LNCS 663. Montreal, Canada, 1992: 410-422

[19] Cleaveland R. Tableau-based model checking in the propositional mu-calculus. Acta Information, 1990, 27(8): 725-747

[20] Long D, Browne A, Clarke E, et al. An improved algorithm for the valuation of fixpoint expressions//Proceedings of the 6th International Conference on Computer Aided Verification. LNCS 818. Stanford, USA, 1994: 338-350

[21] Jiang Hua. Efficient global model-checking for propositional μ-calculus. Journal of Computer Research and Development, 2010, 47(8): 1424-1433(in Chinese)
(江华. 命题 μ-演算全局模型检测的高效算法设计. 计算机研究与发展, 2010, 47(8): 1424-1433)

[22] Liu X, Ramakrishnan C R, Smolka S A. Fully local and efficient evaluation of alternating fixed points//Proceedings of the 4th International Conference on Tools and Algorithms for the Construction and Analysis of System (ACAS'98). Lisbon, Portugal, 1998: 5-19

[23] Liu Jian, Lin Hui-Min. Consistency between the predicate μ-calculus and modal graphs. Journal of Software, 2003, 14(10): 1672-1680(in Chinese)
(刘剑，林惠民. 谓词 μ-演算和模态图的语义一致性. 软件学报，2003，14(10)：1672-1680)

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

[25] Brodo L. On the expressiveness of the π-calculus and the mobile ambients//Proceedings of the 13th International Conference on Algebraic Methodology and Software Technology. QC, Canada, 2011: 44-59

[26] Aman B, Ciobanu G. Coordinating parallel mobile ambients to solve SAT problem in polynomial number of steps//Proceedings of the 7th International Federated Conference on Distributed Computing Techniques. Stockholm, Sweden, 2012: 122-136

[27] Ali N, Fei Chen, Solis C. Modeling support for mobile ambients in service oriented architecture//Proceedings of the 2012 IEEE 1st International Conference on Mobile Services (MS). Honolulu, USA, 2012: 1-8

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

[29] Bodei C, Brodo L, Bruni R. Open multiparty interaction//Proceedings of the 21st International Workshop on Algebraic Development Techniques. Salamanca, Spain, 2013: 1-23

[30] Aman B, Ciobann G. Expressing mobile ambients in temporal logic of actions. Proceedings of the Romanian Academy Series A: Mathematics, Physics, Technical Sciences, Information Science, 2014, 15(1): 95-104

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

附录 1.

算法 1[10]. 命题转换为嵌套谓词等式系.

$formulaChangToequation(F)$ //$F\rightarrow\mathcal{E}$

{

对公式 F 做 α-变换，要求受囿变量两两不同；

计算公式 F 中各谓词变量的嵌套深度.

$(X(\bar{x}),\mathcal{E},j)=formulatoequation(F,\sigma,1,0)$;

return$(X(\bar{x}),\mathcal{E})$;

}

$formulatoequation(\phi,\sigma,i,id)$

{case ϕ of

p: return$(X_i(\bar{x}_i),\{X_i(\bar{x}_i),\sigma,p,id\},i+1)$

$|\phi_1\circledast\phi_2$: $(Y(\bar{y}),\mathcal{E}_1,i_1)=formulatoequation(\phi_1,\sigma,i+1,id)$,
$(Z(\bar{z}),\mathcal{E}_2,i_2)=formulatoequation(\phi_2,\sigma,i_1,id)$
return$(X_i(\bar{x}_i),\{X_i(\bar{x}_i),\sigma,Y(\bar{y})\circledast Z(\bar{z}),id\}::\mathcal{E}_1::\mathcal{E}_2,i_2)$

$|\circledcirc(\phi')$: $(Y(\bar{y}),\mathcal{E}_1,i_1)=formulatoequation(\phi',\sigma,i+1,id)$
return$(X_i(\bar{x}_i),\{X_i(\bar{x}_i),\sigma,\circledcirc(Y(\bar{y})),id\}::\mathcal{E}_1,i_1)$

$|\ominus\phi'$: $(Y(\bar{y}),\mathcal{E}_1,i_1)=formulatoequation(\phi',\sigma,i+1,id)$
return$(X_i(\bar{x}_i),\{X_i(\bar{x}_i),\sigma,\ominus Y(\bar{y}),id\}::\mathcal{E}_1,i_1)$

$|((\bar{x})\phi')(\bar{e})$: $(Y(\bar{y}),\mathcal{E}_1,i_1)=formulatoequation(\phi',\sigma,i+1,id)$
return$(X_i(\bar{x}_i),\{X_i(\bar{x}_i),\sigma,((\bar{y})(Y(\bar{y})))(\bar{e}),id\}::\mathcal{E}_1,i_1)$

$|Y(\bar{e})$: if $Y=X_j(j<i)$, 记$\{\bar{x}_j\}=fnv(X_j)$
return$(X_i(\bar{x}_i),\{X_i(\bar{x}_i),\sigma,((\bar{x}_j)(X_j(\bar{x}_j)))(\bar{e}),id\},i+1)$
else return$(X_i(\bar{x}_i),\{X_i(\bar{x}_i),\sigma,Y(\bar{e}),id\},i+1)$

$|(\sigma' X.((\bar{x})\phi'))(\bar{e})$: $(Y(\bar{y}),\mathcal{E}_1,i_1)=$
$formulatoequation(\phi'[X_{i+1}/X],\sigma',i+2,ad(X))$
return$(X_i(\bar{x}_i),\{X_i(\bar{x}_i),\sigma,((\bar{y})(X_{i+1}(\bar{y})))(\bar{e}),id\}::$
$\{X_{i+1}(\bar{y}),\sigma',Y(\bar{y}),ad(X)\}::\mathcal{E}_1,i_1)$

}

在算法 1 中，函数 $formulaChangToequation(F)$ 将命题 F 转换成嵌套谓词等式系 $\mathcal{E}$，通过嵌套调用 $formulatoequation()$ 函数来完成具体的转换，$formulatoequation()$ 有 4 个参数，其中 ϕ 是待转换的命题，σ 是当前谓词变量的不动点类型，i 是可用的下一个谓词变量下标，id 是当前谓词变量的交错嵌套深度，$ad(X)$ 是谓词变量 X 的交错嵌套深度. $formulatoequation()$ 返回一个三元组，其中 $X(\bar{x})$ 与命题 ϕ 对应，$\mathcal{E}$ 是经转换后的嵌套谓词等式系（其中的每一个等式均由等式的左边部分、等式的不动点类型、等式的右边部分和等式所处的交错嵌套层次这 4 个参数来确定），j 是可用的下一个谓词变量下标. 约定嵌套谓词等式系中的第一个谓词变量下标为 1.

附录 2.

依据 Tarski 不动点定理，采用目标驱动策略，有限控制移动界程上基于 μ-演算的一阶谓词界程逻辑模型检测的具体算法如下.

算法 2[10]. 有限控制移动界程上的基于 μ-演算的一阶谓词界程逻辑模型检测算法.

函数 $new(W)$ 产生一个未在名字集 W 中出现过的新名字.

$ModelCheck(P,F)$

{

$(X(\bar{x}),\mathcal{E})=formulaChangToequation(F)$;

for$(i=1;i\leq d;i++)$ $S_{i0}=S_{i1}=S_{i2}=\varnothing$;

//S_{i0} 中保存未计算的结点，S_{i1} 中保存当前值确定的结点，

//S_{i2} 保存当前值不确定的结点，i 表示嵌套层次

$InstallSet((P,X(\bar{x},\phi,\sigma,id)))$;//产生全部计算结点

for$(i=1;i\leq d;i++)$

```
  for(j=0; j≤2; j++)R_ij = S_ij ;
//复制结点到工作区
ValueSet();
return((P,X(x̄,ϕ,σ,id),val).val);
}
InstallSet((P,X(x̄,ϕ,σ,id)))
//产生全部计算结点并初始化存储空间
{
if S_k0 ⋂{(P,X(x̄,ϕ,σ,id),false),(P,X(x̄,ϕ,σ,id),true)}=
∅ then
{
if(σ=μ) add(S_k0,(P,X(x̄,ϕ,σ,id),false));
if(σ=ν) add(S_k0,(P,X(x̄,ϕ,σ,id),true));
//下面生成的结点构成结点(P,X(x̄,ϕ,σ,id),val)的
//直接依赖集 DDS,结点(P,X(x̄,ϕ,σ,id),val)属于
//下面生成结点的直接作用集 DAS.
case ϕ of
Y(ȳ) ⋁ Z(z̄): InstallSet((P,Y(ȳ,ϕ1,σ,id)));
              InstallSet((P,Z(z̄,ϕ2,σ,id)));
|Y(ȳ) ⋀ Z(z̄): InstallSet((P,Y(ȳ,ϕ1,σ,id)));
              InstallSet((P,Z(z̄,ϕ2,σ,id)));
|Y(ȳ)|Z(z̄): ∀P1|P2≡P, InstallSet((P1,Y(ȳ,ϕ1,σ,id)));
              InstallSet((P2,Z(z̄,ϕ2,σ,id)));
|n[Y(ȳ)]: if P≡n[P'] then InstallSet((P',Y(ȳ,ϕ',σ,id)));
|n®Y(ȳ): if n∈fn(P) then return;
              InstallSet((P,Y(ȳ,ϕ',σ,id)));
          if bn(P)≠∅ then ∀m∈bn(P) ⋀ P≡(νm)Q
              InstallSet((Q[n/m],Y(ȳ,ϕ',σ,id)));
|Y(ȳ)@n: InstallSet((n[P],Y(ȳ,ϕ',σ,id)));
|Y(ȳ)⊘n: InstallSet(((νn)P,Y(ȳ,ϕ',σ,id)));
|□Y(ȳ): ∀Q if: P→Q then
              InstallSet((Q,Y(ȳ,ϕ',σ,id)));
|◇Y(ȳ): ∀Q if: P→Q then
              InstallSet((Q,Y(ȳ,ϕ',σ,id)));
|∀x.Y(ȳ): ∀n∈fn(P,ϕ)⋃{new(fn(P,ϕ))}
          InstallSet((P,Y(ȳ,ϕ'[n/x],σ,id)));
|∃x.Y(ȳ): ∀n∈fn(P,ϕ)⋃{new(fn(P,ϕ))}
          InstallSet((P,Y(ȳ,ϕ'[n/x],σ,id)));
|И x.Y(ȳ):
      InstallSet((P,Y(ȳ,ϕ'[new(fn(P,ϕ))/x],σ,id)));
|((ȳ)(Y(ȳ)))(ē):
      InstallSet((P,Y(ȳ,ϕ'[ē/ȳ],σ,id)));
}}
ValueRecord((P,X(x̄,ϕ,σ,i),val))
//计算具体结点值
{
case ϕ of
⊤: return(true);
|⊥: return(false);
|0: {if(P≡0) return(true); else return(false);
|Y(ȳ) ⋁ Z(z̄) and DDS((P,X(x̄,ϕ,σ,i),val))=
  {(P,Y(ȳ,ϕ1,σ1,i1),val1),(P,Z(z̄,ϕ2,σ2,i2),val2)}:
      return(val1 ⋁ val2);
|Y(ȳ) ⋀ Z(z̄) and DDS((P,X(x̄,ϕ,σ,i),val))=
  {(P,Y(ȳ,ϕ1,σ1,i1),val1),(P,Z(z̄,ϕ2,σ2,i2),val2)}:
      return(val1 ⋀ val2);
|Y(ȳ)|Z(z̄) and DDS((P,X(x̄,ϕ,σ,i),val))=
  ⋃_{P1|P2≡P} {(P1,Y(ȳ,ϕ1,σ1,i1),val1),(P2,Z(z̄,ϕ2,σ2,i2),val2)}:
      return(⋁_{P1|P2≡P} val1 ⋀ val2);
|Y(ȳ)@n: and DDS((P,X(x̄,ϕ,σ,i),val))=
      {(P1,Y(ȳ,ϕ1,σ1,i1),val1)}: return(val1);
|Y(ȳ)⊘n: and DDS((P,X(x̄,ϕ,σ,i),val))=
      {(P1,Y(ȳ,ϕ1,σ1,i1),val1)}: return(val1);
|n[Y(ȳ)]: if DDS((P,X(x̄,ϕ,σ,i),val))=
      {(P1,Y(ȳ,ϕ1,σ1,i1),val1)}: return(val1);
    else if DDS(P,X(x̄,ϕ,σ))=∅: return(false);
|n®Y(ȳ): if DDS((P,X(x̄,ϕ,σ,i),val))=
      ⋃_j {(Pj,Yj(ȳj,ϕj,σj,ij),valj)}: return(⋁j valj);
  else if DDS((P,X(x̄,ϕ,σ,i),val))=∅: return(false);
|□Y(ȳ): if(DDS((P,X(x̄,ϕ,σ,i),val))=∅)
      then return(true);
      else if(DDS((P,X(x̄,ϕ,σ,i),val))=
      ⋃_j {(Pj,Yj(ȳj,ϕj,σj,ij),valj)}) return(⋀j valj);
|◇Y(ȳ): if(DDS((P,X(x̄,ϕ,σ,i),val))=∅)
      then return(false);
      else if(DDS((P,X(x̄,ϕ,σ,i),val))=
      ⋃_j {(Pj,Yj(ȳj,ϕj,σj,ij),valj)}) return(⋁i valj);
|∃x.Y(ȳ): and DDS((P,X(x̄,ϕ,σ,i),val))=
      ⋃_j {(Pj,Yj(ȳj,ϕj,σj,ij),valj)}: return(⋁j valj);
|∀x.Y(ȳ): and DDS((P,X(x̄,ϕ,σ,i),val))=
      ⋃_j {(Pj,Yj(ȳj,ϕj,σj,ij),valj)}: return(⋀j valj);
|И x.Y(ȳ): and DDS((P,X(x̄,ϕ,σ,i),val))=
      {(P1,Y(ȳ,ϕ1,σ1,i1),val1)}: return(val1);
|((ȳ)(Y(ȳ)))(n̄): and DDS((P,X(x̄,ϕ,σ,i),val))=
      {(P1,Y(ȳ,ϕ1,σ1,i1),val1)}: return(val1);
}
ValueSet()//计算嵌套谓词等式系
{i=d;//当前等式块块号
while(i≠0)
{
if(R_i0=∅) then i=i−1;
//R_i0保存第 i 等式块上未计算结点
else {
  (P,X(x̄,ϕ,σ,i),val)=readrecord(R_i0);
    //从 R_i0中提取结点
```

$del(R_{i0},(P,X(\bar{x},\phi,\sigma,i),val))$

$val'=val$;

$(P,X(\bar{x},\phi,\sigma,i),val).val=$

$ValueRecord((P,X(\bar{x},\phi,\sigma,i),val))$;

//计算并更新结点 val 值

if($val'=val$) $add(R_{i2},(P,X(\bar{x},\phi,\sigma,i),val))$;

//结点值不变

else{$add(R_{i1},(P,X(\bar{x},\phi,\sigma,i),val))$;

for($j=1;j\leq i;j++$)//从 R_{j2} 中提取属于结点

//$(P,X(\bar{x},\phi,\sigma,i),val)$直接作用集的结点

{$r=DAS((P,X(\bar{x},\phi,\sigma,i),val))\cap R_{j2}$;

$R_{j2}=R_{j2}-r$; $R_{j0}=R_{j0}+r$;

}

if($i<d$) then {for($j=i+1;j\leq d;j++$)

for($k=0;k\leq 2;k++$)$R_{jk}=S_{jk}$;

//恢复工作区中最内层结点的初始状态

$i=d$;//重新计算各内层结点

}}}}}

附录 3.

引理 1. 对于式(1),用算法 2 进行迭代计算,有

(1) $R_4^{j_1(j_2+1)j_3\omega}\in R_4^{j_1j_2j_3\omega}$;

(2) $R_3^{j_1j_2\omega}\in R_3^{(j_1+1)j_2\omega}$.

证明.

(1) 在算法 2 中,直接依据 Tarski 不动点定理计算不动点,结点值一旦发生改变,则不动点嵌套层次比该结点深的所有结点均需从初值重新计算,直至最外层($id=1$)的所有结点到达不动点,计算停止.

在式(1)中,$\sigma_1=\sigma_3=\mu,\sigma_2=\sigma_4=\nu,\mathcal{E}_1,\mathcal{E}_2,\mathcal{E}_3,\mathcal{E}_4$ 等式块对应的结点集为 R_1,R_2,R_3,R_4,其中 $R_i=R_{i0}\cup R_{i1}\cup R_{i2}$ ($1\leq i\leq 4$). 对于任意的 j_1,j_2,j_3 有 $R_2^{j_1(j_2+1)}\in R_2^{j_1j_2}$, $R_3^{j_1(j_2+1)0}=R_3^{j_1j_20}$, $R_4^{j_1j_200}=R_4^{j_1(j_2+1)00}$, $R_4^{j_1j_201}=f_4(R_1^{j_1},R_2^{j_1j_2},R_3^{j_1j_20},R_4^{j_1j_200})$, $R_4^{j_1(j_2+1)01}=f_4(R_1^{j_1},R_2^{j_1(j_2+1)},R_3^{j_1(j_2+1)0},R_4^{j_1(j_2+1)00})$.

根据 f_4 的单调性有 $R_4^{j_1(j_2+1)01}\in R_4^{j_1j_201}$;类推有 $R_4^{j_1(j_2+1)0\omega}\in R_4^{j_1j_20\omega}$.

$R_3^{j_1j_21}=f_3(R_1^{j_1},R_2^{j_1j_2},R_3^{j_1j_20},R_4^{j_1j_20\omega})$,

$R_3^{j_1(j_2+1)1}=f_3(R_1^{j_1},R_2^{j_1(j_2+1)},R_3^{j_1(j_2+1)0},R_4^{j_1(j_2+1)0\omega})$.

因 $R_2^{j_1(j_2+1)}\in R_2^{j_1j_2}$, $R_3^{j_1(j_2+1)0}=R_3^{j_1j_20}$, $R_4^{j_1(j_2+1)0\omega}\in R_4^{j_1j_20\omega}$, 根据 f_3 的单调性有 $R_3^{j_1(j_2+1)1}\in R_3^{j_1j_21}$.

$R_4^{j_1j_211}=f_4(R_1^{j_1},R_2^{j_1j_2},R_3^{j_1j_21},R_4^{j_1j_210})$,

$R_4^{j_1(j_2+1)11}=f_4(R_1^{j_1},R_2^{j_1(j_2+1)},R_3^{j_1(j_2+1)1},R_4^{j_1(j_2+1)10})$.

因 $R_2^{j_1(j_2+1)}\in R_2^{j_1j_2}$, $R_3^{j_1(j_2+1)1}\in R_3^{j_1j_21}$, $R_4^{j_1(j_2+1)10}=R_4^{j_1j_210}$, 根据 f_4 的单调性有 $R_4^{j_1(j_2+1)11}\in R_4^{j_1j_211}$;类推有 $R_4^{j_1(j_2+1)1\omega}\in R_4^{j_1j_21\omega}$.

重复以上步骤有 $R_4^{j_1(j_2+1)j_3\omega}\in R_4^{j_1j_2j_3\omega}$.

(2) $R_4^{j_1001}=f_4(R_1^{j_1},R_2^{j_10},R_3^{j_100},R_4^{j_1000})$, $R_4^{(j_1+1)001}=f_4(R_1^{j_1+1},R_2^{(j_1+1)0},R_3^{(j_1+1)00},R_4^{(j_1+1)000})$.

由于 $R_1^{j_1}\in R_1^{j_1+1}$, $R_2^{j_10}=R_2^{(j_1+1)0}$, $R_3^{j_100}=R_3^{(j_1+1)00}$, $R_4^{j_1000}=R_4^{(j_1+1)000}$, 根据 f_4 的单调性有 $R_4^{j_1001}\in R_4^{(j_1+1)001}$. 类推有 $R_4^{j_100\omega}\in R_4^{(j_1+1)00\omega}$.

$R_3^{j_101}=f_3(R_1^{j_1},R_2^{j_10},R_3^{j_100},R_4^{j_100\omega})$,

$R_3^{(j_1+1)01}=f_3(R_1^{j_1+1},R_2^{(j_1+1)0},R_3^{(j_1+1)00},R_4^{(j_1+1)00\omega})$.

由于 $R_1^{j_1}\in R_1^{j_1+1}$, $R_2^{j_10}=R_2^{(j_1+1)0}$, $R_3^{j_100}=R_3^{(j_1+1)00}$, $R_4^{j_100\omega}\in R_4^{(j_1+1)00\omega}$, 根据 f_3 的单调性有 $R_3^{j_101}\in R_3^{(j_1+1)01}$.

$R_4^{j_1011}=f_4(R_1^{j_1},R_2^{j_10},R_3^{j_101},R_4^{j_1010})$,

$R_4^{(j_1+1)011}=f_4(R_1^{j_1+1},R_2^{(j_1+1)0},R_3^{(j_1+1)01},R_4^{(j_1+1)010})$.

由于 $R_1^{j_1}\in R_1^{j_1+1}$, $R_2^{j_10}=R_2^{(j_1+1)0}$, $R_3^{j_101}\in R_3^{(j_1+1)01}$, $R_4^{j_1010}=R_4^{(j_1+1)010}$, 根据 f_4 的单调性有 $R_4^{j_1011}\in R_4^{(j_1+1)011}$. 类推有 $R_4^{j_101\omega}\in R_4^{(j_1+1)01\omega}$.

重复以上步骤可得 $R_3^{j_10k}\in R_3^{(j_1+1)0k}$, $R_4^{j_10k\omega}\in R_4^{(j_1+1)0k\omega}$, $R_3^{j_10\omega}\in R_3^{(j_1+1)0\omega}$, $R_4^{j_10\omega\omega}\in R_4^{(j_1+1)0\omega\omega}$.

$R_2^{j_11}=f_2(R_1^{j_1},R_2^{j_10},R_3^{j_10\omega},R_4^{j_10\omega\omega})$,

$R_2^{(j_1+1)1}=f_2(R_1^{j_1+1},R_2^{(j_1+1)0},R_3^{(j_1+1)0\omega},R_4^{(j_1+1)0\omega\omega})$.

由于 $R_1^{j_1}\in R_1^{j_1+1}$, $R_2^{j_10}=R_2^{(j_1+1)0}$, $R_3^{j_10\omega}\in R_3^{(j_1+1)0\omega}$, $R_4^{j_10\omega\omega}\in R_4^{(j_1+1)0\omega\omega}$, 根据 f_2 的单调性有 $R_2^{j_11}\in R_2^{(j_1+1)1}$.

由 $R_1^{j_1}\in R_1^{j_1+1}$, $R_2^{j_11}\in R_2^{(j_1+1)1}$, $R_3^{j_110}=R_3^{(j_1+1)10}$, $R_4^{j_1100}=R_4^{(j_1+1)100}$ 可得 $R_4^{j_1101}\in R_4^{(j_1+1)101}$, 类推有 $R_4^{j_110\omega}\in R_4^{(j_1+1)10\omega}$.

由于 $R_1^{j_1}\in R_1^{j_1+1}$, $R_2^{j_11}\in R_2^{(j_1+1)1}$, $R_3^{j_110}=R_3^{(j_1+1)10}$, $R_4^{j_110\omega}\in R_4^{(j_1+1)10\omega}$, 根据 f_3 的单调性有 $R_3^{j_111}\in R_3^{(j_1+1)11}$, 类推有 $R_3^{j_11k}\in R_3^{(j_1+1)1k}$, $R_3^{j_11\omega}\in R_3^{(j_1+1)1\omega}$.

重复以上步骤有 $R_3^{j_1j_2\omega}\in R_3^{(j_1+1)j_2\omega}$. 证毕.

附录 4.

定理 1. 根据算法 3 的计算,如果 $ModelCheck(P,F)$ 最后计算结果$(P,X(\bar{x},\phi,\sigma,id),val)$. $val=$true,则 $P\models F$,否则 $P\not\models F$.

证明(简要).

在算法 3 中,函数 $formulaChangToequation()$ 将基于 μ-演算的一阶谓词界程逻辑公式转换成嵌套谓词等式系,两者在语义上是一致的[10]. 函数 $InstallSet((P,X(\bar{n},\phi,\sigma)))$和函数 $ValueRecord((P,X(\bar{x},\phi,\sigma)))$严格遵从移动界程的语义以及基于 μ-演算的一阶谓词界程逻辑的语义进行计算,其正确性是显然的. 算法 3 中函数 $ValueSet()$的执行首先从

嵌套谓词等式系中最内层等式块$\mathcal{E}_4$对应的结点集开始执行. 等式块间的跳转,有两种情况:一是当结点值发生改变时,嵌套层次比当前层次深的所有结点的当前不动点值均需重新计算,计算跳转到最内层结点重新开始计算;二是当前层所有结点均到达不动点状态,计算跳转到外一层结点集,这两种跳转都是严格遵从嵌套不动点等式块求解顺序进行的,正确性是显然的. 同层结点集合中结点的计算是在 R_{i0} 中找出值改变的结点,并移到 R_{i1} 中. 在外层结点值不变的情况下,根据单调性,结点值发生改变后是不会再改变的,这样的处理是正确的. 在算法 3 中,用 $ValueSet()$ 函数计算时,当等式块$\mathcal{E}_3$对应的结点集有结点值改变时,等式块$\mathcal{E}_4$对应的结点集不总是从初值 S_{40} 开始计算,而是从序列 $R_4^{j_1 0k0}, R_4^{j_1 0k\omega}, R_4^{j_1 1k\omega}, \cdots, R_4^{j_1 \omega k\omega}$ 中找到最近计算过的状态开始计算;当等式块$\mathcal{E}_2$对应的结点集有结点值改变时,等式块$\mathcal{E}_3$对应的结点集从序列 $R_3^{0i0}, R_3^{0i\omega}, R_3^{1i\omega}, \cdots, R_3^{\omega i\omega}$ 中找到最近计算过的状态开始计算. 根据引理 1,这样的处理是正确的,并且能减少算法的计算量,改善算法的时间复杂度. 当等式系最外层$\mathcal{E}_1$对应的结点集中所有结点值均到达不动点状态时,整个计算结束. 此时若$(P, X_1(\bar{x}, \phi, \sigma))$中 $X_1.\sigma=$ true,则说明 $P \models \phi$,否则 $P \not\models \phi$.　　证毕.

附录 5.

设 $m_i=|R_i|$ 为等式块$\mathcal{E}_i$对应的结点集规模,调用 f_i 函数的次数作为等式块$\mathcal{E}_i$对应结点集的计算量,记为$|f_i|$.

定理 2.　利用算法 3 对嵌套谓词等式系(1)进行计算,时间复杂度为 $O((M+d)^2)$.

证明(简要).

根据算法 3,等式块$\mathcal{E}_1$对应的结点计算序列为 $R_1^0, R_1^1, R_1^2, \cdots, R_1^\omega$,计算结点总数为 m_1. 由于 $\sigma_1=\mu$,各结点 val 初值均为 false. 在计算过程中结点 val 值一旦变成 true,结点进入 R_{11}. 由 f_1 的单调性可知,该结点在整个计算过程中的 val 值将不再发生改变. 在最坏的情况下,R_{10} 中的结点 val 值最终均由 false 变成 true 才能到达不动点,因此在最坏的情况下等式块$\mathcal{E}_1$调用 f_1 的次数为 $|f_1|=m_1+1$,加 1 是因为 $R_{10}=\varnothing$时也调用了一次 f_1.

R_1 取值为 $R_1^{j_1}$ 时,等式块$\mathcal{E}_2$对应的结点计算序列为 $R_2^{j_1 0}, R_2^{j_1 1}, R_2^{j_1 2}, \cdots, R_2^{j_1 \omega}$. 在最坏的情况下,等式块$\mathcal{E}_2$对应的结点调用 f_2 的次数$=m_2+1$,R_1 的不同取值至多有 m_1+1 个,所以$|f_2|=(m_1+1)\cdot(m_2+1)$.

计算$\mathcal{E}_3$对应的结点 R_3 时,从 R_3^{000} 开始计算的情形至多有 $R_3^{000}, R_3^{010}, \cdots, R_3^{0\omega 0}$ 共 m_2+1 次,每次从 R_3^{000} 开始计算的序列是 $R_3^{0j0}, R_3^{0j1}, \cdots, R_3^{0j\omega_1}, R_3^{1j\omega_1}=R_3^{0j\omega_1}, R_3^{1j(\omega_1+1)}, R_3^{1j(\omega_1+2)}, \cdots, R_3^{2j\omega_2}=R_3^{1j\omega_2}, \cdots, R_3^{\omega j\omega}$.

不考虑重复的取值,序列的长度最大为 m_3+1. 不考虑重复计算,计算$\mathcal{E}_3$对应结点调用f_3的次数$=(m_2+1)\cdot(m_3+1)$.

每计算一个 $R_3^{j_1 j_2 \omega}$ 时须重复计算一次,重复计算量与 R_2 的不同取值数量一致,所以重复计算量$=(m_1+1)\cdot(m_2+1)$,所以$|f_3|=(m_1+1)\cdot(m_2+1)+(m_2+1)\cdot(m_3+1)$.

在计算 R_3 时,当 $j=0$ 时,序列 $R_3^{000}, R_3^{001}, \cdots, R_3^{00\omega_1}, R_3^{10\omega_1}=R_3^{00\omega_1}, R_3^{10(\omega_1+1)}, \cdots, R_3^{10\omega_2}, R_3^{20\omega_2}=R_3^{10\omega_2}, \cdots, R_3^{\omega 0\omega}$,序列最大长度为 m_3+1. 计算 R_4 时,从 R_4^{0000} 开始计算的情形至多有 m_3+1 次. 每次从 R_4^{0000} 开始计算的序列为 $R_4^{i0k0}, R_4^{i0k1}, \cdots, R_4^{i0k\omega_1}, R_4^{i1k\omega_1}=R_4^{i0k\omega_1}, R_4^{i1k(\omega_1+1)}, \cdots, R_4^{i1k\omega_2}, R_4^{i2k\omega_2}=R_4^{i1k\omega_2}, \cdots, R_4^{i\omega k\omega}$,序列最大长度为 m_4+1. 不考虑重复计算,一个序列调用 f_4 的次数$=m_4+1$,m_3+1 个序列调用 f_4 的次数$=(m_3+1)\cdot(m_4+1)$.

每计算一个$R_4^{j_1 j_2 j_3 \omega}$,需重复计算一次,总的重复次数与 R_3 的不同取值数量一致,所以重复计算量$=(m_2+1)(m_3+1)$,计算 R_4 的总调用 f_4 的次数$|f_4|=(m_2+1)(m_3+1)+(m_3+1)\cdot(m_4+1)$.

计算嵌套谓词等式系$\mathcal{E}_1 :: \mathcal{E}_2 :: \mathcal{E}_3 :: \mathcal{E}_4$对应计算结点的计算量为

$$\begin{aligned}|f| &= |f_1|+|f_2|+|f_3|+|f_4| \\ &= (m_1+1)+(m_1+1)\cdot(m_2+1)+(m_1+1)\cdot(m_2+1)+ \\ &\quad (m_2+1)\cdot(m_3+1)+(m_2+1)(m_3+1)+(m_3+1)\cdot(m_4+1) \\ &\le (m_1+m_2+m_3+m_4+4)^2=O((M+d)^2).\end{aligned}$$

证毕.

附录 6.

定理 3.　算法 3 的空间复杂度为 $O(M)$.

证明(简略).

用算法 3 计算时,存储空间开销主要是 a,b 数组,分别存放 R_3,R_4 在迭代计算过程中的迭代序号以及迭代计算中间值. $a[i]$保存序列 $R_3^{0i0}, R_3^{0i\omega}, R_3^{1i\omega}, \cdots, R_3^{\omega i\omega}$ 中计算的最新数据,a 数组长度$=|R_2|+1$. $b[k]$保存序列 $R_4^{j_1 0k0}, R_4^{j_1 0k\omega}, R_4^{j_1 1k\omega}, \cdots, R_4^{j_1 \omega k\omega}$ 中计算的最新数据. b 数组长度$=|R_3|+1$. 其它存放数据的临时变量以及控制循环的循环变量总的存储开销是常量 c. 因此算法 3 总的空间需求量$=|R_2|+1+|R_3|+1+c=O(M)$.　　证毕.

附录 7.

定理 4.　用算法 4 对嵌套谓词等式系(3)对应的结点集进行计算,算法时间复杂度为 $O((M+d)^{\lfloor d/2 \rfloor+1})$.

证明.

m_i,$|f_i|$定义见附录 5.

由附录 5 可知,编号为 1 的等式块中各等式系对应的计算结点的计算量分别为

$|f_1|=m_1+1$,

$|f_2|=(m_1+1)\cdot(m_2+1)$,

$|f_3|=(m_1+1)\cdot(m_2+1)+(m_2+1)\cdot(m_3+1)$,

$|f_4|=(m_2+1)(m_3+1)+(m_3+1)\cdot(m_4+1)$.

从算法 4 可知,在计算$f_{x,4}()$之前,均会调用$ValueSet_{x+1}()$计算嵌套层次更深的计算结点的当前不动点值. 因此编号为 2 的等式块中各等式系对应的计算结点的计算量分别为

$|f_5|=|f_4|\cdot(m_5+1)$,

$|f_6|=|f_4|\cdot(m_5+1)\cdot(m_6+1)$,

$|f_7|=|f_4|\cdot((m_5+1)\cdot(m_6+1)+(m_6+1)\cdot(m_7+1))$,

$|f_8|=|f_4|\cdot((m_6+1)(m_7+1)+(m_7+1)\cdot(m_8+1))$.

以此类推,当 $i>0$ 时,均有

$|f_{4i+1}|=|f_{4i}|\cdot(m_{4i+1}+1)$,

$|f_{4i+2}|=|f_{4i}|\cdot(m_{4i+1}+1)\cdot(m_{4i+2}+1)$,

$|f_{4i+3}|=|f_{4i}|\cdot((m_{4i+1}+1)\cdot(m_{4i+2}+1)+(m_{4i+2}+1)\cdot(m_{4i+3}+1))$,

$|f_{4i+4}|=|f_{4i}|\cdot((m_{4i+2}+1)(m_{4i+3}+1)+(m_{4i+3}+1)\cdot(m_{4i+4}+1))$.

当 $d \bmod 4=0$ 时,可将等式系分为 $n=d/4$ 块,每块不动点算子嵌套深度均为 4. 用算法 4 进行计算,则有

$$|f|=\sum_{j=1}^{d}|f_j|$$
$$\leq\prod_{i=0}^{n-1}((m_{4i+1}+1)+(m_{4i+1}+1)\cdot(m_{4i+2}+1)+(m_{4i+1}+1)\cdot(m_{4i+2}+1)+(m_{4i+2}+1)\cdot(m_{4i+3}+1)+(m_{4i+3}+1)\cdot(m_{4i+4}+1))$$
$$\leq\prod_{i=0}^{n-1}(m_{4i+1}+m_{4i+2}+m_{4i+3}+m_{4i+4}+4)^2$$
$$\leq O((M+d)^{d/2}).$$

当 $d \bmod 4\neq 0$ 时,则有

(1) $d \bmod 4=1$

$$|f|\leq\Big(\prod_{i=0}^{n-2}(|f_{4i+1}|+|f_{4i+2}|+|f_{4i+3}|+|f_{4i+4}|)\Big)\cdot(m_d+1)$$
$$\leq\prod_{i=0}^{n-2}(m_{4i+1}+m_{4i+2}+m_{4i+3}+m_{4i+4}+4)^2\cdot(m_d+1)$$
$$=O((M+d)^{\lfloor d/2\rfloor+1}).$$

(2) $d \bmod 4=2$

$$|f|\leq\Big(\prod_{i=0}^{n-2}(|f_{4i+1}|+|f_{4i+2}|+|f_{4i+3}|+|f_{4i+4}|)\Big)\cdot((m_{d-1}+1)+(m_{d-1}+1)\cdot(m_d+1))$$
$$=O((M+d)^{\lfloor d/2\rfloor+1}).$$

(3) $d \bmod 4=3$

$$|f|\leq\Big(\prod_{i=0}^{n-2}(|f_{4i+1}|+|f_{4i+2}|+|f_{4i+3}|+|f_{4i+4}|)\Big)\cdot((m_{d-2}+1)+2(m_{d-2}+1)\cdot(m_{d-1}+1)+(m_{d-1}+1)\cdot(m_d+1))$$
$$=O((M+d)^{\lfloor d/2\rfloor+1}).$$

综上所述:$|f|=O((M+d)^{\lfloor d/2\rfloor+1})$. 证毕.

附录 8.

定理 5. 算法 4 对嵌套谓词等式系(3)进行计算,算法空间复杂度为 $O(M)$.

证明(简略).

在算法 4 中,$ValueSet_x()$所需存储空间主要是 a_x,b_x 数组,分别存放 R_{4x+3},R_{4x+4} 在迭代计算过程中的中间值以及迭代序号. 其长度分别为 $|R_{4x+2}|+1$ 和 $|R_{4x+3}|+1$. 其他存放数据的临时变量以及控制循环的循环变量总的需求量是常量 c. 因此,算法 4 总的空间需求量 $=\sum_{i=1}^{n}(|R_{2i+2}|+1+|R_{2i+3}|+1+c)=O(M)$. 证毕.

JIANG Hua, born in 1970, Ph. D., associate professor. His main research interests include model checking, algorithm design and analysis.

Background

Model checking is an algorithm technology used to verify whether system satisfies given property. First-order predicate ambient logic based on μ-calculus was put forward by Academician Lin Huimin in Chinese Academy of Sciences in 2004. It uses predicate variable to construct fixpoint formula which is convenient to describe the properties of closed-loop system. The formula's semantics is brief. Mobile ambient calculus was first proposed by Cardelli, et al. in 2000. It is used to describe formal model for distributed mobile computing. Process and nesting sub-ambient constitute concurrent computing environment, nesting sub-ambient constitutes tree structure and has strong space expression ability.

The first model checking algorithm for first-order predicate ambient logic in finite control mobile ambient based on μ-calculus was put forward by Academician Lin Huimin in Chinese Academy of Sciences in 2004. The second kindred algorithm was proposed by the author of this paper in 2009. For the present best model checking algorithm for first-order predicate ambient logic in finite control mobile ambient based on μ-calculus, its time complexity has exponent relation to d and space complexity has linear relation to d, d is the alternating nesting depth of the formula in fixpoint. For the algorithm designed in this paper, its time complexity has exponent relation to $d/2+1$ and space complexity has linear relation to d, which is the third presently known model checking algorithm for first-order predicate ambient logic in finite control mobile ambient based on μ-calculus.

This work is a part of "The Model Checking and Controller Synthesis of Metric Interval Temporal Logic MITL", which is mainly supported by the National Natural Science Foundation of China under Grant No. 61472406. The project mainly aims to solve the problem of implementing MITL model checking tool and controller synthesis tool. The algorithm's core idea in this paper is expected to be used in MITL model checking tool after real-time extension. This work is also a part of "Research on the Core Algorithm of the Mobile Ambients Model Checking", which is supported by the Natural Science Foundation of Fujian under Grant Nos. 2015J01269 and 2016J01304. The project mainly aims to improve the performance of existing algorithm on the Mobile Ambients Model Checking. The algorithm in this paper is the core task of the project.