π-演算互模拟判定算法的优化和实现*

许 文, 方 海, 林惠民

(中国科学院 软件研究所 计算机科学开放研究实验室,北京 100080)

E-mail: lhm@ox.ios.ac.cn

http://www.ios.ac.cn/openlab

摘要: π-演算采用互模拟作为进程等价的准则.对有限状态的π-演算进程互模拟等价是可判定的,由此提出了互模拟检测算法的一种优化技术.该技术基于只将输入名字实例化为在以后的等名测试中用到的自由名字.通过实例说明这一优化技术可大大减少所用的时间和空间,并证明了优化算法的正确性.

关键词: 进程代数;π-演算;互模拟;验证算法

中图法分类号: TP301 **文献标识码**: A

互模拟等价是用于刻画进程性质的语义理论.对于早先的进程代数模型,如Milner的通信进程演算(calculus of communicating systems,简称CCS),已有成熟的技术和工具进行互模拟等价的验证.为了描述通信拓扑结构可动态改变的并发系统,Milner等人又提出了π-演算[2].π-演算中不区分变量和常量,而统一代之以"名字"的概念,所以适用于传值CCS的互模拟算法和工具不能直接应用于π-演算.

π-演算中输入前缀引入受囿名字,这是π-演算区别于CCS的主要特征.受囿名字的作用类似于变量,而自由名字则类似于常量.B. Victor等人实现了一个检验π-演算进程间互模拟等价关系的系统MWB[3,4],但目前的版本(Polyadic Version 3.122)只检查开互模拟关系(open bisimulation)[5].文献[6]提出了符号互模拟的概念.而后符号互模拟被引入π-演算进程的互模拟验证,并得到了有效算法[1].我们在该算法的基础上实现了π-演算进程间迟(早)互模拟等价的验证.

文献[1]的算法在验证输入动作时只需把输入名字实例化成有限个名字,但是当需要验证的进程有较复杂的结构时,在实际应用过程中仍然需要耗费相当的时间和空间.我们发现,在某些情况下可以在该算法的基础上缩小输入名字的实例化范围,从而提高算法的效率.从一些实例的测试结果来看,优化算法可以极大地提高时间和空间效率.

本文首先回顾π-演算和互模拟中有关的术语和概念,接着介绍验证系统的主要实现技术,然后描述我们对验证算法所进行的优化及其正确性证明.

1 π-演算和模式互模拟

1.1 π-演算中的相关定义及记号

我们预先假定有序可数名字集N,其中的元素用$a,b,x,y,w,\ldots$表示;条件是以常见的布尔操

* 收稿日期: 2000-05-11; 修改日期: 2000-11-27

基金项目: 国家自然科学基金资助项目(69833020)

作者简介: 许文(1975-),女,湖南长沙人,硕士,主要研究领域为并发理论,形式化方法;方海(1976-),男,天津人,硕士,主要研究领域为形式化方法,并发理论;林惠民(1947-),男,福建福州人,研究员,博士生导师,中国科学院院士,主要研究领域为进程代数,并发理论,模型检测,形式化方法.

作符作用于形如 $x=y$ 的等名测试而成的布尔表达式,用 $\varphi,\psi,\ldots$ 表示. $\neg(x=y)$ 简写为 $x\neq y$. 名字替换 σ 是从 N 到 N 的部分映射,$\lfloor\tilde{y}/\tilde{x}\rfloor$ 表示将 $\tilde{x}$ 映射为 $\tilde{y}$ 的名字替换,其中 $\tilde{x}$ 和 $\tilde{y}$ 是长度相等的名字向量,且 $\tilde{x}$ 中的名字互不相同. $\sigma\{x\mapsto v\}$ 表示将 σ 在 x 处的值改为 v,其余不变. 空替换记为 $\varnothing$. $\varphi\sigma=\text{true}$ 记作 $\sigma\vdash\varphi$;如果对任意替换 $\sigma,\sigma\models\varphi$ 蕴含 $\sigma\models\psi$,记作 $\varphi\Rightarrow\psi$;如果 $\varphi\Rightarrow\psi$ 且 $\psi\Rightarrow\varphi$,记作 $\varphi=\psi$. $\sigma\sigma'$ 是 σ 与 σ' 的复合,即对任意表达式 $e,e(\sigma\sigma')=(e\sigma)\sigma'$.

π 演算的进程由以下 BNF 范式定义:

$$p::=\mathbf{0}\mid\alpha.p\mid p+p\mid\varphi p\mid\upsilon xp\mid p\quad p\mid A(x_1,\ldots,x_n),$$
$$\alpha::=\tau\mid\overline{a}x\mid a(x).$$

关于这些操作符的含义以及受囿名字、自由名字和 α 变换的定义请参见文献[2]. $\upsilon\overline{x}ax.p$ 简写为 $\overline{a}(x).p$. 进程 p 中的自由名集合记作 $fn(p)$,动作的自由名集和受囿名集被定义为 $fn(\tau)=bn(\tau)=bn(\overline{a}x)=\varnothing,fn(\overline{a}x)=\{a,x\},fn(a(x))=fn(\overline{a}(x))=\{a\},bn(a(x))=bn(\overline{a}(x))=\{x\}$. 进程标识符集的元素用 $A,B,\ldots$ 表示. 对应于每个进程标识符 A 都有一个形如 $A(x_1,\ldots,x_n)\overset{\text{def}}{=}p$ 的递归定义,其中 $fn(p)\subseteq\{x_1,\ldots,x_n\}$. 此外,我们不允许在递归定义体中出现并行复合操作符,经过这样限制的语言称为"有穷控制"π-演算. 对动作的替换定义为 $\tau\sigma=\tau,(\overline{a}x)\sigma=\overline{a\sigma}x\sigma,(a(x))\sigma=(a\sigma)(x),(\overline{a}(x))\sigma=\overline{a\sigma}(x)$.

定义 1. 进程上的对称二元关系 $\Im$ 是一个强互模拟,如果 $p\Im q$ 时有:

(1) 如果 $p\xrightarrow{\alpha}p'$,α 为不受囿的输出动作或 τ 动作,那么存在 q',$q\xrightarrow{\alpha}q'$ 且 $p'\Im q'$.

(2) 如果 $p\xrightarrow{x(y)}p'$ 且 $y\notin n(p,q)$,那么存在 q',$q\xrightarrow{x(y)}q'$ 且对任意名字 w,$p'\{w/y\}\Im q'\{w/y\}$.

(3) 如果 $p\xrightarrow{\overline{x}(y)}p'$ 且 $y\notin n(p,q)$,那么存在 q',$q\xrightarrow{\overline{x}(y)}q'$ 且 $p'\Im q'$.

若存在一个强互模拟 $\Im$ 使得 $p\Im q$,则记作 $p\sim q$. 易见"$\sim$"是最大的强互模拟关系.

1.2 符号操作语义与符号迁移图

定义 2. 符号迁移图(symbolic transition graph,简称 STG)是一个有向图,其每个节点 n 标有名字的有穷集 $fn(n)$,每条边标有由布尔表达式和动作组成的二元组,它们满足:如果 $n\overset{\varphi,\alpha}{\longmapsto}m$ 是从 n 到 m 的一条边,则有 $fn(\varphi)\cup fn(\alpha)\subseteq fn(n),bn(\alpha)\cap fn(n)=\varnothing,fn(m)\subseteq fn(n)\cup bn(\alpha)$.

下面列出 π-演算的符号操作语义规则(略去了与 Sum 和 Par 对称的规则). 其中 Res 和 Open 规则中使用了对条件的操作 ν_x,其定义如下:

$$\nu_x\text{true}=\text{true},\qquad \nu_x[x=x]=\text{true},$$
$$\nu_x[x=y]=\text{false},\qquad \nu_x[y=z]=[y=z],$$
$$\nu_x\neg\varphi=\neg\nu_x\varphi,\qquad \nu_x(\varphi\wedge\psi)=\nu_x\varphi\wedge\nu_x\psi.$$

显然有 $x\notin n(\nu_x\varphi)$.

$$\text{Pre}\ \frac{}{\alpha.p\xrightarrow{\text{true},\alpha}p},\qquad \text{Cond}\ \frac{p\xrightarrow{\psi,\alpha}p'}{\varphi p\xrightarrow{\psi\wedge\varphi,\alpha}p'}\,n(\varphi)\cap bn(\alpha)=\varnothing,$$

$$\text{Sum}\ \frac{p\xrightarrow{\varphi,\alpha}p'}{p+q\xrightarrow{\varphi,\alpha}p'},\qquad \text{Par}\ \frac{p\xrightarrow{\varphi,\alpha}p'}{p\mid q\xrightarrow{\varphi,\alpha}p'\mid q}\,bn(\alpha)\cap fn(q)=\varnothing,$$

$$\text{Res}\ \frac{p\xrightarrow{\varphi,\alpha}p'}{\upsilon xp\xrightarrow{\nu_x\varphi,\alpha}\upsilon xp'}\,x\notin n(\alpha),\qquad \text{Com}\ \frac{p\xrightarrow{\varphi,\alpha(x)}p'\quad q\xrightarrow{\psi,\overline{\alpha}y}q'}{p\mid q\xrightarrow{[\alpha=b]\wedge\varphi\wedge\psi,\tau}p'[y/x]\mid q'},$$

$$\text{Open}\ \frac{p\xrightarrow{\varphi,\overline{a}x}p'}{\upsilon xp\xrightarrow{\nu_x\varphi,\overline{a}(x)}p'}\,x\neq a,\qquad \text{Close}\ \frac{p\xrightarrow{\varphi,\alpha(x)}p'\quad q\xrightarrow{\psi,\overline{\alpha}(y)}q'}{p\mid q\xrightarrow{[\alpha=b]\wedge\varphi\wedge\psi,\tau}\upsilon y(p'[y/x]\mid q')}.$$

$$\mathrm{Id}\ \frac{p[y_1,\ldots,y_n/x_1,\ldots,x_n]\xrightarrow{M,\alpha}p'}{A(y_1,\ldots,y_n)\xrightarrow{M,\alpha}p'}\ A(x_1,\ldots,x_n)\overset{\mathrm{def}}{=}p.$$

根据符号操作语义,我们可以对任意给定的π-演算进程生成相应的符号迁移图.给定一个符号迁移图$\mathscr{G}$,$\mathscr{G}$上的状态空间$S=\{n_\sigma\mid n$是$\mathscr{G}$的节点,σ是名字上的替换$\}$.STG的迟操作语义是状态上的迁移关系,由下面的规则给出:

$$\frac{m\overset{\varphi,\tau}{\longmapsto}n}{m_\sigma\xrightarrow{\tau}n_\sigma}\sigma\models\varphi,\qquad \frac{m\overset{\varphi,\overline{x}z}{\longmapsto}n}{m_\sigma\xrightarrow{\overline{\sigma(x)}\sigma(z)}n_\sigma}\sigma\models\varphi,\qquad \frac{m\overset{\varphi,x(y)}{\longmapsto}n}{m_\sigma\xrightarrow{\sigma(x)(y)}n_\sigma}\sigma\models\varphi.$$

我们也称$\mathscr{G}$上的状态为进程.对$p=n_\sigma$,规定$p\sigma'=n_{\sigma\sigma'}$.

模式互模拟

定义3. 进程上的对称二元关系R是一个模式互模拟,如果$(p,q)\in R$时有:

(1) 如果$p\overset{a(x)}{\longmapsto}p'$,那么存在$q'$,$q\overset{a(y)}{\longmapsto}q'$且有

◇对任意$v\in fn(p,q)$,$(p'[v/x],q'[v/y])\in R$;

◇对一个新名字$z\notin fn(p,q)$,$(p'[z/x],q'[z/y])\in R$.

(2) 如果$p\overset{\overline{a}(x)}{\longmapsto}p'$,那么存在$q'$,$q\overset{\overline{a}(y)}{\longmapsto}q'$,并对一个新名字$z\notin fn(p,q)$,$(p'[z/x],q'[z/y])\in R$.

(3) 对其他动作α,如果$p\xrightarrow{\alpha}p'$,那么存在q',$q\xrightarrow{\alpha}q'$,并有$(p',q')\in R$.

当存在一个模式互模拟R使得$(p,q)\in R$时,记作$p\dot{\sim}q$.易见,$\dot{\sim}$是最大的模式互模拟关系.模式互模拟和互模拟之间的关系是[1],$p\sim q$当且仅当$p\dot{\sim}q$.

2 算法实现及优化

2.1 算法及验证系统组成

根据模式互模拟的定义,文献[1]中给出了相应的互模拟验证算法,它是从为传值CCS设计的"on-the-fly"互模拟验证算法[7]中演化而来的.下面我们列出该算法中比较输入动作的部分,算法的其余部分以及对算法的直观解释请参看文献[1].其中nextSN函数的作用是,给定一个(或多个)进程,返回一个不在进程中出现的最小名字.

$\mathrm{match}_{\{a(x)\}}(p,q)=$

for each $p\overset{a(x)}{\longmapsto}p_i$, $q\overset{a(y)}{\longmapsto}q_j$

for each $v\in(fn(p,q)\cup\{\mathrm{nextSN}(p,q)\}$

$b_{ij}^v=\mathrm{close}(p_i[v/x],q_j[v/y])$

$b_{ij}=\bigwedge_{v\in fn(p,q)\cup\{\mathrm{nextSN}(p,q)\}}b_{ij}^v$

return $(\bigwedge_i(\bigvee_j b_{ij}))\wedge(\bigwedge_j(\bigvee_i b_{ij}))$

我们现在的π-演算互模拟验证系统是在传值CCS互模拟验证系统的基础上,用SML/NJ (Version 110.0.3)加以实现的.

验证系统从输入文件中读入两个进程的定义以后,先根据π-演算的符号操作语义分别构造对应的符号迁移图,再在符号迁移图上使用"on-the-fly"验证算法进行互模拟验证.

在构造符号迁移图的时候,由于动作的受囿名字可能与当前的自由名字冲突,因此需要进行换名.我们的系统现在采取的是动态换名的方法,即当发现名字冲突时,把动作的受囿名字替换成一个新名字.在第2.2节中将介绍系统目前所采用的换名算法.

同时,对某些输入变量可以缩小实例化的范围,从而大大减少验证中所生成的状态数,节省时间和空间.在第2.3节中将介绍这个优化算法,并给出相应的证明.

2.2 实现中的动态换名算法

当 $p \xrightarrow{\psi,\alpha} p'$ 时,如果 $bn(\alpha) \cap fn(p) \neq \varnothing$,那么要将 α 中的受囿名字替换成与 p 中的自由名字均不相同的新名字 nextSN(p),以避免发生名字冲突.

举例来说,考虑进程 $a(x).x(b).\mathbf{0}|x(c).\mathbf{0}$,当做输入动作 $a(x)$ 时,x 与 $x(c).\mathbf{0}$ 中的自由名字 x 冲突,因此,要对受囿名字 x 进行换名,设 nextSN($a(x).x(b).\mathbf{0}|x(c).\mathbf{0}$)$=n_1$,可得

$$a(x).x(b).\mathbf{0}|x(c).\mathbf{0} \xrightarrow{\text{true},a(n_1)} n_1(b).\mathbf{0}|x(c).\mathbf{0}.$$

复合进程所生成的符号迁移图中可能有一些冗余的节点.例如:

$$a(x).A(x)|a(x).A(x)|a(x).A(x) \xrightarrow{\text{true},a(n_1)} A(n_1)|a(x).A(x)|a(x).A(x),$$

$$a(x).A(x)|a(x).A(x)|a(x).A(x) \xrightarrow{\text{true},a(n_1)} a(x).A(x)|A(n_1)|a(x).A(x).$$

$A(n_1)|a(x).A(x)|a(x).A(x)=a(x).A(x)|A(n_1)|a(x).A(x)$,在图上可以合并成一个节点.目前,系统中实现了以下等价规则以减少图中的冗余节点:

$$p+q=q+p,$$

$$p+(q+r)=(p+q)+r,$$

$$\nu x p=p \quad (\text{当 } x \notin fn(p) \text{时}),$$

$$\nu x \nu y p=\nu y \nu x p,$$

$$\nu x \alpha.p=\alpha.\nu x p \quad (\text{当 } x \notin \alpha \text{ 时}),$$

$$p|q=q|p,$$

$$p|(q|r)=(p|q)|r.$$

2.3 减少实例化状态的优化算法

按照第2.1节中的算法,在对输入动作进行比较的时候,必须把输入变量实例化为当前进程的所有自由名字和一个新名字.当自由名字较多时,会导致状态数的剧烈增加.但是在某些情况下,并不是所有自由变量都需要被实例化.

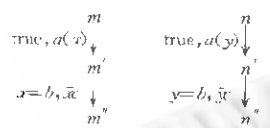

Fig. 1
图1

举例来说,考虑图1中的两个符号迁移图,按原算法,在比较 $a(x)$ 和 $a(y)$ 时,应将 x 和 y 实例化为 b,c 以及一个新名字 z.但是,因为 x 和 y 在后继边的布尔表达式($x=b,y=b$)中没有与 c 比较过,所以,当 x 和 y 取 c 和取 z 的时候,后继边中布尔表达式的值是相同的(均为假).如果当 x 和 y 实例化为 z 的时候,进程 $m'_{[z/x]}$ 与进程 $n'_{[z/y]}$ 互模拟,对于 $m'_{[z/x]}$ 的动作 $\bar{z}c$,$n'_{[z/y]}$ 有一个相应的动作 $\bar{z}c$,那么,当 x 和 y 实例化为 c 的时候,$m'_{[c/x]}$ 一定会有相应的 $\bar{c}c$ 动作,同样地,$n'_{[c/y]}$ 也有相应的 $\bar{c}c$ 动作.因此,当 x 和 y 实例化为 c 的时候,进程 $m'_{[c/x]}$ 与进程 $n'_{[c/y]}$ 也互模拟.

直观地说,当 z 为一个新名字,且名字 c 在后继边的布尔表达式中不出现时,x 和 y 实例化为 z 时如果发现后继节点互模拟,则可知实例化为 c 时后继节点也应该是互模拟的;而如果此时已经发现后继节点不互模拟,已经可以得出两个进程不互模拟的结论,就不必实例化为 c 再进行验证了.

根据以上思想,下面引入进程的比较相关集的概念.

定义4. 对符号迁移图 $\mathcal{G}$ 上的节点 n,定义其比较相关集(match relative set)为 $Rel(n)=\{x$

$x\in n(\varphi)$,φ为在$\mathcal{G}$中由n出发可达的任意边上出现的布尔表达式$\}\bigcap fn(n)$.对进程$p\equiv n_n$,定义其比较相关集为$Rel(p)=(Rel(n))\sigma$(将σ作用于$Rel(n)$中的每个元素所得到的集合).对于多个进程,记号$Rel(p_1,\ldots,p_n)$表示$\bigcup_{1\leqslant i\leqslant n} Rel(p_i)$.

某个进程的"比较相关集"是由于在图上可能参与比较,从而在实例化时必须加以考虑的名字集合.如果我们在比较输入动作时只将输入名字实例化为比较相关集中的名字,则可以得到一种新的进程间的"紧互模拟"关系,其定义如下:

定义5. 进程之间的对称关系S为紧模式互模拟,若$(p,q)\in S$时有:

(1) 如果$p\xrightarrow{a(x)}p'$,那么存在q',$q\xrightarrow{a(y)}q'$且

◇对任意$v\in Rel(p,q)$,有$(p'[v/x],q'[v/y])\in S$,

◇对某个新名字$z\notin fn(p,q)$,有$(p'[z/x],q'[z/y])\in S$;

(2) 如果$p\xrightarrow{\overline{a}(x)}p'$,那么存在$q'$,$q\xrightarrow{\overline{a}(y)}$且对某个新名字$z\notin fn(p,q)$有$(p'[z/x],q'[z/y])\in S$;

(3) 对其他动作α,如果$p\xrightarrow{\alpha}p'$,那么存在q',$q\xrightarrow{\alpha}q'$且$(p',q')\in S$.

当存在一个紧模式互模拟S使得$(p,q)\in S$时,记作$p\asymp q$.易见$\asymp$是最大的紧模式互模拟关系.

如果能够证明紧模式互模拟和模式互模拟等价,就可以根据紧模式互模拟的定义改进算法,在比较输入动作时缩小实例化的名字范围.证明中要用到几个引理.

引理1. 对布尔表达式φ及替换σ,若$b\notin n(\varphi\sigma)$,则对任意名字z,有$\sigma\models\varphi\Leftrightarrow\sigma[b/z]\models\varphi$.

证明:记$\sigma'=\sigma[b/z]$.对φ中任意一个等名测试$(x=y)$,

若$z\notin\{\sigma(x),\sigma(y)\}$,$\sigma'(x)=\sigma(x)$,$\sigma'(y)=\sigma(y)$,则$\sigma\models(x=y)\Leftrightarrow\sigma'\models(x=y)$;

否则,设$\sigma(x)=z$,则$\sigma'(x)=b$,且$\sigma\models(x=y)\Leftrightarrow\sigma(y)=z$,而$\sigma'\models(x=y)\Leftrightarrow\sigma'(y)=b$,由$b\notin(n(\varphi\sigma)$,$\sigma'(y)=b\Leftrightarrow\sigma(y)=z$,所以$\sigma\models(x=y)\Leftrightarrow\sigma'\models(x=y)$.

引理2. 设$b\notin Rel(p)$,$z\notin fn(p)$,

(1) 若$p[z/x]\xrightarrow{\alpha}p'$且$bn(\alpha)\bigcap\{b,z\}=\varnothing$,则$p[b/x]\xrightarrow{\alpha[b/z]}p'[b/z]$;

(2) 若$p[b/x]\xrightarrow{\alpha'}p''$且$bn(\alpha')\bigcap\{b,z\}=\varnothing$,则存在$\alpha$,$p'$,使$\alpha'=\alpha[b/z]$,$p''=p'[b/z]$,且$p[z/x]\xrightarrow{\alpha}p'$.

证明:见附录.

根据定义,要证明$\asymp$是一个模式互模拟,只要设法构造一个包含$\asymp$的二元关系,然后证明它是一个模式互模拟关系即可.

定义6. 定义关系族$\{S_i\mid i\in\omega\}$如下:

$$S_0=\asymp,$$

$$S_{i+1}=\{(p[b/x],q[b/y])\mid(p[z/x],q[z/y])\in S_i,b\in fn(p,q)\backslash Rel(p,q),\ z\notin fn(p,q),x\in fn(p),y\in fn(q)\}\cup S_i.$$

$$S^*=\bigcup_{i\in\omega}S_i.$$

引理3. 对于任意$n\in\omega$,S_n具有以下性质.对所有$(p,q)\in S_n$,

(1) 如果$p\xrightarrow{a(x)}p'$,则存在q',$q\xrightarrow{a(y)}q'$且

(a) 对任意$v\in Rel(p,q)$,$(p'[v/x],q'[v/y])\in S_n$;

(b) 对某个新名字$z\notin fn(p,q)$,$(p'[z/x],q'[z/y])\in S_n$;

(c) 对任意$u\in fn(p,q)\backslash Rel(p,q)$,$(p'[u/x],q'[u/y])\in S_{n-1}$.

(2) 如果 $p \xrightarrow{\overline{a}(x)} p'$,则存在 q',$q \xrightarrow{\overline{a}(y)} q'$ 且对某个新名字 $z \notin fn(p,q)$,$(p'\{z/x\},q'\{z/y\}) \in S_n$.

(3) 对其他动作 α,如果 $p \xrightarrow{\alpha} p'$,则存在 q',$q \xrightarrow{\alpha} q'$,且 $(p',q') \in S_n$.

证明:见附录.

由以上引理容易证明,模式互模拟和紧模式互模拟是等价的.

定理1. $p \asymp q$ 当且仅当 $p \times q$.

证明:由 $\times$ 和 $\asymp$ 的定义可知,$\asymp \subseteq \times$;又由引理3可知,$\times \subseteq S^* \subseteq \asymp$. 故命题得证. □

计算比较相关集时需要考虑符号迁移图中所出现的环,当图的结构比较复杂时(例如,存在多个互相嵌套的环),生成比较相关集可能需要耗费较长的时间. 目前,在我们的系统实现中没有采用对每个节点计算比较相关集的办法,而是在进行验证前对两个符号迁移图做一次扫描,把两个图所有边的布尔表达式中出现的名字放入集合 ComparedNames. 由于对这两个图上的任意节点 n,一定有 $Rel(n) \subseteq ComparedNames$,因此可以保证算法正确.

根据紧模式互模拟的定义,对原互模拟验证算法中比较输入动作的部分修改如下:

$\text{match}_{(a(x),a(y))}(p \text{ as } (m,\sigma), q \text{ as } (n,(\sigma')))=$

for each $p \xrightarrow{a(x)} p_i, q \xrightarrow{a(y)} q_j$ {

$S=((fn(p) \cap (ComparedNames\ \sigma)) \cup (fn(q) \cap (ComparedNames\ \sigma'))) \cup \{nextSN(p,q)\}$

for each $v \in S$ $b_{ij}^v = \text{close}\ (p_i\{v/x\}, q_j\{v/y\})$

$b_{ij} = \bigwedge_{v \in S} b_{ij}^v$

return $((\bigwedge_i (\bigvee_j b_{ij})) \wedge (\bigwedge_j (\bigvee_i b_{ij})))$

以上介绍的是强互模拟验证算法. 对于弱互模拟,与此不同的只是要将某些单箭头迁移改为双箭头迁移. 我们的优化算法同样适用于弱互模拟的情形.

3 实例及性能分析

下面给出4个测试例子的运行结果. 前3个例子来自 MWB(Polyadic Version 3.122)[3],第4个例子是我们自己编写的用 π-演算进程模拟缓存数据结构的测试例子. 以下例子的测试环境均为 SPARC-1000E 服务器.

在表1中,MWB 验证的是两个 π-演算进程是否为开互模拟,而我们的系统验证的是两个 π-演算进程是否为迟(或早)互模拟. 在第4个例子中,当模拟查找缓存中的数据时,如果输入的名字和已存储的名字都不相同,则需要给出"查找失败"的答复;由于所存储的名字是前面通过输入动作输入的,无法穷举出所有可能的输入名字,因而这种情形只能用 if...then...else... 句式来描述. 在我们的系统中,if...then...else... 可表示为 $\varphi p + \neg\varphi p$. 而 MWB 只能验证开互模拟[5],没有不等名测试,if...then...else... 句式在 MWB 中无法表示,所以,这个例子在 MWB 中无法进行验证.

从表中可以看到,系统中所实现的优化算法大大减少了互模拟所需的时间和空间,有效地提高了验证的效率.

4 结束语

本文所介绍的对 π-演算进程进行互模拟验证的工具,可以有效地验证进程之间是否存在迟互模拟关系. 系统中所实现的优化算法,能够提高某些例子中互模拟验证的效率. 文献[1]中还给出了

验证早互模拟的算法,对于紧模式互模拟同样有其"早"的版本,并已在目前的系统中实现,限于篇幅,不再赘述.

Table 1 The results of four benchmark examples

表1 4个测试例子的验证效果对照表

	MWB		Our Alg. (before opt.)①		Our Alg. (after opt.)②	
	Time (s)③	Number of states④	Time (s)	Number of states	Time (s)	Number of states
Minihandover	2.11	22	0.41	20	0.36	20
Bool	0.24	21	0.97	215	0.77	18
Handover	202.45	323	270.06	10 350	35.10	465
Cache	—	—	6.27	4 354	0.48	37

①本系统(优化前),②本系统(优化后),③时间(秒),④状态数.

由于符号迁移图中的一个节点可能对应于实例化后的多个状态,在符号迁移图上可以进行一些有效的优化.文献[8]中提供了针对传值CCS互模拟验证算法的优化算法,其中与赋值无关的优化方法可推广到π-演算中.目前系统中只实现了最简单的(弱互模拟验证时的)窥孔优化.我们将在对系统的进一步完善时实现其他优化方法.

此外,在实现换名和判断两个项之间是否相等这些问题上还有其他实现方法可供选择.不同的实现方案将影响最终生成的符号迁移图的大小,从而直接影响整个互模拟验证的效率.这也是将来我们改进系统时需要仔细研究的一个问题.

References:

[1] Lin, H. Computing bisimulations for finite-control π-Calculus. Journal of Computer Science and Technology, 2000,15(1): 1~9.

[2] Milner, R., Parrow, J., Walker, D. A calculus of mobile processes, Part I, II. Information and Computation, 1992,100 (1),1~77.

[3] Victor, B. A Verification Tool for the Polyadic π-Calculus [Licentiate Thesis]. Sweden: Department of Computer Systems, Uppsala University, 1994.

[4] Victor, B., Moller, F. The mobility workbench——a tool of the π-Calculus. In: Dill, D. ed. Proceedings of CAV'94. Lecture Notes in Computer Science, Vol 818. Berlin: Springer-Verlag, 1994. 428~440.

[5] Sangiorgi, D. A theory of bisimulation for the π-Calculus. Acta Informatica, 1996,33(1):69~97.

[6] Hennessy, M., Lin, H. Symbolic bisimulations. Theoretical Computer Science, 1995,138(2):353~389.

[7] Lin, H. "On-the-fly Instantiation" of value-passing processes. In: Formal Description Techniques and Protocol Specification, Testing and Verification. Paris: Kluwer Academic Publishers, 1998. 215~230.

[8] Fang, Hai, Xu, Wen, Lin, Hui-min. A local optimization algorithm for symbolic transition graphs with assignment. Computer Research and Development, 2000,37(1):95~101 (in Chinese).

附中文参考文献:

[8] 方海,许文,林惠民.带赋值符号迁移图的局部优化算法.计算机研究与发展,2000,37(1):95~101.

附录 两个引理的证明

引理2的证明:

(1) 若 $p[z/x]\xrightarrow{\alpha}p'$,则由符号迁移图的(迟)操作语义,在符号迁移图上存在一条边 $m\xrightarrow{\varphi,\beta}n$,且 $m\sigma=p[z/x]$, $\sigma\models\varphi$, $\beta\sigma=\alpha$, $n\sigma=p'$, $\sigma(x)=z$.

取 σ' 为 $\sigma\{x\mapsto b\}$,因为 $z\notin Range(\sigma')$,有 $\sigma'=\sigma[b/z]$.

由 $b\notin Rel(p)$ 可知 $b\notin Rel(p[z/x])$,从而有 $b\notin n(\varphi\sigma)$,由引理1得 $\sigma'\models\varphi$,

由符号迁移图的(迟)操作语义得 $m\sigma'\xrightarrow{\beta\sigma'}n\sigma'$.

由 $m\sigma'=m(\sigma\{x\mapsto b\})=p[b/x]$,取 α' 为 $\beta\alpha'$,则 $\alpha'=\beta\sigma[b/z]=\alpha[b/z]$,

$n\sigma'=n(\sigma[b/z])=p'[b/z]$,命题得证.

(2) 若 $p[b/x] \xrightarrow{\varphi} p''$,同样,由符号迁移图的(迟)操作语义,可得符号迁移图上对应的边 $m \xrightarrow{\varphi,\beta} n$,且 $m\sigma = p[b/x], \sigma \models \varphi, \beta\sigma = \alpha', n\sigma = p'', \sigma(x) = b$.

取 σ' 为 $\sigma\{x \mapsto z\}$,同上,有 $\sigma = \sigma'[b/z], \sigma' \models \varphi, m\sigma' \xrightarrow{\beta\sigma'} n\sigma'$.

由 $m\sigma' = m(\sigma\{x \mapsto z\}) = p[z/x]$,取 α 为 $\beta\sigma'$,则 $\alpha' = \alpha[b/z]$,取 p' 为 $n\sigma'$,$p'' = p'[b/z]$,命题得证.

引理3的证明:

对下标 n 进行归纳证明.

$n=0$时,对任意$(p,q) \in S_1 = \sim$,由~的定义可知,对于引理3中,除了(1)中的(c)以外的所有情形,命题均成立;

对于引理3的(1)中的(c)这种情形,任取 $u \in fn(p,q) \backslash Rel(p,q)$,由引理3的(1)中的(b)和 S_1 的定义可知,$(p'[u/x], q'[u/y]) \in S_1$. 因此,$n=0$时命题成立.

假设 $n=k$ 时命题成立,考虑 $n=k+1$时的情形:

任取 S_{k+1} 中的元素,如果它属于 S_k,由归纳假设可知命题成立;否则,由定义可知其形如$(p[b/x], q[b/y])$.

对于引理3的(1)中的(a)这种情形,首先通过 α 变换使输入动作中的受囿名字不与 z 以及进程的其他自由名字冲突. 若 $p[b/x] \xrightarrow{\alpha} p''$,由于 S_{k+1} 定义中的 z 满足 $z \notin fn(p,q)$,又 $b \notin Rel(p)$,由引理2的(2),存在 α 和 p',使 $a(u) = \alpha[b/z], p'' = p'[b/z]$,且 $p[z/x] \xrightarrow{\alpha} p'$;由$(p[z/x], q[z/y]) \in S_k$,由归纳假设,存在 q',$q[z/y] \xrightarrow{\alpha} q'$;

由引理2的(1),得 $q[b/y] \xrightarrow{\alpha[b/z]} q'[b/z]$,即 $q[b/y] \xrightarrow{\alpha(w)} q'[b/z]$,

由$(p[z/x], q[z/y]) \in S_k$,对任意 $v \in Rel(p[z/x], q[z/y])$,$(p'[v/u], q'[v/w]) \notin S_k$;取 $z' \notin fn(p', q') \cup \{b, z, u, v, w\}$,$(p'[v/u], q'[v/w]) = (p'[z'/z][v/u][z/z'], q'[z'/z][v/w][z/z'])$,令 $p_{new} = p'[z'/z][v/u], q_{new} = q'[z'/z][v/w]$,$(p_{new}[z/z'], q_{new}[z/z']) = (p'[v/u], q'[v/w]) \in S_k$;显然,对任意 $v \in Rel(p[b/x], q[b/y]) \backslash \{b\}$,有 $v \in Rel(p[z/x], q[z/y])$;

由 $b \notin Rel(p[z/x])$可得 $b \notin Rel(p_{new})$,同样,$b \notin Rel(q_{new})$. 同时,显然有 $z \notin fn(p_{new}, q_{new})$. 若 $z \in fn(p', q')$,$z' \in fn(p_{new}, q_{new})$,则由定义,$(p_{new}[b/z'], q_{new}[b/z']) = (p'[b/z][v/u], q'[b/z][v/w]) = (p''[v/u], q''[v/w]) \in S_{k+1}$.

(当 $v=b$ 时,$(p'[z/u], q'[z/w]) = (p'[z'/z][z'/u][z/z'], q'[z'/z][z'/w][z/z'])$有相同结论;如果 $z \notin fn(p', q')$,那么 $p'' = p', q'' = q'$,由$(p'[v/u], q'[v/w]) \in S_k$,也有$(p''[v/u], q''[v/w]) \in S_k \subseteq S_{k+1}$的结论)

对于引理3的(1)中的(b)和引理3的(2)、(3)这3种情形,可类似加以证明.

对于引理1的(c)的情形,由于引理3的(1)中的(b)成立,存在 $z' \notin fn(p[b/x], q[b/y])$,$(p''[z'/u], q''[z'/w]) \in S_{k+1}$,由 S_{k+2}的定义可知,引理3的(1)中的(c)成立.

Optimization and Implementation of a Bisimulation Checking Algorithm for the π-Calculus*

XU Wen, FANG Hai, LIN Hui-min

(*Laboratory for Computer Science, Institute of Software, The Chinese Academy of Sciences, Beijing* 100080; *China*)

E-mail: lhm@ox.ios.ac.cn

http://www.jos.ac.cn/openlab

Abstract: Bisimulation is adopted in the π-calculus as the criterion to identify processes. For finite-state processes, bisimulation equivalence is decidable. In this paper, an optimization technique for a bisimulation checking algorithm is presented. It works by instantiating input names only to those free names which are used in subsequent matches. The technique can significantly reduce the required time and space, as illustrated by several benchmark examples. The correctness of the optimization is also proven.

Key words: process algebra; π-calculus; bisimulation; verification algorithm

* Received May 11, 2000; accepted November 27, 2000

Supported by the National Natural Science Foundation of China under Grant No. 69833020