基于下推系统可达性分析的输出信道信息流检测

孙 聪 唐礼勇 陈 钟

(北京大学信息科学技术学院软件研究所 北京 100871) (高可信软件技术教育部重点实验室 北京 100871)

摘 要 提出了一种对含输出信道的命令式语言进行信息流安全性分析的方法。将程序抽象为下推系统,通过自合成将不干涉性转化为安全性属性,将两次相关执行中向输出信道的输出操作分别抽象为由下推规则表示的存储和匹配操作,通过对标错状态的可达性分析验证程序是否满足终止不敏感不干涉性。演化后的方法支持程序的发散执行,通过上界回退算法找到强制终止首次执行所需的最大输出信道上界。实验说明该方法与现有工作相比具有更高的精确性和验证效率。

关键词 信息流,不干涉性,下推系统,可达性分析

中图法分类号 TP311 **文献标识码** A

Checking Information Flow on Output Channel with Reachability Analysis of Pushdown System

SUN Cong TANG Li-yong CHEN Zhong

(Institute of Software, School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China)

(Key Laboratory of High Confidence Software Technologies, Ministry of Education, Beijing 100871, China)

Abstract We proposed an approach for analyzing information-flow security of imperative language with output channels. The program is abstracted with pushdown system, which is then self-composed in order to adapt noninterference as a safety property. The output operations in the two relevant runs are respectively modeled as storing and matching procedure by pushdown rules. Then the termination-insensitive noninterference is verified by a reachability analysis of illegal-flow state. A variation of this approach can deal with program containing divergent run. An upper-bound regression algorithm was proposed to find the maximum upper-bound in order to trigger coercive termination of divergent run. The experimental results show that the approach is more precise and efficient than existing work.

Keywords Information flow, Noninterference, Pushdown system, Reachability analysis

1 引言

在程序语言信息流安全研究中,输出信道作为程序运行时的数据边界常为研究者特别关注。终止不敏感不干涉性(Termination-Insensitive Noninterference, TINI)的一般定义[1,2]可简述为

$(LH_1,P)\Downarrow(L'H_1')\wedge(LH_2,P)\Downarrow(L''H_2'')\Rightarrow(L'=L'')$

式中,L 和 H 分别表示程序 P 的低安全级输入输出和高安全级输入输出。该定义并未考虑程序的执行不终止的情况。当两次相关执行中的一次不终止时,命题前件不成立,故不会对终态的低安全级部分是否可分辨作出判断。批处理模型(Batch-job Model)指在执行前获得输入并在执行终止后生成输出的系统模型,其特点在于执行过程中无中间输入输出。上述 TINI 定义因假定攻击者只能观察到程序终态的低安全级计算结果,故又称批处理终止不敏感不干涉性(BTINI)。BTINI 只允许程序终止行为引起的信息泄露,亦即满足 BTINI 的程序实际上最多泄露一位信息,即程序是否终止。

当程序包含输出信道时,输出信道所输出的信息成为攻击者能观察到的程序的中间结果。程序不再满足批处理模型特点。相应地,BTINI 定义面临以下问题:首先,引入输出信道可导致实际泄露的信息多于一位,如对以下程序

```
for(i=0;i<h;i++) output(i)
```

攻击者观察输出信道可获得 h 的具体值而不仅仅是一位终止性信息。其次,当程序存在发散执行(divergent run)时,不终止的执行使得 BTINI 无法判定终态下低安全级不可分辨性,因而无法将程序规约为不安全。如对以下程序

```
output(h);while(true)
```

BTINI 定义无法将其判定为不安全,而实际上该程序已将高安全级变量信息完全泄露给攻击者。针对 BTINI 定义存在的这两个问题,本文工作讨论如下两个问题:

1)在程序的任一执行终止的前提下,根据输出信道的特点,定义区别于 BTINI 的终止不敏感不干涉性,给出如何使用下推系统可达性分析验证新的终止不敏感不干涉性。

2)当程序存在发散执行时不干涉性定义及验证方法的演化。

依照惯例,我们假定所有输出信道均为低安全级信道,即

到稿日期:2010-08-13 返修日期:2010-11-15 本文受国家自然科学基金(60773163,60911140102)资助。

孙 聪(1982—),男,博士生,主要研究方向为基于程序语言的安全、软件模型检测,E-mail:suncong.pku@gmail.com;唐礼勇(1972—),男,博士,副研究员,主要研究方向为信息安全、系统软件;陈 钟(1963—),男,博士,教授,博士生导师,CCF 高级会员,主要研究方向为信息安全、系统软件与嵌入式系统、软件工程。

攻击者可以获得信道上的任何信息。为简化起见，假设攻击者只通过输出信道获得程序中的低安全级变量值。

当前与确定性语言顺序程序中输出信道相关的信息流安全分析参见文献[4-7]。基于自动机的动态信息流监测[5]未考虑程序的发散执行，且动态方法会增加程序的运行时开销，而其他自动机模型[8]则无法直接应用于程序语言。其他方法均为静态分析，文献[4,7]基于对类型系统[3]的改进，这类方法具有一般类型系统的保守性且非流敏感[9]。基于抽象解释的方法[6]将变量值抽象为安全级并将程序执行抽象为安全级计算，此方法虽较类型系统更精确，但对如下程序(输入信道 a 为高安全级且输出信道 b 为低安全级)

$a?\ x;y=x-x;b!\ y$

仍会误判为不安全，且该方法亦未考虑发散执行的问题。相比而言，本文给出的基于下推系统可达性分析的方法是流敏感的，且考虑了发散执行。实验说明本文工具较文献[6]工具[12]更精确且验证效率更高。

2 语义描述及 TINI 演化

为沿用此前工作[14]中的命令式语言语义，首先定义用于存储信道输出的语义结构。在此使用全局数组 O 依次存储程序向低安全级信道的输出，使用全局变量 p 指向下一个信道输出应存放的数组位置(p 初始化为 0)。这时文献[14]图 2 中的 μ 实际包含两部分，一部分是原语义中的 μ，另一部分是(O,p)。因为文献[14]图 2 中的语义规则不涉及输出信道，故迁移前后(O,p)保持不变。而本文方法将 output 看作特殊的过程调用，output 对应的语义规则形如

$$\frac{((O,p)\bigcup\mu,\lambda,e)\Downarrow v \quad ((O,p)\bigcup\mu,\lambda\uplus[l:=v],[l/x]S_{out})\Downarrow((O',p')\bigcup\mu,\lambda)}{((O,p)\bigcup\mu,\lambda,(output(x)S_{out})(e))\Downarrow((O',p')\bigcup\mu,\lambda)}$$

式中，过程体 S_{out} 向输出信道(O,p)中记录输出信息。

下面根据输出信道特点对终止不敏感不干涉性作出定义。本节及第 3 节均假定程序的任一执行终止。非形式化地讲，终止不敏感不干涉性应该由“高安全级初态变量的变化不会导致低安全级终态变量的变化”转变为“高安全级初态变量的变化不会导致输出信道上的输出序列发生变化”。在此首先定义特定低安全级输入所对应的观察结果。

定义 1(观察结果) 程序 P 在低安全级输入 L 下的观察结果 $Obs(P,L)$定义为

$Obs(P,L)=\{(O',p')\mid(LH\bigcup\{O,0\},P)\Downarrow(L'H'\bigcup\{O',p'\})\}$

使用观察结果的等价性定义终止不敏感不干涉性如下。

定义 2(TINI) 程序 P 满足终止不敏感不干涉性，若对任意 L 有$(O_1,p_1)\in Obs(P,L)\wedge(O_2,p_2)\in Obs(P,L)\Rightarrow(O_1,p_1)=(O_2,p_2)$，其中$(O_1,p_1)=(O_2,p_2)$当且仅当 $p_1=p_2(\geqslant 0)\wedge\forall 0\leqslant i<p_1, O_1[i]=O_2[i]$。

以下具体介绍 TINI 的验证方法。

3 基于存储-匹配模式的可达性分析

此前工作[14,15]已介绍了通过各种自合成方法将不干涉性转化为安全性属性。文献[15]还给出了如何将对安全性属性的验证规约为抽象模型上的可达性分析，其基本思想是将模型构造过程分为 3 部分：(a)基本自合成；(b)初始交叉赋值；(c)标错状态构造。以程序 $l:=h$ 为例，自合成结果形如

$l:=l'$ (a)

$l:=h;l':=h'$ (b)

if $l!=l'$ then goto IFL (c)

IFL 可达则程序不满足 BTINI。本文延续可达性分析的基本思想，以紧凑自合成[14](Compact Self-Composition)作为基本自合成。由于单次执行的所有输出之间不通过存储位置发生关系，输出在时间上具有单向性，而定义 2 的 TINI 所要求的对输出序列的比较又具有一次性，因而在模拟程序执行时，第二次执行只需将输出序列按元素依次与第一次执行所记录的输出序列相比较即可，而不需要再用额外的空间记录第二次执行的输出序列。简言之，即“一次存储二次匹配”。这种非对称的处理方式使得对于 output 的过程体 S_{out}，不能简单地采取抽象后进行紧凑自合成的方式，文献[14]图 5 中的下推规则生成过程 $\Phi((output(x)S_{out})(e),n_i,n_j,R)$此时应变为$\{\langle n_i\rangle\rightarrow\langle n_p n_j\rangle\mid\rho_p=\rho_i\wedge\eta_p[x]=(\rho_i,\eta_i)[e]\wedge\eta_j=\eta_i\}\bigcup\{\langle n_q\rangle\rightarrow\langle\varepsilon\rangle\mid\rho_q=\rho_j\}$，其中 n_p 和 n_q 分别为 S_{out} 的入口点和出口点。注意，S_{out}无需归纳调用 $\Phi(S_{out},n_p,n_q,R)$生成其对应的下推规则，而只需将分别模拟存储和匹配操作的下推规则直接加入紧凑自合成结果中。表 2 给出了存储和匹配分别对应的下推规则以及这些下推规则实际模拟的 Remopla[10] 语句。

表 1 程序抽象及自合成结果示例

int l,h;	q⟨main⟩−−>q⟨DEFAULT_1_⟩
init main;	(x′=l+3&l′=l&h′=h&y′=y&z′=z)
module void main(){	q⟨DEFAULT_1_⟩−−>q⟨DEFAULT_2_⟩
int x,y,z;	(x>10&y′=h&l′=l&h′=h&x′=x&z′=z)
x=l+3;	q⟨DEFAULT_2_⟩−−>q⟨output DEFAULT_3_⟩
if	(x′=x&l′=l&h′=h&x″=x&y″=y&z″=z)
::(x>10)−>y=h;	q⟨DEFAULT_3_⟩−−>q⟨output DEFAULT_4_⟩
output(x);	(x′=y&l′=l&h′=h&x″=x&y″=y&z″=z)
output(y);	q⟨DEFAULT_4_⟩−−>q⟨DEFAULT_5_⟩
if	(h!=0&z′=0&l′=l&h′=h&x′=x&y′=y)
::h!=0−>z=0;	q⟨DEFAULT_5_⟩−−>q⟨output DEFAULT_6_⟩
output(x);	(x′=x&l′=l&h′=h&x″=x&y″=y&z″=z)
::else−>x=1;	q⟨DEFAULT_4_⟩−−>q⟨DEFAULT_6_⟩
fi	(!(h!=0)&x′=1&l′=l&h′=h&y′=y&z′=z)
::else−>skip;	q⟨DEFAULT_1_⟩−−>q⟨DEFAULT_6_⟩
fi	(!(x>10)&l′=l&h′=h&x′=x&y′=y&z′=z)
}	q⟨DEFAULT_6_⟩−−>q⟨⟩ (l′=l&h′=h)

q⟨main⟩−−>q⟨DEFAULT_1_⟩(x′=l+3&l′=l&h′=h&y′=y&z′=z&lt′=lt&ht′=ht&(Ai(0,7)(O′[i]=O[i]))&p′=p)

q⟨maint⟩−−>q⟨DEFAULT_1_t⟩(x′=lt+3&l′=lt&ht′=ht&y′=y&z′=z&l′=l&h′=h&(Ai(0,7)(O′[i]=O[i]))&p′=p)

……

q⟨DEFAULT_6_⟩−−>q⟨maint⟩

(l′=l&h′=h&lt′=lt&ht′=ht&(Ai(0,7)(O′[i]=O[i]))&p′=0)

q⟨DEFAULT_6_t⟩−−>q⟨DEFAULT_6_t⟩

(lt′=lt&ht′=ht&l′=l&h′=h&(Ai(0,7)(O′[i]=O[i]))&p′=p)

q⟨output⟩−−>q⟨output1⟩(O′[p]=x&p′=p+1&(Ai(0,7)((i=p|O′[i]=O[i]))&x′=x&l′=l&h′=h&lt′=lt&ht′=ht)

q⟨output1⟩−−>q⟨⟩

((Ai(0,7)(O′[i]=O[i]))&p′=p&l′=l&h′=h&lt′=lt&ht′=ht)

q⟨outputt⟩−−>q⟨IFL⟩

(O[p]!=x&(Ai(0,7)(O′[i]=O[i]))&p′=p&l′=l&h′=h&lt′=lt&ht′=ht)

q⟨outputt⟩−−>q⟨outputt1⟩(!(O[p]!=x)&p′=p+1&(Ai(0,7)(O′[i]=O[i]))&x′=x&l′=l&h′=h&lt′=lt&ht′=ht)

q⟨outputt1⟩−−>q⟨⟩

((Ai(0,7)(O′[i]=O[i]))&p′=p&l′=l&h′=h&lt′=lt&ht′=ht)

q⟨tmp_start⟩−−>q⟨main⟩(x′=x&l′=l&h′=h&y′=y&z′=z&lt′=l&ht′=ht&(Ai(0,7)(O′[i]=O[i]))&p′=0)

q⟨IFL⟩−−>q⟨IFL⟩

((Ai(0,7)(O′[i]=O[i]))&p′=p&l′=l&h′=h&lt′=lt&ht′=ht)

表 1[上左]给出了 Remopla 实现的文献[5]表 1 的例子，其抽象结果见表 1[上右]。在抽象结果基础上使用紧凑自合成得到表 1[中上]的基本自合成结果(以第一条和最后一条下推规则为例)。然后向其中加入存储-匹配下推规则(表 1[中下])，进行初始状态下的交叉赋值($lt'=l$)并构造标错状态对应的下推规则(表 1[下])。注意，非存储-匹配下推规则都考虑了作为全局变量的(O,p)，而存储-匹配下推规则都考虑了全局变量及其伙伴全局变量。两次执行开始处 p 均置 0。

表 2　存储-匹配下推规则

Remopla	PDS Rules
O[p]=x,p=p+1;	q⟨output⟩−−>q⟨output1⟩　(O′[p]=x&p′=p+1&(Ai(0,7)((i=p|O′[i]=O[i])))&x′=x)
	q⟨output1⟩ −−> q⟨⟩((A i (0,7) (O′[i]=O[i])) & p′=p)
if	q⟨outputt⟩−−>q⟨IFL⟩(O[p]! =x&(Ai(0,7)
:: O[p]! =x	(O′[i]=O[i])) & p′=p)
−> goto IFL;	q ⟨outputt⟩ −−> q ⟨outputt1⟩(! (O[p]! =x) & p′=p+1 &
::else−>p=p+1;	(Ai(0,7)(O′[i]=O[i]))&x′=x)
fi	q⟨outputt1⟩−−>q⟨⟩((Ai(0,7)(O′[i]=O[i]))&p′=p)

Moped 模型检测器[11]的输出结果是标错状态 IFL 的一个可达路径。对于第 1 节中的第一个例子程序，存储-匹配过程会找到一个不确定值(在第一次执行所存储的输出值以外，即 O[h]内容)与一个确定值(由第二次执行所输出的待匹配值)之间的不等关系并到达标错状态，如图 1 所示。

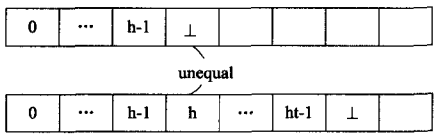

图 1　标错状态的触发

4　上界回退基础上的存储-匹配

定义 1 要求确定性程序的任一执行终止，而当存在发散执行时，此定义不再适用。本节首先考虑含发散执行的程序对应的 TINI 定义，然后给出一种改进的存储-匹配方法，此方法借助一种上界回退算法找到可能的最长输出长度，以此为依据终止抽象程序的发散执行并验证标错状态是否可达。

当存在发散执行时，观察结果定义不能再使用大步语义。假定$(M,P)\rightarrow\perp$表示没有语义规则可以应用于程序 P，观察结果改进定义如下：

定义 3(观察结果′)　程序 P 在低安全级输入 L 下的观察结果 $Obs(P,L)$ 定义为 $Obs(P,L)=\{(O',p')\mid(LH\cup\{O,0\},P)\rightarrow^*(L'H'\cup\{O',p'\},P'),P'=\varepsilon\vee(L'H'\cup\{O',p'\},P')\rightarrow\perp\}$。

在定义 3 基础上仍可用定义 2 作为 TINI 定义，但上节所述的可达性分析方法变得不可用。因为对于任一执行总发散的程序，其抽象模型的符号执行无法进入第二次执行，因而无法匹配输出以确定是否进入标错状态。由于在实际攻击中无限长输出可由足够长的有限长输出近似表示，因而假定任一执行的输出均为有限长并首先定义限 N 观察结果如下：

定义 4(限 N 观察结果)　程序 P 在低安全级输入 L 下的限 N 观察结果 $Obs(P,L,N)$ 定义为 $Obs(P,L,N)=\{(O',p')\mid(LH\cup\{O,0\},P)\rightarrow^*(L'H'\cup\{O',p'\},P'),(p'=N)\vee((P'=\varepsilon\vee(L'H'\cup\{O',p'\},P')\rightarrow\perp)\wedge p'\leqslant N)\}$。

由假定总能找到足够大的 N 使得 $Obs(P,L)=Obs(P,L,N)$。基于限 N 观察结果的 TINI 定义如下：

定义 5(TINI′)　程序 P 满足终止不敏感不干涉性，若对任意 L 和 N 有$(O_1,p_1)\in Obs(P,L,N)\wedge(O_2,p_2)\in Obs(P,L,N)\Rightarrow(O_1,p_1)=_N(O_2,p_2)$，其中$(O_1,p_1)=_N(O_2,p_2)$当且仅当$0\leqslant p_1=p_2\leqslant N\wedge\forall 0\leqslant i<p_1,O_1[i]=O_2[i]$。

由于 TINI′定义涉及程序执行是否收敛/终止，使用可达性分析时需考虑：

1)保证抽象模型在模拟程序执行时，能成功终止两次相关执行中的第一次执行(即任一次执行，无论其是否发散)；

2)在终止发散执行时，被终止的执行与原发散执行从 TINI′的角度看效果应相同。

使程序进入第二次执行的基本思想是，变换抽象模型中有限长队列 O 的长度上限，使得第一次执行向输出信道的输出总能到达此上限，并在到达此上限时显式地终止第一次执行并进入第二次执行。这一思想还可用于攻击者只关心前 N 个输出的情况，此攻击者模型可用于某些时间相关的攻击。

将两次相关执行中的第一次执行称为目标执行，并假定其输出的长度为 N_0，我们希望该目标执行在向输出信道写入第 N_0 个输出后即终止而转入第二次执行。由定义 4 可知，任一$(O,p)\in Obs(P,L,N)$实际上都对应一次执行或初始 H 不同的多次执行。而在 $Obs(P,L,N_0)$中可以找到目标执行的输出结果(O_0,N_0)，使得$(LH\cup\{O,0\},P)\rightarrow^*(L'H'\cup\{O_0,N_0\},P')$，其中$P'=\varepsilon\vee(L'H'\cup\{O_0,N_0\},P')\rightarrow\perp$。根据 TINI′要求，从与目标执行的初态低安全级等价的任一初态起始的执行也必须有 N_0 个输出，且有$(O_0,N_0)=_{N_0}(O',N_0)$。假定程序的两次相关执行总有第一次执行的输出长度大于等于第二次执行的输出长度(因为两次相关执行具有对称性，这一假定平凡)，即第二次执行的输出长度 $N_1\leqslant N_0$，则第二次执行中匹配过程可能存在以下 3 种情况：

1)$N_1<N_0$(无论第二次执行是否发散)；

2)$N_1=N_0$ 但$(O_0,N_0)\neq_{N_0}(O_1,N_1)$；

3)$N_1=N_0$ 且$(O_0,N_0)=_{N_0}(O_1,N_1)$。

前两种情况导致违反 TINI′。因 N_0 与特定的执行相关，故只验证某一个 N_0 是不够的。易知，随 N_0 的减小 $Obs(P,L,N_0)$中单个(O,N_0)对应的执行数增加，且增加的执行亦须按上述 3 种情况判断是否属于 TINI′，故当程序在任意 L 下的所有执行的最长输出长度为 N_{max} 时，需对所有 N_0($0\leqslant N_0\leqslant N_{max}$)验证是否有第二次执行进入上述 3 种情况的前两种情况。因此具体做法分两步：1)找到 N_{max}；2)通过可达性分析验证每个 N_0 下是否进入前两种情况。

寻找 N_{max}使用上界回退算法。将初始的输出信道长度 Max 取为足够大，以使程序在任意 L 下的所有执行的输出长度均小于 Max。若每次验证出的 Max 都使程序末尾标识不可达，则将 Max 置为$\frac{Max}{2}$，直到$\frac{Max}{2^k}$使得程序末尾标识首次

可达，这时$\frac{Max}{2^k} \leqslant N_{max} < \frac{Max}{2^{k-1}}$。再顺次查找$\frac{Max}{2^k}$，$\frac{Max}{2^k}+1$，$\frac{Max}{2^k}+2$，…，直至找到$N_{max}$，如图2所示。

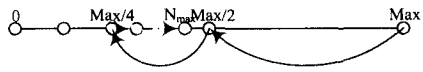

图2　输出信道上界回退

考虑以下违反TINI′的发散程序：

for(i=0;i<h;i++) output(i);

while(true);

假定抽象模型中变量的位数为3(这时变量取值范围为$0\sim2^3-1$)，初始Max值为50，这时若存储操作类似表5中output所解析出的下推规则，则得到$N_{max}=7$所需的时间$T_{MAX}=t_{50}+t_{25}+t_{12}+t_6+t_7+t_8$，详见表3。

表3　回退所需时间

Max	50	25	12	6	7	8
Time($\times10^{-3}$s)	3.81	2.77	2.30	4.19	5.39	2.13
END可达性	×	×	×	√	√	×

得到N_{max}后，即可将从1到N_{max}的每个N_0看作输出信道的实际长度(亦即表5中Remopla代码中的MAX)。为便于说明，将抽象模型的自合成结果也还原成Remopla程序。基本自合成结果见表5[左]，表5[右]的outputt为第二次执行中的实际匹配过程。为了让收敛的程序也可通过上界回退得到正确的MAX，定义状态NOEND；为匹配本节前述的3种情况，定义状态SUCC和IFL。SUCC对应于情况3；outputt中的IFL对应于$N_1 \leqslant N_0 \wedge (O_0,N_1) \neq_{N_1} (O_1,N_1)$，包含了情况1的一部分及情况2；对情况1的余下部分($N_1<N_0 \wedge (O_0,N_1)=_{N_1}(O_1,N_1)$)，需对基本自合成结果的第二次执行部分做以下更改：执行结束处转向IFL状态；每次程序发散前转向IFL状态(见表5[左]代码第19，23行)。这样即可保证情况1的余下部分进入IFL状态。验证TINI′是为了对MAX=1～N_{max}验证IFL状态均不可达。若某个MAX使得IFL状态可达，则违反TINI′，验证终止。表4给出对应于存储-匹配过程的下推规则。再考虑文献[7]中的Program 2，当变量位数为3时，MAX值也会回退到7并得到与上例相同的验证结果。而基于类型系统的方法[7]会将此程序误判为信息流安全，由此体现出本方法相比基于类型系统的方法具有更高的精确性。

表4　存储－匹配下推规则演化

q⟨output⟩－－>q⟨output1⟩　(O′[p]=x&p′=p+1&(Ai(0,6)((i=p|O′[i]=O[i]))&…&x′=x)

q⟨output1⟩－－>q⟨END⟩　(p=7&(Ai(0,6)(O′[i]=O[i]))&p′=p&…)

q⟨output1⟩－－>q⟨output2⟩　(! (p=7)&(Ai(0,6)(O′[i]=O[i]))&p′=p&…&x′=x)

q⟨output2⟩－－>q⟨⟩　((Ai(0,6)(O′[i]=O[i]))&p′=p&…)

q⟨outputt⟩－－>q⟨IFL⟩　(O[p]! =x&(Ai(0,6)(O′[i]=O[i]))&p′=p&…)

q⟨outputt⟩－－>q⟨outputt1⟩　(! (O[p]! =x)&p′=p+1&(Ai(0,6)(O′[i]=O[i]))&…&x′=x)

q⟨outputt1⟩－－>q⟨SUCC⟩　(p=7&(Ai(0,6)(O′[i]=O[i]))&p′=p&…)

q⟨outputt1⟩－－>q⟨outputt2⟩　(! (p=7)&(Ai(0,6)(O′[i]=O[i]))&p′=p&…&x′=x)

q⟨outputt2⟩－－>q⟨⟩　((Ai(0,6)(O′[i]=O[i]))&p′=p&…)

表5　还原为Remopla的自合成结果

```
1 module void main(){                    module void output(int x){
2     int i;                                 O[p]=x,p=p+1;
3     p=0,i=0;                               if
4     do                                     :: (p==MAX) -> goto END;
5     :: (i<h) -> output(i);                 :: else -> skip;
6       i=i+1;                               fi
7     :: else -> break;                }
8     od
9     do                               module void outputt(int x){
10    :: true -> skip;                       if
11    od                                     :: O[p]! =x -> goto IFL;
12    goto NOEND;                            :: else -> p=p+1;
13 END:p=0,i=0;                              fi
14    do                                     if
15    ::(i<ht)->outputt(i);                  ::(p==MAX)->goto SUCC;
16      i=i+1;                               :: else -> skip;
17    :: else -> break;                      fi
18    od                               }
19 # goto IFL;
20    do                               NOEND: goto NOEND;
21    :: true -> skip;                 SUCC: goto SUCC;
22    od                               IFL: goto IFL;
23 # goto IFL;
24 }
```

如果初始MAX足够小，使得MAX下抽象模型存在可达END状态的执行，则省去对N_{max}的回退寻找过程，直接进行存储-匹配。

5　实现及评价

本文工具的实现基于此前工作[14]中的工具，具体见图3。图中N相当于每次生成的下推系统PDS′中的MAX，Remopla语法分析器已在Moped中实现，我们的实现在Remopla语法分析器后端并在Moped模型检测引擎之前。

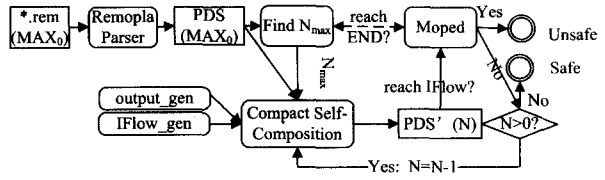

图3　工具实现

实验环境为1.66GHz×2 Intel CPU/1GB RAM。实验目的包括以下两方面：

1)从验证效率和精确性两方面将本文方法与相关工作进行比较。由于第1节相关工作中只有文献[6]给出了具体实现(Iflow[12])，故在此与Iflow比较；又因该方法未考虑发散执行，故用第3节方法与之比较。

2)考察第4节方法相比第3节方法的验证开销增长情况。

针对第一个实验目的，选取以下用例：文献[6]中P1－P8，tax，文献[14]中ttaa1-ttaa3，hu1，hu2，P9为第1节说明抽象解释方法不精确性的程序hu3为文献[13]中的图10。因ttaa1-ttaa3，hu1-hu3本不包含输出信道，在此需改为显式地将低安全级变量向输出信道输出。针对Iflow支持输入信道，本文使用与输出信道等长的数组表示输入信道并对低安全级输入信道进行交叉赋值；使用Remopla语言静态结构体近似表示动态对象。两种方法的具体验证结果如图4所示。

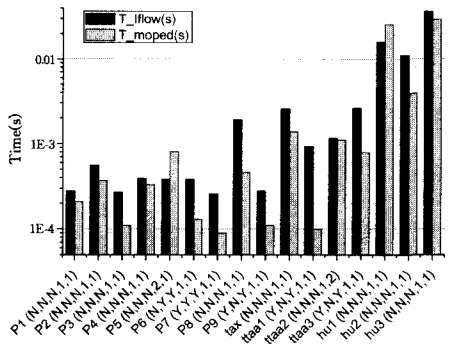

图 4　与文献[6]方法的比较结果

图 4 中用例名后括号中的内容形如($Safe$, $Safe_i$, $Safe_m$, MAX, # bit), $Safe$ 表示根据 TINI'定义该程序是否为安全(Y/N 分别表示安全/不安全), $Safe_i$ 表示用 Iflow 验证出的信息流安全性, $Safe_m$ 表示用本文工具验证出的信息流安全性, MAX 为本文方法使用的输出信道实际长度, # bit 表示变量位数。因文献[6]方法并未实际记录每次向输出信道输出的值,故在绝大多数情况下取 MAX 为 1 即可达到正确的验证结果,唯一例外的 P5 第二次输出的才是高安全级信息,因而需 MAX≥2 才能得到正确的验证结果。

从两工具的验证结果($Safe_i$, $Safe_m$)对实际安全性 $Safe$ 的误判情况看两种方法的验证精确性。P6 因包含发散执行而导致两种方法均产生误判。对于 ttaa1,在 Iflow 计算的终态下 x 具有高安全级而 y 具有低安全级,因而输出信道安全级由低安全级提升到了高安全级,从而得出不安全。相比之下,本文方法考虑了变量的实际计算结果而可以将其验证为安全。类似的情况包括 ttaa3 和 P9。由此可见本文方法较文献[6]方法更精确。

从验证效率的角度看,除 P5 和 hu1 以外,本文方法的验证效率均高于文献[6]方法。P5 由于输出信道长度增长而导致抽象后 BDD 规模增长,从而增加了验证开销。hu1 实际上是变量数极多而违反不干涉性的执行数极少的一种极端情况,这时本文方法记录决定具体路径的变量值在很大程度上增加了开销。虽然 hu3 也具有与 hu1 类似的特点,但 hu3 中高安全级变量数的增加使得 Iflow 的计算开销增长更快。

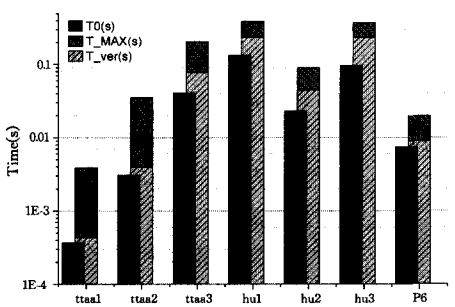

图 5　考虑发散执行导致的验证开销增长

针对第二个实验目的,在 # bit 值为 3、信道初始长度为 50 的情况下比较第 3 节方法与第 4 节方法的验证效率,如图 5 所示。P6 在使用第 4 节方法时验证为不安全。图中 T_{MAX} 为回退算法计算 N_{max} 的时间, T_{ver} 为第 4 节方法的验证时间, T_0 为第 3 节方法的验证时间。由图可知,考虑程序发散执行使得验证开销显著增长,且在过大的初始信道长度下上界回退运算占据了整个验证时间的主要部分。我们将如何更有效地获得输出信道上界作为未来的工作。

结束语　本文实现了一种精确而高效的程序输出信道信息流安全分析方法,其目前可应用于含发散执行的顺序程序。未来工作包括以下 3 方面:①利用并发下推模型将此方法推广到并发程序;②寻找更高效的上界回退算法;③如文献[6]第 8 节建议的将本文方法用于信息流安全中的机密消去机制(declassification)。

参 考 文 献

[1] Barthe G, D'Argenio P R, Rezk T. Secure Information Flow by Self-composition[C]// Proceedings of the 17th Computer Security Foundations Workshop. IEEE Computer Society, 2004: 100-114

[2] Terauchi T, Aiken A. Secure Information Flow as a Safety Problem[C]// SAS. Vol 3672 of LNCS. Springer, 2005: 352-367

[3] Volpano D M, Irvine C E, Smith G. A Sound Type System for Secure Flow Analysis[J]. Journal of Computer Security, 1996, 4(2/3): 167-188

[4] Smith S F, Thober M. Improving usability of information flow security in java[C]// PLAS. ACM Press, 2007: 11-20

[5] Le Guernic G, Banerjee A, Jensen T P, et al. Automata-based Confidentiality Monitoring[C]// ASIAN. Vol 4435 of LNCS. Springer, 2008: 75-89

[6] De Francesco N, Martini L. Instruction-level security typing by abstract interpretation[J]. International Journal of Information Security, 2007, 6(2/3): 85-106

[7] Askarov A, Hunt S, Sabelfeld A, et al. Termination-Insensitive Noninterference Leaks More Than Just a Bit[C]// ESORICS. Vol 5283 of LNCS. Springer, 2008: 333-348

[8] 訾小超,姚立红,李斓. 一种基于有限状态机的隐含信息流分析方法[J]. 计算机学报, 2006, 29(8): 1460-1467

[9] Hunt S, Sands D. On flow-sensitive types[C]// POPL. ACM Press, 2006: 79-90

[10] Introduction to Remopla[EB/OL]. http://www.fmi.uni-stuttgart.de/szs/tools/moped/remopla-intro.pdf

[11] Kiefer S, Schwoon S, Suwimonteerabuth D. Moped: A model checker for pushdown systems[EB/OL]. http://www.fmi.uni-stuttgart.de/szs/ tools/moped, 2006

[12] Martini L. Iflow: a tool for information flow checking[EB/OL]. http://www.iet.unipi.it/l.martini/iflow.html, 2005

[13] Unno H, Kobayashi N, Yonezawa A. Combining type-based analysis and model checking for finding counterexamples against noninterference[C]// PLAS. ACM Press, 2006: 17-26

[14] Sun Cong, Tang Liyong, Chen Zhong. Secure Information Flow by Model Checking Pushdown System[C]// Proceedings of the 2009 Symposia and Workshops on Ubiquitous, Autonomic and Trusted Computing. IEEE Computer Society, 2009: 586-591

[15] Sun Cong, Tang Li-yong, Chen Zhong. Secure Information Flow in Java via Reachability Analysis of Pushdown System[C]// Proceedings of the 10th International Conference on Quality Software. IEEE Computer Society, 2010: 142-150