软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2014,25(6):1143−1153 [doi: 10.13328/j.cnki.jos.004429]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

一种面向非干扰的线程程序逻辑*

李 沁[1], 曾庆凯[2,3], 袁志祥[1]

[1](安徽工业大学 计算机学院,安徽 马鞍山 243032)

[2](计算机软件新技术国家重点实验室(南京大学),江苏 南京 210023)

[3](南京大学 计算机科学与技术系,江苏 南京 210023)

通讯作者: 李沁,曾庆凯, E-mail: linuxos2@163.com, zqk@nju.edu.cn

摘 要: 目前,针对线程信息流的验证研究主要着重于时间信道.然而,由于线程程序中线程控制原语存在函数副作用,对此类原语的不恰当调用亦可引起非法信息流,有意或无意地破坏程序的非干扰属性.因此,提出以验证线程程序信息流为目的依赖逻辑,其可表达线程程序的数据流、控制流以及线程控制函数的副作用,推理程序变量和线程标识符之间的依赖关系,进而判定是否存在高机密性变量对低机密性变量的干扰.

关键词: 非干扰;动态作用域线程;公理语义

中图法分类号: TP301

中文引用格式: 李沁,曾庆凯,袁志祥.一种面向非干扰的线程程序逻辑.软件学报,2014,25(6):1143−1153. http://www.jos.org.cn/1000-9825/4429.htm

英文引用格式: Li Q, Zeng QK, Yuan ZX. Logic of multi-threaded programs for non-interference. Ruan Jian Xue Bao/Journal of Software, 2014,25(6):1143−1153 (in Chinese). http://www.jos.org.cn/1000-9825/4429.htm

Logic of Multi-Threaded Programs for Non-Interference

LI Qin[1], ZENG Qing-Kai[2,3], YUAN Zhi-Xiang[1]

[1](School of Computer, Anhui University of Technology, Maanshan 243032, China)

[2](State Key Laboratory for Novel Software Technology (Nanjing University), Nanjing 210023, China)

[3](Department of Computer Science and Technology, Nanjing University, Nanjing 210023, China)

Corresponding author: LI Qin, ZENG Qing-Kai, E-mail: linuxos2@163.com, zqk@nju.edu.cn

Abstract: Existing work on the verification of information flow in threads mainly focuses on timing channels. However, this paper shows that system calls, such as 'fork' or 'join' with side effects, can also be used to create covert channels intentionally or accidentally. The study results in a dependency logic for verifying information flow in multi-threaded programs where improper calls over thread controlling primitives with side effects could incur illegal information flow. The logic can express the data flow, control flow and the side effects of thread-controlling system calls, and can reason about the dependency relationship among variables and thread identities, to determine whether secret variables interfere with public ones in multi-threaded programs.

Key words: non-interference; dynamic scoping multi-threaded program; axiomatic semantic

线程在操作系统、数据库管理系统等现代软件系统中扮演了极其重要的角色,除了并发以外,线程的动态创建、终止是其显著的特征.信息流安全是线程安全研究的一个重要方面,其目的是保证程序中的高机密性变量不会影响低机密性变量,即所谓的非干扰属性.一般来说,引起程序中的不同安全级别的变量之间的干扰来自于数据流和控制流.数据流是指在赋值表达式的左值为低机密性变量,而右值表达式中包含高机密性变量;控制

* 基金项目: 国家自然科学基金(61170070, 90818022, 61321491); 国家科技支撑计划(2012BAK26B01); 国家高技术研究发展计划(863)(2011AA1A202)

收稿时间: 2012-10-31; 定稿时间: 2013-05-24

流是指在条件判断语句中包含低机密性变量的分支执行依赖于条件中的高机密性变量.无论是在并行组合的并发[1,2]或线程并发[3,4]中,内部或外部时间信道引起的非法信息流都是并发程序独有的现象,因此现有的语言级别的信息流安全研究均着重于在数据流和控制流之上研究对时间信道的验证与分析.

在基于并行组合(parallel composition)的并发程序中,并发的命令形如 $C_1||C_2$,当 C_1 与 C_2 的执行时间有差异,并且当这种差异依赖于高机密性变量时,有可能在程序中出现由这种时间上的差异所导致的低机密性变量取值的不同,从而形成引起非法信息流的内部时间信道.内部时间信道也出现在动态作用域(dynamic scoping)的线程程序中,线程执行时间上的差异可以起到中介作用,形成低机密性变量对高机密性变量的依赖.

然而,现有的线程信息流研究尚未考虑函数副作用对信息流的影响.实际上,支持对全局变量写操作的函数调用和控制流不恰当的组合亦会造成潜在的信息流.特别地,在支持线程编程的 C 语言等系统调用中,线程创建和等待线程终止等线程控制调用在返回时会影响到调用线程状态变量 errno 的值.如果这些线程调用位于条件分支(或循环)语句中,同时在条件分支的判断中又包含高机密性变量,那么变量 errno 将对高机密性变量产生依赖;如果在后续语句的异常处理中存在对低机密性变量的数据流,最终将导致低机密性变量对高机密性变量的依赖,形成非法信息流.

例如,在下面两个例子中,由于不恰当地调用线程函数,程序中都存在非法信息流(两例的结构在 Linux 的进程并发程序或者基于 POSIX 的 Pthread 线程库的程序中是常见的[5]).

例 1 是一个由不平衡条件语句和动态线程控制函数共同引起内部时间信道的例子.在第 4 行的条件分支语句,由于其两个分支在线程 t 的执行时间是不平衡的,线程 t1 中对 low 的赋值(第 11 行)需等待线程 t 执行完毕(第 10 行调用 join 函数等待线程 t 结束)才能执行.主线程第 11 行对变量 low 的赋值既有可能发生在线程 t1 中第 9 行之前,也有可能在其之后,这取决于线程 t 的执行时间,而线程 t 的执行时间又取决于变量 high 的值,这样就形成了低机密性变量 low 对高机密性变量 high 的依赖:如果 low 为 0,则 high 不为 0;反之,high 为 0.

例 1:内部时间信道.

```
1   high=0; pid_t t;
2   t=fork {
3           if high
4               step(1000);
5           else
6               step(1);}
7   t1=fork {
8       join t;
9     low=0;}
10  step(100);
11  low=2;
```

例 2 是一个完全由线程内部状态引起非法信息流的例子.线程 t 的存在取决于高机密性变量 high 的值,第 5 行调用 join 函数等待线程 t 结束.如果线程 t 不存在,则 join 函数会返回,并对主线程的局部变量 errno 赋值为 ESRCH.在第 6 行进行错误处理时,低机密性变量 low 的值根据 errno 的值而不同,从而形成了从变量 high 经由线程 t 的存在性而到达变量 low 的非法信息流.

例 2:线程存在性引起的信息流.

```
1   high=1; pid_t t;
2   if high
3       t=fork step(100);
4       else skip;
5   join t;
```

```
6  if errno==ESRCH
7    low=1;
8    else low=0;
```

上述两例充分说明:线程控制函数的不恰当调用,会导致线程状态对高机密性变量的依赖;而当低机密性变量依赖于线程状态时,则形成了间接的非法控制流.可见,线程程序在非干扰属性方面的行为特征有其自身的特点,除了内部时间信道方面的因素之外,线程控制函数的副作用的影响也很大.

针对以上问题,本文从公理语义出发,研究动态多线程程序的信息流安全问题,包括:

1) 首次提出了线程控制原语的副作用引发的信息流安全问题.分析了控制流和线程控制原语的不恰当组合导致的非法信道,这种信道利用线程内部状态诱导非法信息流的出现.此问题的提出丰富了并发程序信息流问题所涉及的程序类型,缩小了信息流安全研究与实际问题的距离;
2) 提出了针对线程信息流的程序逻辑.虽然已经有应用公理语义方法验证并发程序的工作[6,7],但是尚无针对线程程序中函数副作用对信息流安全(security)的影响的研究.本文利用不同机密性级别的变量间的依赖关系间接表达程序的非干扰属性,进而利用依赖逻辑对变量、线程间的控制依赖关系进行推理验证.借助依赖逻辑的语法可以对线程状态的判断以及线程系统调用的副作用作较为简明的描述,而这些描述却难以在基于类型理论的方法中得到表达.

本文第 1 节介绍并发程序信息流研究的相关工作.第 2 节给出 CWhile 语言的语法与语义.第 3 节给出依赖逻辑的语法和语义.第 4 节证明依赖逻辑的可靠性.第 5 节给出应用依赖逻辑的实例.最后是结论.

1 相关工作

在现有并发程序信息流研究中,无论是并行组合还是线程并发,研究的重点可以总结为:在顺序程序的数据流和控制流分析基础之上,分析执行时间上不平衡的线程如何引发内部或外部时间信道.

对于并行组合程序中的信息流,类型论是主要被采用的验证方法.文献[8,9]主要研究了对可能成为并发程序的内部中间语言的π演算变体,如何构造基于类型理论的技术以跟踪似然性信息流,并在此基础上扩展了引用和高阶过程,使得验证可以面向命令式中间语言.这些工作所提出的类型系统的可靠性建立在观察必然的基础上,其缺点是过于严格.

文献[10]为提高类型论方法的可扩展性,提出了对单个线程对共享变量进行访问时需说明对变量的假设(assumption)和保证(guarantee),从而使得类型检查可以针对单一线程展开,只要该线程、环境对共享变量的使用满足假设和保证的规范,即可允许程序通过类型检查.

文献[11]着重研究了独立于调度模型的非干扰属性定义,并指出统一均匀分布调度模型或轮转调度模型是健壮的,由此放松一些过于严格的对类型系统的限制.文献[12]给出了一种程序和调度模型交互的新的建模方式,侧重于给出并发语言同步模型以及调度与线程之间的规范化交互接口.

Volpano 和 Smith 在文献[13]中针对静态线程提出了概率性非干扰,在语言中加入了 protect 原语以防止产生时间信道.然而,protect 原语本身在实现上的可能性还值得商榷.文献[14]针对类似于 CML 语言的并发演算提出了基于观察必然的非干扰属性,并给出了相应的类型系统.文献[1,15]提出了基于马尔科夫链的观察必然的非干扰属性定义,分别从类型理论和模型检验的角度对时间信道进行了分析.这些工作是基于统一均匀分布调度模型的,不能应用于轮转调度模型.

在线程并发的信息流方面,Sabelfeld 在文献[4]中针对包含同步原语的并发程序信息流提出了类型检查的方法,并给出了一种关闭时间信道的代码填充技术;作者又在文献[3]中给出了针对顺序程序的转换技术,将敏感计算转移至新线程中执行.文献[16]则针对并发程序利用一个 Haskell 库实现了从敏感计算到新线程的转换,同时针对线程所接触的信息的安全级别,动态变化线程的安全级别,达到控制或约束非法信息流的目的.文献[17]研究了在全存储序(total store order)内存模型下,线程程序的验证信息流的类型系统,该系统在免于数据竞争的程序中可以保证其可靠性.

文献[18]首次在并发 C 程序中使用上下文敏感的切片技术分析信息流安全,Krinke 将程序依赖图(PDG)扩展为线程程序依赖图(tPDG),在此基础上提出了较为精确的上下文敏感的切片算法.算法虽然考虑了过程间依赖,但是没有考虑过程调用的副作用.

总之,无论是基于类型论的方法、动态监视转换还是基于切片技术的方法,均未考虑线程动态管理函数的副作用引起的非法信息流问题.

2 Cwhile 语言的语法与语义

本节给出命令式动态多线程 CWhile 语言的语法和语义,在语法上类似于文献[4]中的语言,线程控制函数源于简化后的 POSIX Pthread 线程库函数,其在语义上的非形式化解释来自于 POSIX 的 Linux 帮助文档.

2.1 语 法

CWhile 语法的 BNF 定义如图 1 所示.除了常见的命令式顺序语言的语法构件之外,语言的动态并发行为来自于两个线程控制原语 fork 和 join.语句 x=fork C 用来生成新的线程,返回新线程的标识符赋值于变量 x,新的线程执行命令 C,规定这里的 x 在此语句之前没有被使用过.执行语句 join x 的线程将等待线程 x 终止,如果线程 x 不存在,则返回错误值 NEX,并赋值于该线程内部变量 errno.注释语句% inside E 用来指明当前的执行位于线程 E 中,其作用将在下文中说明;线程利用语句 exit 结束自己的执行.我们假设 CWhile 程序的运行总是可以终止的.

$$E ::= n \mid x \mid E + E \mid E - E \mid \ldots \quad B ::= \text{True} \mid \text{False} \mid B \vee B \mid \neg B \quad A ::= \%\ \text{inside}\ E$$
$$C ::= \text{skip} \mid x = E \mid \text{if}\ B\ \text{then}\ C\ \text{else}\ C \mid \text{while}\ B\ \text{do}\ C \mid C;C \mid x = \text{fork}\ C \mid \text{join}\ E \mid \text{exit} \mid A\ C$$

Fig.1 Syntax of CWhile

图 1 CWhile 的语法

2.2 操作语义

Cwhile 的操作语义牵涉到以下几个定义:

$$Var =_{\text{def}} \{x, y, z, \ldots\} \quad \text{States} =_{\text{def}} \{\sigma \mid Var \to \mathbb{Z} \cup \{\perp\}\}$$
$$\text{ErrVal} =_{\text{def}} \{0, NEX, \ldots\} \quad \text{TState} =_{\text{def}} \{\gamma \mid \mathbb{N} \rightharpoonup \mathcal{C} \times \text{ErrVal} \times \mathcal{P} \times \{0,1\}\}.$$

集合 Var 是变量的可数无穷集;集合 States 的元素为函数 σ,将已定义变量映射到整数集(对未定义变量其取值为未定义符号 $\perp$),用来记录程序运行时状态;集合 ErrVal 包括线程内部错误变量 errno 可能取到的值,errno 的默认值为 0.线程格局是由四元组(C,errno,P,b)定义的,其中,$P \in \mathcal{P}$是线程满足的谓词[5],这里,$\mathcal{P}$代表所有一阶谓词构成的集合.线程格局用来表示单线程所处的状态,第 4 个参数 b 为 1 时,说明线程在线程池中;为 0 时说明线程已经结束.程序格局是由部分函数(partial function)γ以及状态函数 σ定义的,前者将代表线程标识符的自然数集合映射到线程格局集合,程序格局(σ,γ)用来表示一个多线程程序执行时所有线程所处的状态以及所有程序变量的值.

首先给出表达式的语义,由函数 $[\![\ ]\!]_\sigma$ 归纳给出:

$$[\![n]\!]_\sigma = n, [\![\perp]\!]_\sigma = \perp, [\![x]\!]_\sigma = \sigma x, [\![E_1 \pm E_2]\!]_\sigma = [\![E_1]\!]_\sigma \pm [\![E_2]\!]_\sigma.$$

其次,单线程命令的语义是由线程格局间的二元关系 $\rightsquigarrow$ 定义的,单线程命令的语义与一般的命令式顺序程序语言的语义相近,但是其语义动作的结果却有全局和局部之分:全局上,赋值语句(ASS)改变全局状态函数 σ;局部上,单个线程执行的命令序列发生变化,即线程状态函数 γ的值发生变化.

在以下语义规则中,谓词 Inside 是用来指明当前执行命令 join t 的线程.语义规则 EXT 说明在执行 exit 之后,有关线程 t 的记录并不消失,而是保留在线程池中;同时,利用线程记录的第 4 个参数标记此线程已经不再活跃.规则如下(假设当前线程的标识符为 n):

$$(\sigma,\gamma[n\mapsto(\text{skip},0,P,1)])\rightsquigarrow(\sigma,\gamma[n\mapsto(\_,0,P,1)]) \quad \text{(SKP)}$$

$$\frac{[\![E]\!]_\sigma=k}{(\sigma,\gamma[n\mapsto(x=E,0,P,1)])\rightsquigarrow(\sigma[x\mapsto k],\gamma[n\mapsto(\text{skip},0,P,1)])} \quad \text{(ASS)}$$

$$\frac{[\![B]\!]_\sigma=\text{True}}{(\sigma,\gamma[n\mapsto(\text{if }B\text{ then }C_1\text{ else }C_2,0,P,1)])\rightsquigarrow(\sigma,\gamma[n\mapsto(C_1,0,P,1)])} \quad \text{(IFT)}$$

$$\frac{[\![B]\!]_\sigma=\text{False}}{(\sigma,\gamma[n\mapsto(\text{if }B\text{ then }C_1\text{ else }C_2,0,P,1)])\rightsquigarrow(\sigma,\gamma[n\mapsto(C_2,0,P,1)])} \quad \text{(IFF)}$$

$$\frac{[\![B]\!]_\sigma=\text{True}\ \ \sigma,\gamma[n\mapsto(C,0,P\wedge B,1)]\rightsquigarrow\sigma,\gamma[n\mapsto(\text{skip},0,P,1)]}{(\sigma,\gamma[n\mapsto(\text{while }B\text{ do }C,0,P\wedge B,1)])\rightsquigarrow(\sigma,\gamma[n\mapsto(C;\text{while}B\text{ do }C,0,P,1)])} \quad \text{(WHT)}$$

$$\frac{[\![B]\!]_\sigma=\text{False}}{(\sigma,\gamma[n\mapsto(\text{while }B\text{ do }C,0,P,1)])\rightsquigarrow(\sigma,\gamma[n\mapsto(\text{skip},0,P,1)])} \quad \text{(WHF)}$$

$$\frac{\sigma t\in\text{dom}(\gamma)\ \mathsf{Inside}(\sigma t')\ (\gamma\sigma t)_3=Q}{(\sigma,\gamma[\sigma t'\mapsto(\text{join }t,0,P,1)])\rightsquigarrow(\sigma,\gamma[\sigma t'\mapsto(\text{skip},0,P\wedge\mathsf{Dep}(t,t'),1)])} \quad \text{(JN1)}$$

$$\frac{\sigma t\notin\text{dom}(\gamma)\ \mathsf{Inside}(\sigma t')}{(\sigma,\gamma[\sigma t'\mapsto(\text{join }t,0,P,1)])\rightsquigarrow(\sigma,\gamma[\sigma t'\mapsto(\text{skip},NEX,P,1)])} \quad \text{(JN2)}$$

$$\frac{\sigma t=\perp\ \ n\notin\text{dom}(\gamma)\ \mathsf{Inside}(\sigma t')\ \ \sigma,\gamma\vDash P}{(\sigma,\gamma[\sigma t'\mapsto(t=\text{fork}_{[P]}C,0,R,1)])\rightsquigarrow(\sigma[t\mapsto n],\gamma[n\mapsto(C,0,P,1),\sigma t'\mapsto(\text{skip},0,R,1)])} \quad \text{(FRK)}$$

$$\frac{\mathsf{Inside}(\sigma t)}{(\sigma,\gamma[\sigma t\mapsto(\text{exit},0,P,1)])\rightsquigarrow(\sigma,\gamma[\sigma t\mapsto(\text{skip},0,P,0)])} \quad \text{(EXT)}$$

.

3 依赖逻辑

本节给出依赖逻辑的逻辑系统,其目标是对线程程序中的程序变量之间的依赖关系进行推理.

3.1 语法和语义

首先给出依赖逻辑的断言:

$$\begin{aligned}P,Q ::= {}& \mathsf{Inside}(C,E)\mid\mathsf{Dep}(E_1,E_2)\mid\mathsf{Thread}(E,P)\\ &\mid\mathsf{LV}(x)\mid\mathsf{Done}(E)\mid P\to Q\mid P\times Q\\ &\mid P\to^{*}Q\mid\exists x.P\mid P\wedge Q\mid P\oplus_x Q\mid\neg P\mid B.\end{aligned}$$

依赖逻辑的断言是定义在程序状态σ和线程格局γ上的一阶谓词.断言 Inside 指明当前命令所处的线程;断言 Thread 指明线程 E 满足 P,定义空断言 emp 代表无判断发生;断言 Done 指明某线程已终止;断言 Dep 指明 E_2 的取值依赖于 E_1 的取值.为阅读上的简明,当存在多个变量依赖于一个变量时,则将谓词的第 2 个参数写成集合的形式,即:

$$\mathsf{Dep}(t,\{t_1,t_2\})\Leftrightarrow\mathsf{Dep}(t,t_1)\wedge\mathsf{Dep}(t,t_2).$$

$\mathsf{LV}(x)$指变量 x 为左值,此谓词为了表达线程对变量的赋值历史.

类似于分离逻辑,$P*Q$ 指存在两个线程分别满足 P 和 Q;$P\to^{*}Q$ 指已存在线程满足 P,并且在有新生成的线程之后,Q 也被满足;$P\oplus_x Q$ 指 P 和 Q 的成立是不相容的,并且依赖于变量 x.在依赖逻辑中引入这个联结词,是为了表达条件分支对程序变量的依赖:条件语句分支中,以谓词表达的状态将依赖于逻辑判断中的变量,并且两个分支中的谓词状态在一般意义上是不相容的,它的优先级是高于联结词$\wedge$的.此算符的定义为

$$P\oplus_x Q\Leftrightarrow\forall t\in\mathsf{FV}(P\vee Q).\mathsf{LV}(t)\wedge\mathsf{Dep}(x,t)\wedge((P\wedge\neg Q)\vee(\neg P,Q)).$$

为定义逻辑断言的形式语义,首先给出两个语义函数σ和γ在关系$\rightsquigarrow$之上所形成的迹.程序的迹表示程序在一次运行中程序格局所经历的中间状态.

定义 1(迹). 程序的迹 $(\vec{\sigma},\vec{\gamma})$ 是一个以二元组(σ,γ)为元素的序列,相邻序列项之间存在$\rightsquigarrow$关系,即:

$$(\vec{\sigma},\vec{\gamma})=_{def}((\sigma_0,\gamma_0),(\sigma_1,\gamma_1),\ldots,(\sigma_k,\gamma_k)),(\sigma_i,\gamma_i)\rightsquigarrow(\sigma_{i+1},\gamma_{i+1}).$$

图 2 给出依赖逻辑断言的形式语义.注意:函数$()_i$用来返回一个四元组的第 i 个分量;下划线_指其所处位置的值与语义动作无关.

$$\sigma_k,\gamma_k \vDash B \Leftrightarrow [\![B]\!]_{\sigma_k} = tt$$
$$\sigma_k,\gamma_k \vDash \mathsf{Thread}(E,P) \Leftrightarrow (\gamma_k([\![E]\!]_{\sigma_k}))_3 = P$$
$$\sigma_k,\gamma_k \vDash P_1 * P_2 \Leftrightarrow \exists t_1,t_2 \in \mathrm{dom}(\gamma_k).t_1 \neq t_2 \wedge (\gamma_k t_1)_3 = P_1 \wedge (\gamma_k t_2)_3 = P_2$$
$$\sigma_k,\gamma_k \vDash P_1 \rightarrow^* P_2 \Leftrightarrow \exists t_1 \in \mathrm{dom}(\gamma_k), t_2 \in \mathrm{dom}(\gamma').\mathrm{dom}(\gamma_k) \cap \mathrm{dom}(\gamma')=\varnothing \wedge (\sigma_k,\gamma_k \vDash P_1) \Rightarrow (\sigma_k,\gamma_k \uplus \gamma' \vDash P_2)$$
$$\sigma_k,\gamma_k \vDash \mathsf{Inside}(C,E) \Leftrightarrow \exists i,n.n \in \mathrm{dom}(\gamma_{k-1}) \wedge n = [\![E]\!]_{\sigma_{k-i}} = [\![E]\!]_{\sigma_k} \wedge (\gamma_{k-i}n)_1 = \%\mathrm{inside}\ E\ C;C'$$
$$\sigma_k,\gamma_k \vDash \mathsf{LV}(x) \Leftrightarrow \exists m,t,i.i < k \wedge m \in \mathrm{dom}(\gamma_i) \wedge m = \sigma_i t \wedge (\gamma_i m)_1 = (x = \_;C)$$
$$\sigma_k,\gamma_k \vDash \mathsf{Done}(E) \Leftrightarrow (\gamma_k[\![E]\!]_{\sigma_k})_4 = 0$$
$$\vDash \mathsf{Dep}(x,y) \Leftrightarrow \forall \sigma_1,\sigma_2,\gamma_1,\gamma_2.(\sigma_1 x \neq \sigma_2 x \Rightarrow \sigma_1 y \neq \sigma_2 y) \vee (\gamma_1\sigma_1 x \neq \gamma_2\sigma_2 x \Rightarrow \gamma_1\sigma_1 y \neq \gamma_2\sigma_2 y)$$

Fig.2 Formal semantics of dependency logic

图 2 依赖逻辑的形式语义

谓词 Dep 则是根据非干扰的一般规范定义的,即:变量 y 依赖于 x,当且仅当在程序执行的任意状态下,若 x 的取值发生变化,则 y 的取值也不同;析取式的第 2 部分代表线程状态之间的依赖关系.

3.2 证明规则

依赖逻辑的判断(judgement)以霍尔逻辑的三元组形式$\{P\}C\{Q\}$给出,P,C,Q 分别是前置条件、命令和后置条件.非形式地说,三元组的含义为:如果在初始格局下前置条件 P 成立,命令 C 执行并且可以终止,那么后置条件 Q 亦成立.图 3 给出依赖逻辑的公理.

- 公理 A1 和公理 A2 用来按线程标识符规范逻辑公式,以期将针对同一线程的逻辑判断归结到一个式子中去;
- 公理 A3 是针对赋值语句的三元组,在霍尔逻辑规则的基础上引入了左值对右值表达式中变量的依赖,同时记录了对变量 x 的赋值次数;
- 公理 A4 表达依赖关系的传递性;
- 公理 A5 说明在线程执行退出命令后线程不再活动.

$$\vdash P \wedge \mathsf{Incide}(C,E) \Rightarrow \mathsf{Thread}(E,P) \quad \text{(A1)}$$
$$\vdash P \wedge \mathsf{Thread}(E,Q) \Rightarrow \mathsf{Thread}(E,P \wedge Q) \quad \text{(A2)}$$
$$\vdash \{P\}\ x = E\{P \wedge \mathsf{Dep}(\mathsf{FV}(E),x) \wedge \mathsf{LV}(x)\} \quad \text{(A3)}$$
$$\vdash \mathsf{Dep}(x,y) \wedge \mathsf{Dep}(y,z) \Rightarrow \mathsf{Dep}(x,z) \quad \text{(A4)}$$
$$\vdash \{\mathsf{Thread}(t,P)\}\%\mathrm{inside}\ t\ \mathrm{exit}\{\mathsf{Thread}(t,P \wedge \mathsf{Done}(t))\} \quad \text{(A5)}$$

Fig.3 Axioms of dependency logic

图 3 依赖逻辑的公理

图 4 为逻辑的推理规则.

- 规则 R1 说明在程序中存在注释%inside E 时在程序状态中记录当前线程标识符为 E;
- 规则 R2 说明对顺序语句的组合推理和霍尔逻辑是相同的;
- 规则 R3 说明当线程标识符 t 未被使用的情况下,fork 语句将产生一个以 t 为标识符的新线程,运行此线程应满足断言 P';
- 规则 R4 和规则 R5 分别说明了终止线程的两种可能:终止一个存在的线程,则 errno 赋值为 0;否则,将 errno 置为 NEX;
- 规则 R6 将条件语句的两个分支执行后的状态区别开来,表达两种不同状态对判断语句中变量的依赖关系.

- 规则 R7 指当循环语句的循环体中存在赋值语句,则作为左值的变量对判断语句中的变量有依赖关系.

$$\frac{\{P\}C\{Q\}}{\{P\}\%\text{inside } E\ C\{Q\wedge \mathsf{Inside}\{C,E\}\}} \quad (R1)$$

$$\frac{\{P\}C\{Q\}\ \{Q\}C'\{R\}}{\{P\}C;C'\{R\}} \quad (R2)$$

$$\frac{t\notin \mathsf{FV}(P)}{\{P\}t=\text{fork}_{[P']}\ C\{P\wedge \mathsf{LV}(t)*\mathsf{Thread}(t,P')\}} \quad (R3)$$

$$\frac{P\Rightarrow\neg\mathsf{Done}(t)}{\{\mathsf{Thread}(t',P')*\mathsf{Thread}(t,P)\}\%\text{inside } t'\text{ join } t\{\mathsf{Thread}(t',P'\wedge \text{errno}=0\wedge \mathsf{Dep}(t,\{t',\text{errno}\})*\mathsf{Thread}(t,P\wedge\mathsf{Done}(t))\}} \quad (R4)$$

$$\frac{t\notin \mathsf{FV}(P)\vee P\Rightarrow \mathsf{Thread}(t,Q\wedge\mathsf{Done}(t))}{\{\mathsf{Thread}(t',P'\wedge \text{errno}=0)*P\}\%\text{inside } t'\text{ join } t\{\mathsf{Thread}(t',P'\wedge \text{errno}=NEX\wedge\mathsf{Dep}(t,\{t',\text{errno}\})*P\}} \quad (R5)$$

$$\frac{\{P\wedge B\}C_1\{Q_1\}\ \{P\wedge\neg B\}C_2\{Q_2\}\ x\in\mathsf{FV}(B)}{\{P\}\text{if } B \text{ then } C_1 \text{ else } C_2\{Q_1\oplus_x Q_2\}} \quad (R6)$$

$$\frac{\{P\wedge B\}C\{P\wedge \mathsf{LV}(x)\}\ y\in\mathsf{FV}(B)}{\{P\}\text{while } B \text{ do } C\{P\wedge\mathsf{LV}(x)\wedge\neg B\wedge\mathsf{Dep}(x,y)\}} \quad (R7)$$

Fig.4 Inference rules of dependency logic

图 4 依赖逻辑的推理规则

4 可靠性的证明

本节证明依赖逻辑的可靠性,即:

定理 1(**可靠性**). 如果$\vdash\{P\}C\{Q\}$,那么对任意 $\vec{\sigma},\vec{\gamma}$,如果$\sigma_0,\gamma_0\vDash P$,则$\vDash\{P\}C\{Q\}$.

证明:对可靠性的证明从公理可靠性出发,然后证明推理规则的可靠性;对复合规则的证明是归纳进行的.

对公理 A1,需证明:若存在 $\vec{\sigma},\vec{\gamma}$ 且存在 k 满足$\sigma_k,\gamma_k\vDash P\wedge\mathsf{Inside}(C,E)$时,$\sigma_k,\gamma_k\vDash\mathsf{Thread}(E,P)$亦成立.根据假设有$\sigma_k,\gamma_k\vDash P$ 以及$\sigma_k,\gamma_k\vDash\mathsf{Inside}(C,E)$,由谓词 Inside 的语义可知:在$\gamma_k$ 之前,线程$[\![E]\!]_{\sigma_{k-i}}$,其执行的命令为%inside E C,这说明在线程$[\![E]\!]_{\sigma_{k-i}}$ 的代码中存在注释指明当前线程标识符;又由于两个判断是由合取联结词联结的,观察公理和推理规则不难发现:只有规则 R3 产生关于*的断言,而其他规则均是对于单线程语句的判断,所以断言 P 仅仅是对当前线程$[\![E]\!]_{\sigma_{k-i}}$ 的状态判断,即当前线程$[\![E]\!]_{\sigma_{k-i}}$ 满足断言 P.由 Thread 的语义可知,在状态σ_k,γ_k 下,$\mathsf{Thread}(E,P)$成立.

对公理 A2 的证明是类似的.

对公理 A3,需证明:若存在 $\vec{\sigma},\vec{\gamma}$ 且存在 k 满足$\sigma_k,\gamma_k\vDash P$,且当前线程 t 运行命令 $x=E$,那么在此命令执行后,在$\sigma_{k+1},\gamma_{k+1}$下,有$\vDash P\wedge\mathsf{Dep}(\mathsf{FV}(E),x)\wedge\mathsf{LV}(x)$.首先,由于执行的命令是赋值语句,变量 x 成为一个左值,由谓词 LV 的语义可知 $\mathsf{LV}(x)$成立;最后,对表达式 E 中包含的变量的值的改变,显然会改变 x 的取值,因此 $\mathsf{Dep}(\mathsf{FV}(E),x)$亦成立.

对公理 A4,易见依赖关系的传递性是显然的.

对公理 A5,需证明:如果存在 $\vec{\sigma},\vec{\gamma}$ 且存在 k 满足$\sigma_k,\gamma_k\vDash\mathsf{Thread}(t,P)$,则在执行%inside t exit 之后,有$\sigma_k,\gamma_k\vDash\mathsf{Thread}(t,P\wedge\mathsf{Done}(t))$.由 $\mathsf{Thread}(t,P)$成立可知$(\gamma_k\sigma_k t)_3=P$,执行语句之后,由 exit 的语义可知$(\gamma_{k+1}\sigma_{k+1}t)_4=0$,故$(\gamma_{k+1}\sigma_{k+1}t)_3=P\wedge\mathsf{Done}(t)$,因此$\sigma_k,\gamma_k\vDash\mathsf{Thread}(t,P\wedge\mathsf{Done}(t))$.

对规则 R1,需证明:如果存在 $\vec{\sigma},\vec{\gamma}$ 且存在 k 满足$\sigma_k,\gamma_k\vDash\{P\}C\{Q\}$,则在执行语句%inside E C 后,$\sigma_{k+1},\gamma_{k+1}\vDash Q\wedge\mathsf{Inside}(C,E)$.由$\sigma_k,\gamma_k\vDash\{P\}C\{Q\}$可知$\sigma_k,\gamma_k\vDash P$,因为注释并不对程序状态 $\vec{\sigma}$ 产生影响,所以$\sigma_{k+1},\gamma_{k+1}\vDash Q$.再由谓词 Inside 的语义,因为 $(\gamma_k[\![E]\!]_{\sigma_k})_1=$ %inside E C,可知$\sigma_{k+1},\gamma_{k+1}\vDash\mathsf{Inside}(C,E)$.

对规则 R2,需证明:若存在 $\vec{\sigma},\vec{\gamma}$ 且存在 k 满足$\sigma_k,\gamma_k\vDash\{P\}C\{Q\}$,$\sigma_{k+1},\gamma_{k+1}\vDash\{Q\}C'\{R\}$,则若$\sigma_k,\gamma_k\vDash P$,在执行语句 $C;C'$后,有$\sigma_{k+2},\gamma_{k+2}\vDash R$.由归纳假设可知,在语句 C 执行后$\sigma_{k+1},\gamma_{k+1}\vDash Q$;再由归纳假设可知,在执行 C'后断言 R 成立.

对规则 R3,需证明:若存在 $\vec{\sigma},\vec{\gamma}$ 且存在 k 满足$\sigma_k,\gamma_k\vDash P$,且$\exists n.(\gamma_k n)_1=(t=\text{fork}_{[P']}C)\wedge(\gamma_k n)_3=P$,则在生成线程 t 之后,

有$\sigma_{k+1},\gamma_{k+1}\vDash P\wedge \mathsf{LV}(t)*\mathsf{Thread}(t,P')$.首先,由 fork 语句的语义可知,在执行此语句后会生成一个新的线程标识符 $m=\sigma_{k+1}t$,且$(\gamma_{k+1}m)_3=P'$,这正好符合谓词 Thread 的语义;其次,由于本语句也是一个赋值语句,所以也符合 LV 的语义;最后,由于变量 t 不在 P 中自由出现,所以此赋值行为不影响断言 P 的真值.

对规则 R4,需证明:如果存在 $\vec{\sigma},\vec{\gamma}$ 且存在 k 满足$\sigma_k,\gamma_k\vDash\mathsf{Thread}(t',P')*\mathsf{Thread}(t,P)$,且 $P\Rightarrow\neg\mathsf{Done}(t)$,则在执行语句%inside t' join t 之后,有$\sigma_{k+1},\gamma_{k+1}\vDash\mathsf{Thread}(t',P'\wedge\mathrm{errno}=0\wedge\mathsf{Dep}(t,t'))*\mathsf{Thread}(t,P\wedge\mathsf{Done}(t))$.由假设以及谓词 Thread 的语义可知,在当前的程序格局σ_k,γ_k下,$(\gamma_k t)_3=P\wedge(\gamma_k t')_3=P'$;再由注释可知,语句 join t 属于线程 t',从而有 $\mathsf{Thread}(t',P')\wedge\mathsf{Inside}(t',\text{join } t)$;在执行语句 join t 之后,由语义知$(\gamma_{k+1}t')_2=0$,且对于线程 t,有$(\gamma_{k+1}t)_4=0$,所以 $\mathsf{Thread}(t,P\wedge\mathsf{Done}(t))$和 $\mathsf{Thread}(t',\mathrm{errno}=0)$分别成立;最后,线程 t 的存在性(即$(\gamma_k t)_4$)决定线程 t'的四元组的 errno 分量的取值(即$(\gamma_k t')_2$),因此 $\mathsf{Dep}(t',\{t,\mathrm{errno}\})$成立.

对规则 R5,需证明:如果存在 $\vec{\sigma},\vec{\gamma}$ 且存在 k 满足$\sigma_k,\gamma_k\vDash\mathsf{Thread}(t',P'\wedge\mathrm{errno}=0)*P$,且 $t\notin\mathsf{FV}(P)$或者断言 P 蕴涵 $\mathsf{Thread}(t,Q\wedge\mathsf{Done}(t))$,则在执行语句%inside t' join t 后,有$\sigma_{k+1},\gamma_{k+1}\vDash\mathsf{Thread}(t',P'\wedge\mathrm{errno}=NEX\wedge\mathsf{Dep}(t,t'))*P$.首先,由 $t\notin\mathsf{FV}(P)$或者断言 P 蕴涵 $\mathsf{Thread}(t,Q\wedge\mathsf{Done}(t))$可知,线程 t 不存在或者已经结束,按照%inside t' join t 的语义可知,当前执行此语句的线程为 t',并且在此状态下执行 join t 会导致$(\gamma_{k+1}t')_2$ 被置为 NEX,所以在$\sigma_{k+1},\gamma_{k+1}$下,有 $\mathsf{Thread}(t',P'\wedge\mathrm{errno}=NEX)$;其次,与规则 R4 相同,线程 t 的存在性仍然影响了线程 t'的状态以及变量 errno,因此 $\mathsf{Dep}(t',\{t,\mathrm{errno}\})$成立.

对规则 R6,如果对任何程序格局(σ,γ),若存在 k 使得$\sigma_k,\gamma_k\vDash\{P\wedge B\}C_1\{Q_1\}\wedge\{P\wedge\neg B\}C_2\{Q_2\}$,需证明:若$\sigma_k,\gamma_k\vDash P$,则在执行语句 if B then C_1 else C_2 后,有$\sigma_{k+1},\gamma_{k+1}\vDash\forall x\in\mathsf{FV}(B).Q_1\oplus_x Q_2$.考虑在当前格局$\sigma_k$下,若$[\![B]\!]_{\sigma_k}=\mathrm{True}$,则按照语义规则 IFT 可知,语句 C_1 将得到执行;在其执行完毕后,可得$\sigma_{k+1},\gamma_{k+1}\vDash Q_1$;相反若$[\![B]\!]_{\sigma_k}=\mathrm{False}$,同理可知 $\sigma_{k+1},\gamma_{k+1}\vDash Q_2$.因此,断言 Q_1,Q_2 成立与否取决于断言 B 的真值:如果这两个断言中存在谓词 $\mathsf{LV}(x)$,说明在语句 C_1 或 C_2 中存在对变量 x 的赋值,所以其取值依赖于集合 $\mathsf{FV}(B)$中的变量值.另外,易见谓词 Q_1 与 Q_2 的成立是不相容的.综上,对任意变量 $x\in\mathsf{FV}(B)$,有 $Q_1\oplus_x Q_2$.

对规则 R7,如果对任何程序格局(σ,γ),若存在 k 使得$\sigma_k,\gamma_k\vDash\{P\wedge B\}C\{P\wedge\mathsf{LV}(x)\}$,需证明:若$\sigma_k,\gamma_k\vDash P$,则在执行语句 while B do C 之后,有$\exists i.i\geqslant 0.\sigma_{k+1},\gamma_{k+1}\vDash P\wedge\neg B\wedge\mathsf{Dep}(x,y),y\in\mathsf{FV}(B)$.由 while 语句的语义规则 WHT,若$[\![B]\!]_{\sigma_k}$为真,则执行语句 C,并且在执行过程中对变量 x 赋值,继续执行 while 语句,并且有 $\mathsf{LV}(x)$;由程序可终止的假设可知,存在 i 使得在执行循环体 C 之后,$[\![B]\!]_{\sigma_{k+i}}$为假,使得$\neg B$ 成立.由于$[\![B]\!]_{\sigma_k}$的真值决定对 x 的赋值动作是否发生,因此有 $\mathsf{Dep}(x,y)$.所以结论成立. □

5 依赖逻辑的应用

5.1 谓词LV的局部作用域

依赖逻辑的断言中谓词 LV 的作用是表达在已经执行的语句中存在赋值动作,由于同一变量可能被多次赋值,单凭依赖逻辑仅能表达某一变量曾经被赋值,不能表达赋值动作的新鲜性.

例如,当赋值语句 C_1 与条件控制语句 C_2 顺序复合时,如果 C_1,C_2 都包含对同一变量 x 的赋值动作,那么根据推理规则有$\{P\}C_1\{Q\wedge\mathsf{LV}(x)\}$和$\{Q\wedge\mathsf{LV}(x)\}C_2\{R\wedge\mathsf{LV}(x)\}$,进而根据复合规则有$\{P\}C_1;C_2\{R\wedge\mathsf{LV}(x)\}$.注意到:在语句 C_2 的前件和后件中均有断言 $\mathsf{LV}(x)$,实际上这两处 $\mathsf{LV}(x)$的发生分别是前后两次赋值动作的结果.假若在语句 C_2 中没有针对变量 x 的赋值,在 C_2 结束后 $\mathsf{LV}(x)$仍然成立.此处无法分辨 $\mathsf{LV}(x)$是针对哪一个语句进行判断的.对于依赖逻辑来说,如果对赋值动作发生位置不敏感,那么在条件控制语句或者循环语句中将无法判断变量间依赖关系.

问题的根源在于:先行语句的赋值判断 $\mathsf{LV}(x)$覆盖了后续语句中对变量 x 的判断,无论在后续语句中是否存在对 x 的赋值,$\mathsf{LV}(x)$都被继承下来,影响了在控制语句中依赖关系的判断.因此,在利用复合语句的推理规则 R2 时,需做特别处理,类似于一般程序设计语言中的局部变量作用域,当遇到控制语句时需暂时屏蔽先行语句中对

左值的判断,在控制语句结束后再恢复.

对复合规则 R2,在生成验证条件时做如下标记与处理:若$\{P\}C_1\{Q\wedge \mathsf{LV}(x)\}$,$C_2$ 为条件分支语句或循环语句,且$\{Q\wedge \mathsf{LV}(x)\}C_2\{R\wedge \mathsf{LV}(x)\}$,则在将 C_1,C_2 复合时将 C_2 的三元组中的 $\mathsf{LV}(x)$标记为不可见,并在复合语句的后件中取消其不可见的标记,即:

$$\frac{\{P\}C_1\{Q\wedge \mathsf{LV}(x)\}\ \{Q\wedge \boxed{\mathsf{LV}(x)}\}C_2\{R\wedge \boxed{\mathsf{LV}(x)}\}}{\{P\}C_1;C_2\{R\wedge \mathsf{LV}(x)\}}.$$

标记的目的是在语句 C_2 的三元组中仅考虑从 C_2 局部来说变量 x 是否为左值;在 $C_1;C_2$ 结束执行时,作为一个整体在其中变量 x 是左值,故取消标记.

5.2 实 例

本节利用对引言中例 2 的验证说明依赖逻辑的应用.例 2 中的 fork 与 join 函数是 Linux 库函数**的简化版本,控制结构在网络服务程序中是较为普遍的.验证的主要目的是针对已经插入注释的程序,证明低级变量 L 对高级变量 H 存在依赖关系.图 5 是证明过程,为节约空间把谓词 Thread 记为 T.

1 %inside t_m
2 $H=1$;pid_t t;
3 $\{\mathsf{LV}(H)\wedge \mathsf{Inside}(t_m,H=1)\}$
4 $\{\mathsf{T}(t_m,\mathsf{LV}(H))\}$
5 if H
6 then $\{\mathsf{T}(t_m,H\neq 0\wedge \boxed{\mathsf{LV}(H)})\}$
7 $t=$fork {step(100);exit;}
8 $\{\mathsf{T}(t_m,\mathsf{LV}(t)\wedge \boxed{\mathsf{LV}(H)})*\mathsf{T}(t,\mathrm{emp})\}$
9 else $\{\mathsf{T}(t_m,H==0\wedge \boxed{\mathsf{LV}(H)})\}$
10 skip; $\{\mathsf{T}(t_m,H==0\wedge \boxed{\mathsf{LV}(H)})\}$
11 $\{\mathsf{T}(t_m,\mathsf{LV}(H)\wedge \mathsf{LV}(t)\oplus_H \mathrm{emp})*\mathsf{T}(t,\mathrm{emp})\}$
12 join t;
13 $\{\mathsf{T}(t_m,\mathsf{LV}(H)\wedge \mathsf{LV}(t)\oplus_H \mathrm{emp}\wedge \mathsf{Dep}(t,\{t_m,\mathrm{errno}\}))*\mathsf{T}(t,\mathsf{Done}(t))\}$
14 if errno == NEX
15 then $\{\mathsf{T}(t_m,\boxed{\mathsf{LV}(H)\wedge \mathsf{LV}(t)\oplus_H \mathrm{emp}}\wedge \mathsf{Dep}(t,\{t_m,\mathrm{errno}\})\wedge \mathrm{errno}==NEX)*\mathsf{T}(t,\mathsf{Done}(t))\}$
16 $L=1$;
17 $\{\mathsf{T}(t_m,\boxed{\mathsf{LV}(H)\wedge \mathsf{LV}(t)\oplus_H \mathrm{emp}}\wedge \mathsf{Dep}(t,\{t_m,\mathrm{errno}\})\wedge \mathrm{errno}==NEX\wedge \mathsf{LV}(L)\wedge L==1)*\mathsf{T}(t,\mathsf{Done}(t))\}$
18 else $\{\mathsf{T}(t_m,\boxed{\mathsf{LV}(H)\wedge \mathsf{LV}(t)\oplus_H \mathrm{emp}}\wedge \mathsf{Dep}(t,\{t_m,\mathrm{errno}\})\wedge \mathrm{errno}==0)*\mathsf{T}(t,\mathsf{Done}(t))\}$
19 $L=0$;
20 $\{\mathsf{T}(t_m,\boxed{\mathsf{LV}(H)\wedge \mathsf{LV}(t)\oplus_H \mathrm{emp}}\wedge \mathsf{Dep}(t,\{t_m,\mathrm{errno}\})\wedge \mathrm{errno}==0\wedge \mathsf{LV}(L))\wedge L==0)*\mathsf{T}(t,\mathsf{Done}(t))\}$
21 $\{\mathsf{T}(t_m,\mathsf{LV}(t)\oplus_H \mathrm{emp}\wedge \mathsf{Dep}(t,\{t_m,\mathrm{errno}\})\wedge (\mathsf{LV}(L)\wedge L==1\oplus_{\mathrm{errno}}(\mathsf{LV}(L)\wedge L==0)\wedge \mathsf{LV}(H)*\mathsf{T}(t,\mathsf{Done}(t))\}$
22 $\{\mathsf{T}(t_m,\mathsf{LV}(H)\wedge \mathsf{Dep}(H,t)\wedge \mathsf{Dep}(t,\{t_m,\mathrm{errno}\})\wedge \mathsf{Dep}(\mathrm{errno},L))*\mathsf{T}(t,\mathsf{Done}(t))\}$
23 $\{\mathsf{T}(t_m,\mathsf{LV}(H)\wedge \mathsf{Dep}(H,L)\wedge \mathsf{Dep}(t,t_m))*\mathsf{T}(t,\mathsf{Done}(t))\}$

Fig.5 An applied example of dependency logic

图 5 依赖逻辑的应用实例

断言 4 显示,变量 H 在线程 t_m 中为左值;断言 6~断言 10 中的 $\boxed{\mathsf{LV}(H)}$ 表示 $\mathsf{LV}(H)$不参与条件分支语句内的判断;断言 11 显示,线程标识符 t 依赖于变量 H;断言 13 显示,在执行 join t 之后,线程 t_m 及其内部变量 errno 均依赖于线程标识符 t;断言 17、断言 18 以及断言 20 显示,变量 L 由于处于条件分支语句中的原因依赖于线程 t_m

** 如果是进程并发,则 join 对应系统调用 waitpid;若为线程并发,则 fork 对应 Pthread 库中的 Pthread_create,而 join 对应 Pthread_join.

内部变量 errno;断言 21~断言 23 则是利用 Dep 的传递性推理出变量 L 对变量 H 的依赖.

注:定义程序逻辑只是整个程序验证工作的一部分,将其应用于实际程序还需要后续研发的支持.依赖逻辑的应用复杂性来源于 3 个方面:注释插入、验证条件生成、验证条件的证明.首先,注释插入技术和验证条件生成算法已经相当成熟,有现成的开源工具可供二次开发;其次,在定理证明器中证明验证条件,一般有两种技术路线:一是利用相应的证明策略语言(类似 Coq 工具的 Ltac 语言)开发相应的证明脚本,交互式地实现证明过程;二是直接利用 SMT 等全自动证明工具证明验证条件.这些为我们后续研发提供了强有力的工作基础.

6 结 论

本文给出了在线程程序中通过不恰当的线程管理引发非法信息流的例子,首次提出了一种可以对具有副作用的线程管理函数以及相关控制流进行推理的依赖逻辑.该逻辑以变量、线程标识符之间的依赖关系为直接的推理目标,利用依赖关系表达了程序中的信息流.后续的研究工作包括开发依赖逻辑的验证条件生成算法;研究相应的定理证明策略;扩展语言语法,研究一般性过程调用等语法构件对依赖关系的影响等.

致谢 感谢审稿人的仔细评阅和中肯意见.

References:

[1] Smith G. Improved typings for probabilistic noninterference in a multi-threaded language. Journal of Computer Security, 2006, 14(6):591−623.

[2] Barthe G, Nieto LP. Formally verifying information flow type systems for concurrent and thread systems. In: Proc. of the 2004 ACM Workshop on Formal Methods in Security Engineering. Washington: ACM Press, 2004. 13−22. [doi: 10.1145/1029133.1029136]

[3] Russo A, Hughes J, Naumann D, Sabelfeld A. Closing internal timing channels by transformation. In: Proc. of the 11th Asian Computing Science Conf. on Advances in Computer Science: Secure Software and Related Issues. Tokyo: Springer-Verlag, 2007. 120−135. [doi: 10.1007/978-3-540-77505-8_10]

[4] Sabelfeld A. The impact of synchronisation on secure information flow in concurrent programs. In: Bjφrner D, Broy M, Zamulin A, eds. Proc. of the 4th Int'l Andrei Ershov Memorial Conf. on Perspectives of System Informatics. Akademgorodok: Springer-Verlag, 2001. 225−239.

[5] Stevenc WR, Fenner B, Rudoff AM. UNIX Network Programming, Vol.1. 3rd ed., Beijing: China Machine Press, 2004. 121−151.

[6] Dodds M, Feng XY, Parkinson M, Vafeiadis V. Deny-Guarantee reasoning. In: Castagna G, ed. Proc. of the Programming Languages and Systems. Berlin/Heidelberg: Springer-Verlag, 2009. 363−377. [doi: 10.1007/978-3-642-00590-9_26]

[7] Feng XY. Local rely-guarantee reasoning. In: Proc. of the 36th Annual ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages. Savannah: ACM Press, 2009. 315−327. [doi: 10.1145/1480881.1480922]

[8] Honda K, Vasconcelos V, Yoshida N. Secure information flow as typed process behaviour. In: Proc. of the 9th European Symp. on Programming Languages and Systems. Springer-Verlag, 2000. 180−199. [doi: 10.1007/3-540-46425-5_12]

[9] Honda K, Yoshida N. A uniform type structure for secure information flow. ACM Trans. on Programming Languages System, 2007, 29(6):31. [doi: 10.1145/1286821.1286822]

[10] Mantel H, Sands D, Sudbrock H. Assumptions and guarantees for compositional noninterference. In: Proc. of the IEEE 24th Computer Security Foundations Symp. (CSF). IEEE, 2011. 218−232. [doi: 10.11.09/CSF.2011.22]

[11] Mantel H, Sudbrock H. Flexible scheduler-independent security. In: Proc. of the 15th European Conf. on Research in Computer Security. Athens: Springer-Verlag, 2010. 116−133. [doi: 10.1007/978-3-642-15497-3_8]

[12] Russo A, Sabelfeld A. Securing interaction between threads and the scheduler in the presence of synchronization. Journal of Logic and Algebraic Programming, 2009,78(7):593−618. [doi: 10.1016/j.jlap.2008.09.003]

[13] Volpano D, Smith G. Probabilistic noninterference in a concurrent language. In: Proc. of the IEEE 11th Computer Security Foundations Workshop. IEEE, 1998. 34−43. [doi: 10.1109/CSFW.1998.683153]

[14] Zdancewic S, Myers AC. Observational determinism for concurrent program security. In: Proc. of the IEEE 16th Computer Security Foundations Workshop. IEEE Press, 2003. 29−43. [doi: 10.1109/CSFW.2003.1212703]

[15] Huisman M, Worah P, Sunesen K. A temporal logic characterisation of observational determinism. In: Proc. of the IEEE 19th Computer Security Foundations Workshop. IEEE Press, 2006. 1−13. [doi: 10.1099/CSFW.2006.6]

[16] Stefan D, Russo A, Buiras P, Levy A, Mitchell JC, Mazières D. Addressing covert termination and timing channels in concurrent information flow systems. In: Proc. of the 17th ACM SIGPLAN Int'l Conf. on Functional Programming. Copenhagen: ACM Press, 2012. 201−214. [doi: 10.1145/2364527.2364557]

[17] Vaughan JA, Millstein T. Secure information flow for concurrent programs under total store order. In: Proc. of the IEEE 25th Computer Security Foundations Symp. (CSF). IEEE, 2012. 19−29. [doi: 10.1109/CSF.2012.20]

[18] Krinke J. Context-Sensitive slicing of concurrent programs. SIGSOFT Software Engineering Notes, 2003,28(5):178−187. [doi: 10.1145/940071.940096]

李沁(1976－),男,安徽芜湖人,博士,副教授,主要研究领域为信息安全,形式化方法.
E-mail: linuxos2@163.com

曾庆凯(1963－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为信息安全,分布计算.
E-mail: zqk@nju.edu.cn

袁志祥(1973－),男,副教授,主要研究领域为形式化方法.
E-mail: zxyuan@ahut.edu.cn