软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2012,23(8):1935−1949 [doi: 10.3724/SP.J.1001.2012.04132]

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

一种基于变量可达向量的链表抽象方法*

李仁见[1+], 刘万伟[2], 陈立前[1], 王 戟[1]

[1](国防科学技术大学 计算机学院 并行与分布处理国家重点实验室,湖南 长沙 410073)

[2](国防科学技术大学 计算机学院 计算机科学与技术系,湖南 长沙 410073)

List Abstraction Method Based on Variable Reachability Vector

LI Ren-Jian[1+], LIU Wan-Wei[2], CHEN Li-Qian[1], WANG Ji[1]

[1](National Key Laboratory of Parallel and Distributed Processing, College of Computer, National University of Defense Technology, Changsha 410073, China)

[2](Department of Computer Science and Technology, College of Computer, National University of Defense Technology, Changsha 410073, China)

+ Corresponding author: E-mail: li.renjian@gmail.com

Li RJ, Liu WW, Chen LQ, Wang J. List abstraction method based on variable reachability vector. *Journal of Software*, 2012,23(8):1935−1949 (in Chinese). http://www.jos.org.cn/1000-9825/4132.htm

Abstract: This paper presents a list abstraction method. This method enjoys low space overhead by storing the edges between nodes in a list implicitly in a compact manner. It also enjoys high precision by keeping the length of lists. Specifically, the study introduces a so-called variable reachability vector to encode the reachability properties of variables to list nodes, and use variable reachability vector set with counters as an abstract model for each list state. Based on this model, abstract semantics are then defined for basic list operations. This approach could model all singly-linked lists including cyclic cases after a simple extension is brought in. On this basis, the study designs and implements a symbolic execution framework, which could automatically analyze programs manipulating lists automatically. Finally, this approach is applied to analyzing some typical list-manipulating programs for non-trivial properties, such as run-time errors, length related properties and termination.

Key words: list abstraction; symbolic execution; list manipulating program; variable reachability vector

摘 要: 提出了一种链表抽象表示方法.该方法隐式存储链表结点之间的边信息,并采用了一种紧致的链表状态表示,存储开销较低,且维护了链表长度信息,精确度较高.具体而言,根据变量对链表结点的可达性质定义了变量可达向量,采用带计数的变量可达向量集描述链表的形态及数量性质,并定义了基本链表操作的抽象语义.通过简单扩展,该方法可以建模包括环形链表在内的所有单向链表.最后,为了验证该链表抽象方法的正确性,在符号执行框架中进行实验,并对常见链表操作程序的运行时错误、长度相关性质等关键性质进行了分析与验证.

关键词: 链表抽象方法;符号执行;链表操作程序;变量可达向量

中图法分类号: TP311 **文献标识码**: A

* 基金项目: 国家自然科学基金(90818024, 60725206); 国家高技术研究发展计划(863)(2011AA010106)
收稿时间: 2010-12-06; 修改时间: 2011-07-21, 2011-09-22; 定稿时间: 2011-10-08

近年来,针对链表结构的抽象方法以及链表操作程序的自动化分析验证技术引起了国内外研究人员的极大兴趣[1−9].链表是进行堆内存操作时最经常采用的数据结构之一.例如,在操作系统内核代码中就大量采用链表来实现任务调度队列等关键数据结构.此外,很多其他常用数据结构,例如堆和栈等,都可以间接地采用链表来实现.包括链表在内的递归可变数据结构都拥有类似图的特性,均可看作包含若干无名结点和边的图.如何高效地表示和操作图中的结点与边,是链表分析验证方法能否高效自动化实现的一个关键问题.已有的链表抽象表示方法有抽象形态图[1,7−11]、三值逻辑[12]、谓词抽象[7,13,14]等.为了描述链表状态,除了指针变量指向信息以外,已有方法大都需要对无名结点显式命名,并显式地或通过谓词的方式记录所有的边.结点的合并抽象可以将长度不定的链表抽象成一个最小化的抽象形态图,然而对边信息的存储和操作仍将会带来较大的存储开销.

此外,经典形态分析方法仅关注内存结点的形态性质,忽略了其数量信息.但在有些情况下,我们必须结合形态性质与数量性质不变式来验证程序的安全性质以及活性性质.如图 1 中所示的链表操作过程,该示例程序来自文献[11],其语义是将输入的链表逆向复制到一个备份链表中,然后同时删除这两个链表.对于此示例程序,一个重要的数量特性是两个链表的长度始终相同.经典形态分析工具(如 TVLA[12])无法验证该程序的内存安全性及终止性,将会在第 13 行及程序结束时分别产生空指针释放及内存泄漏误报.

```
void copy_and_delete(List* xList) {
1: List *yList, *pList, *qList;          8: yList=xList;
2: yList=xList; qList=pList=null;        9: while (yList!=null) {
3: while (yList!=null) {                 10:   yList=yList→next;
4:    yList=yList→next;                  11:   qList=qList→next;
5:    qList=malloc(·);                   12:   free(xList); xList=yList;
6:    qList→next=pList;                  13:   free(pList); pList=qList;
7:    pList=qList;}                      14: }}
```

Fig.1 A sample example

图 1 一个示例程序

针对上述问题,本文提出了一种紧致的链表抽象方法.该方法隐式存储链表结点之间的边信息,并采用了一种紧致的链表状态表示方法.通过为变量可达向量附加计数信息,并在抽象语义中相应地加以维护,该方法可以自动地维护链表长度信息,提高分析精度,减少误报.例如,文中给出的实验框架即可避免上述误报.

本文第 1 节介绍相关工作.第 2 节给出链表操作程序的语法.第 3 节首先从最简单的单向无环链表入手,介绍基于变量可达向量的链表抽象方法;然后,将其扩展至包括带环链表在内的各种单向链表,并提出带计数的变量可达向量集,作为链表的一种精确抽象模型.第 4 节给出链表操作的抽象语义.第 5 节介绍原型工具的设计框架及实验结果.最后进行总结和下一步的工作目标.

1 相关工作

已有的针对链表的分析与验证方法,大部分基于形态图对链表状态进行建模.文献[1,10]以抽象形态图表示链表状态,但未维护链表长度信息.文献[10]与我们的思想非常接近,为堆内存中每个链表结点维护一个可达变量集合;但从表示形式上来讲,文献[10]维护的仍是一种抽象形态图结构,且需要针对每个结点维护包括 unique 谓词以及 shared 谓词等辅助谓词,才能完整地描述链表抽象形态.为了提高抽象模型精度,文献[8,9,11]采用带计数的抽象形态图对链表状态进行建模,其计数变量的作用与本文针对变量可达向量维护的链表计数信息类似.

TVLA[12]采用三值逻辑的方法对内存形态进行抽象,需为每个状态保存若干一元谓词和二元谓词.存储变量指向信息的一元谓词,其定义域是所有结点;而存储边信息的二元谓词,其定义域是所有可能的结点对.因此,本质上他们的方法同样需要保存变量的指向关系以及结点之间的指向关系.文献[7,13,14]采用谓词抽象的方法描述堆内存形态.缩减谓词数目是采用谓词抽象方法进行形态分析时需要解决的主要问题之一.本文中方法可看作谓词逻辑与形态图的一种特殊结合形式,采用尽量少的谓词,获得了相对精确的抽象模型.分离逻辑也可用于建模链表形态[15].使用分离逻辑公式作为抽象空间,能够更好地对程序内存进行局部推理.文献[3,5,16]研究

了针对链表的分离逻辑片段,但均未对链表进行特殊抽象处理,可看作采用分离逻辑描述具体链表形态.文献[17]中的链表抽象方法相对特殊,将单向链表表示为一个有穷字母表上的字,通过有穷状态自动机描述抽象链表状态,最后采用模型检验的方法验证链表操作程序的 safety 性质.

与上述链表抽象方法不同,本文引入了一种新的链表抽象表示方法.该方法采用了高效的编码方式,可以隐式地记录链表中的边信息,存储开销较低;且通过位向量操作来实现链表操作抽象语义,应该存在高效的实现算法;同时,文中方法维护了链表长度信息,相比文献[1,7,10,12−14,17]中采用的链表模型更加精确.

2 链表操作的抽象语法

单向链表中,每个链表结点通过一个 *next* 指针指向下一个结点.数据域的操作不会影响链表的形态.为简洁起见,文中表述时省略了对链表结点中数据域的操作,仅给出链表操作的一个最小语法集合,如图 2 所示.*PVar* 是程序中指向链表的指针变量集合.允许的语法结构包括赋值语句、条件分支语句和循环语句.为了简化抽象语义,与文献[9]一样,本文假设赋值语句 $p=q,p=q\rightarrow next,p\rightarrow next=q$ 中变量 p,q 为不同的变量,否则可以通过引入临时变量的方法进行等价转换.此外,为了简化抽象语义,我们还假设在对指针变量 p 或者 $p\rightarrow next$ 进行赋值时,首先执行了 $p=null$ 或者 $p\rightarrow next=null$ 操作.这些假设显然不会影响程序的正确性语义.

$p,q\in PVar$

$AsgnStmnt::=p=null|p=q|p=q\rightarrow next|p\rightarrow next=null|$
$p\rightarrow next=q|p=malloc(\cdot)|free(p)$

$Cond::=p==q|p==null|$
$\neg Cond|Cond_1\vee Cond_2|Cond_1\wedge Cond_2|true|false$

$BranchStmnt::=$**if** $Cond$ then $\{Stmnt;\}^*$ [**else** $\{Stmnt;\}^*$] **fi**

$WhileStmnt::=$**while** $Cond$ **do** $\{Stmnt;\}^*$ **od**

$Stmnt::=AsgnStmnt|BranchStmnt|WhileStmnt$

$Program::=\{Stmnt\}^*$

Fig.2 Abstract syntax for list operations

图 2 链表操作的抽象语法

3 基于变量可达向量的链表抽象方法

具体形态图可以完全描述链表的运行时状态.下面首先给出经典形态图的定义.

定义 1. 链表的形态图 *SG* 是一个三元组$\langle N,V,E\rangle$,其中,N 是变量以及链表结点的集合,V 是程序中变量的集合,且 $V\subseteq N$.我们引入一个特殊结点 *NULL* 来表示空指针,并采用记号 N_{nil} 表示 $N\cup\{NULL\}$.$E:N\rightarrow(N_{nil}-V)$是一个函数,用于描述形态图中指针变量的指向关系以及链表结点间的 *next* 指向关系.若结点 $n\in V$,则称其为变量结点,否则称其为链表结点.

图 3(a)给出了一个链表,其形态图可以表示为 $SG_1=\langle N,V,E\rangle,u,v,p,q$ 是指向链表的指针变量,而 $N=\{u,v,p,q,n_1,n_2,n_3,n_4,n_5\},V=\{u,v,p,q\},E(u)=n_1,E(p)=n_2,E(q)=n_2,E(v)=n_5,E(n_1)=n_4,E(n_2)=n_3,E(n_3)=n_4,E(n_4)=n_5,E(n_5)=NULL\}$.

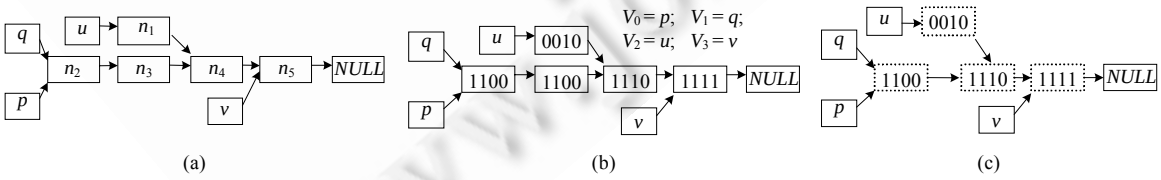

Fig.3 An example for shape graph and its VRVs

图 3 形态图及变量可达向量示例

可以看出,采用形态图建模链表状态时,需对所有无名链表结点显式命名,并保存结点间的所有指向关系.在程序运行过程中,链表结点与边的数目不受限制,其形态图也可能变得极为庞大,产生极大的存储开销.针对

这个问题,本文提出基于变量可达向量的链表抽象方法,根据变量对结点的可达性质描述与记录链表中的无名结点,同时避免了链表结点之间的边信息显式存储.由于环形链表性质相对特殊,首先在第 3.1 节中针对单向无环链表定义该抽象方法,然后在第 3.2 节中扩展至带环的情形,最后在第 3.3 节中给出维护链表长度信息的链表抽象模型.

3.1 单向无环链表的变量可达向量抽象方法

程序运行时,链表中的结点及边组成一个连通图.单向链表中每个结点最多有 1 条出边,但允许存在多条入边.对单向无环链表,由于不存在环路,其形态总是一棵逆向树,树的叶子结点都是变量结点,枝干结点都是链表结点.为了便于描述,首先基于形态图定义链表中结点之间的可达关系谓词.给定链表形态图 $SG=\langle N,V,E\rangle$,

$$Reach(n,n')\triangleq\exists k\in\mathbf{N},\ \forall 0\leqslant i\leqslant k,\ n_i\in N\wedge n_0=n,\ n_k=n'\wedge\forall 0\leqslant j<k,E(n_j)=n_{j+1}.$$

对任意 $n,n'\in N$,$Reach(n,n')$取值为真,当且仅当形态图中存在 1 条从 n 到 n'的路径.我们对程序运行时所有可能指向链表的指针变量进行排序,任意的 $0\leqslant i\leqslant|V|-1$,以 V_i 表示其中的第 i 个变量.若 $Reach(V_i,n)$为真,则称变量 V_i 可达结点 n.在不考虑可能别名而只考虑必然别名时,对任意链表结点 $n\in(N-V)$,所有变量是否可达 n 的性质可以通过一个布尔向量来表示.

定义 2. 任意链表结点 $n\in(N-V)$,其变量可达向量(variable reachability vector,简称 VRV)γ_n 是一个$|V|$维的布尔向量,$\gamma_n\in\{0,1\}^{|V|}$且$\gamma_n[i]=1$,当且仅当 $Reach(V_i,n)=true$.

记所有链表结点 $n\in(N-V)$的变量可达向量所构成的集合为Γ.我们始终假设在常规的变量可达向量集合Γ中不包含全 0 的变量可达向量 $0^{|V|}$,因为该向量表示没有任何变量可以访问的内存结点,也就是发生了内存泄漏.对任意$\gamma\in\Gamma$,若$\gamma[i]=1$,也称变量 V_i 可达γ.我们以 $R_\gamma\triangleq\{i\in\mathbf{N}|\gamma[i]=1\}$表示可达$\gamma$的所有变量对应位的集合,而以$\Gamma_i=\{\gamma|\gamma[i]=1\}$表示变量 V_i 可达的所有变量可达向量的集合.

针对图 3(a)中的形态图,按照 *pquv* 的顺序对指针变量进行排序,可以获得所有链表结点的变量可达向量,如图 3(b)所示.此时,链表对应的变量可达向量集Γ为{1100,0010,1110,1111},按照定义,$R_{1100}=\{0,1\}$,$R_{0010}=\{2\}$,$R_{1110}=\{0,1,2\}$,$R_{1111}=\{0,1,2,3\}$,而$\Gamma_0=\{1100,1110,1111\}$,$\Gamma_1=\{1100,1110,1111\}$,$\Gamma_2=\{0010,1110,1111\}$,$\Gamma_3=\{1111\}$.

定义 3(变量可达向量间的可达关系). 给定两个变量可达向量$\gamma_1,\gamma_2\in\Gamma$:

- 若 $R_{\gamma_1}\subseteq R_{\gamma_2}$,则称$\gamma_1$ 可达γ_2,并简记为$\gamma_1\subseteq\gamma_2$;
- 若 $R_{\gamma_1}\subset R_{\gamma_2}$,则称$\gamma_1$ 严格可达γ_2,并简记为$\gamma_1\subset\gamma_2$;
- 若 $R_{\gamma_1}\cap R_{\gamma_2}=\varnothing$,则称$\gamma_1$ 与γ_2 不可达,并简记为$\gamma_1\cap\gamma_2=\varnothing$.

下面讨论变量可达向量间可达关系与链表形态图中结点之间路径的对应关系.

引理 1. 给定链表中两个结点 n_1,n_2,若形态图中存在一条从 n_1 到 n_2 的路径,则必有 $\gamma_{n_1}\subseteq\gamma_{n_2}$.

证明:根据定义 2 和定义 3 即可证明. □

引理 2. 给定链表中两个不同结点 n_1,n_2,若$\neg(\gamma_{n_1}\cap\gamma_{n_2}=\varnothing)$,则链表的形态图中存在一条从 n_1 到 n_2 或者从 n_2 到 n_1 的路径(证明在附录中给出).

定义 4. 给定两个结点 n_1,n_2,若 $\gamma_{n_1}=\gamma_{n_2}$,则称这两个链表结点变量可达等价.

定理 1. 给定链表中两个非变量可达等价的结点 n_1,n_2,则链表形态图中存在一条从结点 n_1 到 n_2 的路径,当且仅当 $\gamma_{n_1}\subset\gamma_{n_2}$.

证明:"⇒":设存在一条从结点 n_1 到 n_2 的路径.由引理 1 可知 $\gamma_{n_1}\subseteq\gamma_{n_2}$,因 $\gamma_{n_1}\neq\gamma_{n_2}$,故 $\gamma_{n_1}\subset\gamma_{n_2}$;

"⇐":因不考虑所有变量均不可达的结点,故 γ_{n_1},γ_{n_2} 均不为 $0^{|V|}$,$\gamma_{n_1}\cap\gamma_{n_2}\neq\varnothing$.由引理 2 可知,必存在一条从 n_1 到 n_2 或从 n_2 到 n_1 的路径.由引理 1 可知,不可能存在从 n_2 到 n_1 的路径,故必存在一条从 n_1 到 n_2 的路径. □

定义 5. 变量可达向量集Γ是协调的,当且仅当任意两个变量可达向量$\gamma_1,\gamma_2\in\Gamma$,要么$\gamma_1\cap\gamma_2=\varnothing$,要么$\gamma_1\subset\gamma_2$ 或$\gamma_2\subset\gamma_1$.

定理 2. 协调的变量可达向量集Γ满足$|\Gamma|\leqslant 2|V|-1$(证明在附录中给出).

定理 3. 单向无环链表的变量可达向量集是协调的.

证明:由引理 1 与引理 2 可证. □

依据变量可达向量的定义及构造规则,在保证变量及其编码顺序不变的前提下,总可以针对每个单向无环链表生成唯一的一个协调的变量可达向量集Γ.由定理 1 可知,Γ中变量可达向量之间的严格可达关系(一个偏序关系)隐式地描述了链表形态图中对应结点之间的路径可达关系.给定变量 V_i,可以依据变量可达向量间的严格可达关系对Γ_i 中所有元素排序,并以 γ_i^0 表示其中的最小元.易知,γ_i^0 就是 V_i 当前指向的链表结点所对应的变量可达向量.当每个变量可达向量严格对应一个抽象结点时,Γ_i 中向量之间的全序关系则可以完全描述抽象结点之间的指向关系.因此,我们可以采用协调的变量可达向量集作为链表的一个紧致的抽象模型.该模型维护了非变量可达等价结点之间的可达关系,对于变量可达等价的结点,则保守地认为存在任意的一条单向路径,将所有结点连接在一起.

例如,根据图 3(b)得到的变量可达向量集Γ={1100,0010,1110,1111}保留了图 3(a)中链表状态的大部分形态信息,其信息量等价于图 3(c)中给出的最小化的抽象形态图.图 3(c)中链表结点采用虚线框标识,表示该向量代表至少 1 个链表结点.例如,由 1100⊂1110⊂1111 可知,变量 p 与 q 指向同一个链表结点(二者存在别名关系),且从这个结点出发,经过若干个结点之后可以依次到达变量 u 及变量 v 指向的结点.

3.2 对带环链表的扩展策略

环形链表是现实程序中经常采用的一种扩展链表结构.每个环形链表对应形态图中一个带回路的最大连通子图.我们称形态图中链表结点构成的回路为链表环,简称为环.每个环形链表从形态上可看作若干逆向树连接到同一个链表环上.若仍按照第 3.1 节中的方法进行抽象,则所有在环上的链表结点对应的变量可达向量均相同,无法区分变量具体指向的结点及环上结点之间的相对可达性质.

解决这个问题的基本思路是,将环从某条 *next* 边处“切断”,并基于第 3.1 节中方法对获得的“单向无环”链表进行抽象表示,然后为每个变量可达向量γ附加一个特殊标记位 c,构成扩展变量可达向量 $\hat{\gamma}$,其中,$\hat{\gamma}[c]=1$当且仅当对应链表结点在某个环上.“切断”操作的作用是设置一个变量可达信息传播终点,而特殊标记位 c 则标识了环上结点所对应的变量可达向量.给定扩展变量可达向量 $\hat{\gamma}$,称不考虑标记 c 而得到的向量γ为 $\hat{\gamma}$ 的投影向量.

图 4(a)给出了一个环形链表的示例,链表从 n_5 到 n_2 的 *next* 边被“切断”,构成一个“单向无环”链表,按照 *pquv* 的顺序,并在最后添加环标记位 c,可得扩展变量可达向量集 $\hat{\Gamma}$={00100,11001,11101,11111}.

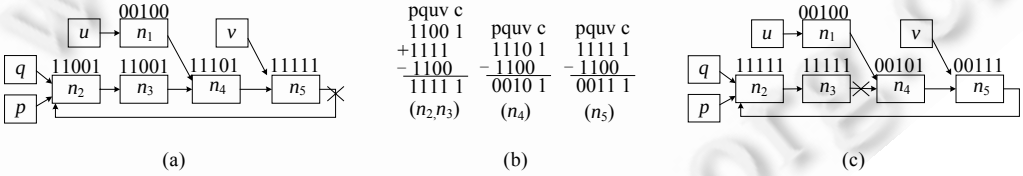

Fig.4 An example for cyclic list

图 4 环形链表示例

对“切断”之后所得的扩展变量可达向量集 $\hat{\Gamma}$,给定任意 $\hat{\gamma}\in\hat{\Gamma}$,定义“切断”之后可达 $\hat{\gamma}$ 的变量集合 $R_{\hat{\gamma}}\triangleq\{i\in\mathbf{N}\mid\hat{\gamma}[i]=1\}$.注意,此时 $R_{\hat{\gamma}}$ 可能包含标记位 c,我们可以等价地认为引入了一个新的变量 V_c,指向了执行“切断”操作的 *next* 边的目的结点.例如,在图 4(a)中可认为变量 V_c 指向了结点 n_2,而在图 4(c)中则可以理解为变量 V_c 指向结点 n_4.定义变量 V_i 可达的扩展变量可达向量集合 $\hat{\Gamma}_i=\{\hat{\gamma}\mid\hat{\gamma}[i]=1\}$.同时,对任意 $\hat{\gamma}_1,\hat{\gamma}_2\in\hat{\Gamma}$,沿用第 3.1 节中定义的变量可达向量间可达关系⊆以及⊂.与单向无环链表一样,$\hat{\Gamma}_i$ 中的元素之间存在全序关系,保持了扩展变量可达向量之间的相对可达特性,其最小元 $\hat{\gamma}_i^0$ 同样代表了变量 V_i 当前指向的结点所对应的扩展变量可达向量.

引理 3. 在单向链表中,一个结点可能且仅可能在 1 个链表环上.

证明:依据单向链表的特性以及图论理论,采用反证法可以很容易地证明该引理的正确性. □

定义 6. 给定两个不同的扩展变量可达向量 $\hat{\gamma}_1,\hat{\gamma}_2$,若 $\hat{\gamma}_1$ 与 $\hat{\gamma}_2$ 位于同一个链表环上,则称 $\hat{\gamma}_1$ 与 $\hat{\gamma}_2$ 是环可达等价的,并简记为 $\hat{\gamma}_1 \doteq \hat{\gamma}_2$.等价地, $\hat{\gamma}_1 \doteq \hat{\gamma}_2$ 当且仅当 $\hat{\gamma}_1[c]=1 \wedge \hat{\gamma}_2[c]=1 \wedge (\hat{\gamma}_1 \subset \hat{\gamma}_2 \vee \hat{\gamma}_2 \subset \hat{\gamma}_1)$.

定义集合 $\underline{\hat{\Gamma}}_i = \{\hat{\gamma} \in \hat{\Gamma} - \hat{\Gamma}_i \mid \exists \hat{\gamma}' \in \hat{\Gamma}_i . \hat{\gamma}' \doteq \hat{\gamma}\} \cup \hat{\Gamma}_i$. 显然, $\underline{\hat{\Gamma}}_i$ 考虑了环可达等价关系,是变量 V_i 真正可达的结点所对应的变量可达向量集合.而相对来说, $\hat{\Gamma}_i$ 则表示不考虑环可达等价关系时的变量可达向量集合.

如前所述,在内存形态的每个可能带环的链表中,标记位 c 的作用相当于引入了一个变量 V_c.因为程序运行时可能操作多个链表,且复用同一个链表环标记位 c,所以在定义协调性时无法完全采用第 3.1 节中的定义.为此,对协调性的定义进行扩展:对于任意两个扩展变量可达向量,要么二者的投影向量相互不可达,要么这两个扩展变量可达向量之间是可达的.

定义 7. 扩展变量可达向量集 $\hat{\Gamma}$ 是环扩展协调的,当且仅当任给 $\hat{\gamma}_1,\hat{\gamma}_2 \in \hat{\Gamma}$ (设其投影向量分别为 γ_1,γ_2),要么 $\gamma_1 \cap \gamma_2 = \varnothing$,要么 $\hat{\gamma}_1 \subset \hat{\gamma}_2 \vee \hat{\gamma}_2 \subset \hat{\gamma}_1$.

定理 4. 单向链表的环扩展变量可达向量集是环扩展协调的.

证明:给定扩展变量可达向量集 $\hat{\Gamma}$,取任意 $\hat{\gamma}_1,\hat{\gamma}_2 \in \hat{\Gamma}$,设其投影向量分别为 γ_1,γ_2.

(i) 若内存形态图中存在多个链表且 $\hat{\gamma}_1,\hat{\gamma}_2$ 位于两个不同链表之中,则由于每个链表对应一个最大连通子图,结合引理 2 可知,必然满足 $\gamma_1 \cap \gamma_2 = \varnothing$.

(ii) 设 $\hat{\gamma}_1,\hat{\gamma}_2$ 位于同一个链表,取链表中结点对应的扩展变量可达向量子集合,记为 $\hat{\Gamma}_{1,2}$,则可等价地认为引入了一个特殊的变量 V_c.如前所述,可将 $\hat{\Gamma}_{1,2}$ 看作在变量集合 $V \cup \{V_c\}$ 之上的一个普通的变量可达向量集.按定义 5 及定理 3 可知,要么 $\hat{\gamma}_1 \cap \hat{\gamma}_2 = \varnothing$,要么 $\hat{\gamma}_1 \subset \hat{\gamma}_2$ 或 $\hat{\gamma}_2 \subset \hat{\gamma}_1$.而若 $\hat{\gamma}_1 \cap \hat{\gamma}_2 = \varnothing$,则自然有 $\gamma_1 \cap \gamma_2 = \varnothing$. □

定理 5. 给定扩展变量可达向量集 $\hat{\Gamma}$,若 $\hat{\Gamma}$ 是环扩展协调的,则 $|\hat{\Gamma}| \leqslant 2|V|$ (证明在附录中给出).

执行“切断”操作的 *next* 边的源结点是变量可达信息传播的终点,我们称其对应的变量可达向量为环扩展根向量.根据定义 7 及定理 4 可知,同一个环上所有的扩展变量可达向量中,环扩展根向量为极大元,且该环上所有结点的变量可达特性实际上均与这个环扩展根向量相同.给定在某个环上的任意变量可达向量 $\hat{\gamma}$,我们以 $Cycle_{\hat{\gamma}}$ 标记 $\hat{\gamma}$ 所在的环,并以 $CycleVar_{\hat{\gamma}}$ 记录所有可达或指向环 $Cycle_{\hat{\gamma}}$ 的变量.若假设该环上的环扩展的根向量为 $\hat{\gamma}^{root}$, 则必有 $CycleVar_{\hat{\gamma}} = R_{\hat{\gamma}^{root}} - \{c\}$.

从不同 *next* 边处将环形链表“切断”,可得到不同的扩展变量可达向量集,这也相当于选取不同的环扩展根向量.事实上,对一个链表状态,可以选择任意的扩展变量可达向量,将其转换为环扩展根向量,从而获得当前链表状态的另一种等价抽象表示.不妨把这个操作定义为 $rotate(\hat{\Gamma},\hat{\gamma})$,其中, $\hat{\Gamma}$ 为已有的扩展变量可达向量集,而 $\hat{\gamma}$ 为新选中的要转换为环扩展根向量的向量.其算法如图 5 所示.

```
Algorithm: rotate( Γ̂, γ̂ )
输入: Γ̂, γ̂.   /*初始的变量可达向量集与选中的变量可达向量*/
输出: Γ̂.     /*变换之后得到的变量可达向量集*/
1:   if ( γ̂[c] == 0 ) return ( Γ̂ );   /*不在环上的变量可达向量*/
2:   for each  γ̂' ∈ Γ̂   {
3:      if ¬( γ̂' ≐ γ̂ ) continue;     /*不在同一个环上的变量可达向量*/
4:      else {
5:          if ( γ̂ ⊂ γ̂' )
6:              for each i∈ CycleVar_γ̂   { γ̂'[i] = γ̂'[i] − γ̂[i] ;}
7:          else
8:              for each i∈ CycleVar_γ̂   { γ̂'[i] = γ̂'[i] + 1 − γ̂[i] ;}
9:    }}
10:  return ( Γ̂ );
```

Fig.5 Algorithm for rotate

图 5 rotate 算法

算法仅对在环 $Cycle_{\hat{\gamma}}$ 上的变量可达向量中包含于 $CycleVar_{\hat{\gamma}}$ 的变量的可达性质产生影响:针对 $CycleVar_{\hat{\gamma}}$ 中每一位,将环上的每个扩展变量可达向量 $\hat{\gamma}'$ 与 $\hat{\gamma}$ 对该位的取值相减,即可得 $\hat{\gamma}'$ 经 rotate 操作之后的表示.根据扩展向量的可达关系及其协调性定义,若 $\hat{\gamma}\subset\hat{\gamma}'$,则针对 $CycleVar_{\hat{\gamma}}$ 中每一位的减法操作都不会出现 0 减 1 的情况;而若 $\hat{\gamma}\lhd\hat{\gamma}'$,则由于二者在同一个环上,必有 $\hat{\gamma}'\subseteq\hat{\gamma}$.此时,算法针对 $CycleVar_{\hat{\gamma}}$ 中每一位的减法操作都首先进行加 1,从而不会产生负数的情况,也不会产生取值为 2 的情况.

例如,若要将图 4(a)中的链表抽象表示改为从结点 n_3 到 n_4 的 *next* 边"切断",则采用 rotate 算法可得如图 4(c)所示的链表形态表示.算法执行过程如图 4(b)所示.在针对 n_2,n_3 对应的扩展变量可达向量进行转换时,由于 11001=$\hat{\gamma}\lhd\hat{\gamma}'$=11001,所以执行 rotate 算法中第 8 行的操作,在对每个变量位执行减法操作时,首先进行加 1 操作.而对 n_4,n_5 对应的扩展变量可达向量进行转换时,由于 $\hat{\gamma}\subset\hat{\gamma}'$,所以执行 rotate 算法中第 6 行的操作,直接对每个变量位执行减法操作.此外,由于结点 n_1 不在环 $Cycle_{11001}$ 上(即¬(00100≐11001)),因此算法将在第 3 行处终止对其进行转换.

"切断"操作是为了处理带环链表的情况,轮转操作可以保证"切断"的边是可以任意选取的.切断操作保持了语法和语义的正确性以及方便统一表示.文中给出的切断操作可以保持变量之间的相对可达性质,而环标记位的引入可以标识出所有在环上的链表结点.通过引入上述环扩展策略,我们可以描述所有单向链表的形态.

按照上述环扩展策略,第 3.1 节中的单向无环链表可看作在所有变量可达向量的尾部同时添加一个始终为 0 的标记位.此时,任意扩展变量可达向量 $\hat{\gamma}$ 及其投影向量 γ 必满足 $R_{\hat{\gamma}}=R_{\gamma}$.显然,环扩展策略不会影响单向无环链表情形的正确性.在后文描述中,我们不再区分变量可达向量与扩展变量可达向量,统称为变量可达向量,其集合统称为变量可达向量集.同时,也不区分变量可以到达链表结点 n 与变量可以到达对应变量可达向量 $\hat{\gamma}_n$ 的概念.

3.3 带计数的变量可达向量抽象方法

变量可达向量集可以作为链表的一个紧致的抽象模型,其保存的信息等价于最小化的抽象形态图.同时,我们还可根据需要为每个变量可达向量维护不同的信息,以形成不同精度的抽象模型.本文主要关注链表的长度相关性质,故为每个变量可达向量维护一个数值属性,用于记录该变量可达向量所代表的具体链表结点的数目.

定义 8. 带计数的变量可达向量集(variable reachability vector set with counter,简称 VRVSC)$T\subseteq\hat{\Gamma}\times N$ 是一个变量可达向量与一个自然数组成的二元偶的集合.对任意 $t\in T$,我们用 *t.vec* 表示该二元偶的第 1 项,对应一个变量可达向量;而以 *t.num* 表示元偶的第 2 项,记录变量可达向量等于 *t.vec* 的链表结点的个数.

带计数的变量可达向量集精确地维护了链表的长度信息,可以提高抽象模型精度,降低因为模型不精确而引起的误报.同时,这对验证链表长度相关性质及终止性也是至关重要的.显然,$\hat{\Gamma}_i$ 中所有元素的 *num* 域之和即为变量 V_i 当前指向的链表片段的长度.例如,针对图 4(a)中的链表状态,由于 $\hat{\Gamma}_0=\{11001,11101,11111\}$,可知变量 p 指向的链表部分长度为 2+1+1=4.

4 链表操作的抽象语义

文中采用带计数的变量可达向量集描述链表状态,而将链表操作语句定义为状态之间的迁移函数.暂不考虑过程间调用,故可将 V 中所有变量看作全局变量.因为只有赋值语句才可能改变链表的形态,所以首先在图 6 中给出赋值语句的抽象语义.为简洁起见,文中始终假设 $p=V_p,q=V_q$,即变量 p,q 分别为第 p 位和第 q 位变量,故而 $\hat{\gamma}_p^0,\hat{\gamma}_q^0$ 分别对应变量 p,q 当前指向的变量可达向量.给定自然数集合 S,$t.\hat{vec}/_{S\leftarrow 0}$ 表示将向量 $t.\hat{vec}$ 与 S 中元素所对应的位全部置 0,且其他位保持不变.同样地,$t.\hat{vec}/_{S\leftarrow j}$ 表示将 $t.\hat{vec}$ 与 S 中元素所对应的位全部采用第 j 位的取值进行替换,且其他位保持不变.*new*(*S*)表示新创建一个变量可达向量,且与 S 中元素对应位设置为 1,其他位为 0.在具体操作时,每条语句都按照语义对 $\hat{T}$ 中包含的所有二元偶进行操作.但为简洁起见,仅给出可能会产生变化的二元偶的操作.

$$[\![p=null]\!](\hat{T})\xrightarrow{\hat{T}'=rotate(\hat{T},\hat{\gamma}_p^0)}\{t'\mid\forall t''\in\hat{T}''\wedge t''.v\hat{e}c\in\hat{\Gamma}_p''.t'.v\hat{e}c=t''.v\hat{e}c/_{\{p\}\leftarrow 0},t'.num=t''.num\}$$

$$[\![p=q]\!](\hat{T})\xrightarrow{\hat{T}'=[\![p=null]\!](\hat{T})}\{t'\mid\forall t''\in\hat{T}''\wedge t''.v\hat{e}c\in\hat{\Gamma}_q''.\ t'.v\hat{e}c=t''.v\hat{e}c/_{\{p\}\leftarrow q},t'.num=t''.num\}$$

$$[\![p=q\rightarrow next]\!](\hat{T})\xrightarrow{\hat{T}'=rotate([\![p=null]\!](\hat{T}),\hat{\gamma}_q^0);\hat{\gamma}_q^{0''}.num>1\vee\hat{\gamma}_q^0[c]==0}\{t'\mid\forall t''\in\hat{T}''\wedge t''.v\hat{e}c\in\hat{\Gamma}_q''.\ t'.v\hat{e}c=t''.v\hat{e}c/_{\{p\}\leftarrow q},$$
$$t'.num=(t''.v\hat{e}c==\hat{\gamma}_q^{0''})?t''.num--:t''.num\}\cup\{\langle new(R_{\hat{\gamma}_q^{0''}}),1\rangle\}$$

$$[\![p=q\rightarrow next]\!](\hat{T})\xrightarrow{\hat{T}'=rotate([\![p=null]\!](\hat{T}),\hat{\gamma}_q^0);\hat{\gamma}_q^{0''}.num==1\wedge\hat{\gamma}_q^0[c]==1}\{t'\mid\forall t''\in\hat{T}''\wedge t''.v\hat{e}c\in\hat{\Gamma}_{\underline{\ }q}''.\ t'.v\hat{e}c=t''.v\hat{e}c/_{\{p\}\leftarrow 1},t'.num=t''.num\}$$

$$[\![p\rightarrow next=null]\!](\hat{T})\xrightarrow{\hat{T}'=rotate(\hat{T},\hat{\gamma}_p^0)}\{t'\mid\forall t''\in\hat{T}''\wedge t''.v\hat{e}c\in\hat{\Gamma}_p''.t'.v\hat{e}c=t''.v\hat{e}c/_{\{R_{\hat{\gamma}_p^{0''}}\}\leftarrow 0},t'.v\hat{e}c[c]=(t''.v\hat{e}c\subseteq\hat{\gamma}_p^{0''})?0:t''.v\hat{e}c[c]$$
$$t'.num=(t''.v\hat{e}c==\hat{\gamma}_p^{0''})?t''.num--:t''.num;\}\cup\{\langle new(R_{\hat{\gamma}_p^{0''}}-\{c\}),1\rangle\}$$

$$[\![p\rightarrow next=q]\!](\hat{T})\xrightarrow{\hat{T}'=[\![p\rightarrow next=null]\!](\hat{T});\hat{\gamma}_q^{0''}[q]==0}\{t'\mid t''\in\hat{T}''\wedge t''.v\hat{e}c\in\hat{\Gamma}_q''.\ t'.v\hat{e}c=t''.v\hat{e}c/_{\{R_{\hat{\gamma}_q^{0''}}\}\leftarrow q},t'.num=t''.num\}$$

$$[\![p\rightarrow next=q]\!](\hat{T})\xrightarrow{\hat{T}'=[\![p\rightarrow next=null]\!](\hat{T});\hat{\gamma}_q^{0''}[q]==1}\{t'\mid t''\in\hat{T}''\wedge t''.v\hat{e}c\in\hat{\Gamma}_q''.\ t'.v\hat{e}c=t''.v\hat{e}c/_{\{c\}\leftarrow 1},t'.num=t''.num\}$$

$$[\![p=malloc(\cdot)]\!](\hat{T})\xrightarrow{\hat{T}'=[\![p=null]\!](\hat{T})}\hat{T}''\cup\{\langle new(\{p\}),1\rangle\}$$

$$[\![free(p)]\!](\hat{T})\xrightarrow{T''=rotate(\hat{T},\hat{\gamma}_p^0)}\{t'\mid t''\in\hat{T}''\wedge t''.v\hat{e}c\in\hat{\Gamma}_p''.t'.v\hat{e}c=t''.v\hat{e}c/_{\{R_{\hat{\gamma}_q^{0''}}\}\leftarrow 0},t'.v\hat{e}c[c]=(t''.v\hat{e}c\subseteq\hat{\gamma}_p^{0''})?0:t''.v\hat{e}c[c]$$
$$t'.num=(t''.v\hat{e}c==\hat{\gamma}_p^{0''})?t''.num--:t''.num;\}$$

注:图中每条语句操作完成之后均需要执行 Compact 操作.

Fig.6 Abstract symantics for *AsgnStmnt*

图 6 *AsgnStmnt* 语句的抽象语义

下面详细解释每条语句的操作语义.

(1) 执行语句 *p=null* 之后,变量 *p* 不可达任何链表结点,故可将 $\hat{\Gamma}_p''$ 中所有变量可达向量的第 *p* 位赋值为 0.在执行抽象语义之前,首先对变量可达向量集进行一次轮转操作,其目的是使变量 *p* 指向环扩展根向量,从而统一处理各种情况,避免产生无意义向量,使得操作语义更加简化、高效.这也是本文方法的一个特点.

(2) 执行语句 *p=q* 之后,变量 *p* 可达且仅可达原先变量 *q* 可达的链表结点.此时,只需将所有变量可达向量第 *p* 位的取值采用第 *q* 位的取值进行覆盖.由于我们假设采用临时变量的方式避免了 *p*,*q* 为相同变量的情形,故可以在执行 *p=q* 之前首先执行 *p=null* 操作,然后只对 $\hat{\Gamma}_q''$ 中向量进行操作.

(3) 执行语句 *p=q→next* 之后,除了变量 *q* 当前指向的链表结点以外,变量 *p* 和 *q* 针对其他链表结点的可达性质完全相同.假设首先执行 *p=null* 操作,然后将变量 *q* 指向的变量可达向量轮转为环扩展根向量,得到 $\hat{T}''$.此时,可以根据变量 *q* 指向的变量可达向量及其对应计数信息分如下两种情况加以处理:

(3.1) 变量 *q* 指向的变量可达向量不在环上或在环上但其计数大于 1.

- 若 $t''.v\hat{e}c\in\hat{\Gamma}_q''$ 但 $t''.v\hat{e}c\neq\hat{\gamma}_q^{0''}$,则变量 *p*,*q* 均可达 $t''.v\hat{e}c$ 对应的链表结点.此时,可将 $t''.v\hat{e}c$ 中第 *p* 位的取值采用第 *q* 位的取值进行覆盖,使得 $t''.v\hat{e}c$ 的第 *p* 位和第 *q* 位的取值均为 1.
- 若 $t''.v\hat{e}c=\hat{\gamma}_q^{0''}$,则因其对应的具体结点包括变量 *q* 当前指向的链表结点,需进行特殊处理. $t''.v\hat{e}c$ 对应的链表结点可划分成两类:首先,变量 *q* 当前指向的链表结点在语句执行结束之后,变量 *p* 不可达(环已经被"切断").因此,可创建一个第 *p* 位为 0、其余位与 $\hat{\gamma}_q^{0''}$ 相同的变量可达向量,并设置其对应的 *num* 计数为 1;其次,对其他变量可达向量为 $t''.v\hat{e}c$ 的链表结点,变量 *p* 与 *q* 均可达,可采用 $t''.v\hat{e}c$ 中第 *q* 位取值覆盖第 *p* 位,使得 $t''.v\hat{e}c$ 的第 *p* 位和第 *q* 位的取值均为 1.但由于排除了变量 *q* 当前指向的链表结点,其 *num* 计数需要在原先取值的基础上减 1.

(3.2) 当 $\hat{\gamma}_q^{0''}$ 是环扩展根向量且其代表的节点数目为 1 时,变量 p 应该可达所有在环 $Cycle_{\hat{\gamma}_q^{0''}}$ 上的结点.因此,此时需要将所有在 $\hat{\Gamma}_q''$ 中的变量可达向量的第 p 位设置为 1.

图 7 中给出了链表从状态(I)到状态(II)执行语句 $p=q\rightarrow next$ 时,链表形态图与带计数的变量可达向量集的变化.图 7 的状态(II)中首先给出了将 p 赋值 *null* 并执行轮转操作之后的变量可达向量集 $\hat{T}''$.最终结果 $\hat{T}'$ 中的标灰项分别代表新创建二元偶和变量 q 在 $\hat{T}''$ 中指向的变量可达向量所对应的二元偶.在进行操作的过程中,因为向量 $00100\notin\hat{\Gamma}_q''$,所以其对应的二元偶保持不变;而因为 $\hat{\gamma}_q^{0''}$ =01111,所以该二元偶的 *num* 域减 1.

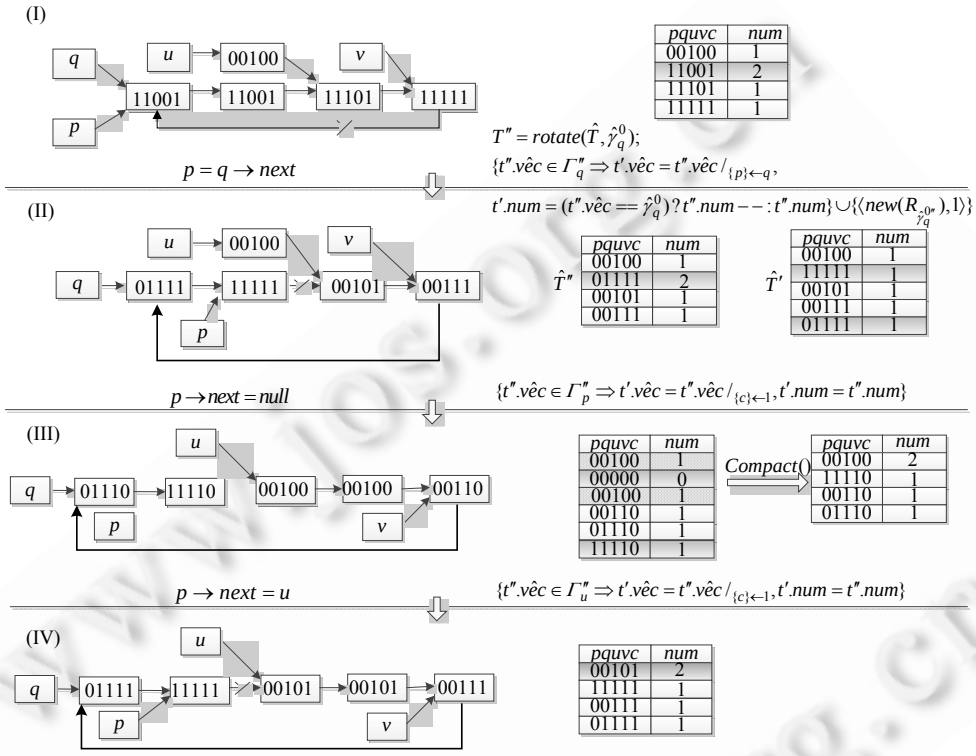

Fig.7 Examples for abstract semantics of assignments

图 7 赋值语句抽象语义示例

(4) 执行语句 $p\rightarrow next=null$ 之前首先使变量 p 指向的向量为环扩展根向量,此时,只有 $\hat{\Gamma}_p''$ 中变量可达向量的可达性质可能会产生改变,而改变的变量集合是所有可达 $\hat{\gamma}_p^{0''}$ 的变量.

- 若 $t''.\hat{vec}\in\hat{\Gamma}_p''$ 且 $t''.\hat{vec}\neq\hat{\gamma}_p^{0''}$,则所有可达 $\hat{\gamma}_p^{0''}$ 的变量均不再可达 $t''.\hat{vec}$,此时,可将 $t''.\hat{vec}$ 的 $R_{\hat{\gamma}_p^{0''}}$ 中元素对应位的取值赋值 0.此外,由于变量 p 指向的向量已经为环扩展根向量,按照该语句的语义,所有可达 $\hat{\gamma}_p^{0''}$ 的向量都不可能在任何环上,故而应该将其环标记位清零.
- 若 $t''.\hat{vec}=\hat{\gamma}_p^{0''}$,则因其对应的具体结点包含变量 p 当前指向的链表结点,需进行特殊处理. $t''.\hat{vec}$ 对应的链表结点可以划分成两类:首先,因为变量 p 当前指向的链表结点的可达性质不变,此时可新创建一个 $\hat{\gamma}_p^{0''}$ 相同的变量可达向量,并设置其对应的 *num* 计数为 1.但因为此结点肯定不可能再在任何环上,此时应该将其环标记位清零;其次,对其他变量可达向量为 $t''.\hat{vec}$ 的链表结点,处理方法与 $t''.\hat{vec}\in\hat{\Gamma}_p''$ 且

$t.vec \neq \gamma_p^0$ 时相同.但由于排除了变量 p 当前指向的链表结点,其 *num* 计数需要在原先取值的基础上减 1.

如图 7 中从状态(II)到状态(III)的迁移,执行了语句 $p\rightarrow next=null$,其操作结果如图 7 的状态(III)所示,图中标灰项分别表示变量可达向量 $\hat{\gamma}_p^{0''}$ 操作之后的结果以及新建的表项.由于变量 p 指向的向量已经是环扩展根向量,该操作把所有可达 $\hat{\gamma}_p^{0''}$ 的变量可达向量的环标记位设置为 0.

(5) 执行语句 $p\rightarrow next=q$ 之前,首先执行了 $p\rightarrow next=null$,故变量 p 指向环扩展根向量.

(5.1) 若 $\hat{\gamma}_p^{0''}[j]=0$,则变量 q 不可达 $\hat{\gamma}_p^{0''}$,执行语句 $p\rightarrow next=q$ 之后不会形成环,此时,可达 $\hat{\gamma}_p^{0''}$ 的所有变量都将可达变量 q 可达的任何向量,其 *num* 域保持不变.

(5.2) 若 $\hat{\gamma}_p^{0''}[q]=1$,则变量 q 可达 $\hat{\gamma}_p^{0''}$,执行语句 $p\rightarrow next=q$ 之后会形成一个新的环,此时,需将变量 q 可达的所有变量可达向量的环标记位设置为 1,其 *num* 域保持不变.

图 7 中链表从状态(III)到状态(IV)的迁移执行了操作 $p\rightarrow next=u$,由于变量 u 可达变量 p 当前指向的变量($\hat{\gamma}_p^{0''}[u]=1$),此时需要将变量 u 可达的所有向量的环标记位设置为 1.操作结果如图 7 的状态(IV)所示.

(6) 执行语句 $p=malloc(\cdot)$时,其语义等价于首先将变量 p 赋值为 *null*,然后新建一个只有第 p 位为 1 且 *num* 域计数为 1 的二元偶,以表示新创建的结点.

(7) 执行语句 *free*(p)时,其语义操作与执行 $p\rightarrow next=null$ 时大概相同,区别在于变量 p 指向的链表结点将不再存在,因此不再需要新建一个表项.

在链表操作语句执行过程中,有可能产生如下几种情况:

① 存在多个 *vêc* 域取值相同的元偶.此时,需要对这些二元偶进行合并.亦即将所有 *vêc* 域取值相同的二元偶合并成一个新的二元偶,其 *num* 域为这些二元偶的 *num* 域之和.

② 产生的 *num* 域为 0 的二元偶.此时不表示任何意义,需要从带计数的变量可达向量集中删除.

③ 产生的 *vêc* 域为全 0 的二元偶.此时,若 *num* 域为 0,则根据情况②中处理方法,可以简单地删除该二元偶;而若 *num* 域大于 0,则表明发生了内存泄漏,文中的策略是另行记录该错误,并从带计数的变量可达向量集中删除该二元偶,继续执行下一条语句.

我们定义操作 *Compact*($\hat{T}$),对带计数的变量可达向量集进行必要的合并操作,并除去 *vêc* 域为全 0 或 *num* 域等于 0 的二元偶,同时记录内存泄漏 bug.每条语句对应的抽象语义之后,都默认执行 *Compact*($\hat{T}$)操作.

例如,在图 7 中执行语句 $p\rightarrow next=null$ 之后,链表状态(III)会产生两个变量可达向量为 00100 的元偶(带斜线背景的表项).这两个结点的变量可达性质完全相同,需要在带计数的变量可达向量集中将其合并成一个二元偶〈00100,2〉.同时,执行该语句过程中会产生一个 *vêc* 域和 *num* 域均为 0 的二元偶(第 1 个标灰项),亦需要把该项进行删除.图 7 的状态(III)中给出了执行 *Compact*($\hat{T}$)操作之后得到的带计数的变量可达向量集.

图 8 中给出两种基本布尔条件在带计数的变量可达向量集上的解释语义.显然,$p==null$ 为真当且仅当 $\hat{\Gamma}_p$ 为空.在具体实现时,只需判断 $\hat{\Gamma}$ 中所有向量的第 p 位为 0 即可;而 $p==q$ 真当且仅当 $\hat{\Gamma}_p=\hat{\Gamma}_q$.在具体实现时,只需判断 $\hat{\Gamma}$ 中所有向量的第 p 位与第 q 位取值相同即可.这些判断操作都存在高效实现算法.

$$[\![p==null]\!](\hat{T})=true \Leftrightarrow \hat{\Gamma}_p=\varnothing$$
$$[\![p==q]\!](\hat{T})=true \Leftrightarrow \hat{\Gamma}_p=\hat{\Gamma}_q$$

Fig.8 Abstract semantics for Boolean conditions

图 8 布尔条件的抽象语义

在定义链表赋值操作和条件语句的抽象语义之后,可将链表操作程序看作一个以带计数的变量可达向量集为状态的迁移系统,3 种基本语法结构的迁移语义如图 9 所示.该迁移系统描述了链表操作程序的抽象语义,且较好地维护了链表结点的指向信息以及长度信息的变化过程.

$$\frac{\hat{T}' = [\![AsgnStmnt_1]\!](\hat{T})}{\langle (AsgnStmnt_1; Stmnt), \hat{T}\rangle \rightarrowtail \langle Stmnt, \hat{T}'\rangle}$$

$$\frac{BranchStmnt = \textbf{if } Cond \textbf{ then } \{Stmnt';^*\} \textbf{ else } \{Stmnt'';^*\} \textbf{ fi}}{\langle (BranchStmnt; Stmnt), \hat{T}\rangle \rightarrowtail ([\![Cond]\!](\hat{T}) == true?\rangle\langle (\{Stmnt';^*\}; Stmnt, \hat{T})\rangle : \langle (\{Stmnt'';^*\}; Stmnt, \hat{T})\rangle}$$

$$\frac{WhileStmnt = \textbf{while } Cond \textbf{ do } \{Stmnt';^*\} \textbf{ od}}{\langle (WhileStmnt; Stmnt), \hat{T}\rangle \rightarrowtail ([\![Cond]\!](\hat{T}) == true)?\langle (\{Stmnt';^*\}; WhileStmnt; Stmnt), \hat{T}\rangle : \langle Stmnt, \hat{T}\rangle}$$

Fig.9 Abstract semantics for syntax structures manipulating lists

图 9 链表操作语法结构的抽象语义

5 链表分析原型工具设计及实验结果

为了验证文中链表抽象方法的有效性,我们基于符号执行技术设计实现了一个链表分析原型工具,采用带计数的变量可达向量集作为符号执行过程中链表的抽象状态,可以分析空指针引用、内存泄漏等运行时错误,以及链表长度相关的安全性质.主要框架如图 10 所示.

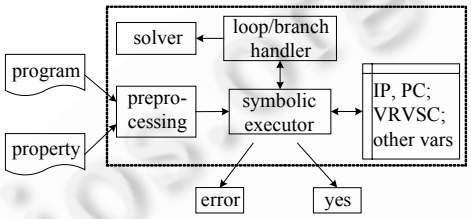

Fig.10 Symbolic execution framework for analyzing lists

图 10 链表分析符号执行框架

具体而言,preprocessing 模块对链表操作程序和性质进行预处理.例如,引入新的临时变量以保证每条赋值语句中最多只有 1 个 *next* 操作子,以及将局部变量全局化等等.针对待分析验证的性质,preprocessing 模块将其转换成带计数的变量可达向量集上的断言.具体实现时,我们默认检测并记录所有的空指针引用错误以及内存泄漏错误等运行时错误,这两类错误也可采用 *assert* 断言进行判断:若访问变量 p 的右值时 $\hat{\Gamma}_p = \varnothing$,则可断定发生了空指针引用错误;而若在抽象语义操作时发现 *node* 域为全 0、*num* 域大于 0 的二元偶,则可断定发生了内存泄漏.symbolic executor 模块是符号执行器,可以从指定的初始状态开始执行,返回结束时的程序状态(可能为状态集合).每个状态除维护带计数的变量可达向量集(VRVSC)外,还需维护路径条件 PC、程序指令指针 IP 以及其他必要变量的取值.loop/branch handler 模块对分支语句与循环结构进行处理.对简单循环结构,该模块采用特殊的处理策略,调用 symbolic executor 和 solver,将其作为一个整体进行处理,避免了经典符号执行中循环展开的处理方法.对其他无法直接进行处理的循环结构,仍采用常规符号执行的方法展开若干次进行分析.图中 solver 模块用于判定当前状态下的分支条件和循环条件取值.我们采用开源符号执行工具 KLEE[18]作为基本的符号执行器,而采用 SMT 工具 Z3[19]作为分支条件及循环特殊处理机制的底层求解器.

在采用符号执行框架设计实验时,*num* 部分有可能是符号表达式.若某个向量对应的 *num* 域为符号表达式且无法判断其是否肯定大于 0,则按照符号执行的语义,分别处理其等于 0 与大于 0 两种情况.在符号执行时孵化出两个对应的状态,继续下一条语句的符号执行.

- 循环处理机制

循环体对链表的影响表现在链表形态及链表片段长度的变化.在以带计数的变量可达向量集作为链表状态的符号执行框架中,也就是要获知带计数的变量可达向量集中所有二元偶的 *vec* 域及 *num* 域的变化规律.具体实现 loop/branch handler 模块时,首先根据形态分析的结果对循环体控制流进行精化,将原始循环转换成若干“小循环”,使得每个“小循环”的循环体入口处仅存在 1 种可能的链表抽象形态(*vec* 部分相同则认为是同一种链

表抽象形态),然后针对每个“小循环”求解其 *num* 部分的变化规律.对无法通过该方法获得循环执行次数的程序,我们仍旧采用循环展开的方法执行符号执行,设置符号执行最大次数为一个相对较大的值.

- 实验结果

我们对常见的链表反转、链表拷贝、链表删除及图 1 中给出的 *copy_and_delete* 等链表操作过程进行实验,除了 prio-与 del-cond 系列程序以外,分别验证单向无环链表及环形链表(以-cyclic-标识)两个版本的源程序.此外,为了验证循环处理机制的有效性,分别设定链表长度为一个较大的值(后缀为-big,等于 10^6)和符号值(后缀为-sym)进行实验.实验中,分别在每个程序中植入空指针引用、内存泄漏以及链表长度相关性质违反错误.由于不关心链表结点存储的具体数据,具体实验时,将程序中所有涉及数据域的操作均采用随机值代替.我们在一台 CPU 为 Pentium 4 1.8G、内存为 512M 的计算机上进行实验.针对上述性质,分析验证过程均给出正确的分析结果,没有产生误报.实验结果见表 1.

Table 1 Experimental results

表 1 实验结果

Programs	Null pointer reference	Memory leak	Length property	Time (s)	Time-Loop (s)
reverse-big	√	√	√	10.365	0.092
reverse-sym	√	√	√	∞	0.134
reverse-err	×	×	×	—	—
copy-big	√	√	√	19.211	0.100
copy-sym	√	√	√	∞	0.156
copy-err	×	×	×	—	—
del-big	√	√	√	4.706	0.056
del-sym	√	√	√	∞	0.098
del-err	×	×	×	—	—
copy_and_delete-big	√	√	√	35.342	0.281
copy_and_delete-sym	√	√	√	∞	0.314
copy_and_delete-err	×	×	×	—	—
reverse-cyclic-big	√	√	√	11.479	0.099
reverse-cyclic-sym	√	√	√	∞	0.153
reverse-cyclic-err	×	×	×	—	—
copy-cyclic-big	√	√	√	20.786	0.112
copy-cyclic-sym	√	√	√	∞	0.168
copy-cyclic -err	×	×	×	—	—
del-cyclic-big	√	√	√	5.325	0.067
del-cylic-sym	√	√	√	∞	0.111
del-cyclic-err	×	×	×	—	—
copy_and_delete-cylic-big	√	√	√	36.320	0.289
copy_and_delete-cylic-sym	√	√	√	∞	0.346
copy_and_delete-cylic-err	×	×	×	—	—
prio-big	√	√	√	229.839	0.632
prio-sym	√	√	√	∞	0.908
prio-err	×	×	×	—	—
del-cond-big	√	√	√	∞	0.883
del-cond-sym	√	√	√	∞	0.928
del-cond-err	×	×	×	—	—

表 1 首先在前 3 列给出针对空指针引用、内存泄漏以及链表长度相关性质的检查的结果,time 列表示采用经典符号执行框架进行程序分析的时间,time-loop 列表示对循环结构采用特殊处理之后进行程序分析的时间开销.若分析过程在 10 分钟时间内仍未结束,则认为该分析过程不结束,并标记为∞.由于我们分别为 3 种错误设计了一个程序版本进行实验,而且不同错误在程序中引入的位置不同,因此没有统计监测错误版本的实验时间.

显然,由于需要针对所有链表结点进行迭代分析,所需时间均较长.其中,当链表长度为符号值时,经典符号执行方法由于无法判断循环终止条件,导致分析过程无法终止.在采用循环特殊处理机制之后,极大地缩短了程序分析的开销,同时还能对链表长度为符号值的程序进行分析.另外,从表 1 中我们还可以看出,环形链表的分析时间均稍大于对应的无环版本,这主要是因为在处理带环链表时需要更多的环标记位判断及相关处理操作.

prio 系列的程序中包含了两层嵌套循环,且外层循环是一个 while(1)类型的非停机的循环.在具体实验时,

我们在-big 版本中针对外层循环设置循环次数为 10,而在-sym 版本中则设置为一个符号值.del-cond-系列的程序根据链表内容相关的条件删除某个链表结点.因为我们对所有的链表内容设置为任意值,在分析 del-cond-big 时,每次均无法判断分支条件真假,从而总是需要维护两条路径.所以,该程序的分析时间相对于普通的链表遍历程序要长很多,在约定时间仍未结束,而在引入循环特殊处理之后,分析过程就可以成功地结束.

6 结论及下一步的工作

本文最主要的创新点在于给出了一种针对单向链表的抽象表示方法.该抽象方法的状态表示是紧致的,能够隐式地表示链表节点之间的边信息,存储开销较低;且能够自动维护链表的长度信息,提供了一种相对精确的链表抽象模型.最后,文中基于符号执行框架设计实现了一个支持多种链表性质的链表分析原型工具,证明了该链表抽象方法的正确性.目前,文中方法仅处理了最常见的单向链表,对双向链表进行处理时,需要进行额外的扩展.已有的思路是分别为 *next* 域以及 *prev* 域维护变量可达信息,并将可达性区分为 *next* 可达以及 *prev* 可达,同时还需要要求双向链表满足良构(well-founded)性质.这也是我们下一步首先将要开展的工作之一.

References:

[1] Bouajjani A, Drăgoi C, Enea C, Rezine A, Sighireanu M. Invariant synthesis for programs manipulating lists with unbounded data. In: Touili T, Cook B, Jackson P, eds. Proc. of the 22th CAV. LNCS 6174, Edinburgh: Springer-Verlag, 2010. 72−88.

[2] Černý P, Radhakrishna A, Chaudhuri S, Alur R. Model checking of linearizability of concurrent list implementations. In: Touili T, Cook B, Jackson P, eds. Proc. of the 22th CAV. LNCS 6174, Edinburgh: Springer-Verlag, 2010. 465−479. [doi: 10.1007/978-3-642-14295-6_41]

[3] Bansal K, Brochenin R, Lozes E. Beyond shapes: Lists with ordered data. In: de Alfaro L, ed. Proc. of the 12th FOSSACS. LNCS 5504, York: Springer-Verlag, 2009. 425−439. [doi: 10.1007/978-3-642-00596-1_30]

[4] Alur R, Černý P. Streaming transducers for algorithmic verification of single-pass list-processing programs. In: Ball T, Sagiv M, eds. Proc. of the 38th POPL. Austin: ACM Press, 2011. 599−610. [doi: 10.1145/1925844.1926454]

[5] Bozga M, Iosif R, Perarnau S. Quantitative separation logic and programs with lists. Journal of Automated Reasoning, 2010,45(2): 131−156. [doi: 10.1007/s10817-010-9179-9]

[6] Bozga M, Iosif R. On flat programs with lists. In: Cook B, Podelski A, eds. Proc. of the 8th VMCAI. LNCS 4349, Nice: Springer-Verlag, 2007. 122−136.

[7] Katoen JP, Noll T, Rieger S. Verifying concurrent list-manipulating programs by LTL model checking. In: Proc. of the Workshop on Heap Analysis and Verification (HAV 2007). 2007. 94−113.

[8] Bozga M, Iosif R. Quantitative verification of programs with lists. In: Proc. of the VISAS 2005. 2005. [doi: 10.1007/s10817-010-9179-9]

[9] Bouajjani A, Bozga M, Habermehl P, Iosif R, Moro P, Vojnar T. Programs with lists are counter automata. In: Ball T, Jones RB, eds. Proc. of the 18th CAV. LNCS 4144, Seattle: Springer-Verlag, 2006. 517−531. [doi: 10.1007/s10703-011-0111-7]

[10] Dor N, Rodeh M, Sagiv S. Checking cleanness in linked lists. In: Palsberg J, ed. Proc. of the 7th SAS. LNCS 1824, Santa Barbara: Springer-Verlag, 2000. 115−135.

[11] Finkel A, Lozes É, Sangnier A. Towards model-checking programs with lists. In: Archibald M, Brattka V, Goranko V, Löwe B, eds. Proc. of the ILC 2007. LNCS 5489, Cape Town: Springer-Verlag, 2007. 56−86. [doi: 10.1007/978-3-642-03092-5_6]

[12] Sagiv S, Reps TW, Wilhelm R. Parametric shape analysis via 3-valued logic. ACM Trans. on Programming Languages and Systems, 2002,24(3):217−298. [doi: 10.1145/292540.292552]

[13] Manevich R, Yahav E, Ramalingam G, Sagiv M. Predicate abstraction and canonical abstraction for singly-linked lists. In: Cousot R, ed. Proc. of the 6th VMCAI. LNCS 3385, Paris: Springer-Verlag, 2005. 181−198. [doi: 10.1007/978-3-540-30579-8_13]

[14] Podelski A, Wies T. Boolean heaps. In: Hankin C, Siveroni I, eds. Proc. of the 12th SAS. LNCS 3672, London: Springer-Verlag, 2005. 268−283. [doi: 10.1007/11547662_19]

[15] John CR. Separation logic: A logic for shared mutable data structures. In: Proc. of the LICS 2002. Copenhagen: IEEE Computer Society, 2002. 55−74.

[16] Berdine J, Calcagno C, O'Hearn PW. A decidable fragment of separation logic. In: Lodaya K, Mahajan M, eds. Proc. of the 24th FSTTCS. LNCS 3328, Chennai: Springer-Verlag, 2004. 97−109. [doi: 10.1007/978-3-540-30538-5_9]

[17] Bouajjani A, Habermehl P, Moro P, Vojnar T. Verifying programs with dynamic 1-selector-linked structures in regular model checking. In: Halbwachs N, Zuck LD, eds. Proc. of the 11th TACAS. LNCS 3440, Edinburgh: Springer-Verlag, 2005. 13−29. [doi: 10.1007/978-3-540-31980-1_2]

[18] Cadar C, Dunbar D, Engler DR. Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs. In: Draves R, van Renesse R, eds. Proc. of the 8th OSDI. San Diego: USENIX Association, 2008. 209−224.

[19] de Moura LM, Bjørner N. Z3: An efficient SMT solver. In: Ramakrishnan CR, Rehof J, eds. Proc. of the TACAS. LNCS 4963, Budapest: Springer-Verlag, 2008. 337−340.

附录

1. 引理 2 的证明.

证明:不妨设 $i\in R_{\gamma_{n_1}}\cap R_{\gamma_{n_2}}$. 由 $i\in R_{\gamma_{n_1}}$ 知 $Reach(V_i,n_1)=true$,故存在 $n_{k_0},n_{k_1},...,n_{k_p}\in V$,其中, $n_{k_0}=V_i,n_{k_p}=n_1$, 使得对任意 $k_0\leqslant k<k_p$ 都有 $E(n_k)=n_{k+1}$.同理,必存在 $n_{t_0},n_{t_1},...,n_{t_q}\in V$,其中, $n_{t_0}=V_i,n_{t_q}=n_2$, 使得对任意 $t_0\leqslant t<t_q$ 都有 $E(n_t)=n_{t+1}$.下面分情况加以讨论:

- 若 $p=q$,则因为链表中每个结点只有 1 个后继结点,所以从同一个结点 V_i 出发,经过相同条 *next* 边,必有 $n_{k_p}=n_{t_q}$,亦即 $n_1=n_2$.这与前提条件冲突;
- 若 $p>q$,则存在 $k_q,\ldots,k_p$,使得 $n_{k_q}=n_2$, 而 $n_{k_p}=n_1$, 亦即存在一条从 n_2 到 n_1 的路径;
- 若 $p<q$,则存在 $t_p,\ldots,t_q$,使得 $n_{t_p}=n_1$, 而 $n_{t_q}=n_2$, 亦即存在一条从 n_1 到 n_2 的路径.

由以上讨论可知,引理 2 成立. □

2. 定理 2 的证明.

证明:给定协调的变量可达向量集 Γ,可以定义每个变量可达向量 $\gamma\in\Gamma$ 的前驱与直接前驱集合

$$pred^*(\gamma)=\{\gamma'\in\Gamma\mid\gamma'\subset\gamma\};\ pred(\gamma)=\{\gamma'\in pred^*(\gamma)\mid\nexists\gamma''\in pred^*(\gamma).\gamma'\subset\gamma''\}.$$

易知结论:

(i) 给定 $\gamma\in\Gamma$,任意 $\gamma_1,\gamma_2\in pred(\gamma)$,均有 $R_{\gamma_1}\cap R_{\gamma_2}=\varnothing$,亦即 $\gamma_1\cap\gamma_2=\varnothing$;

(ii) 任意 $\gamma_1,\gamma_2\in\Gamma$,若 $\gamma_1\cap\gamma_2=\varnothing$,且 $\gamma_1'\in pred^*(\gamma_1),\gamma_2'\in pred^*(\gamma_2)$,则 $\gamma_1'\cap\gamma_2'=\varnothing$.

首先采用第二数学归纳法证明,任意 $\gamma\in\Gamma$,其前驱集合 $pred^*(\gamma)$ 满足 $|pred^*(\gamma)|\leqslant 2(|R_\gamma|-1)$.

(1) 当 $|R_\gamma|=1$ 时,因为任意 $\gamma'\in pred^*(\gamma)$,均有 $R_{\gamma'}\subset R_\gamma$,所以 $|R_{\gamma'}|=0$.因为按照变量可达向量的定义,$|R_{\gamma'}|>0$,所以 $pred^*(\gamma)$ 必然为空集,满足 $|pred^*(\gamma)|\leqslant 2(|R_\gamma|-1)$.

当 $|R_\gamma|=2$ 时,不妨假设 $R_\gamma=\{m,n\}$,此时 $pred^*(\gamma)$ 中任意 γ',其对应的 $R_{\gamma'}$ 只能为 $\{m\}$ 或 $\{n\}$.故 $pred^*(\gamma)$ 最多包含两个元素,亦满足 $|pred^*(\gamma)|\leqslant 2(|R_\gamma|-1)$.

(2) 假设当 $|R_\gamma|\leqslant M$ 时都满足 $|pred^*(\gamma)|\leqslant 2(|R_\gamma|-1)$ 成立,那么当 $|R_\gamma|=M+1$ 时,

(2.1) 若 $pred(\gamma)=\varnothing$,则 $pred^*(\gamma)$ 也必然为空集,显然有 $|pred^*(\gamma)|\leqslant 2(|R_\gamma|-1)$ 成立;

(2.2) 假设 $pred(\gamma)=\{\gamma_1,\ldots,\gamma_s\}$,由结论(i)、结论(ii)可知,任意 $\gamma'\in pred^*(\gamma)$,要么 $\gamma'\in pred(\gamma)$,要么存在且只存在 1 个 $1\leqslant i\leqslant s$,使得 $\gamma'\in pred^*(\gamma_i)$.同时对任意 $1\leqslant i\leqslant s$,显然 $pred^*(\gamma_i)\subseteq pred^*(\gamma)$.所以,

$$|pred^*(\gamma)|=\sum_{1\leqslant i\leqslant s}|pred^*(\gamma_i)|+|pred(\gamma)|.$$

对任意 $1\leqslant i\leqslant s$,因为 $R_{\gamma_i}\subset R_\gamma$, 所以 $|R_{\gamma_i}|\leqslant M$.按照假设,$|pred^*(\gamma_i)|\leqslant 2(|R_{\gamma_i}|-1)$.因此,

$$|pred^*(\gamma)|=\sum_{1\leqslant i\leqslant s}|pred^*(\gamma_i)|+s\leqslant\sum_{1\leqslant i\leqslant s}2(|R_{\gamma_i}|-1)+s=2\sum_{1\leqslant i\leqslant s}|R_{\gamma_i}|-s.$$

- 当 $s\geqslant 2$ 时,因为对任意 $1\leqslant i,j\leqslant s\wedge i\neq j$,都有 $R_{\gamma_i}\subset R_\gamma,R_{\gamma_j}\subset R_\gamma$, 且根据结论(i)可知 $R_{\gamma_i}\cap R_{\gamma_j}=\varnothing$,故 $\sum_{1\leqslant i\leqslant s}|R_{\gamma_i}|\leqslant|R_\gamma|$.于是,$|pred^*(\gamma)|\leqslant 2|R_\gamma|-s\leqslant 2(|R_\gamma|-1)$.

- 当 $s=1$ 时,γ只有 1 个前驱γ_i,因为 $R_{\gamma_i} \subset R_\gamma$,所以,$|R_{\gamma_i}|<|R_\gamma|$.于是,$|pred^*(\gamma)|<2(|R_\gamma|-1)$成立.故此时$|pred^*(\gamma)|\leqslant 2(|R_\gamma|-1)$也成立.

综上所述,根据第二数学归纳法可知,任意$\gamma\in\Gamma$,满足$|pred^*(\gamma)|\leqslant 2(|R_\gamma|-1)$.

设$\gamma\in\Gamma$,若不存在$\gamma'\in\Gamma$使得$\gamma\subset\gamma'$,则称γ为Γ中的极大变量可达向量.假设Γ中所有极大变量可达向量为$\gamma_1,\ldots,\gamma_t$.显然,对任意$1\leqslant i,j\leqslant t$且$i\neq j$,都有$R_{\gamma_i}\cap R_{\gamma_j}=\varnothing$,而任意$x\in R_{\gamma_i}$都有$0\leqslant x\leqslant|V|$,所以$\sum_{1\leqslant i\leqslant t}|R_{\gamma_i}|\leqslant|V|$.结合结论(ii)可知,任意$\gamma'\in\Gamma$,要么$\gamma'\in\{\gamma_1,\ldots,\gamma_t\}$,要么存在且只存在 1 个$1\leqslant i\leqslant t$,使得$\gamma'\in pred^*(\gamma_i)$.

所以,$|\Gamma|=\sum_{1\leqslant i\leqslant t}|pred^*(\gamma_i)|+t\leqslant\sum_{1\leqslant i\leqslant t}2(|R_{\gamma_i}|-1)+t=2\sum_{1\leqslant i\leqslant t}|R_{\gamma_i}|-t\leqslant 2|V|-t\leqslant 2|V|-1$. □

3. 定理 5 的证明.

证明:我们考虑内存形态中存在m个链表的情况.显然,指向不同链表的变量各不相同,故可将变量集合V划分为 m 个集合:$V_{/0},V_{/1},\ldots,V_{/m}$.同时,每个链表的结点所对应的扩展变量可达向量子集合也各不相交.只要证明任意$1\leqslant t\leqslant m$,与该链表对应的扩展变量可达向量子集合$\hat{\Gamma}_{/t}$都满足$|\hat{\Gamma}_{/t}|\leqslant 2|V_{/t}|$即可.

给定一个扩展变量可达向量,若有且仅有 1 个变量可达该向量,则称该向量为一个单变量可达向量.采用反证法的思路容易证明:当一个扩展变量可达向量集中元素个数达到最大时,必然包含变量集合中每个变量所对应的单变量可达向量.然后,与定理 4 的证明思路一样,我们把$\hat{\Gamma}_{/t}$看作变量集合$V_{/t}\cup\{V_c\}$上的一个普通的变量可达向量集,则据定理 2 可知,$|\hat{\Gamma}_{/t}|\leqslant 2(|V_{/t}|+1)-1=2|V_{/t}|+1$.但实际上,新引入的变量$V_c$是受限的,按照本节中的环扩展策略,$\hat{\Gamma}_{/t}$中显然不可能存在一个变量$V_c$对应的单变量可达向量.而当$|\hat{\Gamma}_{/t}|=2|V_{/t}|+1$时,显然包括了该单变量可达向量.排除此向量即可知$|\hat{\Gamma}_{/t}|<2|V_{/t}|+1$,亦即$|\hat{\Gamma}_{/t}|\leqslant 2|V_{/t}|$. □

李仁见(1980－),男,山东高密人,博士,主要研究领域为程序分析与验证,符号执行.

陈立前(1982－),男,博士,助理研究员,主要研究领域为程序分析与验证,抽象解释.

刘万伟(1980－),男,博士,讲师,CCF 会员,主要研究领域为模型检验,定理证明.

王戟(1969－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为高可信软件技术,软件方法学,软件工程.