

# 基于数据流分析的单链表可达性自动化验证

冬雨辰 王寒非 赵建华

(南京大学计算机软件新技术国家重点实验室 南京 210023)

**摘要** 程序验证中的常见情景是判断某个用户指定的性质在程序执行之后或执行过程中的某个程序点上是否成立。人工的形式化验证过程繁琐且容易出错,因此形式化验证的自动化是提高代码验证效率的重要方法。数据流分析技术是一种能够自动发现程序中某类性质的技术。研究了将一种数据流分析技术(单链表形状分析)和基于 Scope Logic 的代码验证过程相结合的方法。通过数据流分析获得所有程序点上的单链表可达性性质,将结果表达为带有递归函数的一阶逻辑公式,并将其插入到相应程序点中。分析程序还根据 Scope Logic 的证明法则设定了这些公式之间的逻辑依赖关系。实例测试表明所提方法可以分析得到单链表可达性性质,并且分析结果能够被基于 Scope Logic 的代码形式化验证过程有效利用,提高了代码形式化证明的效率。

**关键词** 代码验证,数据流分析,Scope Logic

**中图分类号** TP301 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2015.12.011

## Automatic Verification of Singly Linked List Pointer's Reachability Property Using Data-flow Analysis Method

DONG Yu-chen WANG Han-fei ZHAO Jian-hua

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

**Abstract** A common scenario in verifying a program is to find out whether some user-specific properties in some program points hold during or after execution. Manual formal verification is tedious and error-prone, so automatic verification is an important way to improve code verification efficiency. Data-flow analysis technique can automatically discover specific properties in programs points. This paper presented a method that integrates a kind of data-flow analysis technique(singly linked list pointers' reachability) with Scope-Logic-based code verification. We collected reachability properties in program points through data flow analysis, presented the results as first order logic formulas with recursive functions, then inserted these formulas into corresponding program points, proved them and established their dependencies according to rules of Scope Logic. Experiments show that our method can acquire singly linked list pointer's reachability property efficiently, and the results can be effectively used in code verification in Scope Logic.

**Keywords** Code verification, Data-flow analysis, Scope Logic

## 1 引言

在程序验证中,为了分析程序是否满足某个目标,我们通常将验证目标表示为一组逻辑公式的集合,并判断在程序执行完成之后或执行过程中这些公式是否成立,从而判断程序是否满足我们的需求。在霍尔逻辑<sup>[1]</sup>中,霍尔三元组和一系列的推导规则被用来描述代码的执行是如何对程序的状态进行改变的。但是手工代码验证的效率很低。程序分析和验证的自动化是代码验证技术实用化的必要手段。有许多方法可以自动地对程序进行分析,其中数据流分析技术是比较常见的一种方法<sup>[2]</sup>。数据流分析技术被广泛地应用于编译优化<sup>[3]</sup>、程序验证与测试以及代码调错。虽然数据流分析技术通常仅能够分析处理特定类型的程序性质,但这些性质在程序验证中可以作为证明其它性质的基础。比如:单链表可达性的数据流分析技术可以分析某些指针指向的数据结构是一个单链表。而根据这个性质,可以进一步证明其它更加复杂

的性质,比如单链表中数据的变化。因此,本文试图将数据流技术应用于程序的形式化验证,以提高程序验证的效率。

程序的特定性质可以表达为包含用户自定义递归函数的逻辑公式,我们可以根据 Scope Logic<sup>[4]</sup>的推导规则手动地分析程序性质。通过将数据流分析技术结合到 Scope Logic 的证明过程中,我们只需在程序开始处给出初始条件,自动分析过程就可以给出程序执行结束后各个程序点上的性质。我们选择 Scope Logic 逻辑系统作为分析基础的原因是它扩展了霍尔逻辑,提供了分析指针的能力,而我们对单链表可达性的分析依赖于对指针的处理。

本文提出了一种使用数据流分析方法来原因提高形式化验证效率的方法。该方法首先进行单链表可达性分析,并将分析结果用逻辑公式表达出来,使得分析结果可以作为验证的中间结果而被其他验证过程所使用。与通常的数据流分析不同的是,该技术不仅仅能分析得到单列表的形状性质,还基于 Scope Logic 的证明规则给出了这些性质之所以成立的逻辑

到稿日期:2015-02-25 返修日期:2015-07-01 本文受国家自然科学基金资助项目(91118007)资助。

冬雨辰(1990-),男,硕士,主要研究方向为软件工程、形式化方法,E-mail: dongyc@seg.nju.edu.cn;王寒非(1988-),男,博士,主要研究方向为软件工程、形式化方法;赵建华(1971-),男,博士,教授,主要研究方向为形式化方法、软件工程、程序设计语言。

推导,并建立了性质之间的依赖关系。这使得数据流分析技术成为代码形式化验证的有机组成部分。本文第2节简要介绍 Scope Logic 和它的4个基本的证明规则;第3节介绍如何将单链表可达性性质进行表达并对应到数据流分析框架中去,主要介绍如何生成数据流分析的半格和传递函数,以及怎样依据 Scope Logic 的证明过程进行推导,生成验证结果并进行证明;第4节介绍目前采用本方法进行的验证实验和效果;第5节总结全文。

## 2 Scope Logic 简介

Scope Logic 对霍尔逻辑进行扩展,支持对一个精简的 C 语言子集的指针和递归数据结构的分析。Scope Logic 的关键概念是内存范围(memory scope),一个表达式的内存范围表示在对表达式求值过程中访问到的内存单元的集合,这个集合可以基于表达式的语法结构构造得到。很明显,如果一个公式的内存范围在某条语句之前成立,且该语句的执行过程中没有修改其内存范围中的内存单元,那么在语句执行之后该公式仍然成立。我们将首先介绍 Scope Logic 的基本证明方法。这些基本证明方法与 Scope Logic 公理及证明规则是相容的:如果程序点中(除入口处前置条件外的)各个公式都按照这些基本证明方法证明,那么其证明结果也可以通过 Scope Logic 的公理和证明规则得到。在3.2节中,我们将说明这些证明方法可以作为单列表形状分析的逻辑基础;数据流分析过程中的 *gen/kill* 等运算都是这些规则的一种表现形式。

### 2.1 特定程序点表达式的证明规则

首先介绍特定程序点表达式(Program-point-specific Expression)。特定程序点表达式被用来表达不同程序点上的表达式之间的关系。假设  $e$  表示一个表达式,那么  $e@i$  表示  $e$  在程序点  $i$  上进行求值的结果。我们可以在另一个程序点  $j$  上用  $e@i$  表示  $e$  在  $i$  上的值,比如可以在  $j$  点用  $x = x@i+1$  表示  $j$  程序点上  $x$  的值是  $x$  在程序点  $i$  上的值加1。显然,在程序点  $i$  上,表达式  $e@i$  的值就等于  $e$ 。所以当在程序点  $i$  处给定一个包含  $@i$  的公式  $f$  时,我们可以通过将  $f$  中的所有  $@i$  表达式全部去掉而获得新的公式  $f'$ ,只要证明  $f'$  成立, $f$  也成立。

### 2.2 自生公式

自生(spontaneous)公式即语句执行所必然导致的性质。假设我们执行了语句  $e1 := e2$ ,设该语句之前的程序点为  $i$ ,该语句之后的程序点为  $j$ ,那么在程序点  $j$  上,显然公式  $*((\&e1)@i) = e2@i$  成立,而如果这条语句是一个内存分配语句  $e1 := alloc$ ,那么我们可以说在程序点  $j$  上,公式  $*((\&e1)@i) \neq null$  和  $init(*((\&e1)@i))$  成立。这里的  $init(*((\&e1)@i))$  表示表达式  $e1$  指向的内存区域全部被初始化(通常为  $null$ )。这些公式成立的逻辑基础与其它公式无关,可以看作是由这个语句自身得到的。

### 2.3 公式传播

传播证明基于 Scope Logic 中对公式的内存范围的观察:如果某个公式在某条语句执行之前成立,并且它的内存范围在语句的执行过程中没有被修改,那么在语句执行之后该公式将仍然成立。例如,我们有在程序点  $i$  用公式  $p$  表示的程序性质,执行语句  $e := v$  后到达  $j$  点,并且分析得出公式  $p$  的内存范围  $mem(p)$  与执行语句修改的地址集合  $\{\&e\}$  不相交,

那么在语句执行之后  $p$  仍然成立。通过传播证明的公式将依赖于传播前程序点上对应的公式以及在传播前程序点上该公式内存范围的相关公式。比如在这个例子中, $j$  点的公式  $p$  将依赖于  $i$  点的公式  $p$  以及在  $i$  点生成的表示内存范围不相交( $mem(p) \cap \{\&e\} = \emptyset$ )的对应公式。

### 2.4 推导证明

推导证明,即在某个程序点上,我们根据已有的性质使用逻辑推导得到更多的性质。这类处理类似于求已有性质的一个闭包,在工具实现中,我们使用 SMT 求解器 Z3<sup>[5]</sup> 来推导性质,例如,在某个程序点上,有公式  $e1 = e2$  和  $e2 = e3$  成立,那么显然可以推导出  $e1 = e3$ 。其证明方式即为推导证明。并且  $e1 = e3$  这个公式将依赖于  $e1 = e2$  和  $e2 = e3$  这两个公式。对于包含用户自定义递归函数的公式,一般还需依赖于相应递归函数的性质。

### 2.5 公式之间的依赖关系

按照上面的方法证明程序点中的各个性质时,被证明的公式(自生公式除外)总是要依赖于其它的公式。Scope Logic 的证明工具会记录并维护公式的依赖关系链。这些依赖关系链说明了各个性质之间的逻辑关系,是 Scope Logic 证明过程严谨性的保证。如果所有的公式都直接或间接地依赖于程序的前置条件,那么就表明当前值条件成立时,所有这些公式都成立。

## 3 单链表可达性分析

本节将介绍如何将特定类型性质的分析过程映射到基于 Scope Logic 的数据流分析框架上,该框架可以支持多种程序性质的分析,我们在工具中实现了单链表可达性分析、数据区间分析和空指针分析等程序性质分析过程。这里以单链表可达性分析为例说明 Scope Logic 和数据流分析技术可以很好地结合并用于验证特定类型的性质。我们分析的单链表结构是包含数据和指针域的 C 语言结构体类型:

```
struct Node { int data; Node * link; }
```

### 3.1 总体框架

方法总体框架如图1所示,需要根据待分析的用户特定性质,即单链表可达性,设计相应的递归函数并给出这些函数的性质,然后为性质设计数据流值(交半格)和传递函数。分析程序提取程序的控制流图,并根据入口程序点上以公式表示的前置条件(初始数据流值)运行数据流算法直到迭代收敛,得到最终的分析结果。最后为分析结果生成相应的公式,并为这些公式给出基于 Scope Logic 规则的逻辑证明。

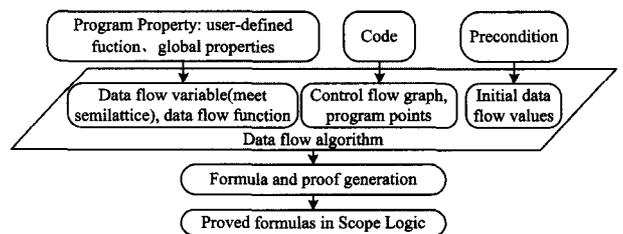


图1 结合数据流分析和 Scope Logic 进行用户特定性质分析的框架

我们分析得到了单链表可达性的性质描述和表示,并实现了工具来进行自动化分析过程,以后所有对单链表可达性进行分析的程序片段,只需给出前置条件即可自动分析并生成结果。

本节接下来分别介绍单链表可达性的递归函数定义和对

应流值表示、各种分析规则及其逻辑基础,以及最终结果的表示和证明。

### 3.2 构造数据流值以及相关的递归函数

为了分析单链表结点指针的可达性性质,我们定义了相应的递归函数来表示可达性信息,包括可达性和可达距离信息;同时还给出了这些递归函数的性质。然后构造了一个交半格来表示可达性信息,作为数据流分析过程中处理的数据流值。

#### 3.2.1 可达性性质的递归函数表示

我们使用表 1 所列递归函数来表达可达性信息,  $isslist(x)$  用来表达单链表指针指向的节点  $x$  是否通过  $link$  链可达  $null$ ,  $isslist(x,y)$  用来表示指针  $x$  是否通过  $link$  域可达  $y$ ,  $lengthslist$  和  $lengthslistseg$  用来描述指针之间的距离,只有当相应的指针可达时其  $length$  信息才有意义。 $nodeset$  和  $nodesetseg$  函数则是用来描述节点集合的信息。这些函数都配备有对应的内存范围表达式来表示访问该函数时所访问的内存地址集合。这些表达式使用  $lambda$  表达式来表示地址的集合,这些集合是将一个用  $lambda$  表达式表示的函数作用于一个集合而得到的集合映像。例如,  $lambda(Node * p) (\&p \rightarrow link) in (nodeset(x))$  表示对于  $nodeset(x)$  集合内的所有结点求其  $link$  字段地址而得到的地址集合。

表 1 使用递归函数表示可达性性质

User-Defined Functions for Singly Linked Lists	Memory Scope Expression
1. $isslist(x) \triangleq (x=null)?True; isslist(x \rightarrow link)$	$lambda(Node * p) (\&p \rightarrow link) in (nodeset(x))$
2. $isslistseg(x,y) \triangleq (x=null)?False; ((x=y)?True; isslist(x \rightarrow link, y))$	$lambda(Node * p) (\&p \rightarrow link) in (nodeset(x)) + makeset(\&y)$
3. $lengthslist(x) \triangleq (x=null)?0 : 1 + lengthslist(x \rightarrow link)$	$lambda(Node * p) (\&p \rightarrow link) in (nodeset(x))$
4. $lengthslistseg(x,y) \triangleq (x=null)?0 : ((x=y)?0 : 1 + lengthslistseg(x \rightarrow link, y))$	$lambda(Node * p) (\&p \rightarrow link) in (nodeset(x)) + makeset(\&y)$
5. $nodeset(x) \triangleq (x=null)?\emptyset; (\{x\} \cup nodeset(x \rightarrow link))$	$lambda(Node * p) (\&p \rightarrow link) in (nodeset(x))$
6. $nodesetseg(x,y) \triangleq (x=null)?\emptyset; (\{x\} \cup nodesetseg(x \rightarrow link, y))$	$lambda(Node * p) (\&p \rightarrow link) in (nodesetseg(x, y))$

注: Memory Scope Expression 表示递归函数的内存范围, makeset 函数表示构建包含单个点的集合

#### 3.2.2 可达性函数的性质

一阶逻辑不能直接处理递归函数,所以我们必须指定一些附加的公式来描述这些递归函数的性质,如表 2 所列。这些性质将在数据流分析过程中辅助推导未知的数据流值。将这些公式描述的性质作为推导的前件,我们可以将这些自定义的递归函数看作未定义函数符号进行逻辑推导。全局性质设计得越详尽越具体,分析得到的结果就越精确。我们为 3.2.1 节中的函数给出了 15 条性质。篇幅所限,表 2 中只列出 4 条典型的性质。

表 2 部分对应于单链表可达性递归函数的性质

PROPERTIES (PART OF)
1. $\forall x(x \neq null \wedge isslist(p \rightarrow link)) \Rightarrow isslist(x)$
2. $\forall x \forall y \forall z(isslistseg(x,y) \wedge isslistseg(y,z)) \Rightarrow isslistseg(x,z)$
3. $\forall x \forall y \forall z(isslistseg(x,y) \wedge isslistseg(y,z) \wedge (\neg(x \in nodeset(y)) \wedge (\neg(y \in nodeset(z)))) \Rightarrow lengthslistseg(x,z) = lengthslistseg(x,y) + lengthslistseg(y,z)$
4. $\forall x \forall y(x=y) \Rightarrow isslistseg(x,y) \wedge lengthslistseg(y,z)=0$

### 3.2.3 数据流值

数据流分析框架需要将我们定义的递归函数映射为数据流值,并且数据流值及其交汇运算要满足交半格的性质才可以进行可收敛的分析<sup>[3,6]</sup>。我们需要分析任意两个  $Node$  类型的指针之间以及任意  $Node$  指针和空指针值  $null$  之间的可达性信息,相同指针之间的可达性无需分析。这是多个流值对应的乘积格,其中每对指针之间的流值取值集合为:

$$\{T, U, B, R(a, b)\}$$

对于待分析的所有可达性流值,我们使用矩阵表示,其中相同指针的可达性不需要考虑:

	P1	P2	P3	null
P1		R(1,1)	B	R(4,4)
P2	B		R(2,2)	R(3,3)
P3	B	U		R(2,2)

用  $matrix[P1, P2]$  表示从指针  $P1$  到  $P2$  的可达性流值。取值集合满足交半格的性质,流值的半格及其交汇运算如图 2 中的箭头所示。设定一个最大长度  $MAXLEN$  来使得格的高度有穷,从而保证迭代分析收敛。流值  $T$  表示对可达性未知,在半格中作为顶;  $B$  表示可达和不可达均有可能,作为半格的底,而  $R(a, b)$  表示可达,且指针之间的距离在  $[a, b]$  范围内 ( $0 \leq a \leq b \leq MAXLEN$ ),例如,  $matrix[x, y] = R(0, 1)$  意味着要么  $x=y$  要么  $x \rightarrow link = y$ 。而  $U$  则表示不可达,即指针之间不可以通过  $link$  域到达。为了后文描述方便,定义流值可达的加减运算  $R(a, b) \pm R(c, d)$  表示  $R(a \pm c, b \pm d)$ 。使用数据流值表达和使用逻辑公式表达的性质有如下的对应关系:可达流值  $matrix[x, y] = R(a, b)$  对应  $isslistseg(x, y) \wedge a \leq lengthslistseg(x, y) \leq b$ , 而  $matrix[x, null] = R(a, b)$  对应的公式为  $isslist(x) \wedge a \leq lengthslist(x) \leq b$ 。对于不可达的流值  $matrix[x, y] = U$ , 使用公式  $y \notin nodeset(x)$  来表示。对于值为  $T$  或  $B$  的数据流值,其对分析可达性信息没有帮助,所以不会为这些流值生成对应的公式。

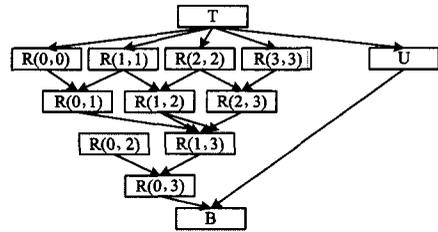


图 2  $MAXLEN$  为 3 的数据流值格表示,箭头体现了交汇运算  $\cap$ 。两个流值的交汇为它们在箭头方向上的最小公共后继。如  $R(0,0) \cap R(1,1) = R(0,1)$ , 而任意的  $R(a, b) \cap U = B$

### 3.3 构造数据流方程

数据流分析算法通过求解数据流方程来传递和修改不同程序点上的数据流值,我们也将针对前向分析,以语句为基本单元来构造数据流约束,这里需要考虑可能改变单链表结构的不同语句类型,即 7 种基本赋值语句类型:

$$x := y; x := alloc; x := null; x := y \rightarrow link;$$

$$x \rightarrow link := y; x \rightarrow link := alloc; x \rightarrow link := null;$$

复杂的赋值语句可以拆分成这 7 种基本语句的顺序组合而逐步处理,这里不再赘述。数据流方程采用数据流前向分析的通用结构<sup>[3]</sup>;对于程序点  $a$ ,  $IN[a]$  表示进入程序点的流

表3 单链表可达性分析流值的 kill 规则

语句	KILL 流值	KILL 条件
$x := rh$	$matrix[x, y]$ $matrix[y, x]$	无  $matrix[y, z] = R \wedge (y = x \vee matrix[y, x] = R \vee matrix[x, y] = B) \wedge (x \rightarrow link = z \vee matrix[x \rightarrow link, z] = R \vee matrix[x \rightarrow link, z] = B)$ 或 $matrix[y, z] = U \wedge (y = x \vee matrix[y, x] = R \vee matrix[y, x] = B)$
$x \rightarrow link := rh$	$matrix[y, z]$	

## 3.3.2 数据流分析的 gen 运算和 Scope Logic 的证明规则

gen 过程将在 kill 过程完成之后执行, gen 操作根据赋值语句  $s$  的类型进行新流值的生成, 对应于 Scope Logic 中的自生证明。gen 过程生成的流值都有相应的自生公式作为依据, 例如, 执行  $x := y$  语句之后, 可自生证明  $x = y$ , 该公式结合表 2 中的性质 4 可推导出如下公式:  $isslistseg(x, y) \wedge lengthsliseg(x, y) = 0$ , 而根据 3.2.3 节叙述, 这个公式对应的就是流值  $matrix[x, y] = R(0, 0)$ , 所以我们可以 gen 过程中生成流值  $matrix[x, y] = R(0, 0)$ 。

表 4 列出了 gen 过程产生流值的规则。流值的生成有时需要考虑某些前提条件, 比如, 对于语句  $x := y \rightarrow link$  语句, 首先要考虑  $y$  是否为 null, 如果  $y$  为 null, 则语句无意义; 在  $y$  不为 null 的情况下, 还要进一步考虑  $y$  是否等于  $y \rightarrow link$ , 如果是, 那么  $y$  自身形成一个环, 执行了  $x := y \rightarrow link$  语句之后,  $y$  直接就等于  $x$ , 并且  $y$  通过 link 链域也可以到达  $x$ , 这样流值  $matrix[y, x]$  就应该为  $R(0, 1)$ 。对于依赖于前提条件产生的流值, 其在生成公式并证明时也要依赖于相应的前提条件对应的公式, 比如这里的  $y \neq null$ , 而  $y \neq null$  公式则来自原程序点上其他的性质, 如流值  $matrix[y, null] = R(2, 2)$  对应的逻辑公式表示  $isslist(y) \wedge lengthsliseg(y) = 2$ , 自然地可以推导证明出  $y \neq null$ 。

表4 单链表可达性分析的 gen 规则

语句	GEN 前提条件	生成流值
$x := null$	无	$matrix[x, null] = R(0, 0)$
$x := y$	无	$matrix[x, y] = R(0, 0)$ $matrix[y, x] = R(0, 0)$ $matrix[x, null] = R(1, 1)$
$x := alloc$	无 $y \neq null$	$matrix[x, y] = U$ $matrix[y, x] = U$
$x := y \rightarrow link$	$y \neq null \wedge y \neq y \rightarrow link$ $y \neq null$	$matrix[y, x] = R(1, 1)$ $matrix[y, x] = R(0, 1)$
$x \rightarrow link := y$	$x \neq null \wedge x \neq y$ $x \neq null$	$matrix[x, y] = R(1, 1)$ $matrix[x, y] = R(0, 1)$
$x \rightarrow link := null$	$x \neq null$	$matrix[x, null] = R(1, 1)$
$x \rightarrow link := alloc$	$x \neq null$	$matrix[x, null] = R(2, 2)$

## 3.3.3 数据流分析中的 prop 运算和 Scope Logic 的证明规则

在 kill 和 gen 过程执行完成之后, 我们还需要执行一个 prop 过程来推导新的公式, 从而产生性质的闭包。prop 是一个迭代的过程, 它将不断产生新的流值, 直到不再变化, prop 处理规则均基于流值对应的公式结合递归函数性质进行的推导。例如, 根据分析, 由已有流值  $matrix[x, y] = R(1, 1)$  可得  $isslistseg(x, y) \wedge lengthsliseg(x, y) = 1$ , 由已有流值  $matrix[y, z] = R(2, 2)$  可得  $isslistseg(y, z) \wedge lengthsliseg(y, z) = 2$ , 由已有流值  $matrix[y, x] = U$  可得  $x \notin nodeset(y)$ , 由已有流值  $matrix[z, y] = U$  可得  $y \notin nodeset(z)$ , 由这 4 个公

值,  $OUT[a]$  表示离开程序点的流值, 入口程序点上的进入流值为用户自定义的前提条件, 其余程序点  $IN[a] = \bigcap_{p \in pred(a)} OUT[p]$  即其前驱结点上流值的交汇, 交汇(meet)运算由前面交半格中定交汇运算  $\cap$  定义, 将在 3.3.4 节中介绍。  $OUT[a] = prop(gen, \cup(IN[a] - kill))$ , 程序点  $a$  的出口流值  $OUT[a]$  要考虑  $a$  点之后的程序语句  $s$  是如何对该流值进行改变的。这里我们将流值的变化过程分为 kill、gen 和 prop 3 个步骤, 其中 kill 和 gen 要考虑到不同的语句类型进行处理, 而 prop 则是一个普适的过程。3 个步骤的执行顺序为先 kill, 再 gen, 最后 prop。数据流算法执行将依赖于对数据流方程的求解。

## 3.3.1 数据流分析的 kill 运算和 Scope Logic 的证明规则

kill 过程对应于 Scope Logic 中对公式内存范围是否被改变的分析, 通过分析语句的赋值操作是否可能修改某些链表的指针信息, 判断对应的可达性信息是否发生变化, 可能发生变化的流值将被 kill 掉, 表示我们失去了该流值信息。从 3.2.3 节可以得到流值与公式的对应, 从而可以分析其内存范围。

对于  $x := rh$  型的语句, 地址  $\&x$  被修改, 可知对于所有形如  $matrix[x, *]$  的流值, 根据 3.2.3 节, 其对应的公式表示为:  $isslist(x) \wedge a \leq lengthsliseg(x) \leq b$  或  $isslistseg(x, y) \wedge a \leq lengthsliseg(x, y) \leq b$  或  $y \notin nodeset(x)$ , 显然这些公式的内存范围都包含地址  $\&x$ , 所以这些公式将会失效, 对应的流值也会被 kill。

对于  $x \rightarrow link := rh$  型的语句,  $\&x \rightarrow link$  域将被修改, 对于任意流值  $matrix[y, z]$ , 如果分析得到  $y$  有可能到达  $x$  且  $x$  有可能到达  $z$ , 那么  $\&x \rightarrow link$  的修改可能会影响  $y$  到  $z$  的可达性, 该流值将被 kill。例如, 假设程序点  $i$  上有流值  $matrix[y, z] = R(3, 3)$ , 其对应公式为  $isslistseg(y, z) \wedge lengthsliseg(x, y) = 3$ , 如果有  $matrix[y, x] = R(1, 1)$  及  $matrix[x, z] = R(2, 2)$ , 可知  $\&x \rightarrow link$  在逻辑公式  $isslistseg(y, z) \wedge lengthsliseg(y, z) = 3$  的内存范围:  $\{\&y\} \cup \lambda(Node * p) (\&p \rightarrow link) in (nodeset(y)) + makeset(\&z)$  中, 该公式将会失效, 其对应流值  $matrix[y, z] = R(3, 3)$  将被 kill。

对于其余的流值, 其对应公式的内存范围如果未被修改, 则不会被 kill 而将被传播下去。因此我们需要根据已有的流值信息分析其对应的内存范围是否被修改。例如, 在某程序点  $i$  有流值  $matrix[y, z] = R(1, 1)$ , 其对应公式为  $isslistseg(y, z) \wedge a \leq lengthsliseg(x, y) \leq b$ , 在  $i$  点还有  $matrix[y, x] = U$ , 其公式表示为  $x \notin nodeset(y)$ , 可推得  $\&x \rightarrow link \notin \lambda(Node * p) (\&p \rightarrow link) in nodeset(y)$ 。执行语句  $x \rightarrow link := rh$ , 由前面分析可知公式  $isslistseg(y, z) \wedge a \leq lengthsliseg(x, y) \leq b$  对应的内存:  $\{\&y\} \cup \lambda(Node * p) (\&p \rightarrow link) in (nodeset(y)) + makeset(\&z)$  与  $\{\&x \rightarrow link\}$  不相交, 从而该公式在语句后仍然成立, 其对应的流值也将被传递下来而不被 kill。未被 kill 而传播下来的流值将被传播证明, 它们将依赖于传播前程序点上对应的流值和内存范围未被修改的证据公式。表 3 列出了对流值进行 kill 的规则, 可见图 3 中的例子。

式结合表 2 中的性质 2 和性质 3 可推导出公式  $isslistseg(x, z) \wedge lengthslistseg(x, z) = 3$ , 而这个公式即对应于流值  $matrix[x, z] = R(3, 3)$ 。从而该流值将在 *prop* 过程中生成。因为推导是在同一个程序点上完成的, *prop* 过程不需考虑赋值语句。*prop* 过程的推导可以依赖于 SMT solver 求解并证明, 我们总结了 7 条基本推导规则, 限于篇幅, 不再详细描述。

### 3.3.4 数据流分析中的 meet 运算

*meet* 过程对应于控制流结构上未经过赋值语句的流值传递, 仅有一个前驱程序点时, *meet* 过程直接将流值拷贝即可; 而对于多个前驱的情况, 比如 IF 语句两个分支的结尾程序点将流值传递到 IF 语句后的程序点, 这时就需要以其中一个前驱程序点的流值为基础, 逐个与其他前驱程序点的流值进行交汇运算, 这里的交汇运算即对流值矩阵中所有同一位置的流值使用图 2 中格的交汇运算进行求解。

### 3.4 结果生成和逻辑依赖关系的设置

数据流分析算法将以前文定义的数据流值和数据流方程为基础迭代求解, 迭代求解框架与文献[3]中所述的相似。可以证明, 这里的数据流值格有穷并且传递函数收敛, 所以迭代算法一定会收敛, 限于篇幅, 不再赘述。分析得到的结果以流值形式体现, 我们根据 3.2.3 节中的对应关系将其转化为公式表示形式, 分析中同时保存了流值对应公式的证明方法和依赖关系, 在对这些公式进行证明时, 相应地设置依赖关系, 具体的证明方法和依赖关系在 3.3 节各个步骤的说明过程中都有描述, 图 3 描述了一个简单的证明片段的例子, 由 *i* 点部分已证明的公式(1, 2, 3, 5)为基础, 在 *j* 点生成数据流值对应公式及其证明。公式号后的括号内注明了依赖关系。*j* 点数据流值对应公式(7)为传播证明, 公式(10)为通过依赖于自生公式(9)和证据公式(8)的证明, 公式(11)则是由公式(7)和公式(11)推导证明, 公式(1)–公式(3)为 *i* 点已证明的公式, 其余公式为辅助证据公式。其中公式(5)由 Scope Logic 公理<sup>[4]</sup>得到。

```
{i:
  1. isslistseg(r, q) ∧ lengthslistseg(r, q) = 1
  2. isslist(p) ∧ lengthslist(p) = 2
  3. p ∉ nodeset(r)
  4(2). p ≠ null(推导证明)
  5. &p ∉ {&p → link}
  6(3). &(p → link) ∉ (lambda (Node * x) (&x → link) in (nodeset(r) + {&q})) (推导证明)
}
p → link := r
{j:
  7(1, 6). isslistsg(r, q) ∧ lengthslistseg(r, q) = 1(传播证明)
  8. (4, 5). p ≠ null(传播证明)
  9. p → link = r(自生证明)
  10(8, 9). isslistseg(p, r) ∧ lengthslist(p, r) = 1(推导证明)
  11(7, 10). isslistseg(p, q) ∧ lengthslistseg(p, q) = 2(推导证明)
}
```

图 3 分析所得流值转化为的公式及证明的例子

## 4 实例研究

为了验证本文分析框架的有效性, 我们选取了常见的单链表操作代码片段, 包括单链表顺序遍历、单链表顺序查找、单链表反转和单链表的插入、删除、拆分等代码, 使用基于本文所述方法开发的工具, 给出各种前置条件进行自动化验证, 实验过程表明分析工具对于正确的程序可以得到我们需要的可达性信息, 而对于错误程序则无法得到。在本文数据流分析框架的基础上, 我们还实现了对空指针性质和整数范围性质的分析。结果表明, 本文方法可自动分析得到用户指定的性质, 并生成 Scope Logic 中的公式及证明, 具有对一般程序性质的一般表达与分析能力。这使得数据流分析成为代码验证中的一个自动化步骤, 且其分析结果能够在后续的代码验证中被使用。

**结束语** 本文讨论了代码验证中手工分析用户特定性质的困难, 提出了一种基于 Scope Logic 代码验证和数据流分析技术自动对用户特定程序性质进行分析的框架, 并以单链表可达性分析为例, 介绍了如何将单链表可达性性质定义为递归函数表示的公式, 如何为其设计数据流值和数据流方程, 并融合 Scope Logic 的基本证明方法进行推导和分析, 最终将分析得到的结果以公式的形式表达, 以便于其他的形式化验证和分析过程使用。不难看出, 本文方法有较强的灵活性, 可以对许多用户定义的性质进行转换和分析, 可以提高程序性质验证的自动化程度和准确性。本文采用的逻辑基础是 Scope Logic, 本文的结果表明 Scope Logic 具有很好的灵活性, 可以成为不同程序分析技术的共同逻辑基础。我们认为本文的思想也可用于其它的代码验证逻辑和数据流分析技术。

## 参考文献

- [1] Hoare C A R. An axiomatic basis for computer programming [J]. Communications of the ACM, 1969, 12(10): 576-580
- [2] D'silva V, Kroening D, Weissenbacher G. A survey of automated techniques for formal software verification [J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2008, 27(7): 1165-1178
- [3] Aho A V. Compilers: Principles, Techniques and Tools [M]. Pearson Education India, 2003
- [4] Z Jian-hua, L Xuan-dong. Scope Logic: An Extension to Hoare Logic for Pointers and Recursive Data Structures [C] // Theoretical Aspects of Computing-ICTAC 2013. Springer Berlin Heidelberg, 2013: 409-426
- [5] De Moura L, Bjørner N. Z3: An efficient SMT solver [M] // Tools and Algorithms for the Construction and Analysis of Systems. Springer Berlin Heidelberg, 2008: 337-340
- [6] Khedker U, Sanyal A, Sathe B. Data flow analysis: theory and practice [M]. CRC Press, 2009