Z 规格说明的前置条件的简化*

缪淮扣

(上海大学计算机科学系 上海 201800)

摘要 在软件方法学中,形式方法越来越受到人们的重视,并已被应用于软件开发. Z 是一种基于数学表示的软件规格说明方法. 前置条件的简化是 Z 规格说明方法中一种标准的检查,本文讨论了 Z 规格说明中关于操作的前置条件及其计算. 提出了简化过程的终止条件,给出了一个用于简化前置条件的算法,该算法可自动产生简化过程的证据.

关键词 形式方法,Z 规格说明,前置条件,简化.

中图法分类号 TP311.1

形式方法是一种基于严格数学表示的软件开发方法. 在软件工程领域中,形式规格说明是关于形式方法的重要技术. 形式规格说明用数学表示法精确地描述了软件系统必须具有的性质. 对软件规格说明技术的日益重视引起了各种形式规格说明语言的开发. Z[1~3]是目前使用较为广泛的一种规格说明语言. 在过去的 10 年中,从各种关于 Z 的实例的研究中获得的经验表明,大量的规格说明问题都可以用 Z 方便地书写. Z 的表示方法构造了建立规格说明所需要的更复杂的数据结构的基础.

为了确信 Z 规格说明具有所需要的特性,就要对其进行形式推理. 此外,在从规格说明到程序的求精(Refinement)过程中也需要形式推理. 用数学的方法对规格说明进行形式推理是 Z 相对于其它规格说明方法的一个主要优点. 在关于 Z 的规格说明的形式推理中,初始化定理的证明[4]和前置条件(Precondition)的简化是两个标准的检查. 前置条件描述这样一些状态,由这些状态,一个操作可以成功地执行.[1,3]前置条件的简化可消除规格说明的冗余性,便于进行软件求精和程序变换. 虽然现在有些 Z 的支撑工具[5,6]也含有前置条件的简化,但都是由用户引导的交互式的过程,并无法保存简化过程中的证据. 本文给出了一个自动的前置条件的计算、预处理和简化的算法. 该算法能自动生成简化过程中的证据.

1 Z 规格说明表示与前置条件及其计算

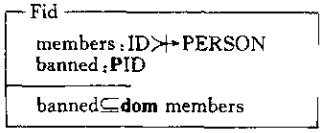
Z 的最主要的结构是模式(Schema). 模式可用来描述软件系统的各种状态和操作. 它支持模块化的设计和软件重用(Reuse). 一个模式是由变量说明和谓词两部分组成的. 我们

* 本文研究得到国家 863 高科技项目基金资助. 作者缪淮扣,1953 年生,副教授,主要研究领域为软件形式方法,自动推理.

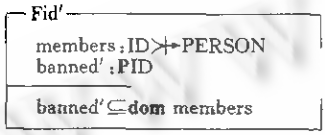
本文通讯联系人:缪淮扣,上海 201800,上海大学计算机科学系

本文 1996-10-17 收到修改稿

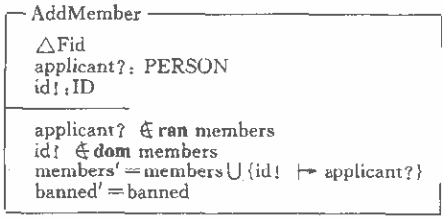
考虑一个关于球迷会员身份的例子. 每个球迷可以有一个唯一的身份码,已被禁止看球赛的肇事者的身份码被列入了一个清单. 给出基本类型 PERSON 和 ID,PERSON 是人的集合,ID 是所有可能的身份码的集合. 我们先说明给定集合(Given Set),[PERSON,ID],并以名为 Fid 的模式来描述这个系统的状态.



这里 members 的类型是一个人射函数(Injective Function),表示每个人只能有 1 个身份码,一个身份码也只能属于一个人,所有的身份码并不一定都被分发完. banned 是集合类型,含有已注册的人的身份码的子集. 谓词刻画了一个不变式. Fid 的后状态以修饰状态 Fid′ 表示.



模式可作为类型和谓词使用. 模式有许多操作. 在 Z 中,输入变量的后面跟上字符"?";输出变量的后面跟上字符"!". 对 Fid,可以有增加一个新成员的操作. 该操作可描述为



这里△Fid 是一个模式,

AddMember 具有如下性质:它的输入是一个申请会员的名字,一个身份码被颁发给他;颁发前,申请者不是一个会员,身份码未被使用;函数 members′ 以新的信息修改之;集合 banned′ 就是 banned. 关于 Z 语言的细节可参阅文献[1~3].

一个操作的前置条件是一个模式,可以用一个模式运算符 Pre 得到. 设操作为 OP,则其前置条件为 PreOP. 对于一个操作 OP,如果对前状态 state 和输入变量的组合存在一个后状态 state′ 和输出变量 out! 满足所说明的变量之间的关系,则 OP 可成功地执行,即有 ∃ state′;outs! · OP. 前置条件中的谓词涉及到一个操作的前状态和后状态,在 VDM[7] 形式规格说明方法中用一对谓词,前置条件和后置条件(Post-condition)来描述一个操作. 但在 Z 形式方法中,后置条件不直接出现,而是用一个前置条件的模式表示前置条件和后置条件的关系. 事实上,该模式是关于操作的前、后置条件断言的规约. Z 的前置条件的计算过程就可用两个步骤来完成,删除 OP 模式中说明部分所含的后状态变量和输出变量,并在谓词部分对它们进行存在量词的量化. 这样,Addmember 的前置条件可以用模式 PreAd-

dmember 描述之. 这里后状态变量是 Fid′,输出变量是 id!.

PreAddmember

Fid
applicant?:PERSON

∃ Fid′;id!:ID ·
　applicant? ∉ **ran** members ∧
　id! ∉ **dom** members ∧
　members′ = members ∪ {id! ↦ applicant?} ∧
　banned′ = banned

量词的引入产生了过于复杂的谓词. 在一般情况下,在一个定律库下,前置条件模式的谓词部分可以被简化为一个比较简洁但逻辑上等价的语句(Statement). 简化的结果是否正确依赖于所用的定律. 形式地讲,应是 $L \Rightarrow P_1 \Leftrightarrow P_2$,这里,$P_1$ 和 P_2 分别是简化前后的前置条件,L 是定律库. 前置条件模式中经过化简后的公式是否为最简公式的问题是不可判定的. 但我们认为,在实际情况下,前置条件的谓词公式中的存在量词和后状态变量都可以被消除. 如进一步再设法消除输出变量,并删除谓词公式中的冗余部分,则得到的谓词相对来说就比较简单了.

2　一个简化前置条件的算法

给出一个前置条件简化算法的困难在于,在简化的过程中,无法判断一个谓词公式是否为最简形式,因而就难以给出简化算法的终止条件. 我们根据 Z 规格说明中前置条件的表示,设计了一个基于定律库的简化过程. 该过程的终止条件是:被简化的谓词中不含输出变量. 我们简化的目的就是用输入变量和系统的其它变量来表达操作的前置条件. 这个简化过程可产生一个相当简单的前置条件模式,并能自动产生简化的证据. 简化过程的核心是选择可用的定律重写谓词公式. 因此和证明 Z 的规格说明的定理一样,简化前置条件也需要一个含有逻辑定律和数学定律的定律库. 一个形式推导应提供详细的证据,即在推导过程的每一步应用了什么规则或定律.

为了便于进行逆向推理,定律的表示方法是,如果定律 L 有前提部分,则前提部分放在"⇐"的右面,表示为 L. right. 关于定律库和证据,我们在文献[3]中已做了叙述. 下面的简化过程是采用逆向推理的方式进行.

```
Procedure Precondition-Simplification
{对于给定的系统状态 SS 的一个操作模式 OP,本过程产生一个经过简化的前置条件模式}
1.  PreOP←calculate(OP); {计算 OP 的前置条件}
2.  pre-simplification; {一个预简化的过程,构成 PreOP 的 dec_part 和 predicate_part}
3.  set-null (sg_list); {目标列表置空}
4.  bt_stk←nil;
5.  put-in-sg_list; {将 predicate_part 中的各子目标依次移入 sg_list}
6.  WHILE sg_list 中的子目标均不含输出变量 DO
7.  [sg←take-head(sg_list); {从 sg_list 的头部取出一个子目标}
8.  IF sg∈dec_part
9.  THEN push-bt_stk(sg,'D',j); {D 是标志,表示 sg 对应于一个说明,j 是该说明在说明部分中的序号}
10. ELSE IF 在定律库中有一条可应用的定律 Lj 与 sg 可匹配
11.      THEN [IF sg 是 Lj 的 instance{即 Lj 没有前提部分,即 Lj. right 为空}
12.            THEN push-bt_stk(sg,'L',j)
13.            ELSE [push-bt_stk(sg,'L',j)
14.                  sgs←substitution(Lj. right); {sgs 为重写的 Lj 的前提部分(即 L. right)的 instance}
15.                  check-push-sg_list(sgs); {对 sgs 中的每一子目标 t,若 t∉sg_list,则将 t 置入 sg_list 的头
```

```
                部}];
16.     ELSE IF 在定律库中有一条可应用于 sg 的子项的重写定律 Lj
17.          THEN [push-bt_stk(sg,'L',j)
18.                rewrite(Lj,sg); {以 Lj 重写 sg 中的子项}
19.                push-sg_list(sg); {将 sg 置入 sg_list 的头部}]
20.          ELSE [remove-exits; {根据有关定律消去存在量词及量词变量}
21.                IF sg 中不含输出变量
22.                THEN en-sg_list(sg) {将 sg 置入 sg_list 的尾部}
23.                ELSE IF not empty(bt_stk)
24.                     THEN [(sg,j)←pop(bt_stk); GOTO 8] {回溯}
25.                     ELSE exception;
                   ]
   ];
26. print-bt_stk; {输出 bt_stk 中的证据}
27. recover-schema; {恢复前置条件的模式表示}
28. END.
```

其中 pre-simplification 为

```
Procedure pre-simplification
{预简化 PreOP}
1. expanding; {由定义,展开 PreOP 中的模式分量}
2. dec  part←PreOP 的说明部分对应的谓词;
3. predicate_part←PreOP 的谓词部分;
4. eliminating; {如果 predicate_part 中出现'|',则根据定律∃ J|P · Q⇔J · P∧Q 和∀ J|P · Q⇔∀ J · P⇒Q 删除'|'}
5. opr-app; {对 predicate_part as 多次应用点规则}
6. transform; {将 predicate_part 转换成子目标集}
7. end;
```

这里,点规则(One Point Rule)为

$$\exists\ x:S \cdot (P \wedge x=t) \Leftrightarrow t \in S \wedge P[t/x] \qquad (X \backslash t)$$

说的是,若有一个存在性量词量化了的谓词,该谓词中的部分量词化的变量有明确的值,那么在该谓词中就可用这些值来替换对应的量词变量的出现.(X\t)是条件,t 中不含 X.

在过程 precondition-simplification 中,栈 bt_stk 的元素是一个三元组(sub-goal,tag,index),它表示了一个子目标、推导步的类型标志和所使用的定律在库中的下标.当一步推导或重写执行前,有关信息被存入 bt_stk,以备后面可能的回溯步骤使用.当所有的子目标都不含输出变量,bt_stk 栈中由栈底到顶的记录构成了证据的集合,每一个记录给出了一个子目标和被应用的定律或前提的下标.sg_list 是子目标的列表.简化过程每次从 sg_list 的头部取一个子目标,如果该子目标与说明部分中的某一说明相同,则删除该子目标,否则,使用定律进行推理或重写."匹配"是一个查找可用的定律或一个前提的子过程,它含有自动定理证明中通常的合一操作(Unification).子目标和定律相同的表示使合一更容易些.rewrite 是一个用重写定律重写子目标中的表达式的过程.有两种重写定律:数学等价定律和逻辑等价定律.如果子目标中的一个表达式和一条数学重写定律"="的一边或逻辑重写定律"⇔"的一边匹配,rewrite 将以"="或"⇔"的替换了的另一边来代替该表达式.重写定律的使用应避免无限重写序列的出现.因此,可能需要对重写定律作特殊的处理.对一个含有输出变量的子目标,如果没有一条可应用的定律并且 bt_stk 为空,过程 exception 则向用户提问是否想增加一条新的定律到库中,如果回答是肯定的,则系统添加一条新定律到库,将子目标 sg 移入 sg_list 并继续 while 循环;否则,显示异常信息,而且过程中止.严格的说,新加入的定律应该用 Z 的定义和现存的定律证明之.过程 substitution(L. right) 根据"匹

配"得到的项对变量的替换集合来对 L. right 中的变量进行替换. 过程 remove-exits 对当前含有存在量词变量的子目标,再取出 sg_list 中其它含有相同存在量词的子目标,查找有关逻辑定律,消除存在量词及其变量,并重写子目标. print-bt_stk 和 recover-schema 都是一些不太困难的操作. 此外,过程中还含有一些常见的关于栈和队列的操作.

3 例　子

假定我们已经有了一个定律库. 为了便于进行逆向推理,定律的表示方法是,如果定律 L 有前提部分,则前提部分放在"⇐"的右面,表示为 L. right. 现在以给定的 AddMember 为操作模式 OP 来说明算法 Precondition-Simplification 的执行. 调用 calculate,产生的前置条件模式 PreOP 为

PreAddmember

Fid
applicant?:PERSON

∃ Fid′;id!:ID •
　applicant? ∉ **ran** members ∧
　id! ∉ **dom** members ∧
　members′ = members ∪ {id! ↦ applicant?} ∧
　banned′ = banned

调用 Pre-Simplification:执行 expanding,即展开 Fid 和 Fid′,dec_part 为{members ∈ ID ⇸ PERSON,banned ∈ **P**ID,applicant? ∈ PERSON,banned ⊆ **dom** members},predicate_part 为

∃ members′:ID ⇸ PERSON; banned′:**P**ID;id!:ID •
　banned′ ⊆ **dom** members′ ∧
　applicant? ∉ **ran** members ∧
　id! ∉ **dom** members ∧
　members′ = members ∪ {id! applicant?} ∧
　banned′ = banned

因无"|",所以执行步骤 eliminating 后该谓词部分不变化,再执行 opr-app,由于量词变量 members′ 和 banned′ 有确定的值,故结果 predicate_part 为

∃ id!:ID •
　members ∪ {id! ↦ applicant?} ∈ ID ⇸ PERSON ∧
　banned ∈ **P**ID
　banned ⊆ **dom** (members ∪ {id! ↦ applicant?}) ∧
　applicant? ∉ **ran** members ∧
　id! ∉ **dom** members

执行步骤 3、4,sg_list 和 bt_stk 均为空,执行 5,sg_list 为{banned ∈ **P**ID, banned ⊆ **dom**(members ∪ {id! ↦ applicant?}),members ∪ {id! ↦ applicant?} ∈ ID ⇸ PERSON, applicant? ∉ **ran** members, id! ∉ **dom** members}.

执行步骤 6,过程进入循环. take-head 取出第 1 个子目标 banned ∈ **P**ID 至 sg,因为 sg ∈ dec_part,所以执行 push-bt_stk,三元组(banned ∈ **P**ID, 'D', 2) 压入栈 bt_stk,这里 'D' 是使用了系统状态说明部分的成分的标志,2 是该成分在 dec_part 中的位置. 第 2 次由 take-head 取出子目标 banned ⊆ **dom**(members ∪ {id! ↦ applicant?}) 至 sg. 查找到重写定律 **dom**

$(A \cup B) = \mathbf{dom}\ A \cup \mathbf{dom}\ B$,(banned$\subseteq$**dom** (members$\cup${id! $\vdash\!\!\rightarrow$applicant?}, 'L',i1)压入栈 bt_stk,其中'L'为使用定律的标志,i1 为重写定律在库中的下标.应用重写定律,rewrite 使 sg 变为 banned$\subseteq$**dom** members$\cup$**dom**{id! $\vdash\!\!\rightarrow$applicant?}.将 sg 置入 sg_list 的头部.又取 sg_list 中的子目标 banned$\subseteq$**dom** members$\cup$**dom**{id! $\vdash\!\!\rightarrow$applicant?}为 sg,查找到逻辑定律 $A \subseteq B \cup C \Leftarrow A \subseteq (B$,(banned$\subseteq$**dom** members$\cup$**dom**{id! $\vdash\!\!\rightarrow$applicant?}, 'L',i2)压入 bt_stk,sg 为该逻辑定律的右部的例示 banned$\subseteq$**dom** members.新的 sg 被置入 sg_list.再取 sg_list 头部的子目标 banned$\subseteq$**dom** members 为 sg,因为 sg$\in$dec_part.所以,(banned$\subseteq$**dom** members, 'D',4)压入 bt_stk.现在取出子目标 members$\cup${id! $\vdash\!\!\rightarrow$applicant?}$\in$ID$\rightarrowtail\!\!\!\!\!\!+$PERSON 进行处理.查找到关于函数的逻辑定律 $f \cup \{x \vdash\!\!\rightarrow y\} \in X \rightarrowtail\!\!\!\!\!\!+ Y \Leftarrow f \in X \rightarrowtail\!\!\!\!\!\!+ Y \wedge x \in X \wedge y \in Y \wedge x \notin \mathbf{domf} \wedge y \notin \mathbf{ranf}$,(members$\cup${id! $\vdash\!\!\rightarrow$applicant?}$\in$ID$\rightarrowtail\!\!\!\!\!\!+$PERSON, 'L',i3)压入 bt_stk.sgs 为重写过的该逻辑定律的右部,含 5 个子目标:members$\in$ID$\rightarrowtail\!\!\!\!\!\!+$PERSON,id! $\in$ID,applicant? $\in$PERSON,id! $\notin$**dom** members,applicant? $\notin$**ran** members.因为后两个子目标已在 sg_list 中,故 check-push-sg_list 只将前 3 个子目标置入 sg_list 的头部.取出 sg_list 中的第 1 个子目标 members$\in$ID$\rightarrowtail\!\!\!\!\!\!+$PERSON 为 sg,因 sg$\in$dec_part,所以(members$\in$ID$\rightarrowtail\!\!\!\!\!\!+$PERSON, 'D',1)压入 bt_stk.取出子目标 id! $\in$ID,它既不属于 dec_part,也没有定律可应用.调用过程 remove-exists,取出 sg_list 中另一子目标 id! $\notin$**dom** members,构成的完整的谓词 sg 是$\exists$ id!:ID $\cdot$ id! $\in$ID$\wedge$id! $\notin$**dom** members,查找到逻辑定律$\exists$ x:Z $\cdot$ x$\in$Z$\wedge$ x$\notin$A$\Leftarrow$A$\neq$Z,相应的三元组也已压入了 bt_stk,sg 重写为 **dom** members$\neq$ID,存在量词及其变量被消除了.新的子目标被置入 sg_list 的尾部.再取出 sg_list 中的子目标 applicant?$\in$PERSON 为 sg,因 sg$\in$dec_part,故(applicant? $\in$PERSON, 'D',3)压入 bt_stk.现在检查 precondition-simplification 中步骤 6 中 WHILE 的条件,此时 sg_list 中的所有子目标均不含输出变量.它们是 applicant?$\notin$**ran** members 和 **dom** members$\neq$ID,退出 WHILE 循环.显示 bt_stk 中的内容.最后调用 recover-schema,获得简化的前置条件模式为

PreAddmember

Fid

applicant?:PERSON

---

applicant? $\notin$**ran** members

dom members$\neq$ID

AddMember 操作可以成功地执行的前置条件是:applicant 还不是一个会员并且 ID 中还有身份码可颁发.

4 结束语

从理论上说,对 Z 规格说明中的任一操作的前置条件模式,不存在一个算法将其化简为最简前置条件模式.但采用本文提出的算法可将一个前置条件简化,使其复杂度明显降低.前置条件简化和初始化定理的证明是 Z 规格说明方法中两个标准的检查,也是 Z 规格说明的形式推理的重要组成部分.

参考文献

1 Diller A. Z an introduction to formal method. London: John Wiley & Sons, 1990.

2 Spivey J M. The Z notation: a reference manual (second edition). London: Prentice Hall, 1992.
3 Potter B, Sinclair J, Till D. An introduction to formal specification and Z. London: Prentice Hall, 1992.
4 Miao Huaikou, McDermid J A, Toyn I. Proving the existence of state in Z specifications. Chinese Journal of Advanced Software Research, 1995,2(4):326~337.
5 Neilson D, Prasad D. zedB: a proof tool for Z built on B. In: Proceedings of the Sixth Annual Z User Meeting, 1992.
6 Jordan D, McDermid J A, Toyn I. CADiZ-computer aided design in Z. In: Nicholls J E ed. London: Springer Verlag, 1991. 93~104.
7 Jones C B. Systematic software development using VDM. Prentice-Hall International, 1986.

THE SIMPLIFICATION OF PRECONDITION IN Z SPECIFICATIONS

MIAO Huaikou

(*Department of Computer Science Shanghai University Shanghai* 201800)

Abstract In software methodology, formal methods is being paid more and more attention and has been applied to software development. Z is a kind of software specification notations based on mathematics. The simplification of precondition is a standard check for Z specifications. This paper discusses the precondition in Z specifications and its calculation, proposes a termination condition for simplifying the precondition and presents a simplifying algorithm which can automatically produce the justifications during the process of simplifying precondition.

Key words Formal methods, Z specifications, precondition, simplification.
Class number TP311.1