Z规格说明 Z语言 定理证明 集合论 (13)

计算机科学1998 Vol. 25 №. 1

51-54

Z规格说明中的定理证明方法*)

A Theorems Proving Method in Z Specifications

缪淮扣 朱关铭

(上海大学计算机科学系 上海201800)

TP301.2

摘 要 This paper discusses the theorems proving in Z specifications, presents a proof algorithm. The proof justifications can be generated automatically by this algorithm. By way of example, two theorems are proved using this algorithm.

关键词 Formal methods, Specification, Z, Theorem proving, Justification

1 引言

形式规格说明使用数学的表示,以一种精确的方法描述了系统要做什么,而不考虑它是如何做的。规格说明本身提供了一个无歧义的、能与用户和同事一起讨论的书面文件,又可作为已完成程序的文档,帮助人们将来进行程序的维护和修改。对形式规格说明的日益重视导致了各种规格说明语言的产生。这些语言一般都以数学概念和表示为基础而构成。Z[1,2]和VDM[3]就是比较著名的形式规格说明的表示。

Z语言是基于一阶逻辑和集合论的数学表示的规格说明语言,Z的设计者的目的就是要设计这样一个规格说明语言,它能精致地描述和建立软件系统的模型,并且能证明程序满足它的规格说明。本文讨论了Z规格说明中的定理证明问题,给出了一个证明算法。该算法是文[8,9]中给出的算法的扩充。

2 Z规格说明语言及其定理证明

Z以一阶逻辑和集合论作为形式语义基础,将函数、映射、关系等数学方法用于规格说明。Z的主要结构是模式,一个模式由模式名、变量说明部分和谓词部分组成。模式将系统状态的信息组合在一起,其主要用途就是说明状态和状态转换,一个状态就是一组集合、对象以及定义在它们上面的谓词,模式可以用一个二维图形来表示。作为一个例子,我们考虑下面一个规格说明。

一个政府提出建立一个足球迷的身份卡的模式,每个球迷可以有一个唯一的身份码,而所有的肇事者的身份号码已被列入了一个清单,这些肇事者已被禁止看球赛。规格说明引入了基本类型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PERSON,再用模式Fid来描述这个系统的状态。这里Fid是模式名,members的类型是一个内射函数(injective function),意味着每个人只能有一个身份码,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的类型是集合类型,含有已注册的人的身份码的子集。谓词banned⊆dom members刻画了一个不变式。

[PERSON, ID]

Fid

members : ID ⤔ PERSON

banned : P ID

---

banned ⊆ dom members

修饰的状态Fid′是Fid的后状态,定义为:

Fid′

members′ : ID ⤔ PERSON

banned′ : P ID

---

banned′ ⊆ dom members

模式可作为类型和谓词使用。模式有许多操作。在Z中约定输入变量的后面跟上字符"?";输出变量的后面加上字符"!"。对Fid可以有增加一个新成员的操作。该操作可表示为:

*)得到国家自然科学基金资助,项目编号:69773038。缪淮扣 教授,主要研究领域为自动推理,软件形式方法。朱关铭 教授,主要研究领域为软件形式方法,人工智能。

AddMember
ΔFid
applicant?:PERSON
id!:ID
---
applicant? ∉ ran members
id! ∉ dom members
members′ = members ∪ {id! ↦ applicant?}
banned′ = banned

这里 ΔFid 是一个模式：

ΔFid
Fid
Fid′

AddMember 具有如下性质：它的输入是一个申请会员的人的名字，一个身份码被颁发给他；申请不能已经是一个会员；身份码不能是一个已经在使用的身份码；函数 members′的新的信息修改之；集合 banned′不改变。禁止一个人看球赛的操作定义为：

BanMember
ΔFid
ban?:ID
---
ban? ∈ dom members
banned′ = banned ∪ {ban?}
members′ = members

关于 Z 语言的细节可参阅[1,2]。用 Z 的表示我们可以写出定理来记录一些重要的事实和性质，其一般形式为：

Decls | Pred ⊢ Conclu

这样一个定理的证明说明了由其前提可推出其结论。一个定理的证明就是在给定的前提下进行推理，给出每一步的证据，最后得到所需要的结论。为了说明：对一个已被禁止看球赛的人，在禁止他看球赛，那么系统的状态不改变。我们需要证明如下定理：

BanMember | ban? ∈ banned ⊢ ΞFid

这里"Ξ"表示系统的状态不改变，即

ΞFid
ΔFid
---
members′ = members
banned′ = banned ∪ {ban?}

在实际的应用中，有一种标准的定理，它可以用规格说明来表示。如果 S 是一个描述了状态的模式名，那么系统可能的初始状态就可以用另一个模式 InitS 来表示。Fid 的初始状态定义为：

InitFid
Fid′
---
members′ = Φ
banned′ = Φ

我们要证明的是，对这个系统有一个合适的初始状态存在。这就是初始化定理，它具有如下形式：

⊢ ∃ Fid′ • InitFid

3 数据表示、定律库和证据

为了证明 Z 规格说明的定理，应该建立一个定律库。库中含有我们需要的逻辑定律和各种数学定律。这些定律已被证明是正确的定律。这样每个人都可放心地使用库中的定律。在[2]中已经列出了几百条这种定律。对于要证明 Z 规格说明定理的人来说，这些定律是最合适的起点。一个定理的前提可以临时地被看成是一个子库，并被放在定律库的前面与定律库连接，新连接而成的库称为 dB。库中的定律和定理中的子目标具有相同的表示。我们的算法是一个逆向证明过程，所以，子目标和定律表示为如下形式：

(1)A　(2)A←$B_1, B_2, \cdots, B_n$

第(2)种形式表示了：如果 $B_1 \wedge B_2 \wedge \cdots \wedge B_n$ 成立，则 A 成立。

用这种表示方法，Φ⊆S，∀ x:Φ • P(x)，dom(f ∪g)=dom f ∪dom g 和 id X⊆Q∧R⊆Q∧Q;Q⊆Q⇒R* ⊆Q 可分别表示为如下形式：

(1)Φ⊆S.

(2)∀ x:Φ • (Px).

(3)dom(f∪g)=dom f∪dom g.

(4)R* ⊆Q←id X⊆Q,R⊆Q,Q;Q⊆Q.

一个形式化的证明应提供详细的证明证据，即在证明过程的每一步应用了什么规则或定律。正确地记录证明证据和证明过程本身一样困难。我们设计了一个栈 bt-stk 记录证据。bt-stk 的元素是一个偶对(sub-goal,index)，它记录了当前子目标，以及所使用的规则或定律。在证明每一个子目标之前，(#,0)被压入 bt-stk，以表示一个子目，下一个子目标的证明开始。证明结束时，由栈底到顶的记录则是整个证明的证据。

4 证明方法

我们的算法称之为 ProTISZ(Proving Theorems In Specification of Z)，它含有 simplify 和 prove 两个过程。我们描述 simplify 如下：

Procedure simplify(P)
{simplify the predicate P}
(1) expanding(P); {expand the schema components in P by their definitions}
(2) eliminating(P); {eliminating '|' if they occur in P by the laws ∃ J|P • Q⇔∃ J • P∧Q and ∀ J|P • Q⇔

∀ J·P⇒Q}
(3)opr-app(P);{apply one point rule to P as many times as possible}
(4)transform(P);{transform P into conjunctive form}
(5)**end**;

点规则(one point)为：

∃ x:S·(P∧x=t)⇔t∈S∧P[t/x]　(X\t)

是指，如果我们有一个存在性量词量化了的谓词，该谓词中的部分量化的变量有明确的值，那么就可以用这些值来替换对应的量词变量的出现。

我们需要一个栈，叫做 sg-stk，来保证证明过程中产生的待证子目标。过程 ProTISZ 依次从 sg-stk 中取一个子目标，并证明之，直至 sg-stk 成为空栈。此时，定理证明完毕。过程 prove 证明了 sg-stk 栈中所有的子目标。为了简单起见，假设变换了的谓词是子目标的合取式。现在给出过程 prove 如下：

```
procedure prove;
{prove all the sub-goals in sg-stk}
(1)bt-stk←nil;
(2)while sg≠nil do
(3)  sg←pop(sg-stk);
(4)if sg is a primitive sub-goal then push((#,0),bt-stk)
   {to indicate that the proof of a new sub-goal begins}
(5)j←0;
(6)search-match(sg,dB,j,u);
   {search a hypothesis or a non-rewriting law,u is the
   most general unifier if the search succeeds and the uni-
   fication takes}
(7)if j≠0 then
(8)  push((sg,j),bt-stk);
(9)if dB[j].right≠nil then
(10)push((dB[j].right)u, sg-stk){put the substituted dB
   [j].right into sg-stk}
(11)
(12)else
(13)  search-rew(sg,dB,j,u);{search a hypothesis or a
      rewriting law}
(14)if j≠0 then
(15)  push((sg,,j),bt-stk);
(16)rewrite(sg,dB[j],u);{rewrite the sg'by the rewriting
    law dB[j]}
(17)push(sg,sg-stk)
(18)
(19)else if top(bt-stk)=(#,0)then
(20)exception
(21)else
(22)  (sg,j)←pop(bt-stk);
(23)goto(6){backtracking}
(24)
(25)
(26)
(27)print(bt-stk);
(28)end
```

prove 中的 search-match 是一个查找可用的定律或一个前提的过程。子目标和定律相同的表示使合一更容易些。u 是最一般的合一替换，j 是定律或前提的下标。search-rew 是一个与 search-match 类似的过程，但它查找的是一条可用的重写定律。rewrite 是一个用重写定律，重写子目标中表达式的过程。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重写定律'='或'⇔'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。

对一个子目标，如果没有一条可应用的定律选择，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，则系统将子目标 sg 压入栈 sg-stk 并继续 **while** 循环；否则显示异常信息而且过程终止。严格地说，新加入的定律应该用 Z 的定义和现存的定律证明之。

我们现在可以描述证明算法 ProTISZ。

```
procedure ProTISZ;
{prove the theorem in Z specification}
(1)if hypotheses≠nil then
(2)
(3)check;{check the variables in conclusion is in the scope
   of the declaration in hypotheses}
(4)simplify(hypotheses);
(5);
(6)simplify(Conclu);
(7)if hypotheses≠nil then{merge hypotheses into library
   and dB is the merged library}
(8)merge(hypotheses,library,dB);
(9)separate(Conclu,sg-stk)
   {separate the Conclu and push the sub-goals onto sg-
   stk};
(10)prove;
(11)end
```

5 例

为了演示例子的方便，我们不列出库中的几百条定律，而只建立一个非常小的定律库。在实际中，一个库应良好地组织，以提高查找的效率。假设这个小的定律库含有如下定律。

(L1)Φ∈S.
(L2)S∈P T←S⊆T
(L3)Φ⊆S.
(L4)Φ∈S≻⊢→T.
*(L5)**dom**Φ=Φ.
*(L6)**ran**Φ=Φ.
*(L7)**dom**(f∪g)=**dom** f∪dom g
(L8)S∪{x}=S←x∈S.
(L9)S∪T=S←T⊆S.
(L10){x}⊆S←x∈S.
*(L11)∃ x:S·x∉T⇔S≠T.
*(L12)∀ x:S·x∈T⇔S=T.
(L13)S⊆T∪R←S⊆T.
(L14)S⊆T∪R←S⊆R.
(L15)f∪{x⊢→y}∈S≻⊢→T←y∉ran T,x ∉ **dom** R.

带有"*"的定律是重写定律。

例1　证明初始化定理

⊢∃ Fid'· InitFid

ProTISZ 调用 simplify，展开 Fid'：

∃ members':ID≻⊢→PERSON ∧ banned':P ID | banned'⊆**dom** members'·members'=Φ∧banned'=Φ

删除"|":

$\exists$ members′:ID≻⊢→PERSON ∧ banned′:P ID •
banned′⊆ **dom** ∧ members′ ∧ members′= Φ ∧
banned′=Φ

应用点规则:

Φ∈ID≻⊢→PERSON ∧Φ∈P ID ∧Φ⊆**dom**Φ

将目标分离成三个子目标:

s1:Φ∈ID≻⊢→PERSON.

s2:Φ∈P ID

s3:Φ⊆**dom** Φ.

并压入栈 sg-stk。现在调用 prove。对 s1,查找到定律 L4,u={ID/S,PERSON/T}。对 s2,查找到定律 L2,u={Φ/S,ID/T},定律的右部替换为 Φ⊆ID 并作为新的子目标压入 sg-stk。对这个新的子目标再查找 dB,它与定律 L3匹配,u={ID/S}。对 S3,查找到定律 L5,**dom** Φ 重写为 Φ,新的子目标 Φ⊆Φ 压入 sg-stk。对子目标 Φ⊆Φ 查找 dB,定律 L3与之匹配。栈 bt-stk 现在为{(Φ⊆Φ,3),(Φ⊆**dom**Φ,5),(#,0),(Φ⊆ID,3),(#,0),(Φ∈ID≻⊢→PERSON,4),(#,0)}。sg-stk 为空,调用过程 print,逆序打印 bt-stk 中的证据:

{(Φ∈ID≻⊢→PERSON,4),(Φ⊆ID,3),(Φ⊆**dom** Φ,5),(Φ⊆Φ,3)}

例2 证明定理:

BenMember|ban?∈banned ⊢ ΞFid′

定理中使用模式 ΞFid′表示一个谓词,因此必须检查该模式中说明的变量是否在模式 BenMember 的范围内。ProTISZ 先调用过程 check,内含的 ΔFid 保证了它的变量在范围内。为了简化前提,调用过程 simplify,展开它:

members:ID≻⊢→PERSON ∧ members′:ID≻⊢→PERSON
∧ banned:P ID ∧ banned′:P ID ∧ ban?:ID
∧ ban?∈**dom** members ∧ banned′=banned∪{ban?}
∧ members′=members|ban?∈banned

删除"|":

members:ID≻⊢→PERSON ∧ members′:ID≻⊢→PERSON
∧ banned:P ID ∧ banned′:P ID ∧ ban?:ID
∧ ban?∈**dom** members ∧ banned′=banned∪{ban?}
∧ members′=members ∧ ban?∈banned

因为没有存在量词和点,故不使用点规则。经过变换,以下九个假设被置于定律库的前部,从而构成 dB。

members∈ID≻⊢→PERSON, members′∈ID≻⊢→PERSON, banned∈P ID, banned′∈P ID, ban?∈ID, ban?∈**dom** members, banned′= banned ∪ {ban?}, members′= members, ban?∈banned

结论 ΞFid′容易地被简化成:

members′=members ∧ banned′=banned

它被分离为:

s1:members′=members

s2:banned′=banned

并被压入 sg-stk。现在调用 prove。对子目标 s1,因为它是前提之一,故在库 dB 中查找到 members′=members,S1得证。对 s2,在 dB 中查找前提 banned′=banned ∪ {ban?},重写 s2得新子目标 banned∪{ban?}=banned。对这个新子目标,定律 L8与之能匹配,u={banned/S,ban?/x},定律 L8的右部被替换成了 ban?∈banned,它又是前提之一,在 dB 中可找到,故得证。现在 sg-stk 为空。调用 print,以逆序打印 bt-stk 中的证据如下:

(members′=members,8),(banned′=banned,7),(banned∪{ban?}=banned,17),(ban?∈banned,9)}.

因为前提被临时地连接在库的前部,所以证据中定律的下标是修改过的下标。

结束语 本文给出的 Z 规格说明中定理的证明方法比我们在[8]中给出的方法更具一般性,它为建立通用的形式规格说明推理的策略模式(tactics)提供了基础,一些实现细节还有待进一步研究。本课题的研究还得到了英国约克大学 John McDermid 教授和 Ian Tony 博士的帮助,在此表示感谢。

参考文献

[1] A. Diller, Z An Introduction To Formal Methods, John Wiley & Sons, 1990

[2] J. M. Spivey, The Z Notation, A Reference Manual, Second Edition, Prentice Hall, 1992

[3] D. Bjorner et al., (eds.), VDM'87: VDM: A Formal Method at Work, Lecture Notes in Computer Science, Vol. 252, London, Springer Verlag, 1987

[4] D. Neilson, D. Prasad, ZedB: A Proof tool for Z built on B, Proc. of the Sixth Annual Z User Meeting, 1992

[5] D. Jordan et al., CADiZ, Computer Aided Design in Z, Z User Workshop, Oxford, 1990

[6] S. Ower et al., PVS: A Prototype Verification System, Automated Deduction CADE-11, Springer Verlag, 1992

[7] R. D. Arthan, HOL Formalised: deductive system, DS/FMU/IED/SP003, London, Intl. Computer Ltd., 1992

[8] 缪淮扣等,Z 规格说明中初始状态存在性的证明,软件学报,6(12)1995

[9] 缪淮扣,Z 规格说明中的前置条件的简化,软件学报,8(9)1997