# Z规格说明中初始状态存在性的证明*

缪淮扣 John McDermid Ian Toyn

(上海大学计算机科学系,上海 201800) (约克大学计算机科学系,英国)

**摘要** Z规格说明中的初始化定理的证明是对基于状态的规格说明的一个标准的检查.本文给出了一个证明初始化定理的过程,该过程可自动地构成证明的证据.作为实例,我们用该过程证明了两个初始化定理.

**关键词** 初始化定理,证明,PEISZ,规格说明,证据.

产生一个对于形式规格说明来说可证明为正确的软件的思想是非常引人入胜的,特别是对于那些与安全性和可靠性有关的系统更具吸引力.使用象Z[1,2]这样一种形式规格说明语言的一个主要优点是能对用它所写的规格说明进行推理.最近我们已经看见了一些支持Z的工具.尽管如此,这都是在语法和类型检查方面所做的工作.它们本身都不能保证能查出规格说明中的所有错误.为了能查出更细微的错误,并对规格说明本身有更深刻的理解,我们就需要有形式的证明[3],并且它应该成为形式规格说明方法的一个组成部分.

用Z来写规格说明,我们应考虑这样的问题:这个规格说明确实具有我们所要求的性质吗?最后的程序和我们所写的规格说明是一致的吗?答案依赖于是否我们能找到一个关于这些问题的定理的形式证明.无论如何,构造一个形式证明也是非常繁琐的,自然也就容易出错.所以期望规格说明的书写者、设计者和实现者完全用手工去做这件事是不合理的.

对于每一个规格说明,我们必须要进行形式证明或以某种其他方式确信的事情是它的初始状态存在.这可表示为一个定理,称之为初始化定理.对于任何以(schema)来描述系统的状态和初始状态的规格说明,都可以写出其初始化定理.

初始化定理的证明是一个标准的检查,可以对任何基于状态的规格说明来执行,模式是用来描述系统状态和初始状态的.和Z规格说明的一般形式的定理比较,初始化定理的前提是空的,并且经常是这样的情形,初始状态模式给出了状态成份明确的值.

一个形式证明应能提供在证明过程中每一个证明步的证据.正确地记录所有的证明证据与证明本身一样困难.现有的关于Z规格说明的证明工具如HOL[4]和ZedB[5]等都是交互式的系统.在交互的方式下,证明是由用户引导的.但为了有效地证明一个定理,用户必须

* 本文1994-05-18收到,1994-11-30定稿

本课题受国家863高科技项目和国家自然科学基金的资助.作者**缪淮扣**,1953年生,副教授,主要研究领域为软件工程,形式方法,自动推理.**John McDermid**,1952年生,教授,主要研究领域为软件工程,安全临界系统.**Ian Toyn**,1961年生,博士,主要研究领域为软件工程,软件工具.

本文通讯联系人:缪淮扣,上海201800,上海大学计算机科学系

清楚地知道整个工作的过程,如证明的上下文,待证定理(目标)中哪些子目标已经得证,哪些子目标还未得到证明,以及哪些推理规则或证明策略可用.在交互的方式下,证明过程中的证据也难以被有效地组织成一个文档.

我们现在考虑的是建立一个自动的证明系统.用户只要输入一个关于Z的规格说明的假设(Conjecture),然后运行该系统,如果该假设是一个定理,即可得证.用户对整个证明过程的细节不必知道,证明过程中的证据可以自动地组织成一个文档.本文介绍的是我们设计的一个Z方法和用于产生并证明Z规格说明中初始化定理的过程PEISZ(PEISZ是由本文英文题目的单词的首字母组合而成,其发音与PIECES类似).它具有我们前面提到的自动证明过程的优点.该过程对用户输入的状态模式和初始状态模式自动产生一个初始化定理,然后证明之并给出其证明证据.我们证明了两个初始化定理.以此来说明PEISZ的执行.

## 1 PEISZ使用的数据结构

PEISZ是基于数学和逻辑定律,并以逆向推理方式进行工作的过程,因此它需要一些与之有关的数据结构.它们是:模式定义数据库;一个子目标栈sg－stk;一个定律库;一个回溯栈bt－stk;一个内定义函数库.

模式定义库存放着描述系统状态的定义,系统状态中各成份的定义以及初始状态的定义.它们均由用户输入,并假定已通过了语法和类型检查.这些定义在PEISZ的执行中用于展开一个初始化的定理.

我们需要一个缓冲区来存放初始化定理,这个定理先被看成是一个目标,随后在证明的过程中将产生子目标.子目标在这个缓冲区中的处理是"后进先出".我们称这个子目标栈为sg－stk.过程PEISZ从栈sg－stk中每次取一个子目标进行证明,直至整个sg－stk为空,这就意味着该初始化定理已被证明,过程PEISZ应该结束.

为了能证明Z规格说明的定理,除了基本的逻辑定律以外,我们还需要大量的数学定律.这些定律告诉我们如何处理在规格说明中所出现的各种结构,如数、集合、关系、函数以及序列等.我们应该有这样一个定律库,这些定律已被证明为正确的,每一个人都可放心地使用它们而不必去担心它们是否可靠.这样的一套关于Z的基本定律在文献[1]中已有描述.对任何一个要证明用Z写的定理的人来说,这都是一个最好的起点.当然为了推理的方便,我们可以增加一些在该书中未列出的确实是很有用的定律.为了提高查找的效率,库中的定律可以按照其结构或形式分成若干个子集.函数find可根据给定的子目标的结构查找可使用的定律,此外为了加快查找的速度,可将一些经常要用的定律如$\varphi \in S, \varphi \subseteq PS$等置于库的前面.

回溯是自动推理中最常用的策略,在PEISZ中,我们提供了一个用于回溯的栈bt－stk,该栈中的每一个元素或是一个偶对(sub－goal,index),它记录了一个子目标和所使用的定律在库中的下标;或是一个特殊的偶对(#,0).当一个子目标得到证明,我们就将(#,0)压入到栈bt－stk中,以表明这个子目标的证明已结束.在bt－stk中除了所有的元素(#,0)以外的信息构成了一个证据的集合.

经常有这样的情形,当处理一个子目标时,需要计算该子目标中的一些表达式,例如是否一个对象是一个数,或检查是否一个表达式满足某一个关系.PEISZ含有一个经常要使用

的内定义的函数的库.

## 2　一个 Z 方法和过程 PEISZ

设 S 是一个描述状态的模式的名,则系统的初始状态可用另一个模式 InitS 来描述. 初始化定理就具有这样的形式: ⊢∃ S′. InitS

在 PEISZ 中,有一个过程 gen,它将产生初始化的定理 G,它是按如下方法来识别初始状态的,初始状态名是在系统原状态的名前加上"Init"前缀,并且以修饰的(decorated)状态名作为其第一个说明(declaration). 一个 Z 方法建立了这样一个规定并且使象前置条件的计算这样的工具可使用,它同样可产生验证条件. 一个例子就是本文引出的推测,即状态中的变量的初值是存在的,这也就是我们所说的初始化定理. 它需要工具来对每一个这样的模式产生和证明初始化定理. PEISZ 的主要步骤是:产生初始化定理,展开它,消除"|",应用点规则(One－Point Rule),分离定理成子目标和使用定律来证明子目标.

PEISZ 含有如下的子过程和函数:

- gen(InitS,G),一个对于给定的初始状态 InitS 能产生初始化定理的过程.
- expand(G),一个根据定义库中的定义来展开目标 G 中的成份的过程.
- elim(G),一个能消除 G 中的"|"而作等价变换的过程.
- opr－app(G),一个对 G 应用点规则的过程.
- separate(G),一个将目标 G 分离成子目标 $g_1,\cdots,g_n$ 的过程. 这是不困难的,因为目标 G 本身具有合取范式的形式,即 $G=g_1\wedge\cdots\wedge g_n$.
- push(d,stk),一个将数据元素 d 压入栈 stk 的过程.
- position(L),一个能得到定律 L 在库中下标的函数.
- pop(stk),一个删除 stk 的顶元素并返回该元素为结果的函数.
- top(stk),一个返回 stk 的顶元素作为结果的函数.
- find(sg,pos,σ),一个查找可用用于子目标 sg 的定律的函数,查找是在定律库中查找,pos 是当前的定律在库中的下标,σ 是一个获得的 mgu(most general unifier),即最一般合一替换.
- app(L,σ,sg,tag),一个将定律 L 和 mgu σ 应用于子目标 sg 的过程,如果 sg 得证,则 tag 等于 0,如果 sg 被证明为是一个矛盾,则 tag 为 1,如果 sg 由重写定律重写,则 tag 为 2,且 sg 是重写后的子目标.
- print(str),一个打印串 str 和栈 bt－stk 中信息的过程.
- exception,一个处理发生了在库中没有可用的定律的情况的过程.

事实上,我们也可以将模式展开,消除"|"和使用点规则看成是证明器的额外的定律,点规则是一个经常要用的规则,它说的是,如果我们有一个存在性量词量化了的谓词,该谓词中的部分量词化的变量有明确的值,那么在该谓词中就可以用这些值来替换对应的量词变量的出现.

为了使用点规则,我们必须:(a)说明替换的项是正确的类型的成员,(b)在谓词中替换这些已知的项的出现.

点规则的使用似乎是比较复杂的,但这个规则是很有用的,并且是值得掌握的技术,某

些软件工具如我们的CADiZ[6]和ZedB[5]已经实现了这个规则.

find是过程PEISZ中最复杂的操作之一,它仔细地分析了子目标的结构和形式,在库中查找可用的定律,以使得子目标或子目标中的一个部分与定律能匹配.find中含有对子目标和定律进行合一的子过程Unification.合一是自动定理证明中典型的问题.在这里的上下文中,一阶逻辑中的合一概念需要扩充.变量和项可以是类型符、集合符、数、序列以及各种数学函数和关系等结构.函数find如果查找到了一条可用的定律,则返回该定律在库中的下标和一个mgu $\sigma$.否则返回数值0和一个替换出错符"$".

过程app如同find一样复杂.它使用了定律来重写子目标,如果子目标恰是定律的一个示例(Instance),则子目标被看成是true;如果子目标是定律的一个矛盾(Contridition),则子目标显然是假,整个目标就不是一个定理.此外,就是子目标被定律重写成一个新的子目标.

过程print先显示了串str,然后显示了bt—stk中的证据,为了能够以正确的次序显示bt—stk中的信息,print逆置bt—stk中的元素,并滤掉了所有的(#,0)偶对.

Procedure PEISZ

{ This is a procedure to prove the initialization theorem in Z specification. The proof justfication can be generated automatically by the procedure.

Input: A state scheme S with its component declarations, and the initialization schema, InitS. }

1. gen(InitS,G);
   {generate the initialization theorem G as a goal from InitS, i.e. G←$\exists$ S' • InitS;}
2. expand(G)
   {expand the schemas in G}
3. elim(G);
   {eliminate '|' if they occur in G by the laws $\exists J \mid P \bullet Q \Leftrightarrow \exists J \bullet P \wedge Q$ and $\forall J \mid p \bullet Q \Leftrightarrow \forall J \bullet P \Rightarrow Q$}
4. opr—apps(G);
   {apply One Point Rule to G as many times as possible};
5. sg—stk←nil;
   bt—stk←nil;
6. separate(G); {separate G into sub—goals $g_1,\ldots,g_n$}
   push($g_i$,sg—stk),i=n,n−1,...,1;
7. **while** sg—stk≠nil **do**
   ⌈
8. sg ←pop(sg—stk);
   pos ←0;{ pos is the current index of the laws in library}
9. L←find(sg,pos,$\sigma$) {find next applicable law in the library}
10. **if** position(L)≠0 **then**
    {L's index is calculated }

```
    ⌈
11.     push ((sg,position(L)),bt−stk);
        {keep(sg, position(L) as backtracking information or a justification }
12.     app(L,σ,sg,tag);
        {apply law L and σ to sg,tag is 0 if sg is proven, 1 if tag is false, 2 if sg is a
        new sub−goal by rewriting}
13.     case tag
        0: push((#,0),bt−stk);
        1: print('This is not a theorem');
           terminate;
        2: push(sg,sg−stk);
    ⌋
14.     else
         if top(bt−stk)=(#,0) then exeception { can't backtrack any further, and
         let user add some laws, or print exception message and end the procedure}
         else ⌈(sg,pos)← pop(bt−stk); goto 9⌋
  ⌋
15. print( 'The initialization theorem has been proven');
```

虽然库中含有相当多的定律,还可能会发生这样一种情形:我们需要的某些定律没有包含在库中.这是很自然的.因为一个库总是有限的,不可能包含所有的定律,对于一个子目标,如果库中没有一条可选择的定律,那么过程 exception 则显示问题来问用户是否他(她)想试图在库中增加一条新的定律.如果回答是肯定的,则用户向库中增加一条新定律,系统将子目标压入 sg−stk,并且令 while 循环继续;如果回答是否定的,则 exception 显示例外信息,过程 PEISZ 终止.现在的问题是,用户增加的定律是否是正确的.这个新增加的定律应该使用 Z 的定义和现存的定律来予以证明,在获得证明以前,它只能是一条引理.

## 3　例

在这一节中,我们考虑两个例,一个是 Storage Allocator(存储分配)模式规格说明[7],另一个是 Library DB(图书馆数据库)模式的规格说明[8].我们给出了它们的初始化定理,并用 PEISZ 来证明.我们不列出库中的几百条定律,为了演示我们的例,而仅建立一个含有一部分定律的很小的库.在实际应用中,一个真正的库应该组织的很合理,以使得查找有较高的效率.

假定这个小的定律库含有如下定律:

$\varnothing \subseteq \mathbf{PS}$　　$L_1$

$\varnothing \in \mathbf{FS}$　　$L_2$

$\varnothing \in S \nrightarrow T$　　$L_3$

$S=S$　　$L_4$

| | |
|---|---|
| $S \subseteq S$ | $L_5$ |
| $S \in \mathbf{P}T \Leftrightarrow S \subseteq T$ | $L_6$ |
| $S \cap \varnothing = \varnothing$ | $L_7$ |
| $S \setminus \varnothing = S$ | $L_8$ |
| $\mathrm{dom}\varnothing = \varnothing$ | $L_9$ |
| $\mathrm{ran}\varnothing = \varnothing$ | $L_{10}$ |
| $R(\!|S|\!) = \mathrm{ran}(S \lhd R)$ | $L_{11}$ |
| $\varnothing^{\sim} = \varnothing$ | $L_{12}$ |
| $\varnothing(\!|S|\!) = \varnothing$ | $L_{13}$ |
| $Q \Rightarrow \mathrm{true}$ | $L_{14}$ |
| $\forall x : T \cdot \mathrm{true}$ | $L_{15}$ |

例1:Storage Allocator

操作系统中有一个用户能访问的存储块的管理问题.设存储块的集合为B,用户的集合为U.设计算机系统中有n个连续的被编了号的块如图1.

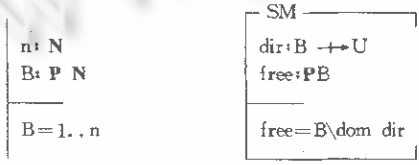


图1 图2 图3

存储管理程序有一个目录记录了哪些用户占用了哪些块,这个信息结构称之为dir,未被任何人占用的块被称为是free的.状态不变式和说明构成了一个模式SM,如图2.

这个系统的初始状态被定义为,没有一个块分配给了任何一个用户,所以每一个块是自由的,如图3.

我们现在说明PEISZ是如何工作的.首先gen产生初始化定理:$\vdash \exists SM' \cdot \mathrm{InitSM}$

调用expand展开初始化定理为:

$$\vdash \exists dir' : B \rightarrowtail U; free' : \mathbf{P}B \mid free' = B \setminus \mathrm{dom}\ dir' \cdot dir' = \varnothing \wedge free' = B$$

调用elim,消除"|",我们有:

$$\vdash \exists dir' : B \rightarrowtail U; free' : \mathbf{P}B \cdot free' = B \setminus \mathrm{dom}\ dir' \wedge dir' = \varnothing \wedge free' = B$$

调用opr—app,使用点规则,我们得到:

$$\vdash \varnothing \in B \rightarrowtail U \wedge B \in \mathbf{P}B \wedge B = B \setminus \mathrm{dom}\varnothing$$

对这个目标调用separate,产生3个子目标$g_1$、$g_2$和$g_3$,并以逆序压入sg—stk.

$g_1$ $\varnothing \in B \rightarrowtail U$

$g_2$ $B \in \mathbf{P}B$

$g_3$ $B = B \setminus (\mathrm{dom}\varnothing)$

第1个子目标$g_1$用定律$L_3$很容易就得到证明.$g_2$先由定律$L_6$重写为$B \subseteq B$,然后由定律$L_5$立即得到证明.现在bt—stk中含有:

$$((\#,0),(B \subseteq B,5),(B \in \mathbf{P}B,6),(\#,0),(\varnothing \in B \rightarrowtail U,3)).$$

$g_3$被弹出sg—stk待证,函数find试图以定律$L_8$与$g_3$匹配,但Unification在求匹配时

失败. 而定律 $L_9$ 可与 $g_3$ 中的"dom∅"匹配,调用 app,$g_3$ 被重写为 B=B\∅,再由定律 $L_8$ 和 $L_4$,该子目标得证. 此时在 bt—stk 顶上的4个元素是(#,0),(B=B,4),(B=B\∅,8),(B=B\(dom ∅),9). 栈 sg—stk 为空,因此 print 被调用,显示了如下信息:

The initialization theorem has been proven!

(∅∈B⇸U,3),(B∈PB,6),(B⊆B,5),(B=B\(dom∅),9),(B=B\∅,8),(B=B,4).

例2: Library DB

该例取自于 Wings Library Problem[2],整个规格说明使用了下面的给定类型(given type):[Book, Copy, Person, Author, Subject, Report]

我们这里仅需要 copy 和 person 两个 given 类型. 集合 Book 是可能的书的集合,Copy 是所有的书的可能的拷贝的集合,person 是所有可能的人的集合.

模式 Library DB 和 InitLibrary DB 可描述为:

$$
\begin{array}{|l}
\text{MaxcopiesAllowed}: \mathbb{N} \\
\hline
\text{LibraryDB} \\
\text{borrower, staff}: \mathbb{F}\ \text{Person} \\
\text{available, checkedout}: \mathbb{F}\ \text{Copy} \\
\text{previouslyborrowedby, borrowedby}: \text{Copy} \rightarrow\!\!\!\!\!\!\rightarrow \text{Person} \\
\hline
\text{borrower} \cap \text{staff} = \varnothing \\
\text{available} \cap \text{checkedout} = \varnothing \\
\text{dom borrowedby} = \text{checkedout} \\
\text{ran borrowedby} \subseteq \text{borrower} \\
\forall\ p: \text{Person} \mid p \in \text{borrower} \bullet \#(\text{borrowedby}^{\sim}(\!|\{p\}|\!)) \leqslant \text{MaxcopiesAllowed} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\text{InitLibraryDB} \\
\hline
\text{LibraryDB}' \\
\hline
\text{borrower}' = \varnothing \\
\text{staff}' = \varnothing \\
\text{available}' = \varnothing \\
\text{checkedout}' = \varnothing \\
\text{previouslyborrowedby}' = \varnothing \\
\text{borrowedby}' = \varnothing \\
\hline
\end{array}
$$

过程 PEISZ 中的 gen 先产生初始化定理　　$\vdash \exists\ \text{LibraryDB}'.\ \text{InitLibraryDB}$

expand 将其展开成:

$$
\begin{aligned}
\vdash \exists\ & \text{borrower}', \text{staff}': \mathbb{F}\ \text{Person}; \text{available}', \text{checkedout}': \mathbb{F}\ \text{Copy}; \\
& \text{previouslyborrowedby}', \text{borrowedby}': \text{Copy} \rightarrow\!\!\!\!\!\!\rightarrow \text{Person} \mid \\
& \text{borrower}' \cap \text{staff}' = \varnothing \wedge \text{available}' \cap \text{checkedout}' = \varnothing \wedge \\
& \text{dom borrowedby}' = \text{checkedout}' \wedge \text{ran borrowedby}' \subseteq \text{borrower}' \wedge \\
& (\forall\ p: \text{Person} \mid p \in \text{borrower}' \bullet \\
& \#(\text{borrowedby}'^{\sim}(\!|\{p\}|\!)) \leqslant \text{MaxcopiesAllowed}) \bullet \\
& \text{borrower}' = \varnothing \wedge \text{staff}' = \varnothing \wedge \text{available}' = \varnothing \wedge \\
& \text{checkedout}' = \varnothing \wedge \text{previouslyborrowedby}' = \varnothing \wedge \\
& \text{borrowedby}' = \varnothing
\end{aligned}
$$

调用 elim,消除"|",我们有:

$\vdash \exists$ borrower′,staff′:**F** Person;available′,checkedout′:**F** Copy;
previouslyborrowedby′,borrowedby′:Copy⇸Person·
borrower′ ∩ staff′ = ∅ ∧ available′ ∩ checkedout′ = ∅ ∧
dom borrowedby′ = checkedout′ ∧ ran borrowedby′ ⊆ borrower′ ∧
(∀ p:Person·
p ∈ borrower′·
# (borrowedby′ ~ (| {p} |) ≤ MaxcopiesAllowed) ∧
borrower′ = ∅ ∧ staff′ = ∅ ∧ available′ = ∅ ∧
checkedout′ = ∅ ∧ previouslyborrowedby′ = ∅ ∧
borrowedby′ = ∅

应用点规则,即调用 opr—app,初始化定理为:

$\vdash \varnothing \in \mathbf{F}$Person ∧ ∅ ∈ **F**Person ∧ ∅ ∈ **F**Copy ∧ ∅ ∈ **F**Copy ∧
∅ ∈ Copy⇸Person ∧ ∅ ∈ Copy⇸Person ∧ ∅ ∩ ∅ = ∅ ∧
∅ ∩ ∅ = ∅ ∧ dom∅ = ∅ ∧ ran∅ ⊆ ∅ ∧
(∀ p:Person·p ∈ ∅ ⇒ # (∅ ~ (| {p} |) ≤ MaxcopiesAllowed)

由 Separate,该目标被分离成如下的子目标,并被压入 sg—stk 中.

$g_1$ ∅ ∈ **F**Person
$g_2$ ∅ ∈ **F**Person
$g_3$ ∅ ∈ **F**Copy
$g_4$ ∅ ∈ **F**Copy
$g_5$ ∅ ∈ Copy⇸Person
$g_6$ ∅ ∈ Copy⇸Person
$g_7$ ∅ ∩ ∅ = ∅
$g_8$ ∅ ∩ ∅ = ∅
$g_9$ dom∅ = ∅
$g_{10}$ ran∅ ⊆ ∅
$g_{11}$ ∀ p:Person·p ∈ ∅ ⇒ # (∅ ~ (| {p} |) ≤ MaxcopiesAllowed

PEISZ 使用定律 $L_2$很容易就证明了 $g_1$、$g_2$、$g_3$和 $g_4$,同样 $g_5$和 $g_6$可由定律 $L_3$得证,$g_7$和 $g_8$可由定律 $L_7$得证,$g_9$由定律 $L_9$得证,$g_{10}$由定律 $L_{10}$和 $L_5$得证,所有的证据均被压入栈 bt—stk. 子目标 $g_{11}$的证明就比较复杂了. 表1解释了 $g_{11}$的部分证明过程.

| sg | L=find | bt—stk |
|---|---|---|
| | | … |
| ∀ p:Person·p ∈ ∅ ⇒ # (∅ ~ (\| {p} \|) ≤ MaxcopiesAllowed | 12 | (∀ p:Person·p ∈ ∅ ⇒ # (∅ ~ (\| {p} \|) ≤ MaxcopiesAllowed, 12) |
| ∀ p:Person·p ∈ ∅ ⇒ # (∅ (\| {p} \|) ≤ MaxcopiesAllowed | 11 | (∀ p:Person·p ∈ ∅ ⇒ # (∅ (\| {p} \|) ≤ MaxcopiesAllowed, 11) |
| ∀ p:Person·p ∈ ∅ ⇒ # (ran(p ◁ ∅) ≤ MaxcopiesAllowed | | |

在现在的状态下,在我们假设的库中没有合适的定律可用,因此 PEISZ 回溯,pop bt—stk,sg 为∀ p:Person·p∈∅⇒#(∅⦇{p}⦈)≤MaxcopiesAllowed,pos 为11. 我们以表2显示了子目标 $g_{11}$ 的证明过程,其中 f—LENG 是一个内定义的函数,它计算了一个集合中元素的个数. f—LEQ 也是一个内定义函数,判定一个表达式是否满足小于等于关系.

| sg | L=find | bt—stk |
|---|---|---|
| | | … |
| ∀ p:Person·p∈∅⇒<br>#(∅~⦇{p}⦈)≤MaxcopiesAllowed | 12 | (∀ p:Person·p∈∅⇒<br>#(∅~⦇{p}⦈)≤MaxcopiesAllowed, 12) |
| ∀ p:Person·p∈∅⇒<br>#(∅⦇{p}⦈)≤MaxcopiesAllowed | 13 | (∀ p:Person·p∈∅⇒<br>#(∅⦇{p}⦈)≤MaxcopiesAllowed, 13) |
| ∀ p:Person·p∈∅⇒<br>#(∅)≤MaxcopiesAllowed | f—LENG | (∀ p:Person·p∈∅⇒<br>#(∅)≤MaxcopiesAllowed, f—LENG) |
| ∀ p:Person·p∈∅⇒<br>0≤MaxcopiesAllowed | f—LEQ | (∀ p:Person·p∈∅⇒<br>(0≤MaxcopiesAllowed, f—LEQ) |
| ∀ p:Person·p∈∅⇒true | 14 | (∀ p:Person·p∈∅⇒true, 14) |
| ∀ p:Person·true | 15 | (∀ p:Person·ture, 15) |
| true | | (#, 0) |

最后 print 显示:The initialization theorem has been proven!

(∅∈FPerson,2)　　(∅∈FPerson,2)

(∅∈FCopy,2)　　(∅∈FCopy,2)

(∅∈Copy⟷Person,3)　(∅∈Copy⟷Person,3)

(∅∩∅=∅,7)　　(∅∩∅=∅,7)

(dom∅=∅,9)　　(ran∅⊆∅,10)

(∅⊆∅,5)

(∀ p:Person·p∈∅⇒#(∅~⦇{p}⦈)≤MaxcopiesAllowed, 12)

(∀ p:Person·p∈∅⇒#(∅⦇{p}⦈)≤MaxcopiesAllowed, 13)

(∀ p:Person·p∈∅⇒#(∅)≤MaxcopiesAllowed, f—LENG)

(∀ p:Person·p∈∅⇒0≤MaxcopiesAllowed, f—LEQ)

(∀ p:Person·p∈∅⇒true, 14)

(∀ p:Person·true, 15)

## 4　结　论

我们给出了一个证明初始化定理的过程 PEISZ,对于一个给定的状态和初始状态,PEISZ 产生初始化定理,然后证明它,如果证明成功,则给出证明过程中的证据.

PEISZ 可以作为证明一般 Z 规格说明的定理的基础而加以扩充. gen、expend、elim、opr—app、separate 和 subsTitution 等子过程和函数已在我们的 CADiZ 系统中得到实现,其余的正在实现过程中. 目前我们正在研究 Z 规格说明中前置条件的简化和一般定理的证明的策略(tactics).

## 参考文献


1 Spivery J M. The Z notation: a reference manual. 2nd ed., London: Prentice Hall, 1992.

2 Diller A. Z an introduction to formal methods. London: John Wiley and Son, 1990.

3 Jordan D, McDermid J A, Toyn I. CADiZ—computer aided design in Z. In: Nicholls J E ed. Workshops in Computing: Z User Workshop, Oxford, 1990, London: Spinger— Verlag, 1991. 93—104.

4 Potter B, Sinclair J, Till D. An introduction to formal specification and Z. London: Prentice Hall, 1991.

5 Arthan R D. HOL formalised: deductive system. DS/FMU/IED/SP003, London: International Computer Ltd., 1992.

6 Neilson D, Prasad D. zedB: a proof tool for Z bulit on B. In: Nicholls J E ed., Workshops in Computing: Z User Workshop, York, U.K. 1991, London: Springer—Verlag, 1992. 243—258.

7 Toyn I. CADiZ quick reference guide. university of York, U.K.: York Software Engineering Limited, 1990.

8 Woodcock J, Loomes M. Software engineering mathematics. London: Pitman, 1988.


# PROVING THE EXISTENCE OF INITIAL STATE IN Z SPECIFICATIONS


Miao Huaikou

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

John McDermid Ian Toyn

*(Department of Computer Science, University of York, U.K.)*



**Abstract** Proof of the initialization theorem is a standard check that may be carried out for any state—based specification. A procedure for proving initialization theorems is presented. The proof justifications can be generated automatically by this procedure. By way of example, two initialization theorems are proved using this procedure.



**Key words** Initialization theorems, proof, PEISZ, specifications, justification.