计算机科学 1997 Vol. 24 №. 1

Z规格说明求精的正确性判定*)

李 刚 朱关铭 童 频

(上海大学计算机科学系 上海 201800)

摘 要 From a view of Abstract Data Type, this paper analyzes the structure of the Z specification and discusses the refinement of the Z specification, then gives a criterion for judging the correctness of the Z specification refinement. This criterion can be used in every step of the Z specification refinement, it also provides a base for the automatic proving of the consistency between Z specifications.

关键词 Software refinement, Z specification, Schema, State space.

软件求精是自动推理和形式化开发方法相结合而形成的一门新技术，它研究从抽象的形式规格说明推演出具体的面向计算机的程序代码的全过程。其基本思想是用一个抽象程度低、过程性强的程序去代替一个抽象程度高、过程性弱的程序，并保持它们之间功能的一致[1]。根据软件求精的观点，程序开发过程可以看成是从软件的形式规格说明开始，通过一系列的求精步骤，每一步都降低一些抽象程度或增强一些过程性，形成如下的序列：

形式规格说明⇒混合体$_0$⇒混合体$_1$⇒…⇒混合体$_n$⇒程序代码

最终得到能够指导计算机明确执行的程序代码，在文[1]中，提供了许多经过证明的软件求精规则，软件求精有可能通过人工智能的方法自动实现。但是，在具体实现中，软件求精规则的使用要求编程人员具有很强的数学基础，而且对于一些较复杂的软件规格说明来讲，从规格说明到代码要经过许多步求精才能实现。这使得软件求精规则的使用受到了极大的限制。所以，在实际应用中，大多数采用形式化方法进行编程的程序员仍是依靠手工进行求精，以求避开那些对数学知识要求过高的软件求精规则，并希望能够减少软件求精的中间步骤，达到跨越式求精甚至直接求精的效果。但是，由于缺少软件求精规则支持，这种手工方法往往不能完全保证软件求精的正确性，在软件求精过程中，有可能会逐步偏离原始的形式规格说明。因此，在这种情况下如何判定软件求精结果是否正确，成为保证最终代码和原始的形式规格说明一致的关键问题。我们以形式规格说明语言Z为例，来探讨用手工进行软件求精时的正确性判定。

1 Z规格说明

形式规格说明是用户与软件开发人员之间对目标软件系统的一种规约。牛津大学PRG小组设计的Z语言是目前流行的一种形式规格说明语言，它以一阶谓词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之中，表达能力强，可产生简洁、无歧义且易于自动证明的形式规格说明，特别适合于编写"安全-紧急"系统的规格说明[2]。

1.1 Z规格说明结构

在利用Z语言来书写软件的规格说明时，首先是辅助部分，它包括：给定类型声明、常量声明、缩写声明以及公理化描述等。其中，给定类型声明给出了规格说明中使用的数据类型；常量声明部分给出了规格说明中使用的常数及数学符号；缩写声明给出了规格说明中使用的缩写；公理化描述则给出了规格说明中的一些常数之间的约束[3]。

除了辅助部分之外，Z规格说明的其余部分都由模式(Schema)组成，根据各自的用途，我们把这些模式分为两种：状态模式和操作模式。它们是Z规格说明中最核心的部分。

1.1.1 **状态模式** 是对系统的状态空间及其约束特性(系统状态不变式)的描述。系统的状态存在于状态空间之中。图1表示了状态模式的概念。

*)本文受国家863高技术计划资助。李刚 硕士生，主要研究领域为面向对象的软件工程，人工智能。朱关铭 教授，主要研究领域为软件工程、逻辑程序设计。童频，教授，博士生导师，主要研究领域为知识工程。

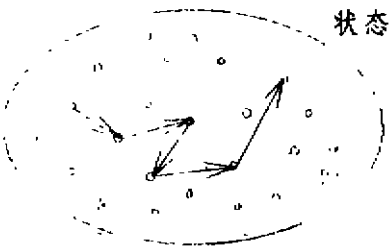

图 1　状态模式的概念

在图中，系统的状态随着操作的进行而在状态空间内改变。系统在任何时刻的状态都满足于状态模式的描述。

在 Z 规格说明中还有一种特殊的模式：初始状态模式。它描述了系统的初始状态。我们把它也看成是一种状态模式。

1.1.2　操作模式　定义了系统在状态空间上的操作特性。图 2 表示了操作模式的概念。它描述操作前的系统状态（前状态）和操作后的系统状态（后状态）之间的关系。其中，前后状态都处于状态空间之中，也都满足于系统状态模式的描述。前状态由操作模式的前置条件来描述。在 Z 规格说明中，若有一操作模式名为 Op，则它的前置条件为 Pre Op。在本文中，我们说定义在状态模式 State 描述的系统状态空间上的操作模式 Op 终止，指的是：系统的前状态 State 满足于操作模式 Op 的前置条件（Pre Op），并且在 Op 作用后，系统可以到达某状态，该状态可以用 State′来表示，且满足于操作模式 Op 中的描述。

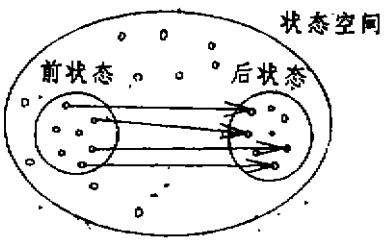

图 2　操作模式的概念

1.2　Z 规格说明与抽象数据类型

七十年代逐步形成了抽象数据类型的概念。抽象数据类型是指一个数学模型以及定义在该模型上的一组操作，在抽象数据类型中，“抽象”的意义在于数据类型的数学抽象特性，用户不涉及一个类型的具体表示，对于类型的访问必须通过其定义的操作来进行。抽象数据类型的定义仅取决于它的一组逻辑特性，而与其在计算机内部如何实现无关[4]。

Z 规格说明也对目标系统建立了一个数学模型，从整体的观点来看，构成 Z 规格说明的状态模式和操作模式可以看成一个抽象数据类型[5]，如下图：

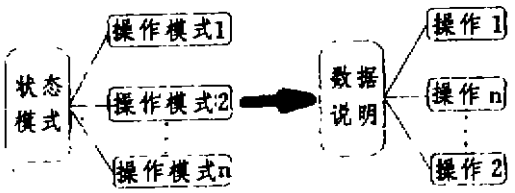

图 3　Z 规格说明与抽象数据类型

图中状态模式的描述看成是该抽象数据类型的数据说明，操作模式的描述看成是定义在该抽象数据类型上的操作。这样，我们就在 Z 规格说明和抽象数据类型之间建立了一种对应关系，在本文的第二部分，我们将从抽象数据类型的角度对 Z 规格说明的求精进行讨论。

1.3　一个 Z 规格说明实例

我们用 Z 语言来书写一个用来记录人名及其生日的统计系统的形式规格说明。它由一个记录数据项的数据库及一组对该数据库进行的操作（添加、删除、查询等等）组成。为简单计，我们只描述添加数据项这一种操作。该系统中，需要用到的数据类型有两种：人名和日期。根据 Z 语言的特点，可以先不必考虑这两种数据类型的具体结构。因此，在辅助部分我们引入如下的给定类型声明：

[NAME，DATE]　其中，NAME 是人名类型，DATE 是日期类型。

我们再考虑状态模式，该系统的状态空间由状态模式 BirthdayBook 描述：

BirthdayBook

known：P NAME

birthday：NAME ⇸ DATE

known = dom birthday

其中，known 是人名的集合，birthday 是一个从人名到日期的部分函数，用以记录人名及其生日，我们可以把它看成是一个数据库。该模式的谓词部分给出了系统状态空间的约束条件，集合 known 等于函数 birthday 的定义域。

系统的初始状态由初始状态模式 InitBirthdayBook 来描述：

InitBirthdayBook

BirthdayBook

known = ∅

该模式表明，在系统初始时，集合 known 为空

集。由此条件及 BirthdayBook 的谓词部分可以得出：在系统初始时函数 birthday 为空函数。

最后我们考虑操作模式。操作模式通过描述系统的前状态和后状态之间的关系来定义系统状态的改变方式。添加数据项这种操作要求提供人名和生日作为输入，然后将接收到的人名、生日信息加入到数据库中，它用操作模式 AddBirthday 描述：

AddBirthday

ΔBirthdayBook
name?：NAME
date?：DATE
name? ∉ known
birthday′ = birthday ∪ {name? ↦ date?}

该模式要求 name?、date?作为输入，当 name?不存在于当前 known 集合中时，将序偶 name?↦date? 加入到 birthday 之中。•

这里，整个系统可以看成是一个抽象数据类型，其数据说明由状态模式 BirthdayBook 描述，其操作说明由 AddBirthday 等操作模式来描述。

可以看出，Z 规格说明对目标系统建立了一个数学模型，必须用较具体的数据结构去实现状态模式中使用的抽象程度高的数据结构，即对状态模式进行精化；并用过程性强的算法去实现操作模式中的操作描述，即对操作模式进行精化。我们以"Z 规格说明求精"这个概念来统一看待这两种精化。

2 Z 规格说明求精的正确性判定准则

从上面的分析可以看出，Z 规格说明求精包括两个方面：

• 用较具体的数据结构去实现状态模式中使用的抽象程度高的数据结构，我们称之为数据求精；

• 用过程性强的算法去实现操作模式中的操作描述，我们称之为过程求精。根据这两个方面，Z 规格说明求精又可以分为两种方式：

• 不进行数据求精，单独进行过程求精。

• 数据求精和过程求精同时进行。

2.1 单独进行过程求精

先看单独进行过程求精时的情况。此时，状态模式没有受到精化，而操作模式受到了精化，精化前后的操作模式作用的状态空间是一致的，它们都由同一个状态模式来描述。如下图：

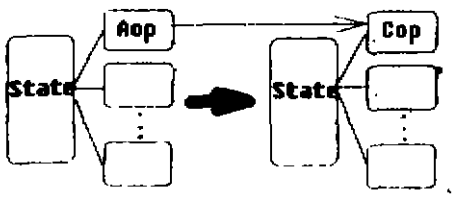

图 4 单独进行过程求精

图中，操作模式 Aop 作用在状态模式 State 描述的状态空间上，进行求精之后，Aop 对应于较具体的操作模式 Cop，Cop 也作用在状态模式 State 描述的状态空间上。

设 x?：X 是 Aop 和 Cop 的输入变量声明，y!：Y 是 Aop 和 Cop 的输出变量声明，Cop 是对 Aop 的求精，等价于如下两个条件：

1)对于给定的系统状态 State，如果 Aop 能够终止，那么 Cop 也能够终止。

2)对于给定的系统状态 State，如果 Aop 和 Cop 都终止，那么 Cop 作用后的系统状态 State′也满足 Aop 中关于后状态的描述。

要满足第一个条件，要求 Cop 能够处理的状态要比 Aop 能够处理的状态多，也即是要求 Cop 的前置条件应该比 Aop 的前置条件弱，因此我们得到如下谓词：

$$\forall State; x? : X \bullet Pre\ Aop \Rightarrow Pre\ Cop$$

要满足第二个条件，要求状态 State 满足 Aop 的前置条件时，Cop 能够终止且 Cop 终止后的系统状态 State′也满足 Aop 中关于后状态的描述。所以我们得到如下谓词：

$$\forall State; State'; x? : X; y! : Y \bullet Pre\ Aop \wedge Cop \Rightarrow Aop$$

当这两个条件满足时，我们可以说操作模式 Cop 适用于操作模式 Aop 适用的所有状态。即 Cop 是对 Aop 的精化。

2.2 数据求精和过程求精同时进行

上面的求精是 Z 规格说明求精时的一种简单情况。然而在大多数 Z 规格说明中，状态模式所使用的数据结构都比较抽象，都要求进行数据求精，当进行了数据求精后，操作模式作用的状态空间也随之改变，所以此时对每一个操作模式都必须进行过程求精。这种 Z 规格说明求精如下图：

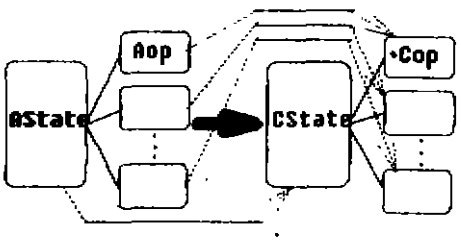

图 5 数据求精和过程求精同时进行

图中，使用较抽象数据结构的状态模式为 AState，使用较具体数据结构的状态模式为 CState，

Aop 为定义在 AState 所描述的状态空间上的一个操作模式，Cop 为定义在 CState 所描述的状态空间上的操作模式，它对应于定义在 AState 上的操作模式 Aop。

设 x? : X 为 Aop 和 Cop 的输入变量声明，y! : Y 为 Aop 和 Cop 的输出变量声明。

在 Z 规格说明求精中，当数据求精和过程求精都进行时，定义在 CState 上的操作模式 Cop 是定义在 AState 上的操作模式 Aop 的精化，它也等价于下面两个条件：

1）AState 描述的某一状态上，如果操作模式 Aop 能够终止，那么该状态在 CState 描述的状态空间上的对应状态，在操作模式 Cop 作用后也能够终止。

2）AState 描述的某一状态，该状态在 CState 描述的状态空间上的对应状态如果被 Cop 作用后能够终止，且到达一后状态，那么该后状态可以映射到 Aop 作用于 AState 后的某后状态上。

为了描述上面两个条件，我们引入一个模式 RelAStateCState 来描述 AState 和 CState 之间的关系，它的声明部分与模式"AState ∧ CState"的声明部分相同，它的谓词部分则描述了状态模式 CState 与状态模式 AState 之间的对应关系（包括系统不变式间的关系）。由第一个条件，可以得到：

判定条件 1 $\forall$ AState; CState; x? : X • Pre Aop ∧ RelAStateCState ⇒ Pre Cop

它表示：如果 AState 描述的当前系统状态满足于操作模式 Aop 的前置条件，那么对应的 CState 描述的系统状态也满足于操作模式 Cop 的前置条件。

由第二个条件，可以得到：

判定条件 2 $\forall$ AState; CState; CState'; x? : X; y! : Y • Pre Aop ∧ RelAStateCState ∧ Cop ⇒ ($\exists$ Astate' • RelAStateCState' ∧ Aop)

它表示：AState 描述的某一状态，该状态在 CState 描述的状态空间上的对应状态上，如果操作模式 Cop 可以终止且终止状态满足 CState' 的描述，那么 CState' 也必对应一个 AState'，AState' 为 Aop 作用于 AState 后的终止状态。

在 Z 规格说明中，由于存在着多个操作模式，所以只有当每一个操作模式的求精都满足这两个条件时，我们才可以说数据求精和过程求精是正确的。同时，考虑到许多 Z 规格说明中，还有初始状态空间的描述，设 Ainit 和 Cinit 分别描述了求精前后系统的初始状态，我们引入如下条件：

判定条件 3 $\forall$ Cstate • Cinit ⇒ ($\exists$ Astate • RelAStateCState ∧ Ainit)

它表示：求精后的初始状态一定对应着一个求精前的初始状态。

综合以上三个条件，我们给出如下 Z 规格说明求精的正确性判定准则：

在 Z 规格说明的某一步求精过程中，如果对于每一个过程求精都能够证明判定条件 1 和判定条件 2 成立，而且对于系统的初始状态模式能够证明判定条件 3 成立，那么这一步求精是正确的求精。

3 举例

下面是对本文第一部分中的 Z 规格说明进行求精过程的一个步骤。首先，引入两个数组类型 names 和 dates，用 Z 语言可以表示为：

$names : N_1 \rightarrow NAME$

$dates : N_1 \rightarrow DATE$

其中，N_1 为非零的自然数集合。此外，还引入了变量 hwm，用来记录当前数组中被使用元素的个数。状态模式 BirthdayBook 求精后，新的状态模式为 BirthdayBook1：

BirthdayBook1

$names : N_1 \rightarrow NAME$

$dates : N_1 \rightarrow DATE$

$hwm : N$

$\forall i, j : 1..hwm \bullet i \neq j \Rightarrow names(i) \neq names(j)$

初始状态模式 InitBirthdayBook 求精后，新的初始状态模式为 InitBirthdayBook1：

InitBirthdayBook1

BirthdayBook1

$hwm = 0$

操作模式 AddBirthday 求精后，新的操作模式为 AddBirthday1，它作用在 BirthdayBook1 所描述的系统状态空间上：

AddBirthday1

ΔBirthdayBook1

$name? : NAME$

$date? : DATE$

$\forall i : 1..hwm \bullet name? \neq names(i)$

$hwm' = hwm + 1$

$names' = names \oplus \{hwm' \mapsto name?\}$

$dates' = dates \oplus \{hwm' \mapsto date?\}$

为了证明这一步求精是正确的，我们引入如下的模式 RelAStateCState：

RelAStateCState

BirthdayBook

BirthdayBook1

known = {i : 1..hwm • names(i)}

∀ i : 1..hwm • birthday(names(i)) = dates(i)

(1)对于操作模式 AddBirthday 的求精，先考虑判定条件 1：

∀ AState; CState; x? : X • Pre Aop ∧ RelAStateCState ⇒ Pre Cop

在这里，AState 为 BirthdayBook，CState 为 BirthdayBook1，输入变量声明为 name? : NAME; date? : DATE，Aop 为 AddBirthday，Cop 为 AddBirthday1，因此，即要证明：

∀ BirthdayBook; BirthdayBook1; name? : NAME; date? : DATE • Pre AddBirthday ∧ RelAStateCState ⇒ Pre AddBirthday1

根据各个模式中的前置条件，得：

Pre AddBirthday 为 name? ∉ known

Pre AddBirthday1 为 ∀ i : 1..hwm • names? ≠ names(i)

由 RelAStateCState 可得 known = {i : 1..hwm • names(i)}

容易得出：

∀ BirthdayBook; BirthdayBook1; name? : NAME; date? : DATE • name? ∉ known ∧ known = {i : 1..hwm • names(i)} ⇒ ∀ i : 1..hwm • names? ≠ names(i)

即判定条件 1 满足。

(2)再考虑判定条件 2：

∀ AState; CState; CState'; x? : X; y! : Y • Pre Aop ∧ RelAStateCState ∧ Cop ⇒ (∃ Astate' • RelAStateCState' ∧ Aop)

即要证明：

∀ BirthdayBook; BirthdayBook1; BirthdayBook1'; name? : NAME; date? : DATE • Pre AddBirthday ∧ RelAStateCState ∧ AddBirthday1 ⇒ (∃ BirthdayBook' • RelAStateCState' ∧ AddBirthday)

其中，(∃ BirthdayBook' • RelAStateCState' ∧ AddBirthday)是形为(∃ x : X • x = E ∧ P(x))的谓词，逻辑上等价于删除存在量词和等式 x = E，而用 E 来代替余下部分中出现的 x。我们把这种方法应用到 BirthdayBook' 中定义的变量 known' 和 birthday' 上去，由于 Addbirthday 中有等式 birthday' = birthday ∪ {name? ↦ date?}，在 BirthdayBook' 中又有等式 known' = dom birthday'，所以(∃ BirthdayBook' • RelAStateCState' ∧ AddBirthday)可变为：

dom(birthday ∪ {name? ↦ date?}) = {i : 1..hwm' • names'(i)} ∧ (∀ i : 1..hwm' • birthday'(names'(i)) = date'(i)) ∧ name? ∉ known

由 Pre AddBirthday，RelAStateCState 和 AddBirthday1 可以容易地得出上面合取式。所以也可以得出判定条件 2 满足。

(3)最后考虑判定条件 3：

∀ Cstate • CInit ⇒ (∃ Astate • RelAStateCState ∧ AInit)代入各模式名得：

∀ BirthdayBook1 • InitBirthdayBook1 ⇒ (∃ BirthdayBook' • RelAStateCState ∧ InitBirthdayBook)

为了证明该式，我们将(∃ BirthdayBook' • RelAStateCState ∧ InitBirthdayBook)变化为：

known = ∅ ∧ birthday = ∅ ∧ known = {i : 1..hwm • names(i)}，

由 InitBirthdayBook1 中 hwm = 0，可以得出 known = ∅，继而可以推出 birthday = ∅，从 RelAStateCState 中可以直接得到 known = {i : 1..hwm • names(i)}，所以可以证得判定条件 3 满足。

当我们对于所有的操作模式都证明这几个条件成立时，可以认为这一步求精是正确的。

后记 本文结合抽象数据类型的概念来看待 Z 规格说明的结构和其求精，并提出了 Z 规格说明求精的正确性判定准则，通过该准则可以判定 Z 规格说明求精时的正确性，并且可以被用来判定两个 Z 规格说明之间功能的一致性。在缺乏强有力的软件求精辅助工具时，如何利用该判定准则构造 Z 规格说明求精正确性判定软件，用以保证 Z 规格说明求精结果与原始的形式规格说明之间的一致，也是我们今后研究的一个方面。

参考文献

[1] Carroll Morgan, Programming from Specifications, Prentice Hall, 1994

[2] Antoni Diller, Z : An Introduction to formal methods, Chichester, UK : John Wiley&Sons, 1990

[3] J. M. Spivey, The Z Notation : A Reference Mannual, 2nd Edition. Prentice Hall, 1992

[4] 朱海滨，面向对象技术—原理和设计，国防科技大学出版社，1992 年

[5] Kevin Lano, Howard Haughton. Object-Oriented Specification Case Studies, Prentice Hall, 1994