计算机科学1998Vol. 25№. 6

关于Z规格说明语言中模式的剖析*)

Dissection of Schema in Z Specification Language

高晓雷　缪淮扣

（上海大学计算机科学系　上海嘉定201800）

摘　要　This paper analys the state schema, operation schema, initial state schema and state invariant in Z language, instantiates their function, and shows the relation among them by way of an example. Finally it points out that the automatic extracting of state invariant is impossible.

关键词　Formal methods, Specification, Schema, Invariant.

1. 引言

形式规格说明语言一般是提供一套称为语法域的记号系统和一个称为语义域的对象集合，以及一组精确地定义哪些对象满足哪个规格说明的规则。规格说明是语法域中的句子。它用数学表示法精确地描述了软件系统必须具备的性质。

Z是目前比较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为基础，通过系统状态及对状态的操作来刻画客观世界。其主要表达形式是模式(状态模式和操作模式)。该语言提供了一种能独立于实现的、可推理的系统数学模型，具有简洁、无歧义性、可视性好的优点，并有利于进行程序正确性的证明。该语言非常适合于航空航天、军事、银行等无法重复测试的关键系统(Critical System)领域。

Z语言运用模式和模式演算来描述目标软件系统，Z规格说明的主体由一系列模式组成，这些模式对目标软件系统各部分的结构和行为特征进行抽象描述，根据模式的不同功能，模式可以分为两类：

1. 状态模式：主要用来描述系统的静态特性(结构特性)，对目标软件系统某部分的结构特征进行抽象，集中描述了该部分的状态变量，并定义了这些量之间的约束关系。

2. 操作模式：主要用来描述系统的动态特性(行为特性)，对目标软件系统某部分的行为特征进行抽象，通过描述操作前后状态值之间的关系，定义了与若干个状态模式相关的该部分的行为。

Z语言中，不论是状态模式还是操作模式，一律用如下所示的E型框架的形式来表示。

SchemaName
Declare-Part
Predicate-Part

其中Declare_Part是模式的声明部分，Predicate-Part是谓词部分，该部分在状态模式中称为不变式。

Z语言与传统编程语言如C、PASCAL和BASIC等是不同的，传统编程语言描述"怎么做"，而Z语言描述"做什么"，忽略实现的细节。通过下面的对比例子可以清楚看出它们的不同，例子是典型的求平方根，上面是用Z书写的规格说明，下面是用PASCAL语言书写的算法。

SquareRootSpec
radicand?, squareroot! : R
radicand? ≥ 0 squareroot!² = radicand? squareroot! ≥ 0

```
function SquareRoot(radicand:real):real;
const epsilon=0.0001;
var guess:real;
begin
    guess:=10;
while abs(radicand-(guess*guess))>epsilon do
    guess:=(guess+(radicand/guess))/2
SquareRoot:=guess;
end;
```

Z模式表示没有给出平方根的具体实现方法，只是说明平方根必须满足的条件，即做什么；至于怎

*)本文受国家自然科学基金(编号：69773038)和上海市教委科技发展基金(编号：97A42)资助。高晓雷　硕士研究生，主要研究方向为软件方法。缪淮扣　教授，主要研究方向为软件方法、自动推理等。

样做，可以有不同的实现方法，该例中 PASCAL 的实现只是众多实现的一种。

2. Z 的各种机制及其作用

Z 是面向模型的语言，模型的抽象就是表示抽象，即模型如何表示、结构如何表示；操作抽象描述模型如何运作；表示抽象是关于系统状态的抽象描述；操作抽象是围绕表示抽象建立的若干操作的规范描述，操作的行为与系统状态有关。Z 语言描述系统可以用一个三元组表示：$M=(S,S_0,\Omega)$，这里 M 表示整个系统状态空间，S 是 M 的所有可能状态的集合，对应 Z 的状态模式；S_0是系统的初始状态集合且 $S_0\subseteq S$。初始状态一般只有一个，对应 Z 的初始状态模式；Ω 是若干操作(运算)的集合，对应 Z 的一组操作模式；Ω 中的每个操作可抽象地表示成一个二元组：$op=(S_{pre},R)$，其中 $S_{pre}\subseteq S$，S_{pre}规定了该操作终止的起始(前)状态集合，$R\subseteq S\times S$ 是状态间的关系，$domR=\{t\in S\mid \exists\, t'\in S\cdot(t,t')\in R\}$，$S_{pre}\subseteq$ **dom**R，R 是同该操作相联系的状态转换关系，为了下面讨论方便，可达性定义为可操作性、可终止性，即操作能在有限步内完成，且对应的初态和终态满足关系 R。即$\forall\, t_0\in S_{pre}\cdot op(t_0)\Rightarrow op$ 终止于 $t\wedge(t_0,t)\in R$[4]。

·状态模式：状态模式描述了系统的全部可能状态。由于 Z 语言是基于一阶构造逻辑(直觉逻辑)的，所以状态模式只是提供了一个框架，在初始时状态空间不包含任何状态，系统的各个可能状态是通过操作模式构造出来的并由系统状态不变式所决定。状态模式的作用是描述系统的本质特征和结构，所谓系统的全部可能状态是指所描述的状态既不多也不少。例如描述开关状态用 Switch 表示，则集合{on，off}就是 Switch 的全部可能状态，用如下模式表示为：

Switch
switch：{on，off}

数字表 Timer 可用三个整型变量 hours，minutes，seconds 表示，其全部可能状态由系统不变式来约束，可表示为如下模式：

Timer
hours：N
minutes，seconds：N

0≤hours≤23
0≤minutes≤59
0≤seconds≤59

·操作模式：操作模式描述系统的动态特性，描述操作前后状态值之间的关系，其作用是根据初始化模式和系统状态不变式构造出系统全部可能状态，通俗讲就是求值。我们可以通过计数器 Counter 例清晰地看出状态空间是由操作动态构造出来的。

Counter
counter，N

InitCounter
Counter
---
counter=0

Add
ΔCounter
---
counter'=counter+1

很显然 Counter 的各个状态值是由 Add 通过 InitCounter 不断迭加而构造出来的。

·初始化模式：初始化模式保证所描述的空间是良序的，也保证操作在构造系统可能状态时能够在有限步内完成。如果没有初始状态，则状态空间中不存在底元或最小元；对状态空间中任一状态、操作都不可能在有限步内结束，这违反了操作定义中的可达性关系。通过上面的例子可以看到初始状态是可以任意指定的，即可以指定开关状态的初始状态为开，也可以为关，不会影响系统的完整性；一般来说应按常识选择合理的初始状态。但是，不是所有系统的初始状态都是可以任意指定的，只有在状态空间中的所有状态通过初始状态都是可达的条件下才行，否则不能任意指定初始状态。如果任意指定，则系统的某些状态就不可能通过操作模式构造出来，构造的系统也是不完整的。在开关的例中，下面的两个初始状态模式是正确的。

InitSwitch
Switch
---
switch=on

InitSwitch
Switch
---
switch=off

在计数器例中，初始状态只能是 counter=0，如果想使 counter 有值2，则通过 Add 操作，由状态 counter=0和 counter=1永远不会得到 counter=2，所建模型是不完备的。数字表例的状态空间中的所有状态通过初始状态都是可达的，所以初始状态是

可以任意指定的。

·系统状态不变式：不变式是对系统固有的不变性质的刻画；无论系统如何变化，都要满足不变式的要求；其作用是维护系统的封闭性、完整性，保证操作的完整性、合法性，同时也是程序正确性证明、求精的必要条件；还能为后续开发和维护提供依据。在上面数字表例中的状态不变式为0≤hours≤23∧0≤minutes≤59∧0≤seconds≤59，只有满足状态不变式的状态才是系统的合法状态，才能被系统所接受。在开关和计数器例中，我们没有显式给出系统的状态不变式，这是因为系统的状态不变式被系统的变量类型所蕴涵；如计数器的系统状态不变式是counter∈N，由于counter的类型是N，counter当然属于N；开关的系统状态不变式是switch＝on∨switch＝off，由于switch的类型是{on，off}，所以switch：{on，off}⇒switch＝on∨switch＝off。一般情况下，如果类型蕴涵了系统状态不变式，则系统状态不变式可以不必在状态模式中显式给出。系统状态不变式是和变量类型密切相关的；在数字表例中如果选择的数据类型为hours：0..23，minutes：0..59，seconds：0..59，则系统状态不变式可以不必在状态模式中显式给出，其相应的状态模式为：

Timer

hours：0..23

minutes，seconds：0..59

3. 实例

尽管上面我们把Z的各种机制分开来讨论，但应该看到各种机制是一个相互依赖、不可分割的整体，在这个整体里状态模式定义了系统的结构，初始化模式定义保证系统是可进行推理的良序集，保证系统的可操作性，操作模式通过初始化模式构造出系统状态模式中定义的各种可能状态，且各个状态满足系统状态不变式，它们相辅相成，构成了一个有机的整体。下面我们通过一个自行设计的实例表明它们之间的关系。

宇宙的本质特性是空间和时间，按照爱因斯坦的宇宙爆炸理论，可建立如下宇宙状态：用Time表示时间，Vector表示宇宙空间半径矢量；时间是有始无终的，t∈Time⇒t≥0；而空间是有始有终的，任何物质的运动速度不能超过光速，因此宇宙是封闭的，它的极限半径r为C·t_{bm}，C为光速，t_{bm}为宇宙运动速度到达光速时的时间；宇宙是在不断加速膨胀的，我们用时间t和空间半径矢量r对宇宙建模；给定任意时刻都对应唯一的宇宙半径，时间到宇宙半径矢量的一对一映射为Time⤔Vector满足：

$$r=\frac{dr}{dt}(t)+\frac{1}{2}\frac{d^2r}{dt^2}(t)^2。$$

基本数据类型：[Time，Vector]

宇宙的状态模式：Universe

宇宙的状态不变式：t：Time·$t\geq 0\wedge r$：Vector·$\frac{dr}{dt}\leq C$

初始条件：$t=0\wedge r=0$

操作模式：Enlarge　宇宙的膨胀

FindRvector　某时刻宇宙半径

FindTime　给定某个宇宙半径获取宇宙年龄

Universe

space：Time ⤔ Vector

t：Time

r：Vector

$t\geq 0$

$r=\frac{dr}{dt}(t)+\frac{1}{2}\frac{d^2r}{dt^2}(t)^2$

$\frac{dr}{dt}\leq C$

InitUniverse

Universe

---

space＝{}

FindRvector

ΞUniverse

t?：Time

r!：Vector

---

t?∈dom(space)

r!＝space(t?)

FindTime

ΞUniverse

t!：Time

r?：Vector

---

r?∈ran(space)

t!＝$space^{-1}$(r?)

Enlarge

ΔUniverse

$t\mapsto r\in space\wedge t'\mapsto r'\notin space\Rightarrow space'=space\oplus(t'\mapsto r')$

以上所建的宇宙模型包含了系统的全部可能状态，并通过系统不变式限定了系统的可能状态，对于$t<0$和$\frac{dr}{dt}>C$不是我们生活中的宇宙，所以这些状态应从系统中排除出去，初始状态是显然的，因为我们生活中的宇宙是从无到有起源于奇点，所以我们

生活中的宇宙的初始状态是空的。宇宙的不断膨胀构造出我们生活中的宇宙的全部可能状态，宇宙每时每刻都在构造出新的状态。该例充分表明系统的动态构造特性，没有初始状态就没有我们生活中的宇宙，没有系统不变式宇宙中就会包含我们生活中的宇宙之外的状态，而这种状态对我们来说是毫无意义的，没有宇宙的不断膨胀也就不会产生我们生活中的宇宙。该例还表明系统的不变式不可能实现自动抽取。不变式抽取是指，在状态模式中未写出或未写完全的不变式，而通过从一系列的操作模式中抽取。在本例中，因为$\frac{dr}{dt}\leqslant C$在所有模式中都不会出现，所以通过模式自动抽取系统的不变式是不可能实现的。

结论 综上所述，用Z开发系统是将问题域分解成相互独立的子系统，然后为各个子系统建模，其实质是对各个子系统构造一个符合问题陈述的封闭系统。其方法是用状态模式描述系统的所有可能状态，给出系统的不变式和初始状态，然后定义一组恰当的操作模式。这里状态的抽取是整个建模的关键，状态要反应系统的本质特征，既不要丢掉系统的特征，又不要将与系统无关的、非本质的东西加进系统中，以保证对系统的描述准确、简洁，使得后续工作（如正确性证明、求精等）顺利进行。

找出正确的初始状态。初始状态模式是构造和检验操作模式的关键，所以找出正确的初始状态模式也是非常重要的工作。尽管我们能够证明初始状态的存在[5]，但不能证明其不存在；换句话讲，就是在所给初始状态确实存在的前提下，我们能够证明初始状态的存在；如果所给初始状态不存在，我们不能够证明其不存在。

通过上面的例子我们看到系统不变式是不可能实现自动抽去的，许多系统不变式与描述系统的模式中的符号无关，系统不变式的确定要靠开发人员对问题域的把握和理解。

目前还没有一种既方便又能保证正确的开发方法，即使以后出现了较方便的开发方法，我们认为对问题域的抽象还是需要靠开发人员完成；每一个软件工作者都应该对所面对的问题域进行深入的了解、学习、研究、努力把握问题的本质，只有这样才能开发出既正确又符合实际的规格说明；一个拿着傻瓜相机的人，如果他不深入生活作细致的观察，是不可能创作出优秀的作品来的。

参考文献

[1] A. Diller, Z. An Introduction to Formal Methods, Chichester, UK: John Wiley & Sons, 1990

[2] H.K. 伯格，程序验证和规范的形式方法，科学出版社，1988.4

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

[4] 朱海滨，面向对象技术——原理与设计，国防科大出版社，1992.10

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

[6] 陈涵生，面向对象开发技术及应用，上海科技文献出版社，1995

[7] Bryan Ratcliff, Introducing Specifidation Using Z—A Practical Case Study Approach, London: McGraw-Hills, 1994