计算机科学2000Vol. 27№. 2

一种从Z到精化演算的软件开发方法*)

A Software Development Method: From Z to Refinement Calculus

查 鸣 王云峰 郑国梁

(南京大学计算机科学系 软件新技术国家重点实验室 南京210093)

Abstract A software development method was introduced in the paper. First we wrote formal specification in Z and implemented the data refinement. Then we translated it into the refinement calculus notation. Finally we used the laws of the refinement calculus to develop abstract programs into executable code. Translation rules from Z specification to abstract programs were represented.

Keywords Z, Specification, Refinement calculus

一、引言

形式化方法的研究和应用已有二十多年的历史，源于Dijkstra和Hoare的程序验证以及Scott、Stratchey等人的程序语义研究，指为保证复杂系统的可靠性、以数学为基础对其进行精确描述和验证的语言、技术和工具。形式化方法的关键在于形式规约语言。通过语法和语义有严格数学定义的形式规约语言对系统及其各方面性能的描述，产生系统的形式规约，可以帮助开发者获得对所描述系统的深刻理解，并通过揭示隐含的不一致性、二义性、不完整性发现设计中的错误和缺陷，而非形式化方法往往难以发现这些错误。形式规约为系统的设计、实现和验证提供了准确、一致的文档依据，成为用户和设计者之间、设计者和实现者之间、实现者和测试者之间充分沟通的可靠桥梁[1]。

近年来，Z[2]作为一应用广泛的形式描述语言，由于其良好的描述特性，在欧洲、美国的工业界以及其他一些地区的软、硬件开发项目中都得到应用，但大多数都是用来描述形式化规约，而极少在其后的开发阶段得到应用，因为Z不是一种实现级的语言，无法直接得到可执行程序，因而无法在软件生命周期的各个阶段都得到应用，也就不能形成完整的开发方法。

由Carroll Morgan提出的精化演算理论[3]通过扩充Dijkstra的卫式命令语言，将规约看作抽象的程序，利用给出的精化规则对形式化规约进行精化，逐步降低规约的抽象级，最终得到可执行程序。

本文比较了Z和精化演算的表示体系，给出了二者之间的转换规则，将二者集成到一个开发方法中，利用Z的描述能力实现规约的数据精化，而利用精化演算实现操作精化，形成一种完整的形式化开发方法，最后给出一个简单的例子。

二、Z和精化演算的表示体系

2.1 Z的表示体系

Z的数学基础是Zermelo-Fraenkel集合论和一阶谓词逻辑，借助于模式(Schema)来表达系统结构。它提供了一种能独立于实现的、可推理的系统数学模型，具有精确、简洁、无二义性的优点，有利于保证程序的正确性，尤其适用于无法进行现场调试的高安全性系统的开发。模式是Z的一大特点，它由说明和约束两部分组成，既可以描述状态，也可以通过状态转换的前、后置条件来描述操作。Z还提供了一组针对模式的运算符，用于进行模式之间的联结，称为模式演算。Z的另一特点是有着丰富的集合运算符号集，关系、映射、函数等作为特殊类型的集合都有专门的表示和运算符号，从而大大增强了Z的表达能力[2,4]。

2.2 精化演算

精化演算由Carroll Morgan提出，通过对Dijkstra的卫式命令语言进行扩充将语言本身和规约语句、精化规则结合起来。精化演算可以将抽象的规约逐步精化成一个可执行的程序，而此程序的正确性依赖

*)本课题得到"九五"攻关项目(98-780-01-07-06)和国家自然科学基金(69673006)资助。查鸣 硕士生，主要研究领域：对象技术，形式化方法，软件工程；王云峰 博士生，主要研究领域：对象技术，形式化方法，软件工程；郑国梁 教授，博士生导师，主要研究领域：软件工程。

于精化过程中每一步骤的正确性[3]。精化演算中的规约语句具有如下形式：

$w:[pre, post]$

如果该语句能推导出可执行代码，那么它具有以下的含义：从一个满足前置条件 *pre* 的初始状态开始，通过改变状态变量列表 w 中的状态变量，建立一个满足后置条件 *post* 的终止状态。

2.3 两种表示体系的区别

精化演算利用保留字 **var** 和 **and** 引入状态变量和状态不变式来表示一个有效状态，这与 Z 用状态模式来描述状态非常相似。然而二者对于某个状态上操作的描述有很大的区别。在 Z 中，操作模式的说明部分给出了操作涉及的状态以及输入输出变量，约束部分通过谓词给出了状态变化前后状态变量以及输入输出变量之间应满足的关系。而相应的操作模式在精化演算中则表示为一个规约语句。

另外，二者对于初始状态变量的引用也不尽相同。在 Z 中，x 表示状态变量 x 在操作执行前的值，x' 表示 x 在操作执行后的值。在规约语句中，x_0 表示状态变量 x 在操作执行前的值，x 表示 x 在操作执行后的值。

三、Z 和精化演算相结合的开发方法

基于形式化描述语言 Z 和精化演算理论，我们提出以下形式化软件系统开发过程：首先进行领域分析，在和用户交流的基础上得到系统的非形式化需求描述，这是非形式化分析部分；在形式化开发阶段，首先用 Z 写出其形式化规约，并对 Z 规约进行数据精化，随后进行由 Z 规约向精化演算的转换，再对其进行逐步求精得到程序代码。

我们首先假设待开发的系统有足够的大小和复杂度，利用模式来表示其规约能有助于开发。如果待开发系统数据结构非常简单，则可直接用精化演算给出其规约进而得到程序代码，无须用模式来描述。

在此前提下，形式化开发的第一步就是写出 Z 的规约，利用模式演算给出系统完整的形式化描述。

第二步进行数据精化。数据精化过程中，先针对抽象状态模式给出其具体状态模式，以及抽象模式和具体模式之间的对应关系，然后根据此关系重写所有的模式。最后还需证明重写后的模式的正确性。值得注意的是，具体状态会引入复杂的数据结构，这使得改写后的模式相应地变得复杂起来。

第三步进行表示体系的转换。

1. 状态模式的转换。只需将其说明部分和状态不变式分别置于精化演算的 **var** 子句和 **and** 子句。

2. 操作模式的转换。设 Op 为基于状态模式 S 的操作模式，Op_1 和 S_1 分别表示经过数据精化的对应模式，并且 Op_1 具有如下的形式：

Op_1

$\triangle S\_1$

$i?:I$

$o!:O$

$pred$

那么 Op_1 可转换为规约语句：

$w:[(\exists\ w:T \mid inv \cdot pred)[w_0\backslash w], pred]$

其中 w 包括 S_1 中的状态变量及输出变量 o；inv 为状态模式 S_1 的不变式；$[w_0\backslash w]$ 保证了前置条件中的变量不出现下标0。

3. 模式演算的转换。对于模式演算，可根据以下的规则直接转换为已具有某些结构的抽象程序，而不是转换为一个规约语句。

规则1 设 $Op \triangleq Op1 \vee Op2$，则 Op 可转换为如下的条件语句：

if pre $Op1 \rightarrow Op1^*$
[]pre $Op2 \rightarrow Op2^*$
fi

其中 $Op1$，$Op2$ 为经过数据精化的操作模式，并且 $Op1$，$Op2$ 的前置条件在目标语言中均是简单表达式，$Op1^*$，$Op2^*$ 表示与 $Op1$，$Op2$ 相对应的规约语句。

规则2 设 $Op \triangleq Op1 \vee Op2$，则 Op 可转换为如下的条件语句：

|[**var** b: *Boolean* ·
$b:[true, b \Leftrightarrow$ pre $Op1]$;
if $b \rightarrow Op1^*$
[]pre $Op2 \rightarrow Op2^*$
fi
]|

其中 $Op1$，$Op2$ 为经过数据精化的操作模式，并且 $Op1$ 的前置条件在目标语言中是一个复杂表达式，$Op1^*$，$Op2^*$ 表示与 $Op1$，$Op2$ 相对应的规约语句，b 是一个新引入的局部变量。

规则3 设 $Op \triangleq Op1 \vee Op2$，则 Op 可转换为如下的条件语句：

|[**var** $r:T$ ·
$r:[true, \varphi]$;
if $\psi_1 \rightarrow w:[\varphi \wedge \psi_1, Op1]$
[]$\psi_2 \rightarrow w:[\varphi \wedge \psi_2, Op2]$
fi
]|

其中 $Op1$，$Op2$ 为经过数据精化的操作模式，r 是新引入的局部变量，φ、ψ_1、ψ_2 为任意的谓词并满足：①$\varphi \wedge (pre\ Op1 \vee pre\ Op2) \Rightarrow (\psi_1 \vee \psi_2)$；②$\varphi \wedge (pre\ Op1 \vee pre\ Op2) \Rightarrow (\psi_i \Rightarrow pre\ Opi)$，$i = 1, 2$。

规则4 设 $Op \triangleq Op1 \wedge Op2$，且 $Op1$，$Op2$ 如下形式：

$Op1 \triangleq [\triangle S \mid s'_1 = s_1 \wedge P1(s_2, s'_2)]$
$Op2 \triangleq [\triangle S \mid s'_2 = s_2 \wedge P2(s_1, s'_1)]$

其中 s_1 和 s_2 是不相交的状态变量集，谓词 $P1, P2$ 分别描述了对部分状态变量的改变。则 Op 可转换为如下任意的顺序结构之一：

$s_1 : [\text{pre } Op2, P2]$;
$s_2 : [\text{pre } Op1, P1]$

或 $s_2 : [\text{pre } Op1, P1]$;
$s_1 : [\text{pre } Op2, P2]$

在完成由模式到抽象程序的转换后，即可利用精化演算的精化规则对抽象程序进行逐步求精，最终得到用卫式命令语言实现的程序代码。以下将通过一个例子说明整个开发过程。

四、例子

有如下非形式化需求规约：一计算机培训班，当有学生报名时，如果本班人数未满，则此学生可进入本班学习并且他此时未通过测验。当学生通过测检后，应作相应记录。学生在通过测验后得到合格证书，未通过测验中途退出的学生将没有合格证书[5]。由于篇幅所限，以下仅给出完成测验的操作，有关报名入学的操作与此类似。

4.1 形式化规约

首先给出一个表示学生的给定数据类型：[Student]

本班可招收学生的最大数为一全局常量：

max : ℕ
max > 0

抽象状态包括两个不相交的集合：集合 y 表示已通过测验的学生，集合 n 表示已录取但未通过测验的学生。

Class
y, n : ℙ Student
y ∩ n = {}
(y ∪ n) ≤ max

初始状态两集合均为空。

Init
Class'
y' = {}
n' = {}

通过测验的学生将从 n 转移到 y 中。

Compl_ok
ΔClass
s? : Student
s? ∈ n
y' = y ∪ {s?}
n' = n \ {s?}

引入一新类型用于显示系统信息：

Response ::= ok | found | full | missing

对于通过测验的学生，返回一个已通过的系统信息：

Success
Resp! : Response
Resp! = ok

对于完成测验的异常情况处理：

Missing
ΞClass
s? : Student
resp! : Response
s? ∉ n
resp! = missing

综上可得通过测验的完整操作：

Complete ≙ (Compl_ok ∧ Success) ∨ Missing

4.2 数据精化

引入两个数组，一个用于存放本班的所有学生，另一个用于存放布尔值，表示对应的学生是否通过测验，引入变量 num 表示本班现有学生数。

精化后的状态模式：

Class_1
cl : 1..max → Student
ex : 1..max → Boolean
num : 0..max
((1..num) ◁ cl) ∈ (ℕ ≻→ Student)

抽象状态与具体状态之间的对应关系如下：

Retr

Class

Class_1

$y=\{i:1..num \mid (ex\ i)=true \cdot (cl\ i)\}$

$n=\{i:1..num \mid (ex\ i)=false \cdot (cl\ i)\}$

精化后的初始状态为：

Init_1

Class_1'

$num'=0$

对于 Init_1，需证明与 Init 具有对应关系：

$Init\_1 \vdash (\exists\ Class' \cdot (Init \wedge Retr'))$

对操作模式 Compl_ok 精化后得到：

Compl_ok_1

$\Delta Class\_1$

$s?:Student$

$\exists\ i:1..num \cdot (cl\ i=s? \wedge ex\ i=false \wedge$
$cl'=cl \wedge$
$ex'=ex \oplus \{i \mapsto true\}$
$num'=num)$

对操作模式 Missing 精化后得到：

Missing_1

$\Xi Class\_1$

$s?:Student$

$resp!:Response$

$\forall\ i:1..num \cdot cl\ i \neq s? \vee ex\ i=true$

$resp!=missing$

模式演算 complete 精化为 complete_1：

$Complete\_1 \triangleq (Compl\_ok\_1 \wedge Success) \vee Missing\_1$

在进行表示体系转换之前，需证明以下公式成立，以保证数据精化的正确性：

$(pre\ Complete) \wedge retr \vdash pre\ Complete\_1$

$(pre\ Complete) \wedge retr \wedge complete\_1 \vdash (\exists\ Class\_1 \cdot complete \wedge retr')$

由于篇幅所限，不再给出以上两公式的证明过程。

4.3 转换为精化演算的抽象程序

根据状态模式转换规则，状态模式 Class_1 转换为：

$$\begin{array}{l}\textbf{var}\ cl:1..max \rightarrow Student;\\ \quad ex:1..max \rightarrow Boolean;\\ \quad num:0..max;\\ \textbf{and}((1..num) \lhd cl) \in N > \rightarrowtail Student\end{array}$$

初始化模式 $Init\_1$ 转换为：

$$\textbf{initially}\ num=0$$

为了利用规则3对模式演算 $Complete\_1$ 进行转换，首先定义谓词 H：

$$H \triangleq (w \in 1..num \wedge cl[w]=s) \vee (w=num+1 \wedge s \notin cl[1..num])$$

应用规则3，$Complete\_1$ 可转换为：

$$\begin{array}{l}Complete(\textbf{value}\ s:Student;\textbf{result}\ r:Response)\\ \sqsubseteq \textbf{var}\ w:0..max+1\cdot\\ w:[true, H]; \qquad (1)\\ \textbf{if}\ w \in 1..num \wedge ex[w]=false \rightarrow r, ex:\\ \left[\begin{pmatrix} H \\ w \in 1..num \\ ex[w]=false \end{pmatrix}, \begin{pmatrix} r=ok \\ ex=ex_0 \oplus \{w \mapsto true\} \end{pmatrix}\right] \qquad (2)\\ [] w=num+1 \vee ex[w]=true \rightarrow\\ \quad r:[H \wedge (w=num+1 \vee ex[w]=true),\\ \quad r=missing] \qquad (3)\\ \textbf{fi}\end{array}$$

4.4 操作精化

由精化演算的精化规则可以得到：

$$\begin{array}{l}(2) \sqsubseteq r, ex[w]:=ok, true\\ (3) \sqsubseteq r:=missing\\ (1) \sqsubseteq w:=1;\\ \quad \textbf{do}\ w \neq num+1 \wedge cl[w] \neq s \rightarrow\\ \quad\quad w:=w+1\\ \quad \textbf{od}\end{array}$$

至此，我们得到了 *Complete* 的完整的可执行程序。

$$\begin{array}{l}Complete(\textbf{value}\ s:student;\textbf{result}\ r:Response) \triangleq\\ |[\textbf{var}\ w:0..max+1\cdot\\ \quad w:=1;\\ \quad \textbf{do}\ w \neq num+1 \wedge cl[w] \neq s \rightarrow\\ \quad\quad w:=w+1\\ \quad \textbf{od};\\ \quad \textbf{if}\ w \in 1..num \wedge ex[w]=false \rightarrow\\ \quad\quad r:=ok;\\ \quad\quad ex[w]:=true\\ \quad [] w=num+1 \vee ex[w]=true \rightarrow\\ \quad\quad r:=missing\\ \quad \textbf{fi}\\ ]|\end{array}$$

结论 本文着重论述了如何将形式化规约从 Z 转换至精化演算的表示体系，进而将两种不同的表示体系有机地集成，形成一种完整的形式化开发方法。此方法的核心就是分别利用这两种表示体系的特点完成各自最适合的开发阶段的工作：利用 Z 良好的描述特性进行分析和设计工作，得到完整的系统规约并完成规约的数据精化；然后转换成精化演算的规约语句，得

（下转第68页）

属于自己的视点类的层次。

VORD 将视点的识别分为五个阶段：

(1)去掉视点类层次图中与系统无关的视点类；

(2)系统的风险承担者，即那些将受到系统影响的人，如果他们没有被包括在视点类层次中，那么在层次中填加这些类；

(3)通过系统结构模型来识别于系统的视点；

(4)通过区分经常使用系统的用户，偶尔使用系统的用户和间接使用系统的用户，得到系统潜在的视点；

(5)对于那些间接视点类，考虑与它们相关的分析人员的视点。

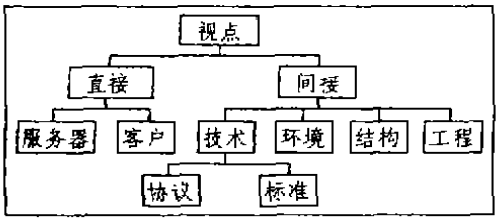

图6 一个视点类的例子

结束语 本文中我们集中介绍了一些比较典型的基于多视点的需求工程方法，同时，我们用一个简单的分布式多媒体会议系统的例子，说明了它们是如何进行视点的识别，视点的分类，视点的说明，视点的集成的。这个领域还有许多问题，如视点间或视点内不一致性和冲突的管理等，都亟待更好地解决，目前已有一些学者开始了这方面的研究工作。

参 考 文 献

1 Ross D, Schoman K E. Structured analysis for requirements definition. IEEE Trans, 1977,3(1):6～15

2 Ross D T. Structured analysis: a language for communicating ideas. IEEE Trans., 1977,SE-3(1):16～34

3 Ross D T. Applications and extensions of SADT. Computer, 1985,18(4):25～34

4 Mullery G P. CORE—— a method for controlled requirement specifications. In: 4th IEEE Computer Socty Int Conf. On Software Engineering. Munich, Germany, 1979.126～135

5 Finkelstein A, et al. Viewpoint: a framework for integrating multiple perspectives in system development. Int. J. Softw Eng. Knowl. Eng., 1992,2(1):31～57

6 Finkelstein A, Fuks H. Multi-party specification. In 5th Int Workshop on Software Specification and Design, IEEE Computer Society, 1989.185～195

7 Nuseibeh B, et al. A framework for expressing the relationships between multiple views in requirements specification. IEEE trans. on Softw. Eng., 1994,20(10):760～773

8 Leite J C S P. Viewpoint analysis: a case study. In Proc. 5th Int. Workshop on Software Specification and Design (IWSSD-5). IEEE Computer Society Press, 1989.111～119

9 Kotonya G, Sommerville I. Requirements engineering with viewpoints. Softw. Eng. J. 1996,11(1):5～18

10 Roman G-C. A taxonomy of current issues in requirements engineering. Computer, 1985(April):14～21

(上接第17页)

到抽象的程序；最后，利用精化演算的求精规则由抽象程序得到可执行程序代码。

目前从形式化规约到程序代码的求精过程主要由人工实现，自动化程度较低。如何在求精过程中尽可能多地应用机器辅助技术是形式化方法能否成功的关键问题之一。例如：利用机器辅助技术进行数据精化的正确性证明，机器辅助实现从 Z 到精化演算表示体系的转换，机器辅助实现操作精化。但在实际应用中还有大量工作要做，如对各求精步骤具体、实用策略的深入研究、完备的谓词转换定理系统的建立及其选用策略的研究等。

参 考 文 献

1 袁晓东. Z 的面向对象扩充和程序转换系统的研究：[南京大学博士毕业论文]. 1997.12

2 Spivey J M. Understanding Z: a specification language and its formal semantics. Number 3 in Cambridge tracts in theoretical computer science. Cambridge University Press, Cambridge, 1988

3 Morgan C C. Programming from Specifications. Prentice-Hall International series in computer science/C. A. R Hore, series editor. Prentice-Hall International, Englewood Cliffs, N. J.; London, 1990

4 Hayes I J. Specification case studies. Prentice-Hall International series in computer science/C. A. R. Hore, series editor. Prentice-Hall International, Englewood Cliffs, N. J.; London, 1980

5 King S. A refinement calculus case study: [Technical Report PRG-TR-7-90]. Programming Research Group, 1990