VDM 和 Z 两种规范描述语言的比较*

朱 玉 ○　　陈 忠 民　　张 乃 孝
（湖州师专）　　（北京大学）

摘要

本文以关系数据库的规范为例，详细讨论了两种重要的规范描述语言 VDM 和 Z 的主要区别，对它们的共性和发展史也作了简单介绍。

关键字：VDM，Z，形式方法，规范描述语言

前言

60 年代初期，IBM 决定开发一种新的程序设计语言来代替 FORTRAN 和 COBOL，这个语言后来称为 PL/1。由于这个语言比较庞大，在设计过程中使用了形式化的方法，首先定义了 PL/1 的操作语义，1972 年，奥地利的 IBM 维也纳实验室做编译系统时又为 PL/1 写了指称语义，这些工作直接导致了 VDM 的产生。VDM 即维也纳开发方法，是一种使用较早、传播较广的开发方法。VDM 的规范描述语言称为 Meta-IV，是一种比较松散的语言，它有一个基本的核心，使用者可以根据需要增加新的成分。1976 年 IBM 维也纳实验室开展新的研究方向，VDM 的主要创始人 Dines Bjorner 和 Cliff Jones 各自回国继续从事 VDM 的研究，在描述语言、开发方法、应用和理论等多方面取得进展[1][2]。VDM 的成果影响了后来的 RAISE、COLD_K 和 VVSL 等项目的研究。

Z 产生于 70 年代末期，其早期的发展与 Jean_Raymond Abrial 在牛津大学从事的工作是分不开的，当时他给牛津大学的 Programming Research Group(PRG)讲授 Z 方法，就使用了集合理论和谓词演算等许多数学工具。Z 的早期应用主要是描述一些工业实例，Ian Hayes 将这些实例收集起来，于 1987 年出版了关于 Z 的第一本书（该书于 1993 年再版）[3]。Z 的一些重要特性（如模式和相关的模式操作）就是这时候出现的。模式的基本操作如包含和联接等经过扩展形成了更复杂的操作，进一步发展成模式演算。1989 年末成立了一个小组开始开发 Z 的标准，1992 年 12 月在 Z 用户会议上发表了标准的第一稿[4]。

本文以一个简单的关系数据库 NDB 为例，用 VDM 和 Z 分别写出它的部分规范，以此来比较这两种描述语言的异同。基本不涉及形式开发方法，工具和环境的研究。文中用来描述 NDB 基本成分的符号有：

Eid　实体标识符集合
Value　实体值集合
Esetnm　实体集合(或类型)的名字集合
Rnm　关系名字集合

另外为排版方便使用以下表示符号：

△　定义符　　A　全称量词　　→　部分函数　　→m　映射

* 本项研究得到国家自然科学基金的资助

$\rightarrow$f **有限部分函数** $\rightarrowtail$ **单射部分函数**

一、规范形式及整体结构

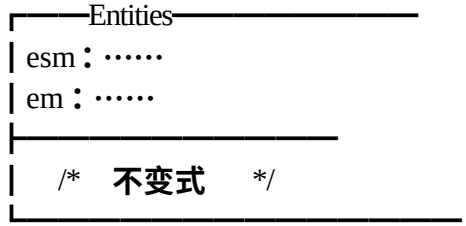
从整体结构上看，VDM **规范和** Z **规范的书写形式具有明显的不同。在** VDM **规范中包含有许多关键字，与一般高级语言类似，用它们来指明规范中不同成分。例如在** VDM **中用一个模块来刻划关系数据库规范时其结构可以为：**

```
module NDB
/*  以下为接口部分    */
parameters
  types Value,Esetnm,Rnm：……
exports
   operations ADDES ……
/*  以下为定义部分    */
definitions
/*  类型定义    */
defined types
Eid=token
……
Rkey ：: nm : Rnm
          fs : Esetnm
          ts : Esetnm
/*  状态定义   */
state
Ndb : esm : Esetnm →m  Eid-set
        em  : Eid →m  Value
        rm  : Rkey →m  Rinf
inv(mk-Ndb(esm,nm,rm)) ……
/*  初始状态定义   */
init(ndb)  ……
/*  操作定义    */
defined functions
……
endmodule NDB
```

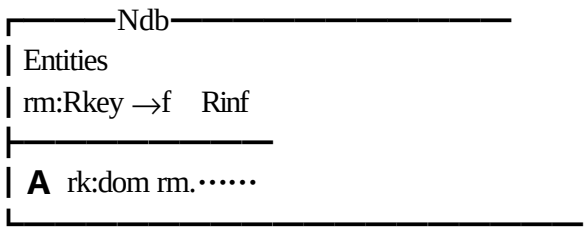
其中 module, parameters, exports, operations, definitions, state……**均为关键字。在** VDM **中用一个模块表示一个抽象机，除接口外主要包含该抽象机的状态和操作的描述，状态包含若干分量说明以及这些分量必须满足的不变式，操作给出使用说明和应满足的前置条件和后置条件。这些不同的成分由于它们在规范中的不同作用而用不同的关键字指明。**

在用 Z **描述一个规范时，规范整体结构是不明显的，典型的** Z **规范由许多称为模式**(schema)**的定义组成，这些模式看起来象“盒子”。模式用来描述状态和操作，**

也可用来作为描述状态或操作的构造模块。对 NDB 而言，它的规范可以分解成三个部分：第一部分，库中实体和实体类型的描述；第二部分，库中单个关系的描述；第三部分多重关系的描述。这些规范合在一起形成完整的 NDB 规范。例如用 Z 可先定义 NDB 中的 Entities 状态及其操作的模式：



然后在下面的 NDB 模式中引用已定义的 Entities 模式:



二、基本数据类型及其表示方法

VDM 中的基本数据类型是集合、组合类型、映射和序列，而 Z 是以集合、关系、函数和序列作为它的基本数据类型。集合、序列是 VDM 和 Z 共有的数据类型，它们有类似的性质和操作，只是符号不同。

用 VDM 方法构造 NDB 规范，使用的基本成分是 Eid, Value, Esetnm, Rnm, 用它们构造下列类型：

Tuple：：fv:Eid
tv:Eid

Relation = Tuple_set

Maptp = OneOne | OneMany | ManyOne | ManyMany

Rinf：：tp:Maptp
r:Relation

其中 Rinf 需满足的不变式为

inv(mk_Rinf(tp,r)) △
(tp=OneMany => A t1,t2∈r・t1.tv = t2.tv => t1.fv = t2.fv) ∧
(tp=ManyOne => A t1,t2∈r・t1.fv = t2.fv => t1.tv = t2.tv) ∧
(tp=OneOne => A t1,t2∈r・t1.fv = t2.fv <=> t1.tv = t2.tv)

值得注意的是，不变式是类型中所有值都必须满足的约束条件。因此若 rel∈Rinf 则必需满足①rel.r∈Relation, ②rel.tp∈Maptp, ③rel.tp 和 rel.r 满足上述不变式。

另外定义：

Rkey：：nm:Rnm
fs:Esetnm

ts:Esetnm

以上我们定义了许多类型，这些类型都是为最后描述状态作准备的。为定义这些类型，我们使用了组合类型（例如 Tuple,Rinf,Rkey），集合类型（Relation）和枚举类型（Maptp）。

Z 中描述 NDB 规范的基本集合也是：[Eid,Esetnm,Value,Rnm]。对于一个数据库，我们需要按实体集来组织实体。下面将它的状态元素与连接这些元素的不变式用模式 Entities 一块来描述。

```
┌──Entities──────────────
│ esm:Esetnm →f  (P Eid)
│ em:Eid →f  Value
├────────────────────
│ dom em = ⋃(ran esm)
└────────────────────────
```

其中 →f 表示有限部分函数。

另外可定义：

Tuple == Eid × Eid

Relation == Eid <—> Eid

VDM 和 Z 都使用对偶的集合来定义关系，VDM 中没有关系这种类型，而用组合类型和集合类型来定义关系，在 Z 中关系是基本数据类型，在关系上定义了丰富的操作符。其中 Eid<—>Eid 是 P(Eid×Eid)的简写形式，即 Eid×Eid 的幂集，“ P ”与 VDM 中的集合构造符“ _set ”作用相同。“ × ”代表笛卡尔集，而在 VDM 中笛卡尔集是通过组合类型定义的。

Z 中 Maptp 的定义与 VDM 实际上是一样的

Maptp ::= OneOne | OneMany | ManyOne | ManyMany

当一个关系被创建时，它被描述为以下四种类型之一：一对一的关系（即单射部分函数），一对多的关系（即它的逆是部分函数），多对一的关系（即部分函数），或者多对多的关系。

```
┌──Rinf──────────────────────────
│ tp:Maptp
│ r:Relation
├──────────
│ (tp = OneOne => r∈(Eid ≻→ Eid)∧
│ (tp = ManyOne => r∈(Eid → Eid)∧
│ (tp = OneMany => r~ ∈(Eid → Eid)
└──────────────────────────────────
```

在 Z 中，X 和 Y 之间的二元关系的集合记作 X<—>Y，部分函数的集合记作 X → Y，单射部分函数的集合记作 X ≻→ Y，关系 r 的逆记作 r~ 。

```
┌──Rkey──────────────
```

```
| nm:Rnm
| fs,ts:Esetnm
└──────────────────────────
```

由此可见，Z 中定义的 Tuple、Relation、Maptp、Rinf 和 Rkey 与 VDM 中含义相同，只是具体语法以及定义方法不一样。Z 中还多定义了 Entities，这是为定义最后的状态而引入的类型。

三、状态描述及初始状态定义

有了基本的数据类型，我们就可以描述状态（state）了。

用 VDM 描述关系数据库 NDB 的状态包括三个组成元素：esm, em, rm。定义：

Ndb::esm:Esetnm →m Eid_set
em:Eid →m Value
rm:Rkey →m Rinf
inv(mk_Ndb(esm,em,rm)) △
dom em = ∪ rng esm ∧
∀ rk∈dom rm ·{rk.fs, rk.ts} ⊆ dom esm ∧
∀ mk_tuple(fv,tv) ∈rm(rk).r · (fv ∈esm(fs) ∧ tv ∈esm(ts))

在 Z 规范中这个状态为

```
┌──────Ndb──────────────────
| Entities
| rm:Rkey →f Rinf
├──────────────
| ∀ rk:dom rm ·
|       {rk.fs, rk.ts} ⊆ dom esm ∧
|       (∀ t:(rm rk).r ·
|           first t ∈esm(rk.fs)∧
|           second t ∈esm(rk.ts) )
└──────────────────────────────
```

这里，VDM 和 Z 描述状态时除了具体语法不同外，不变式的表示方法也不一样：VDM 中的不变式是一个独立的从状态到真值的函数，而 Z 中的不变式是在模式的下半部分给出的谓词。

VDM 中初始状态定义为

init(ndb) △ ndb = mk_Ndb(,,)。

而 Z 中初始状态由如下模式定义：

```
┌──────Ndb_Init──────────────
| Ndb
├──────
| esm = {} ∧ em = {} ∧ rm = {}
```

如果为了遵循 NDB 的更结构化的表示，我们也可以先定义 Entities 的初始集合，然后再在 Entities 的初始集合之上定义 NDB 的初始状态。

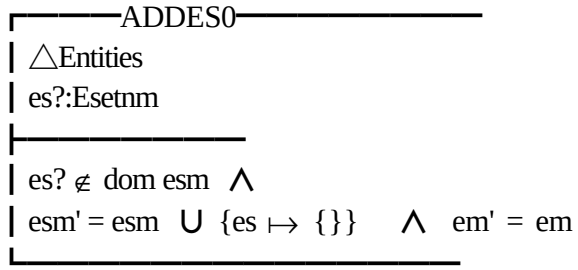
四、操作的定义

下面以往已知的实体集合 esm 加入一个新的实体集合名字的操作为例，对 VDM 和 Z 操作的定义方式进行比较。

在 VDM 中这个操作可定义为

ADDES(es:Esetnm)
ext wr esm:Esetnm →m Eid_set
pre es ∉ dom esm
post esm = esm∪ {es ↦ {}}

在 Z 中则为



其中△Entities 引进了前后状态：

△Entities ≙ Entities ∧ Entities'

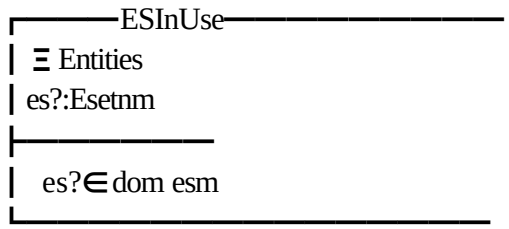
一个明显的语法上的区别是：VDM 使用上面带箭头线(这里为了排版方便改用下划线)的变量表示执行前状态，未带的变量表示执行后状态，而 Z 使用未加任何标记的变量表示执行前的状态，带"'"的变量表示执行后的状态。在 Z 中带"?"的变量名作为输入（带"!"的变量作为输出）。除了语法上的区别外，VDM 使用了外部子句并且区分前后置条件。Z 中没有与外部子句对应的部分。谓词必须定义所有的变量的最后的值(即使变量值未被修改，如 em 也得定义)。因此，对于一个包含很多状态元素的较大的规范来说，如果我们不得不在整个状态上定义操作，那么可能要写上许多说明变量未被修改的谓词。如果将一个状态分解成若干小的元素组，在每一组中定义子操作，然后再组合这些子操作来定义整个的操作，则可避免这个问题。

VDM 操作中有独立的前置条件和后置条件，Z 模式中表示前置条件的谓词有时不能直观地看出来，要通过推算才能得到。Z 模式 ADDES0 的前置条件与 VDM 操作 ADDES 一样，这是能直观地看出来的。

在 VDM 方法中，一个操作的规范由正常的前后置条件和异常前后置条件组成。而在 Z 方法中则分别描述一个操作正常情况和异常情况下的模式，然后再合并这两种模式得到包含错误处理的操作。例如，当增加一个新的实体集合，而这个集合的名字已被使用时，VDM 方法在操作 ADDES 的定义通过增加

err ESInUse es ∈dom esm。

来描述这种异常情况。而在 Z 中这种异常情况的描述为：



其中 Ξ Entities 引进执行前后状态并限制它们是同样的：

Ξ Entities = [△Entities| Entities' = Entities]

增加了错误处理的操作为

ADDESX = ADDES0 ∨ ESInUse。

五、其他区别

在 VDM 方法中，可满足性证明是写规范者进行相容性(consistency)检查的主要方法。而 Z 方法中前置条件的推算可看作主要的相容性检查，推算出来的前置条件与写规范者的期望一致则说明规范是正确的。

VDM 使用的是三值逻辑，而 Z 使用二值逻辑。

Z 规范中避免递归结构，而是使用一种展开的表示法。VDM 方法中则可以定义递归结构。例如，考虑一棵简单的二叉树。在 Z 中描述为：

T ::= nil | binnode<<T×N×T>>

上式引进了新的类型 T 及其常量 nil，和一个单射的构造函数 binnode，当给出了一个左子树，一个自然数和一个右子树时它返回一棵二叉树。

在 VDM 中将它描述为

binnode :: l : T
v : N
r : T

T = [binnode]

这里用 binnode 引进一个新类型。

结束语

前面我们对 VDM 和 Z 作了全面的比较。VDM 和 Z 都用来写抽象机的规范，而且都采用了“面向模型”的方法：都给出抽象机状态的明确的模型，而抽象机的操作则定义在状态之上。和面向模型的思想不同的另一类规范语言——代数规范语言采用“面向特性”的方法，它把重点放在抽象数据类型的描述上，抽象数据类型是通过公理来描述的，这些公理定义了抽象数据类型上的操作之间的关系，它并没有给出类型的明确模型。例如，在 VDM 和 Z 中栈的典型模型是序列，而在“代数”方法中则给出 pop(push(x,s))=s 等公理来描述栈。“代数”方法的抽象程度更高，

但“面向模型”的方法比较具体，写出来的规范可读性好，程序员易于熟悉掌握，而且也易于实现原型。

VDM 和 Z 的研究现在还是一个比较受人重视的课题，每隔 1-2 年就有一次关于 VDM 和 Z 等形式方法的国际会议，有关的理论、应用以及标准化工作在继续。

参考文献

[1] C.B.Jones Systematic Software Development Using VDM
PHI Ltd,1990

[2]J.Dawes The VDM-SL Reference Guide
Pitman, 1991.

[3] Ian Hayes,editor. Specification Case Studies.
Prentice Hall International,second edition,1993.

[4] J.M.Spivey. The Z Notation:A Reference Manual
Prentice Hall International, second edition, 1993.

[5] I.J.Hayers,C.B.Jones,J.E.Nicholls
Understanding the differences between VDM and Z
ACM SIGSOFT Software Engineering Notes vol 19 no 3,1994

The Contrasts Between VDM and Z

Zhu Yujie
(Huzhou Teachers College)
Chen Zhongmin Zhang Naixiao
(Peking University)

ABSTRACT: In this paper, the contrasts between VDM and Z are exposed by means of a database specification. The main features in these specification languages are discussed in detail.

KEYWORDS: VDM, Z, Formal Methods, Specification Languages