形式化软件开发方法 DD－VDM*

吕 建　张建莹

（南京大学计算机软件研究所　南京　210093）

（南京大学计算机软件新技术国家重点实验室　南京　210093）

摘要　在指出 VDM 方法用于较大程序开发的不足的基础上，本文提出了基于模型分解、模块化和运算分解思想的数据分解的概念，并将其引入 VDM，从而得到比 VDM 更加一般的软件形式化开发方法 DD－VDM. 在 DD－VDM 中，可采用较为灵活的开发策略，并且开发过程的复杂性也可得到有效控制.

关键词　形式化方法，数据分解，VDM，模块化，复杂性.

形式化开发方法 VDM 是一种基于模型的方法[1~3]它包括形式规约和形式设计 2 部分. 其形式规约包括：(1)状态集(即状态空间)的定义(可能包括相关的不变式)；(2)初始状态的定义；(3)运算集的定义. 通常，状态空间由面向数学的数据类型，如集合、序列和映射等刻画. 而运算集的定义则是在描述状态空间的抽象模型的基础上，用前后断言方法来刻画每一运算的功能. 对运算而言，其功能规约的有效性由可满足性规则来保证. VDM 的形式设计包括数据精化和运算分解 2 种手段，数据精化主要用于将抽象的模型变换为较具体的模型而运算分解则用高级语言或支撑软件的基本原语来实现由前后断言刻画的运算的功能. 为了保证形式设计过程的正确性，VDM 采用 Adequency 规则、Domain 规则和 Result 规则来刻画数据精化的正确性. 并提供了各种运算分解规则来保证运算分解的正确性. 形式设计的基本过程如图 1. 其中 abstract model，reified model 和 concrete model 分别表示抽象模型、精化模型和具体模型.

分析 VDM 的形式设计过程，我们可以发现：

(1)通常，数据精化和运算分解分别进行且运算分解在数据精化之后；

(2)概念上看，数据精化、运算分解及其相应的证明涉及整个模型而不支持模型分解；实际上许多运算只存取状态空间的一部分；

(3)缺乏引入过程抽象的自然手段，这可从运算分

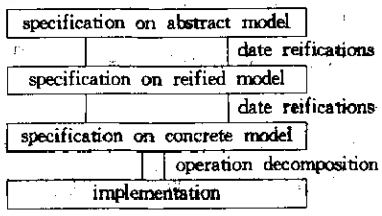

图1 VDM的形式设计过程

* 本文研究得到跨世纪优秀人才基金和国家攀登计划基金资助. 作者吕建，1960 年生，教授，主要研究领域为软件自动化，形式化方法. 张建莹，女，1963 年生，助工，主要研究领域为软件自动化.

本文通讯联系人：吕建，南京 210093，南京大学计算机软件研究所

本文 1995－10－09 收到修改稿

解的形式规则中看出.

尽管上述构架使用简单,易于理解和掌握,但是它不太适合于较大程序的开发,其主要问题在于:随着越来越多的实现细节被引入精化模型之中,整个规约变得越来越复杂,这主要表现在 2 个方面:①精化模型中的数据类型越来越细,从而导致复杂的运算规约;②涉及整个状态空间的不变式出现在所有的开发步中,从而使得开发过程的正确性证明变得十分困难.问题的症结在于 VDM 只支持数据精化而不支持数据分解,所以其开发手段在某种意义下是不完备的.从概念上讲,从软件规约到相应实现的开发包括精化和分解 2 个方面;精化支持从抽象到具体的变换,而分解支持从整体到部分的变换,尽管 VDM 提供了有效的精化手段,但在分解手段方面,它仅支持运算的分解而未提供模型分解的手段.

为了解决上述问题,本文基于模型分解、模块化和运算分解的概念,提出了数据分解思想,数据分解主要包括模型分解、子规约形成、规约重构和运算分裂 4 个步骤;采用数据分解方法,开发者可按如下方式将基于模型的规约分解成若干基于子模型的规约:(1)正确性证明只基于规约本身而无须涉及其内部实现细节;(2)每一子规约可独立开发.由于子规约只涉及部分状态,从而使得其中的不变式和运算规约较为简单,并且在子规约中的运算规约可自然地被处理成过程抽象,从而可方便地引入过程结构.将数据分解引入 VDM,我们可得到比 VDM 更加一般的形式化开发方法 DD—VDM.在 DD—VDM 中,数据分解和数据精化可任意夹插,并且可直接在子规约的基础上进行运算分解,为开发者提供了有效和灵活的开发手段.

1 形式化开发方法 DD—VDM

与 VDM 方法类似,DD—VDM 包括形式规约和形式设计;其形式规约与 VDM 中的形式规约相同.DD—VDM 与 VDM 的主要差别在于形式设计,DD—VDM 的形式设计中增加了数据分解手段,并且它可与数据精化和运算分解手段灵活组合.采用 DD—VDM 方法,从形式规约到相应实现的开发过程如下:从给定的形式规约开始,开发过程包括 2 个阶段:设计阶段和实现阶段.设计阶段由一系列自顶向下的设计步构成,其中每一设计步要么用数据分解手段将形式规约分解成若干子规约,要么用数据精化手段将抽象模型变换成较为具体的模型.此过程一直进行到所有的子模型均是可直接实现的具体模型为止.然后,实现阶段开始,实现阶段采用运算分解手段用高级语言结构实现由前后断言刻画的运算.整个开发过程如图 2 所示.

DD—VDM 的主要特征是引入了数据分解手段.数据分解主要包括以下步骤:

(1)模型分解:根据实际问题的需要,通过分析数据结构,将模型(包括不变式)分解成若干相对独立的子模型(包括子不变式);

(2)子规约形成:通过分析初始模型中的运算规约而得到与每一子模型相关的运算,将子模型和相应的运算组合在一起即可得到相应的子规约;值得注意的是,各子规约是独立的,相互间并无直接关系;

(3)规约重构:在子规约的基础上,将原有模型的规约加以重构;凡涉及 2 个或多个子模型的运算被保留在原模型中并加以重新规约,而其他的运算则被直接安排到相应的子规约中,在这一步骤的进行过程中,模块化的思想起着重要的作用.

(4)运算分裂:通过将子规约中运算处理成过程抽象,对重构后的运算规约进行分解,用子规约中的运算实现相应的运算规约.

现从文献[3]中抽取一简单实例 World 来说明上述思想.

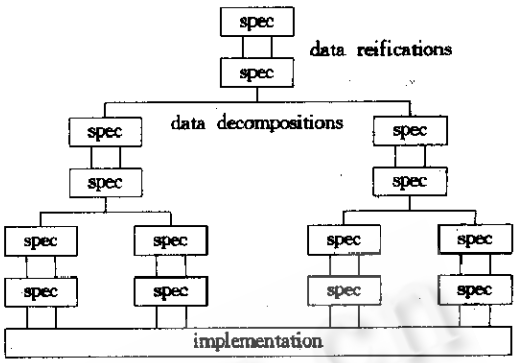

图2 DD—VDM开发过程

$world_1$: $singfem$: $Name$—set
　　$marfem$: $Name$—set
　　$singmale$: $Name$—set
　　$marmale$: $Name$—set
where
$inv-world(mk-world(sf,mf,sm,mm)) \triangleq is-disjoint-4(sf,mf,sm,mm)$
$is-disjoint-4$: $Name$—set × $Name$—set × $Name$—set × $Name$—set→B
$is-disjoint-4(s_1,s_2,s_3,s_4) \triangleq s_1 \cap (s_2 \cup s_3 \cup s_4) = \{\} \wedge s_2 \cap (s_1 \cup s_3 \cup s_4) = \{\}$
$\wedge s_3 \cap (s_1 \cup s_2 \cup s_4) = \{\} \wedge s_4 \cap (s_1 \cup s_2 \cup s_3) = \{\}$
$MARMALE$ () mm : $Name$—set
ext rd $marmale$: $Name$—set
post $mm = marmale$
$MARFEM$ () mf : $Name$—set
ext rd $marfem$: $Name$—set
post $mf = marfem$
$SINGMALE$ () sm : $Name$—set
ext rd $singmale$: $Name$—set
post $sm = singmale$
$SINGFEM$ () sf : $Name$—set
ext rd $singfem$: $Name$—set
post $sf = singfem$
$NEWFEM$ (f : $Name$)
ext wr $singfem$: $Name$—set
　rd $marfem$: $Name$—set
　rd $singmale$: $Name$—set
　rd $marmale$: $Name$—set
pre $f \notin (singfem \cup marfem \cup singmale \cup marmale)$
post $singfem = singfem' \cup \{f\}$
$NEWMALE$ (m : $Name$)
ext wr $singmale$: $Name$—set
　rd $marmale$: $Name$—set
　rd $singfem$: $Name$—set
　rd $marfem$: $Name$—set
pre $m \notin (singfem \cup marfem \cup singmale \cup marmale)$
post $singmale = singmale' \cup \{m\}$
$MARRIAGE$ (m : $Name$, f : $Name$)
ext wr $singmale$: $Name$—set
　wr $marmale$: $Name$—set
　wr $singfem$: $Name$—set
　wr $marfem$: $Name$—set
pre $m \in singmale \wedge f \in singfem$
post $marmale = marmale' \cup \{m\} \wedge marfem = marfem' \cup \{f\}$
　$\wedge singmale = singmale' - \{m\} \wedge singfem = singfem' - \{f\}$

第1步,将 $world$ 模型分解成2个子模型(包括不变式的分解):

$sub-model-1$:
　$male$: : $singmale$: $Name$—set
　　$marmale$: $Name$—set

where

$inv-male(mk-male(sm,mm)) \triangleq is-disjoint-2(sm, mm)$

$is-disjoint-2 : Name-set \times Name-set \rightarrow B$

$is-disjoint-2(s_1,s_2) \triangleq s_1 \cap s_2 = \{\}$

sub-model-2 :

female :: *singfem* : *Name*-set

marfem : *Name*-set

where

$inv-female(mk-female(sf,mf)) \triangleq is-disjoint-2(sf, mf)$

第 2 步,分析初始规约中的运算规约以获取子运算以及相应的规约;可以看出运算 *MARMALE*,*SINGMALE* 仅涉及子模型 *male*,而运算 *MARFEM*,*SINGFEM* 仅涉及子模型 *female*,虽然从规约的观点,运算 *NEWMALE* 和 *NEWFEM* 既涉及 *male* 又涉及 *female*,但从实现的角度,*NEWMALE* 仅涉及 *male* 而 *NEWFEM* 仅涉及 *female*,从而在子模型 *female* 中,运算 *NEWFEM* 的规约变为如下形式:

NEWFEM (*f* : *Name*)

ext wr *singfem* : *Name*-set

rd *marfem* : *Name*-set

pre $f \notin (singfem \cup marmale)$

post $singfem = singfem' \cup \{f\}$

在子模型 *male* 中,运算 *NEWMALE* 的规约如下:

NEWMALE (*m* : *Name*)

ext wr *singmale* : *Name*-set

rd *marmale* : *Name*-set

pre $m \in (singmale \cup singmale)$

post $singmale = singmale' \cup \{m\}$

根据 *MARRIAGE* 的运算规约,可产生如下 2 个子运算规约:

MARRIAGEFEM (*f* : *Name*)

ext wr *singfem* : *Name*-set

wr *marfem* : *Name*-set

pre $f \in singfem$

post $marfem = marfem' \cup \{f\} \wedge singfem = singfem' - \{f\}$

MARRIAGEMALE (*m* : *Name*)

ext wr *singmale* : *Name*-set

wr *marmale* : *Name*-set

pre $m \in singmale$

post $marmale = marmale' \cup \{m\} \wedge singmale = singmale' - \{m\}$

第 3 步,可形成如下 2 个子规约:

sub-specification-1 (*Male*):

sub-model-1

MARMALE

SINGMALE

NEWMALE

sub-specification-2 (*Female*):

sub-model-2

MARFEM

SINGFEM

NEWFEM

第 4 步,基于以上 2 个子规约,原来的规约可重构如下:

world :: *female* : *Female*

male : *Male*

where

```
  inv—new—world (mk—world(f1, m1)) △
let f1 = mk—female(sm, mm) ∧ m1 = mk—male(sf, mf) in
      is—disjoint—2(sm ∪ mm, sf ∪ mf)
MARRIAGE (m : Name, f : Name)
ext wr female : Female
    wr male : Male
pre pre—MARRIAGEFEM(f, female) ∧ pre—MARRIAGEMALE(m, male)
post post—MARRIAGEFEM(f, female', female) ∧ post—MARRIAGEMALE (m, male', male)
```

可以看出,*VDM* 中提出的"运算摘引"("Operation Quotation")在运算规约的重构过程中起着重要的作用,这是由于重构的运算规约常常可由子规约中的谓词直接定义.从规约的角度,此方法提供了由较小规约组成较大规约的手段;而从实现的角度,也可为运算分裂提供某些启发信息.

最后,基于子规约 *Male* 和 *Female* 中的运算,运算 *MARRIAGE* 可分解如下:

```
MARRAGE(m : Name, f : Name)
  MARRIAGEFEM(f); MARRIAGEMALE(m)
end
```

值得注意的是,重构的规约只是初始规约的精化.两者之间并不等价.例如,可使用运算 *NEWMALE* 和 *NEWFEM* 引入同一人,既作为 *male* 又作为 *female*.但是这并不影响使用.形式规约的用户所面临的接口只是原来的规约.

经过上述步骤,初始的规约 *World* 可分解成2个较为简单的规约 *Male* 和 *Female*.实际上,子规约 *Male* 和 *Female* 可进一步分解成更小的子规约,例如,*Male* 可分解成2个更小的子规约 *Singmale* 和 *Marmale*.

```
sub—specification—11 : (Singmale)
    singmale = Name—set
            SINGMALE () sm : Name—set
            ext rd singmale : Name—set
            post sm=singmale
            CHGSINGMALE (sm : Name)
            ext wr singmale : Name—set
            pre sm ∈ singmale
            post singmale = singmale' — {sm}
sub—specification—12 : (Marmale)
    marmale = Name—set
            MARMALE () mm : Name—set
            ext rd marmale : Name—set
            post mm = marmale
            CHGMARMALE (mm : Name)
            ext wr marmale : Name—set
            pre mm ∈ marmale
            post marmale = marmale' ∪ {mm}
```

然后,*Male* 可重构如下:

```
male :: singmale : Singmale
        marmale : Marmale
where
inv—new—male(mk—male(sm, mm)) △ is—disjoint—2(sm, mm)
MARRIAGEMALE (m : Name)
ext wr marmale : Name—set
    wr singmale : Name—set
pre pre—CHGSINGMALE(m, singmale)
post post —CHGMARMALE(m, singmale', marmale)
```

$\wedge$ *post*−*CHGSINGMALE*(*m*, *singmale*′, *singmale*)

最后,运算 *MARRIAGEMALE* 可分解如下:

MARRIAGEMALE(*m*, *Name*)
CHGSINGMALE(*m*); *CHGMARMALE*(*m*)
end.

对于 *Female* 可类似处理.

当然,并不是所有的模型分解均是有用的.一个模型的分解是否有效,取决于具体的应用.一般地,我们希望每一子规约尽可能的独立和完整,也就是说,每一子规约应表示一个固有的概念并具有良好性质;不变式和运算应尽可能的简单.

此外,还使用 *DD*−*VDM* 方法对文献[2]中的其他实例,如 *Spell*−*checker* 等进行了使用.结果表明,*DD*−*VDM* 与 *VDM* 相比有以下特征:

1) *DD*−*VDM* 提供了 2 种设计步,即数据精化和数据分解供开发者灵活使用.而 *VDM* 中只提供了数据精化这一种设计步;

2) 由于数据精化和数据分解可任意夹插,从而可在某些数据精化步之前完成运算分解,开发者不必严守 *VDM* 中"数据精化在前,运算分解在后"的次序;

3) 可在子模型上完成数据精化和运算分解,从而使得相应的正确性证明过程变得容易.而在 *VDM* 中,相应的步骤只能对整个模型进行;

4) 由于引入了过程抽象的手段,从而可在运算分解过程中自然的引入过程结构;而 *VDM* 却无法做到这一点.

当然,要使 *DD*−*VDM* 是可行的,必须给出数据分解的形式规则,这是下节的内容.

2 数据分解的形式规则

从研究形式规则的角度,数据分解包括数据模块化和运算分裂,其中数据模块化由模型分解、子规约形成和规约重构组成.对于数据模块化而言,建立其形式规则的主要问题是,首先确定模型和相应的若干子模型之间的关系,然后给出规则以保证重构规约中的运算规约满足初始规约中相应的运算规约;而对运算分裂而言,首先应搞清运算分裂与运算分解之间的主要差别,然后给出相应规则对之作出相应处理.

如何保证模型分解的正确性呢?从状态空间的角度,由初始模型构成的状态空间和由相应的若干子模型所构成的子状态空间的组合应该是一致的,也就是说,可用双射来刻画它们之间的关系并反映分解的正确性.更精确地,设有规约(X : *TYPE*, *INV* : *TYPE* $\to B$, *OPERATION*−*SET*),其相应的状态空间可定义为 $\Sigma=\{e \mid e \in TYPE \wedge inv(e)\}$.现设 Σ 表示初始的状态空间, $\Sigma_1,\cdots,\Sigma_n$ 分别表示相应的子状态空间.这些子状态空间可由如下不变式加以限定:

$$new\text{−}inv : \Sigma_1\times\cdots\times\Sigma_N \to B$$

基于上述定义,所谓模型分解是正确的,是指能够找到一组函数 *comp*(或分解函数 *decomp*):

$$comp : \Sigma_1\times\cdots\times\Sigma_n \mid new\text{−}inv \to \Sigma \ (or\ decomp : \Sigma \to \Sigma_1\times\cdots\times\Sigma_n \mid new\text{−}inv)$$

它满足:

1) $comp\ (i\text{−}r_1,\cdots,i\text{−}r_n)=i\text{−}r$ (or $decomp(i\text{−}r)=(i\text{−}r_1,\cdots,i\text{−}r_n)$)

2) *comp* 是双射(or *decomp* 是双射),其中

① $\Sigma_1\times\cdots\times\Sigma_n|new-inv=\{(r_1,\cdots,r_n)|(r_1,\cdots,r_n)\in\Sigma_1\times\cdots\times\Sigma_n\wedge new-inv(r_1,\cdots,r_n)\}$

② $i-r$, $i-r_j(1\leqslant j\leqslant n)$ 表示相应的初始状态;

例如,要证明 *world* 中的模型分解是正确的,首先需要找一复合函数 *comp* 或分解函数 *decomp*;对此例,可定义如下分解函数 *decomp*:

$decomp:\Sigma\to\Sigma_1\times\Sigma_2|\ inv-new-world$

$decomp(mk-world(sm,mm,sf,mf))=(mk-female(sf,mf),mk-male(sm,mm))$

其次,可证明以下两式成立:

1) *decomp* 是一对一映射,即

$\forall\ (mk-world(sm,mm,sf,mf),mk-world(sm',mm',sf',mf'))\in\Sigma.$

$(mk-world(sm,mm,sf,mf)\neq mk-world(sm',mm',sf',mf')$

$\Rightarrow(decomp(mk-world(sm,mm,sf,mf))\neq decomp(mk-world(sm',mm',sf',mf'))$

2) *decomp* 是从 Σ 到 $\Sigma_1\times\Sigma_2$ 上的映射,即

$\forall\ (mk-female(sf,mf),mk-male(sm,mm))\in\Sigma_1\times\Sigma_2.\exists\ mk-world(sm1,mm1,sf1,mf1)\in\Sigma.$

$decomp(mk-world(sm1,mm1,sf1,mf1))=(mk-female(sf,mf),mk-male(sm,mm))$

对于重构规约中的每一新的运算规约,可采用如下规则保证其正确性:

1) $\forall\ (r_1,\cdots,r_n)\in\Sigma_1\times\cdots\times\Sigma_n|new-inv.\ pre(comp(r_1,\cdots,r_n))\Rightarrow pre-new(r_1,\cdots,r_n)$

(or $\forall\ r\in\Sigma.\ pre(r)\Rightarrow pre-new(decomp(r))$)

2) $\forall\ (r'_1,\cdots,r'_n),(r_1,\cdots,r_n)\in\Sigma_1\times\cdots\times\Sigma_n|new-inv.\ pre(comp(r_1,\cdots,r_n))$

$\wedge post-new((r'_1,\cdots,r'_n),(r_1,\cdots,r_n))\Rightarrow post(comp(r'_1,\cdots,r'_n),comp(r'_1,\cdots,r_n))$

(or $\forall\ r',r\in\Sigma.\ pre(r)\wedge post-new(decomp(r'),decomp(r))\Rightarrow post(r',r)$)

对于运算分裂,首先引入 2 个符号 *MOD* 和 *OUTPUT* 来刻画运算的不相干性,符号 *MOD* 用来描述运算或谓词所涉及的规约或子规约的名.例如:

$MOD(MARRIAGE)=\{World,Female,Male,Singmale,Marmale,Singfem,Marfem\}$

$MOD(inv-world)=\{World,Female,Male,Singmale,Marmale,Singfem,Marfem\}$

$MOD(MARRIAGEMALE)=\{Male,Singmale,Marmale\}$

$MOD(post-MARRIAGEFEM)=\{Female,Singfem,Marfem\}$

而符号 *OUTPUT* 则用于描述运算所涉及的实际参数.

运算分解和运算分裂的主要区别在于:

1) 运算分裂中的基本原语是过程抽象,它们面向子模型,由输入输出断言刻画;因此,对于子规约中的每一运算 $\{PRE\}\ op\ \{POST\}$ 可定义如下的实例化规则:

$$DD-bop-1:\frac{\{PRE\}\ OP\ \{POST\}}{\{PRE\}\ OP\ \{POST\}\ |\ (A/F)}$$

其中 $\{PRE\}\ OP\ \{POST\}\ |\ (A/F)$ 表示在 $\{PRE\}\ OP\ \{POST\}$ 中用实参 A 替换形参 F 后所得的结果. $\{PRE\}\ OP\ \{POST\}\ |\ (A/F)$ 被称为实例化的运算规约,它给出了运算分裂的基本原语.

对于所谓的“框架问题”，可用如下规则来刻画：

$$DD-fra-1:\frac{\{PRE\}\ OP\ \{POST\},\ MOD(PRED)\cap MOD(OP)=\{\}}{\{PRE\wedge PRED\}\ OP\ \{POST\wedge PRED\}}$$

对于特例，我们有 $DD-fra-2$

$$DD-fra-2:\frac{\{PRE\}\ OP\ \{POST\},MOD(X)\cap MOD(OP)=\{\}}{\{PRE\}\ OP\ \{POST\wedge X=X'\}}$$

2) *VDM* 中的所有形式规则均基于整个状态空间. 而在 *DD−VDM* 中，运算分裂主要基于各个独立的子模型. 因此可引入若干规则将子模型的性质方便地组合成整个模型的性质. 2 种常用的组合形式是逻辑与和逻辑或. 逻辑与可由顺序组合结构得到；逻辑或可由条件结构得到. 相应的规则可定义如下：

$$OUTPUT(OP1)\cap OUTPUT(OP2)=\{\}$$

$$MOD(OP1)\cap MOD(OP2)=\{\}$$

$$DD-sequ-1:\frac{\{PRE1\}OP1\{POST1\};\ \{PRE2\}OP2\{POST2\}}{\{PRE1\wedge PRE2\}OP1;OP2\{POST1\wedge POST2\}}$$

$$DD-cond-1:\frac{\{PRE1\wedge TEST\}\ OP1\{POST1\};\ \{PRE2\wedge\neg TEST\}\ OP2\{POST2\}}{\{PRE1\wedge PRE2\}\ \text{if } TEST \text{ then } OP1 \text{ else } OP2\ \{POST1\vee POST2\}}$$

例如，*MARRIAGE* 的正确性可证明如下：

首先，由 $DD-bop-1$ 规则可得：

$$\{\text{pre}-MARRIAGEFEM(f,\ female)\}MARRIAGEFEM(f)$$
$$\{\text{post}-MARRIAGEFEM(f,female',female)\}$$
$$\{\text{pre}-MARRIAGEMALE(m,\ male)\}MARRIAGEMALE(m)$$
$$\{\text{post}-MARRIAGEMAL\exists\ (m,male',male)\}$$

然后，由于

$$MOD(MARRIAGEFEM)\cap MOD(MARRIAGEMALE)=\{\}$$
$$OUTPUT(MARRIAGEFEM)\cap OUTPUT(MARRIAGEMALE)=\{\}$$

使用 $DD-sequ-1$ 规则可得：

$$\{\text{pre}-MARRIAGEFEM(f,female)\wedge\text{pre}-MARRIAGEMALE(m,male)\}$$
$$MARRIAGEFEM(f);MARRIAGEMALE(m)$$
$$\{\text{post}-MARRIAGEFEM(f,female',female)\wedge\text{post}-MARRIAGEMALE(m,male',male)\}$$

因此对 *MARRIAGE* 所做的运算分裂是正确的.

归结起来，只要将子模型中的运算看作是整个模型上的运算，即可将 *VDM* 中关于运算分解的规则直接用于 *DD−VDM* 中的运算分裂. 上述规则主要涉及高级基本原语的引入和子模型性质的组合.

3 结 语

DD−VDM 为数据精化、数据分解和运算分解的灵活应用提供了统一的构架，其层次性使之较为适合较大程序的开发. 进一步的工作包括：(1)为子模型的选取、运算的分裂等提供系统的方法；(2)在具体的软件工程实践中使用该方法，发现问题，进一步的改进、完善和提高；(3)基于文献[4～7]，DD−VDM 也为并行面向对象程序的开发提供了可能性. 这部

分内容将另文讨论.

致谢　本项研究工作是在Cliff. B. Jones教授(The University of Manchester, U. K.)指导下完成的,T. Clement提供了有益的建议.此外,作者也得到了徐家福教授的大力支持,在此一并向他们表示衷心的感谢.

参考文献

1 Jones C B. Software development. A rigorous approach. London: Prentice—Hall International, 1980.

2 Jones C B. Systematic software development using VDM, 2th ed. London: Prentice—Hall International, 1990.

3 Jones C B, Shaw R C F. Case studies in systematic software development. London: Prentice—Hall International, 1990.

4 Jones C B. Development methods for computer programs including a notion of interference. Dissertation, Oxford University, 1981.

5 Jones C B. Constraining interference in an object—oriented design method. In: Lecture Notes in Computer Science, Springer—Verlag, 1993, **668**: 136～150.

6 Jones C B. A pi—calculus semantics for an object—based design method. In: Lecture Notes in Computer Science, Springer—Verlag, 1993, **715**: 158～172.

7 Jones C B. Reasoning about interference in an object—based design method. In: Lecture Notes in Computer Science, Springer—Verlag, 1993, **670**: 1～18.

A FORMAL SOFTWARE DEVELOPMENT METHOD DD—VDM

Lü Jian　Zhang Jianying

(*Institute of Computer Software　Nanjing University　Nanjing* 210093)

(*State Key Laboratory for Novel Software Technology　Nanjing University　Nanjing* 210093)

Abstract　After pointing out the main shortcomings of VDM in developing larger programs, this paper introduces a new concept—DD(date decomposition) which is based on the ideas of model split, modularisation and operation decomposition, and combines it with VDM to form a more general formal development method DD—VDM. As a result, a more flexible development strategy can be adopted and the development complexity can be effectively controlled.

Key words　Formal method, data decomposition, VDM, modularisation, complexity.