一种从面向对象Z规约到代码的精化演算方法*

王云峰[1,2] 庞军[1] 查鸣[1] 杨朝晖[1] 郑国梁[1]

[1](南京大学计算机软件新技术国家重点实验室 南京 210093)

[2](解放军理工大学气象学院 南京 211101)

E-mail: wangyf@seg.nju.edu.cn

摘要 COOZ(complete object-oriented Z)的优势在于精确描述大型程序的规约.COOZ本身的结构不支持精化演算,这限制了COOZ的应用能力,使COOZ难以作为完整的方法应用于软件的开发.将精化演算引入COOZ,弥补了COOZ在设计和实现阶段的不足,同时也消除了规约与实现之间在结构和表示方法上的完全分离,使程序开发在一个完整的框架下平滑进行.该文提出了基于COOZ和精化演算的软件开发模型,通过实例讨论了数据精化和操作精化问题.在精化演算实现技术方面构造了一种数据精化算子,提出一种基于数据精化演算和程序窗口推理的数据精化的方法.

关键词 形式化开发方法,精化演算,形式规约,面向对象.

中图法分类号 TP311

形式方法与面向对象方法相结合已成为软件开发方法研究的一个重要方向.90年代初,将形式规约语言Z进行面向对象的扩充成为研究热点,先后产生了若干个Z的面向对象扩充版本[1].COOZ(complete object-oriented Z)[2]是在分析以往Z的面向对象扩充的基础上,采用更为先进、合理的技术对Z进行面向对象的扩充.

COOZ的优势在于精确描述大型程序的规约.COOZ本身的结构不支持精化演算,需要用证明的方法进行程序精化.这对于面向对象规约而言是极其困难的,特别是大型的复杂系统尤其如此.所以,COOZ不适用软件开发的后期,如软件的设计与实现.这限制了COOZ的应用能力,不能充分体现出面向对象规约技术在设计与实现阶段的优越性,使COOZ难以作为完整的方法应用于软件的开发.将精化演算引入COOZ,弥补了COOZ在设计和实现阶段的不足,同时也消除了规约与实现之间在结构和表示方法上的完全分离,使程序开发在一个完整的框架下平滑进行.

将精化演算与COOZ结合,建立一个面向对象的精化演算框架.在这一框架下,可以采用COOZ的面向对象形式规约技术描述软件的规约,然后采用精化演算的方法进行程序精化.程序精化是将程序规约转换成可执行代码的过程.一般将其分为数据精化和算法精化[3]两种形式.数据精化是把抽象的数据结构转换为可以高效实现的形式,算法精化是将程序逐步转换为更加便于实现的形式,直至代码.

我们提出了基于COOZ和精化演算的软件开发模型,通过实例讨论了数据精化和操作精化.另外,我们还构造了一种数据精化算子,提出了一种基于数据精化演算和程序窗口推理的数据精化的方法,作为实现支持精化演算工具的技术基础.

1 基本概念

1.1 COOZ

在吸取现有各种Z面向对象扩充的长处、弥补其不足的基础上,我们设计了能充分支持面向对象机制、表达

* 本文研究得到国家自然科学基金(No.69673006)和国家"九五"重点科技攻关项目基金(No.98-780-01-07-06)资助.作者王云峰,1964年生,博士生,主要研究领域为形式化方法,面向对象技术.庞军,1976年生,硕士生,主要研究领域为形式化方法,精化证明工具.查鸣,1975年生,硕士生,主要研究领域为形式化方法,精化证明工具.杨朝晖,1976年生,硕士生,主要研究领域为形式化方法,定理证明工具.郑国梁,1937年生,教授,博士生导师,主要研究领域为软件工程.

本文通讯联系人:王云峰,南京 210093,南京大学计算机科学与技术系

本文2000-01-17收到原稿,2000-04-03收到修改稿

能力强的Z扩充版本COOZ,给出其完整的BNF语法定义和扩充部分的形式语义.类模式由类名、类参数、父类列表、对象接口、局部定义、状态模式、初始化模式、方法模式以及表示对象实时和历史约束的类不变式组成,如图1所示,其详细语法定义参见文献[2].

例如,描述一个班级学生的类ClassStudent,该班级学生分为两种:通过考试和未通过考试的学生,分别用变量y,n表示,为了以后讨论方便,仅给出简化的类模式,如图2所示.其中,仅给出学生注册的方法模式Enroll_OK,其他的方法模式省略.

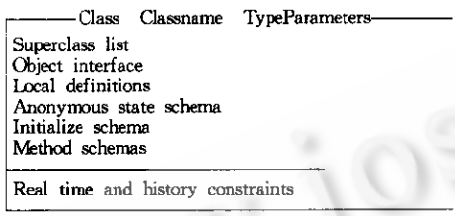

Fig. 1 Class schema in COOZ
图1 类模式COOZ

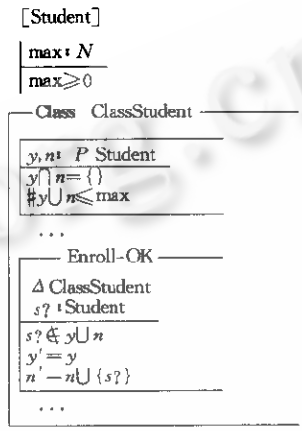

Fig. 2 Class schema ClassStudent
图2 类模式ClassStudent

1.2 精化演算

精化演算扩充了Dijkstra的卫式命令语言,在可执行的结构中加入规约语句(specification statement).规约和执行程序集成在一种语言里,可以通过在这一语言内的一系列转换来开发程序,对程序的平滑开发起到了重要作用.在这一开发过程中,初始程序是一种典型的规约语句,最终的程序只包含可执行代码,而开发过程中的程序是两种结构的混合体.

• 规约语句

规约语句是精化演算中一种重要的程序表示方法.规约语句提供了一种方便的方法,将抽象规约嵌入程序.通常,程序S规约写为:$pre \Rightarrow [S]post$,规约语句将其表示为$x:[pre, post]$,其中x表示在程序中发生改变的变量列表,其他的变量在程序中不发生变化.规约语句的语义定义如下:

$$[x:[pre, post]]P == pre \wedge (\forall x' \cdot post \Rightarrow P).$$

下面,给出一个简单程序的规约语句及其精化:

$$x:[\text{true}, x = \max(a,b)]$$
$$\sqsubseteq (\text{if } a \geq b \rightarrow x := a \,[]\, b \geq a \rightarrow x := b \text{ fi}.$$

其中[]表示精化关系.

• 数据精化

数据精化可视为程序精化的一个特例:程序中的抽象局部变量被另一组具体变量所替换,而程序的结构尽可能不变.下面是据此给出的数据精化的形式定义:

如果sim属于simulation,S'数据精化S(记为:$S <= S'$)当且仅当

$$S; sim \sqsubseteq sim; S'.$$

同样,如果sim属于co-simulation,S'数据精化S(记为:$S <= S'$)当且仅当

$$sim; S \sqsubseteq S'; sim.$$

实际上,对状态转换程序加上某些约束,可以保证数据精化关系在不同的程序结构中满足分配律,在数据精化过程中维持抽象程序的结构不变.例如:

$$a,g:[pre,post]<=c,g:[sim\ pre,sim\ post].$$

式中 a,c 分别表示抽象和具体变量,g 表示全球变量.该式表示可以从抽象规约中演算出具体规约.Co-simulation一般定义为:对任意谓词 P,

$$sim\ P==(\exists\ a\cdot R\wedge P).$$

其中 R 是抽象变量和具体变量之间的关系.对于 simulation,sim 一般定义为

$$sim\ P==\forall\ a\cdot R\Rightarrow P.$$

1.3 程序窗口推理

窗口推理技术是由 Robinson 和 Staples 等人提出来的,用于机器定理证明的一种证明转换技术.原来的焦点是$\Pi[F]$,它通过转换 F 实现自身的转换.其中被选作子窗口焦点的部分加方框,如$\boxed{F}$.子窗口的假设条件 Γ' 依赖于上下文 $\Pi[]$和原来窗口的假设条件 Γ.通常,Γ' 是强于 Γ 的假设.同时,假定子窗口保持新的前序关系$\sqsubseteq'$,则原来窗口保持前序关系$\sqsubseteq$.当聚焦原来焦点的某一部分时,可以自动形成窗口开启规则.程序窗口推理系统是窗口推理系统的一种扩充[4],可以显式地处理程序的上下文,更好地支持自动精化.程序窗口推理系统可以处理程序变量,应用精化规则,管理精化过程中产生的各种上下文环境信息.下面主要讨论程序窗口推理系统的前置条件处理.

考虑如下选择命令的第 1 个分支精化的窗口开启规则:

$$\frac{\Gamma\ \models\ \boxed{S}\ \sqsubseteq\ S'}{\Gamma\ \models\ \text{if } B\rightarrow\boxed{S}\,[]\,C\rightarrow T\text{ fi}\sqsubseteq\text{if } B\rightarrow S'\,[]\,C\rightarrow T\text{ fi}}$$

以及下面的选择命令:

$$\{P\}\text{ if } B\rightarrow S\,[]\,C\rightarrow T\text{ fi}.$$

在 S 的精化过程中,希望能够利用前置条件 P 以及选择条件 B,如果使用上面的窗口开启规则精化 S,无法用到 P 和 B.为此,前置条件需要显式给出,即把上式改写为

$$\{P\}\text{ if } B\rightarrow\{P\wedge B\}S\,[]\,C\rightarrow\{P\wedge C\}T\text{ fi}.$$

这样表示出现重复信息,尤其是当 P 较为复杂时,上式将极为繁琐.为此,才引入一种新型窗口:程序窗口,它把前置条件和通常的假设条件放在一起.例如,上述窗口开启规则表示为

$$\frac{\Gamma,pre\ P\wedge B\ \models\ \boxed{S}\ \sqsubseteq S'}{\begin{array}{l}\Gamma\models\text{ if } B\rightarrow\boxed{S}\,[]\,C\rightarrow T\text{ fi}\\ pre\ P\qquad\sqsubseteq\text{if } B\rightarrow S'\,[]\,C\rightarrow T\text{ fi}.\end{array}},$$

在这一规则中,选择命令的前置条件 $pre\ P$ 自动加上选择条件 B 作为 S 精化的前置条件.标识符 pre(和下面的 inv 及 $lval$ 一样)作为程序窗口的语法中的关键字.其结合性要强于"⊨="和";",但却弱于逻辑连接符.例如,$pre\ P\wedge B$ 实际上表示 $pre(P\wedge B)$.

2 软件开发模型

我们从用 COOZ 书写的功能规约出发,进行进一步的细化,根据对象类型的规约进行类的设计.通过类的设计,即得到符合原功能规约的设计规约,设计规约经过类中状态模式和方法模式的逐步精化,逐步降低其抽象级,每一次得到更为具体、更易于实现的规约.

2.1 开发模型

这一开发模型是以 COOZ 为主的开发模型,如图 3 所示.

首先用 COOZ 形式描述软件的规约,然后将操作模式转换为精化演算中的规约语句,在对状态模式进行精化后,根据精化演算中的数据精化演算出精化后的操作.根据精化演算中的精化规则,对规约语句进行操作精化,得到 COOZ 框架下的抽象程序.最后,利用系统转换程序将抽象程序转换为某种编程语言的程序代码,如 JAVA,C++等.

2.2 数据精化

文献[5]中提出,将规约和程序代码都看作是程序,从而扩展了程序的定义.在程序中用规约语句表示待求

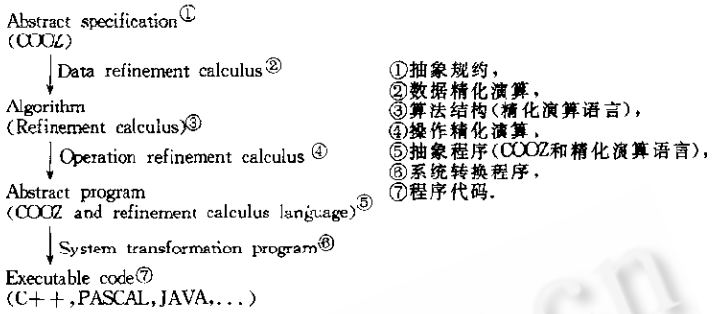

Fig. 3　Develop model 1
图3　开发模型1

精的不可执行部分,而将程序中的可执行语句称为代码.我们进一步允许程序中使用COOZ中的数学数据类型,包含规约语句和数学数据类型的程序称为抽象程序.这样,将规约和程序的概念统一起来,规约的求精过程即是从抽象程序到代码的转换过程.规约语句由前置条件、后置条件和可修改的变量列表组成,为了描述上的方便,还可以定义一些局部常量和变量.它和COOZ中的方法模式有着直接的对应关系,如图4所示,方法模式中由$\Delta Id\text{-}list$指出可修改的状态量,在谓词部分用横线将不同的前置条件处理分开,用关键字if将方法的前置条件和后置条件分开.因而可以通过辅助工具将COOZ的方法模式自动转换成相应的规约语句.为与COOZ中的方法模式相对应,我们对规约语句作了相应的修改:针对不同的前置条件分别给出其相应的后置条件,因而,一个规约语句中允许有多对$[pre,post]$.

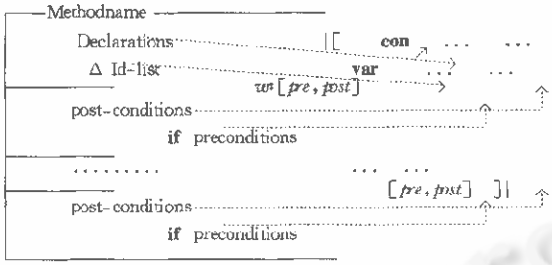

Fig. 4　The corespondent relation of specification statement and operation schema
图4　规约语句与方法模式的对应关系

例如,图2中的方法模式Enroll_OK可以转换为规约语句:

$$n:[s?\notin y\cup n, n=n0\cup\{s?\}]. \tag{1}$$

其中$n0$表示对变量n的操作前变量值的引用.下面分析对图2类模式的数据精化.

用两个数组分别描述学生和学生是否通过考试的状态,用一个整数变量作为计数,表示数组使用的情况.在Z中,可以用全函数来描述数组,该全函数的定义域为一个索引集合{1..max},则类模式ClassStudent精化为ClassStudent_1,其无名状态模式如图5所示.

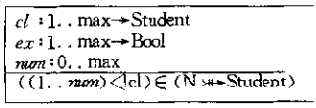

Fig. 5　State schems of ClassStudent
图5　类ClassStudent的状态

其中状态不变式表示cl数组中前num个元素不能有重复的,即不允许已经注册的学生重复注册.精化前后的状态模式可以用谓词R描述:

$$R\triangleq(y=\{i:1..num\mid(ex\ i)=\text{true}\cdot(cl\ i)\})\wedge\{i:1..num\mid(e\hat{x}\ i)=\text{false}\cdot(cl\ i)\}).$$

根据数据精化演算:$a:[pre,post]\prec c:[Sim\ pre, Sim\ post]$,可以由式(1)及$R$演算得到基于如图5所示状态模式

的操作.这里,数据精化演算中的 Sim 定义为:$Sim\ Q \triangleq \exists\ a \cdot R \wedge Q$.

$$n:[s? \notin y \cup n, n = n0 \cup \{s?\}] \prec cl, ex, num:[\exists\ n \cdot R \wedge s? \notin y \cup n, \exists\ n \cdot R \wedge n = n0 \cup \{s?\}] =$$
$$cl, ex, num:[(num < max) \wedge (s? \notin \{1..num \cdot cl\ i\}), (num = num0 + 1) \wedge$$
$$(cl = cl0 \oplus (num \mapsto s?)) \wedge ((ex = ex0 \oplus ((num \mapsto \text{false}))]. \quad (2)$$

2.3 操作精化

对于面向对象的形式规约,操作精化有两方面的内容:

(1) 基于精化演算,按照精化规则进行操作,从类方法的规约逐步精化得到代码.

(2) 基于面向对象规约技术,根据子类型的定义和构造方法,通过继承实现操作精化.

这两种方法都实现了对类的精化,但是前者是关于软件实现的精化,而后者一般仅仅局限于软件的规约.在COOZ规约中引入精化演算,可以将两种操作精化结合起来,不再严格区分软件的规约与实现.

例如,对于数据精化后的操作模式 Enroll_OK_1,即相应的规约语句式(2),根据精化演算规则,容易得到如下的程序代码:

(2)式 $\sqsubseteq num, cl[num+1], ex[num+1] = num+1, s, \text{false}$

$\sqsubseteq num = num + 1;$

$cl[num+1] = s$

$ex[num+1] = \text{false}.$

3 程序窗口推理中的数据精化

在对程序进行部分精化时,其上下文信息是非常重要的.前文介绍的窗口推理和程序窗口推理提供了处理这种上下文信息的方法,但均未包含数据精化.由于数据精化需要针对更大的程序块,所以它比一般的算法精化更加复杂.

在这一节里,我们将提出一种利用程序窗口推理进行数据精化的方法.

3.1 程序块

由于局部变量的作用范围是一块语句,所以数据精化可以描述成块语句的转换.故数据精化可以看作是算法精化的特例;精化仅限于块语句.

块语句表示为

$$|[\text{var } a \mid \text{Init} \cdot P\ ]|, \quad (3)$$

此即为数据精化的对象,其中 var a 为引入程序的局部变量,根据谓词 Init 对其进行初始化.块语句的语义以最弱前置条件定义:对于任意一个不包含自由变量 a 的谓词 φ,

$$|[\square(|[\text{var } a \mid \text{Init} \cdot P]|)]|\varphi \triangleq \forall\ a \cdot \text{Init} \Rightarrow |[P|]\varphi. \quad (4)$$

3.2 数据精化算子

在对块语句进行数据精化时,将抽象和具体变量的关系表示为抽象不变式 AI,对于谓词和程序分别引入精化算子 D_{AI} 和 D_{AI},将块语句的数据精化表示为

$$|[\text{var } a \mid \text{Init} \cdot P]| \sqsubseteq |[\text{var } c \mid D_{AI}(\text{Init}) \cdot D_{AI}(P)]|. \quad (5)$$

任意给定一谓词 φ,D_{AI} 及其对偶算子 $D*_{AI}$ 定义为

$$D_{AI}(\varphi) \triangleq \exists\ a \cdot AI \wedge \varphi, \quad (6)$$

$$D*_{AI}(\varphi) \triangleq \forall\ a \cdot AI \Rightarrow \varphi. \quad (7)$$

根据第 2.1 节以及数据精化的谓词转换子的"对偶和邻接[6]"概念,我们有

$$P \prec P' \text{ if } sim; P; sim* \sqsubseteq P'. \quad (8)$$

这样,可以将 $(sim;P;sim*)$ 看作是"最小"(最抽象)精化,则对于语句 P 有

$$sim;P;sim* \sqsubseteq D_{AI}(P). \quad (9)$$

3.3 与算法精化的比较

本文将数据精化当作算法精化的特例来处理,但是二者的窗口开启规则和窗口转换规则还是有所不同的.

3.3.1 窗口开启规则

对于数据精化,仅有一条针对变量声明的块语句的窗口开启规则.其焦点是针对块语句中所有的命令,而非其中的部分命令.由于程序不能同时处于抽象和具体状态,所以对于抽象变量的替换必须一步完成.当焦点移进块语句时,抽象不变式 AI 和由其他转换得到的谓词,如 D_{AI}都归结到新窗口的上下文信息中.

3.3.2 窗口转换规则

对于算法精化,焦点转换的根据是 Morgan[5]的精化演算规则,将程序语句精化成不同的结构(如迭代、选择、循环等结构).而对于数据精化,焦点转换一般不改变命令的结构,它只是用数据精化算子从基于较抽象的数据结构的命令演算出基于较为具体的数据结构的命令.

4 结 语

将精化演算与 COOZ 结合,建立一个面向对象的精化演算框架.在这一框架下,可以采用 COOZ 的面向对象形式规约技术描述软件的规约,然后采用精化演算的方法进行程序精化.程序精化是将程序规约转换成可执行代码的过程.一般将其分为数据精化和算法精化.数据精化是把抽象的数据结构转换为可以高效实现的形式,算法精化是将程序逐步转换为更加便于实现的形式,直至代码.

我们提出了基于 COOZ 和精化演算的软件开发模型,通过实例讨论了数据精化和操作精化问题.另外,构造了一种数据精化算子,提出了一种基于数据精化演算和程序窗口推理的数据精化方法,作为实现支持精化演算工具的技术基础.由于操作精化演算的研究已经较为成熟,本文未作详细讨论,相关内容可参见文献[5].许多具体技术问题尚未研究,尤其是针对该软件开发方法的支撑工具的研究,这是目前我们正在进行的主要工作.

参考文献

1 Stepney S, Barden R, Cooper D. Object Orientation in Z. London: Springer-Verlag, 1992

2 Yuan Xiao-dong, Hu De-qiang, Xu Hao *et al*. COOZ: a complete object-oriented extension to Z. ACM Software Engineering Notes, 1998,23(4):78~81

3 Hoare C A R. Proofs of correctness of data representation. Acta Informatica, 1972,1(4):271~281

4 Nickson R, Utting M. A new face for Ergo: adding a user interface to a programmable theorem prover. Technical Report, TR-95-42. Software Verification Research Centre, University of Queensland, 1995

5 Morgan C C. Programming from Specifications. 2nd edition, London: Prentice Hall, 1994

From Object-Oriented Z Specification to Code by Refinement Calculus

WANG Yun-feng[1,2] PANG Jun[1] ZHA Ming[1] YANG Zhao-hui[1] ZHENG Guo-liang[1]

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

[2](*Meteorology College PLA University of Technology Nanjing* 211101)

Abstract The advantage of COOZ (complete object-oriented Z) is to specify large scale software, but it does not support refinement calculus. Thus its application is confined and it can not be taken as a complete method for software development. Including refinement calculus into COOZ remedies its disadvantage during design and implementation. The separation between the design and implementation for structure and notation is removed as well. Then the software can be developed smoothly in the same frame. In this paper, development model is established, which is based on COOZ and refinement calculus. Data refinement and operation refinement are debated with a example. As for implementary technology of refinement calculus, a data refinement calculator is constructed and an approach for data refinement which is based on data refinement calculus and program window inference is provided.

Key words Formal development method, refinement calculus, formal specification, object-oriented.