计算机科学1998Vol. 25№. 6

机器辅助下的形式化规格说明求精技术*)

Machine-aided Refinement Techniques of Formal Specification

袁晓东

(电力部电力自动化研究院仿真室　南京210003)

郑国梁

(南京大学计算机软件新技术国家重点实验室　计算机科学与技术系　南京210093)

摘　要　Presently the refinement from formal specifications to program codes is mainly completed by manual work with little automatic techniques. To adopt machine-aided techniques in the process of refinement as more as possible, this paper introduces a new refinement steps from formal specifications written in an object-oriented extension to Z named COOZ to C++ program, and describes the strategies and steps of operation refinement such as automatic generation of program frame, stepped implementation of specification sentence and predicate transition in detail. This paper also introduces data refinement rules from abstract types of COOZ to concrete types of C++, and illustrates how to implement various abstract mathematical operators using code library written in templates of C++.

关键词　Formal specification, COOZ, C++, Refinement, Machine-aided

形式化规格说明为我们提供一个简洁、精确且能被很好理解的系统描述。但我们还需要从规格说明得到其合适的代码实现，这一开发过程称为形式化规格说明的求精[1]。规格说明的求精技术已经有比较深入的研究，一般要通过数据求精和操作求精逐步精化的过程，逐步降低抽象级，最终得到规格说明的程序实现代码。但现有的精化理论较少考虑机器辅助技术的应用，而机器辅助技术的成熟是形式化方法得以推广应用的前提和保证，因而有必要研究有利于应用机器辅助技术的形式化规格说明求精过程。

本文介绍一种能充分利用机器辅助技术的形式化规格说明语言 COOZ 的求精过程。COOZ(Complete Object-Oriented Z)[3]是我们在 Z 语言基础上自行设计的充分支持面向对象思想和概念描述机制的 Z 面向对象扩充语言。本文中以 COOZ 为形式化规格说明样板语言，以 C++ 为实现语言具体阐述了程序框架自动生成、规格说明语句分步实现、谓词转换等操作求精的策略、步骤，以及 COOZ 中各种数据类型的求精规则、抽象数学运算的程序代码库等数据求精的策略、规则。这些机器辅助下的求精技术对 Z 的各种扩充版本以及其他形式化描述语言的求精过程都有普遍的参考价值。

1. 基于 COOZ 的系统开发

基于面向对象的形式化描述语言 COOZ，我们提出如图1的应用机器辅助技术的软件系统开发过程：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；然后用 COOZ 写出其形式化规格说明，对其进行逐步求精得到程序代码。

注意图1中有两种不同的求精次序：可以先进行数据求精，也可以先进行操作求精。一般的求精次序是首先进行数据求精，然后再操作求精得到程序代码。但数据求精后得重写规格说明，为了使得数据求精后的规格说明简洁、明了、合乎逻辑，需要系统开发人员的智慧，而依照规则由机器自动产生的结果则往往冗长、难懂、不利于下一步操作求精，因而很不适用。为了能充分利用机器辅助技术，减轻开发人员的劳动，我们对求精次序作了改进。首先扩展了程

*)本课题系国家九五攻关项目、国家自然科学基金资助。

序的概念，允许在程序中使用规格说明语句和各种数学数据类型，从而可以首先进行操作求精，在转化为最终的程序代码前再进行数据求精，这时不必再重写规格说明，而是通过数学运算程序代码库直接将抽象的数学运算转化为程序代码，这样机器辅助技术可以在更大程度上发挥效用，成为系统开发人员的得力助手。

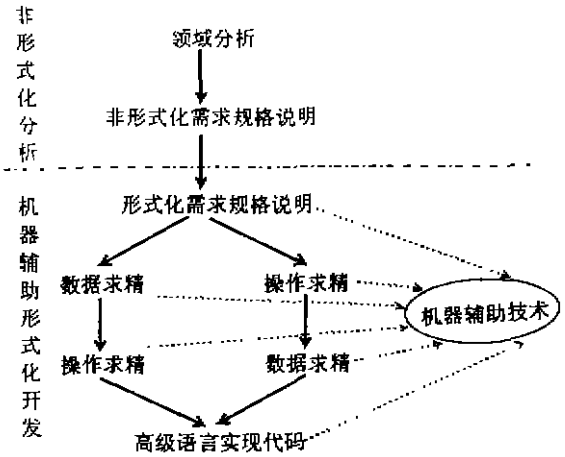

图1 基于COOZ规格说明语言的系统开发过程

2. 操作求精

2.1 规格说明语句和广义数据类型

文献[4]中提出，将规格说明和程序代码都看作是程序，从而扩展了程序的定义。在程序中用规格说明语句表示待求精的不可执行部分，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，包含规格说明语句和数学数据类型的程序称为抽象程序。这样将规格说明和程序的概念统一起来，规格说明的求精过程即是从抽象程序到代码的转换过程。规格说明语句由前置条件、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[3]有着直接的对应关系，如图2所示，方法模式中由 ΔId-list 指出可修改的状态量，在谓词部分用关键字 **if** 将方法的前置条件和后置条件分开。因而通过辅助工具可将COOZ的方法模式自动转换为相应的规格说明语句。

在对复杂数据类型的求精过程中，往往会出现既非COOZ中的数据类型，也非程序设计语言的数据类型，而是介于两者之间的一种混合类型。为使程序中允许出现这样的数据类型，我们提出广义数据类型的概念：

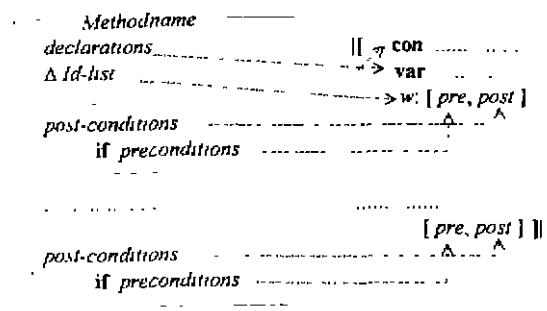

图2 规格说明语句与方法模式的对应关系

定义1 在抽象程序中，允许使用以下三种数据类型：

Ⅰ. 符合COOZ语法的数据类型；

Ⅱ. 符合程序设计语言语法的数据类型；

Ⅲ. 多层嵌套数据类型，其中上层类型为程序设计语言中的数据类型，底层类型为COOZ中数据类型，即允许记录类型的域类型、数组类型的下标分量类型、指针类型所指向的静态数据类型为COOZ中的抽象数据类型。

以上三种数据类型统称为广义数据类型。

2.2 程序框架自动生成

COOZ设计规格说明由若干个 **Class** 模式（图3）组成，它与C++中的类结构有着很好的对应关系，规则1说明如何将 **Class** 模式求精为C++的类。

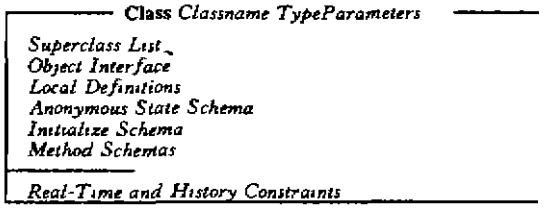

图3 COOZ中 **Class** 模式的组成

规则1 图3是COOZ中 **Class** 模式的一般格式，它可求精为如下的C++的类：

```
class classname:ancestor_names
{public:constants
    class_attributes
    methods_signature
};
classname::classname()
{
    initial_values
}
methods_implementation
classname::~classname()
{
}
```

其中直体书写部分为C++中的符号和关键字，斜体部分是指代、实际求精时需由相应内容替换。*ancestor_names*、*constants*、*class_attributes*、*methods_signature*、*initial_values*、*methods_implementation*分别由图3中*Superclass List*、*Local Definitions*、*Anonymous State Schema*、*Object Interface*、*Initialize Schema*、*Method Schemas*转化而来，转化方法如下：

Ⅰ 在*Superclass List*中每一类名前加上公有访问控制public则得到*ancestor_names*；

Ⅱ. 在*Local Definitions*中每一对象常量名前加上类型修饰符const与常量类型名，则得到*constants*；

Ⅲ. 对*Anonymous State Schema*中每一项状态量说明，将状态量类型置于状态量前面，则得到*class_attributes*；

Ⅳ. *Object Interface*中给出了消息名、消息参数名，参数类型以及返回类型，因此可直接产生C++的函数型构，即得到*methods_signature*；

Ⅴ. 将*Initialize Schema*改写为规格说明语句，作为构造函数的实现部分，即可得到*initial_values*；

Ⅵ. 将*Method Schemas*中各方法模式改写为规格说明语句，作为类中各成员函数的实现部分，即为*methods_implementation*；

Ⅵ. 析构函数的实现部分为空，留待以后扩充。

2.3 规格说明语句的分步实现

规格说明语句的精化程序代码必须满足相应的前、后置条件，可以采用分步的策略来实现，使每一步所要解决的问题得到明确和简化[5]。我们将规格说明语句进一步细化为如下有次序关系的四部分：

第1步：对前置条件的检查：仅包含方法执行前的状态量和输入变量，可以是任意形式的谓词公式。由于一个规约语句中允许有多对前、后置条件，为与后面的处理步骤相对应，当有多个前置条件时，对每个前置条件分别编号。

第2步：给局部量、方法执行后的状态量和输出变量赋值。赋值通过特殊形式的谓词公式给出：谓词公式必须是等式的形式，等式左边是单个局部量、方法执行后的状态量或输出变量，等式右边是由方法执行前的状态量、输入变量、已被赋值的局部量和后状态量组成的表达式。注意，这些等式的次序是重要的。如果第1步中有多个编号的前置条件，则这里的赋值语句也应该有相应的编号，只有当第1步中相同编号的前置条件为真时，该赋值语句才被执行。

第3步：对方法后置条件的检查。即是对原规约语句中后置条件的检查，目的是检查第2步中对状态量和输出变量的赋值是否使得后置条件成立，从而是原规约语句的一个精化。如后置条件不成立，则说明第2步中的求精是错误的，需要重新考虑。与前面步骤一样，如果方法有多对前、后置条件，则相对应的前、后置条件必须有相同的编号，只有当第1步相同编号的前置条件为真时，该后置条件才被检查。该步可通过C++的断言实现。

第4步：修改状态量。用相应后状态量的值代替前状态量的值、没有后状态量的状态值保持不变。因为规格说明语句中的后状态量在程序中是用临时变量实现的，所以有必要在对象方法执行完毕前对程序中的状态量进行修改，表示对象方法执行完毕、开始进入新的状态。该步是由于规约语句与程序代码的差异而增加的，不需要编号。

要对规格说明语句进行上述细化，必须消除不确定性和对谓词公式的转换，具体方法将在下节介绍。

2.4 谓词转换

要实现上述规格说明语句求精的第2步，必须对模式中后置条件的谓词公式进行转换。一般来说，谓词公式是松散性的，并非前面所要求的等式形式，且使谓词公式成立的变量值不止一个，具有不确定性，将其转换为上节中所要求的等式格式，即成为可执行规格说明语句，需要程序开发人员的智力劳动，目前尚无法完全由机器实现。但这一过程中机器可起到一定的辅助作用，主要表现在：对于一些简单的谓词转换，可以按照定理由机器自动进行；对于机器无法进行自动转换的，可以提供有关规则给用户查询，以方便用户进行转换。例如：

例1：$x=y'^2$

$\equiv y'=\sqrt{x} \vee y'=-\sqrt{x}$ （一元二次方程求解）

$\Leftarrow y'=\sqrt{x}$ （消除不确定性）

由于篇幅限制，谓词转换的有关定理将另文介绍。

3. 数据求精

3.1 数据类型的转化

Z中的数据类型分为基本类型和复合类型两大类，复合类型又分为集合类型、笛卡尔乘积类型、模式类型三种。Z中的基本类型不再有内部结构，求精时有的可直接成为程序设计语言中的数据类型（如整型），其余的经过必要的人工干预后可与具体数据类型具有简单的对应关系（如字符串类型）。COOZ中扩充的对象类型通过上述规则1可精化为C++的类，下面主要是针对三种复合类型及其嵌套类型提

出将其转化为C++中数据类型的求精规则。

3.1.1 简单复合类型的求精 现有的各种基于实现的程序设计语言都不支持或只能在极有限的范围内支持集合类型的实现，我们不得不采用其他数据结构来表示集合类型的数据，能用来表示集合类型的数据结构有两种：数组结构和链表结构，对元素个数限制在一定数目之内的集合数据用数组实现较为方便，而如果集合元素个数难以作出明确的限制，即其动态变化幅度很大，则必须采用链表结构来表示，下面分别就数组结构和链表结构给出集合类型的求精规则。各规则中，a 表示抽象数据，c 表示精化后的具体数据。

规则2 设 X 为基本类型，a:ℙX 为 COOZ 中集合类型数据说明，若 #a<n，n 为常量，则 a 可求精为如下的数组和整型计数器的封装结构：

```
struct X_Set {X body [n];
              int length;
              }c;
```

其中 length 表示集合中的元素个数，body[i](i=0，1，…，a. length-1)表示集合中的各个元素，下标 i 的次序是无关紧要的。

规则3 设 X 为基本类型，a:ℙX 为 COOZ 中集合类型数据说明，则 a 可求精为如下的链表结构：

```
struct X_Link {X value;
               X_Link* next;
               }*c;
```

其中 value 表示集合中元素的值，next 是指向下一个集合元素的指针。

用数组结构和链表结构表示集合类型各有优缺点，数组结构易于运算，但当集合中元素数目变动较大时造成存储空间的浪费，链表结构不浪费存储空间但运算较复杂。

笛卡尔乘积类型和模式类型在性质上是类似的，都是由多个不同类型数据组合而成的复合类型。所不同的仅是模式类型对其各成分变量给定一名称，但各变量之间没有次序关系，而笛卡尔乘积类型中各分量之间有次序关系，但没有变量名。因此，笛卡尔乘积类型和模式类型都可以用记录类型来求精。对模式类型而言，求精后的记录类型各个域名即是原模式类型中的成分变量名，域类型即是对应成分变量的求精类型；对笛卡尔乘积类型而言，我们必须按照各分量的次序为求精后对应的域取一个统一的域名，域类型即为原分量类型的求精类型。

为了保持原笛卡尔乘积类型中只有分量次序而无分量名的语义，必须使所有笛卡尔乘积类型的第 n 个分量对应求精类型中的域名都相同，为此我们可构造统一的域名生成函数为所有笛卡尔乘积类型的各分量生成对应域名：

定义2 函数 fieldname 为笛卡尔乘积类型各分量生成对应域名，称为域名构造函数，其型构如下：

```
char * fieldname(int n)
```

其中参数 n 为正整数，表示该分量为笛卡尔乘积类型中的第 n 个分量，返回类型为域标识符，为了使由域名构造函数产生的标识符与 COOZ 规格说明中原有标识符加以区别，我们规定 fieldname 产生的标识符一律以下划线'_'开头。

规则4 设 X_i(i=1，2，…，n)为基本类型，a:$X_1 \times X_2 \times \ldots \times X_n$ 为 COOZ 中笛卡尔乘积类型数据说明，fieldname(n)=_xn(n=1，2，…)，则 a 可求精为如下结构类型：

```
struct X_Cartesian{X1 _x1;
                   X2 _x2;
                   ……
                   Xn _xn;
                   }c;
```

为了保持模式类型中各成分变量与其次序无关的语义，我们可按成分变量名的字母顺序对其进行排序，并以此作为求精后记录类型中各个域的次序：

规则5 设 X_i(i=1，2，…，n)为基本类型，a:(p_1:X_1；…；p_n:X_n)为 COOZ 中模式类型数据说明，对 p_1，…，p_n 按字母顺序排序后的次序为 p_{k_1}，…，p_{k_n}，则 a 可求精为如下的结构类型：

```
struct X_Schema{X_k1 p_k1;
                ……
                X_kn p_kn;
                }c;
```

对模式类型数据而言，其分量选择符'.'同结构类型的分量选择符一致。对笛卡尔乘积类型数据而言，精化后各分量要通过域名来读取，而不是分量位置，如规则4中若 a 为二维笛卡尔乘积类型 $X_1 \times X_2$，则 first(a)用 c._x1表示，second(a)用 c._x2表示。

3.1.2 多层嵌套类型的求精 在一个较大的系统中，往往会出现上述几种复合类型相互嵌套形成的复杂数据结构，对其尽管也可以分别使用上述单一求精规则，但不可能由原始类型一步求精到程序设计语言中的数据类型，在广义数据类型(见定义1)的基础上我们可以应用下面的分步求精规则来实现 Z 中多层嵌套的复杂类型的求精：

规则6（分步求精规则） 任何 Z 中的嵌套数据类型从最外层看都是下述三种形式之一：(1)ℙX；(2)$X_1 \times X_2 \times \ldots \times X_n$；(3)($p_1$:$X_1$；…；$p_n$:$X_n$)。可将规则2、规则3、规则4、规则5适用范围扩充到类型 X、X_i

(i=1,2,…,n)可为复合类型,然后应用相应规则对其进行求精,其结果为一广义类型。对广义类型中的底层抽象类型自顶向下反复应用分步求精规则并汇合各步结果,最终可将其精化为程序设计语言中的嵌套数据类型。

3.2 用数据转化函数重写规格说明

数据求精之后,必须用具体数据结构重写规格说明。如果存在着从具体数据结构到抽象数据结构间的函数关系,则可直接用数据转化函数重写规格说明[2],机器可根据数据转化函数自动完成该项工作。而一个性能良好的数据求精中应该存在着这样的函数关系。下面我们给出上节中各条数据求精规则所对应的数据转化函数,其中 a 为抽象数据,c 为求精后的具体数据。

规则7 规则2的数据转化函数为:$a=f(c)=\{c.body[i]\mid 0\leqslant i<c.length\}$。

规则8 规则3的数据转化函数为:$a=f(c)=$**if** c=NULL **then** $\varnothing$ **else** $c->value\cup f(c->next)$。

规则9 规则4的数据转化函数为:$a=f(c)=(c._x1,c._x2,\ldots,c._xn)$。

规则10 规则5的数据转化函数为:$a=f(c)=\langle p_1:=c.p_1,p_2:=c.p_2,\ldots,p_n:=c.p_n\rangle$。

3.3 数学运算的程序实现

如果首先进行操作求精,那么在应用数据求精规则之后可以直接用程序代码实现原来的抽象数学运算,而省掉重写规格说明这一中间环节。方法是用数学运算的程序代码库来辅助用户将其转化为程序。我们针对 COOZ 中的各种抽象数学运算准备了较为完善的程序代码库,程序代码库中的各程序用 C++ 的模板(template)编制,将参与运算的操作数类型名等信息作为模板参数。如采用规则2将集合精化为数组,程序代码库中用模板类 set_a 来实现各种集合类型数据的表示及其操作。我们以集合并(∪)操作为例,set_a 中用成员函数 set_union 实现如下,其中模板参数 X_Set 在实例化时用集合中元素类型求精后的具体类型名代替,n 表示集合中允许的最大元素个数。

```
template<class X_Set, int n>class set_a {
      X_Set body [n];
      int length;
      ……
public :
      set_a(X_Set,n)set_union(set_a(X_Set,n)a);
      ……
};
template(class X_Set, int n)
set_a(X_Set,n)set_a(X_Set,n)::set_union(set_a(X_
  Set,n)a)
      {int i,j;
       set_a(X_Set,n)c=a;
       for(i=0;i<length;i++){
        j=0;
        while (j<a.length && body[i]!=a.body[j])j+
        +;
        if (j==a.length){
          try
          {if (c.length>=n)
             throw "Set union overflow!";
             /*当元素个数大于 n 时,溢出错*/
          }
          catch(char* str)
          {
               cout<<"Exception raised:"<<str<<"\n";
               exit;
          }
          c.body[c.length]=body[i];
          c.length++;
        }
       }
       return c;
};
```

基于数学运算程序代码库中上述集合类型实现模板,如程序中有如下集合类型变量说明:

```
set_a(int,100)s1,s2;
```

则集合并操作 s1∪s2可用 C++表达式 s1.set_union(s2)实现。

COOZ 中的各种抽象数据类型如笛卡尔乘积、序列(sequence)、包(bag)等的有关运算都可用类似的方法编制程序模板。

结论 本文基于面向对象的形式化规格说明语言 COOZ 和实现语言 C++ 提出了一种从规格说明到程序代码的求精方案,该方案主要考虑了如何尽可能地在求精过程中应用机器辅助技术,以减轻系统开发人员的负担。虽然规格说明求精的完全自动化在现阶段是不切实际的,但求精过程中大量应用机器辅助技术是必要的和可能的,如本文所提出的谓词转换、程序框架自动生成、数据求精规则和数学运算程序代码库等不仅是规格说明的求精方法,也是很好的机器辅助技术。我们目前正在进行有关辅助精化工具的研制工作。

致谢:在 COOZ 辅助精化工具的研制和本文的写作过程中,王云峰、许皓、李勇提供了很多参考意见,在此深表感谢!

参考文献

1 J. B. Worworth, Software Development with Z. Addison Wesley, 1992
2 C. C. Morgan, T. Vickers, eds. On the refinement calculus. FACIT. Springer-Verlag, 1994
3 袁晓东,郑国梁. Z 的面向对象扩充 COOZ 的设计. 软件学报,8(9),1997
4 C. C. Morgan, Programming from specifications, second edn. Prentice Hall, 1994
5 M. Love, Animating Z specification in SQL* Forms 3.0. In: J. P. Bowen, J. E. Nicholls eds, Z User Workshop, Lodon, 1992, Workshops in Computing. Springer-Verlag, 1993