文章编号:1000-5862(2017)01-0052-04

Apla →Java 程序生成系统中泛型机制实现方法研究

徐华珍[1,2],薛锦云[2*],朱小征[1,2]
(1. 江西师范大学计算机信息与工程学院,江西 南昌 330022;2. 江西师范大学国家网络化支撑软件国际科技合作基地,江西 南昌 330022)

摘要:在对泛型程设本质特征深入研究的基础上提出了新型泛型语言机制构想,并在 Apla →Java 生成系统中具体实现的新方法. 该方法比现有的 Java、C + +、C#等语言中泛型机制的实现方法简单,并通过经典算法实例演示实现效果,大量用例的成功测试表明该实现方法的可靠性和新型泛型语言机制的正确性.

关键词:泛型语言机制;软件可靠性;安全机制;Apla →Java 生成系统
中图分类号:TP 311 文献标志码:A DOI:10.16357/j.cnki.issn1000-5862.2017.01.10

0 引言

随着信息技术发展的日新月异,软件行业进入“互联网 +”[1] 时代,软件的开发效率更加备受关注,已成为计算机科学界普遍关注的焦点. 而泛型程序设计[2] 是解决该问题的一条有效途径,它能够大幅度地简化程序设计过程,提高程序开发效率和可重用性[3].

目前支持泛型语言机制[4] 的语言[5-8] 有:ML[9]、C + +[10]、Java[11]、Eiffel[12]、Haskell[13] 等,但这类高级语言中的泛型语言机制种类偏少,一般只有类型作参数. 少数语言如 Ada[5] 还有子程序作参数. 为更好地避免这些情况,文献[4] 提出了类型(type)、动作(action)和抽象数据类型(ADT)作参数的新型安全泛型语言机制,其中泛型的参数可以是数值、数据类型或动作(包括运算符、方法、过程、函数和其它程序单元等),也可以是抽象数据类型作参数. 这些参数还具有相应的安全机制 region,使泛型程序设计安全性得到了较好的保证.

PAR[15] 由复杂算法建模语言 Radl、抽象程序建模语言 Apla、[16] 从 Radl 算法到 Apla 程序、再由 Apla 到 Java、C + +等可执行语言程序的一系列模型转换系统,以及基于这些建模语言和模型转换系统,进行算法和程序建模,最后生成可执行语言程序代码的方法来构成的[17]. 本文在 Apla →Java 程序的模型转换系统中,有效地实现上述新型安全泛型语言机制.

1 Apla 语言中的泛型语言机制

在 PAR 平台中,原有的泛型语言机制为类型参数化和子程序参数化,现保留类型参数化的机制,将子程序参数化机制扩充为 action 参数化的泛型机制,再新增 ADT 参数化的泛型机制,于是 PAR 平台共有 3 种泛型语言机制:类型参数化、action 参数化和 ADT 参数化.

1.1 类型参数化

在 Apla 语言中,通过关键字 sometype 来定义类型变量,类型变量可以是基本数据类型,也可以是组合数据类型. 例如:

```
program BtreeSortProg(sometype basetype);
    var t: basetype;
```

t 即为简单数据类型,如 integer、char 等. 为了更好地提高安全性,对类型参数设置 region 安全机制,其定义的语法结构如下为

sometype = {所满足的一些特性类型的集合};

如在前面介绍的 BtreeSortProg 实例中,其 sometype 可以定义为

sometype = { char,integer,real}.

1.2 action 参数化

action 包括 Apla 中预先定义的操作、用户自己定义的函数、过程或服务. 用 someaction 关键字进行定义,action region 的定义语法结构为 someaction =

收稿日期:2016-08-09
基金项目:国家自然科学基金重大国际合作研究项目(61020106009),国家自然科学基金面上项目(61272075,61472167)和国家自然科学基金地区科学基金(61462041)资助项目.
通信作者:薛锦云(1947-),江苏海门人,教授,博士生导师,主要从事软件形式化和自动化的研究. E-mail:jinyun@vip.sina.com

{满足一些特性 actions 的集合};下面以插入排序的例子进行说明:

```
procedure sort( someaction comp) ;
…
someaction = { ≤ , ≥} ;
begin
…
end;
procedure sortascending: new sort( ≥) ;
procedure sortdescending: new sort( ≤) ;
```

其中 comp 是一个 action 变量,并对其进行了约束,即在实例化时所带参数只能为≤或≥.

1.3　ADT 参数化

ADT(抽象数据类型)是由一组数据和操作组成,将满足一些特性的 ADT 集合称作 ADT region,其语法结构为 ADT region: someadt = {满足一些特性的 ADT 的集合};下面用实例来加以说明:

```
procedure sort( somadt( elem,comp) ) ;
…
someadt = { ( integer, < ) ,( real, < ) ,( integer, > ) } ;
begin
…
end;
procedure sortdescending: new sort( integer, < ) ;
procedure sortdescending: new sort( real, < ) ;
```

其中 elem 是类型变量,comp 是 action 变量. 2 者组合满足了 ADT 特性,称为 ADT 变量. ADT region 表示实例化所带参数必须满足其定义的范围,否则视参数为不合法.

2　泛型语言机制在 Apla →Java 转换系统中的实现

2.1　Apla →Java 生成系统

Apla →Java 生成系统是 PAR 中的自动转换工具之一,它是由 Apla 语言为源语言,Java 8.0 为目标语言的自动转换生成系统. 其中 Apla(Abstract Programming Language)语言是一种抽象程序设计语言,用来描述抽象程序,用它编写的程序简单易懂,易于形式化推导和证明. 但它所描述的程序是不可执行的,于是 Apla →Java 程序生成系统便成为 Apla 程序自动转换为 Java 程序的重要桥梁,其系统结构图如图 1 所示.

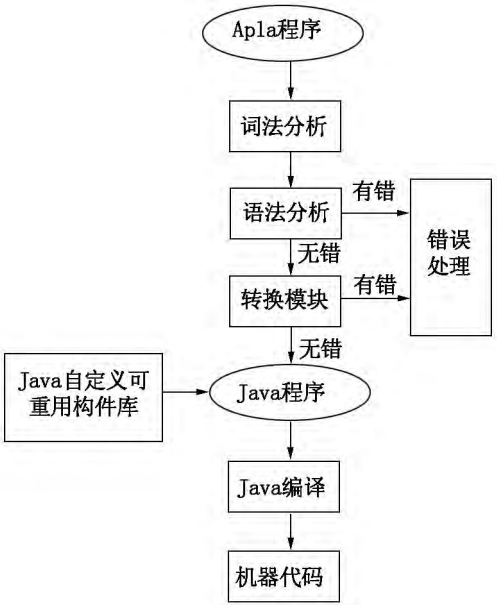

图 1　Apla →Java 程序生成系统的系统结构图

Apla →Java 程序生成系统包括转换器和自定义 Java 可重用构件库. 其中转换器主要由词法分析、语法分析及转换 3 部分组成. 首先对 Apla 源程序进行词法和语法分析,若语法分析不通过,则进行报错处理,否则进入转换模块,若转换也出现错误,同样进行报错处理. 否则根据 Apla 语言到 Java 语言的转换规则并结合自定义 Java 可重用构件库将其转换为对应的 Java 程序.

2.2　泛型实现方法

首先对 Apla 泛型程序进行词法语法分析,如果不符合语法规定,报错处理;再找到泛型关键字和实例化的参数,进行验证,如果参数不合法进行错误处理;转换 Apla 泛型程序,擦除程序中的泛型元素,转换为普通的 Apla 程序,然后直接交给转换器 tran 来运行. 本文所做的主要工作是在原有系统已经实现非泛型程序转换的基础上设计泛型转换模块,将泛型模型转换为非泛型模型,并加入相应的检测机制.

在 Apla 中有 4 种泛型程序单元,program、procedure、function 和 ADT,其中 procedure 和 function 结构类似,所以将 2 者放在一起实现. 根据泛型机制,本文主要对泛型子程序(procedure 和 function) 、泛型 ADT 这 2 大类 Apla 泛型程序单元进行了设计实现. 由于泛型子程序和泛型 ADT 的实现原理相同,限篇幅所致,这里只对泛型子程序的实现做详细介绍. 具体实现步骤如下:

(i) 在普通过程或函数算法的转换模块之前添加泛型子程序的处理模块. 当扫描到 procedure 或 function 关键字时,继续扫描 Apla 程序,若扫到 sometype,someaction 或 someadt 关键字,则进入泛型

子程序的处理模块,否则直接进行普通过程或函数的转换;

(ii) 在处理泛型子程序的模块中,首先用一指针链表保存 procedure 或 function 开始直到 end 的这段 Apla 代码,同时用数组保存 region 中的参数. 数组元素的类型应为字符型;

(iii) 系统依次扫描 Apla 程序,当扫描到实例化部分时,取出实例化的类型参数,在 region 参数数组中进行查找,然后比对,若在数组中没有找到,则进行报错,否则,进入下一步流程;

(iv) 检测通过后,取出前面用指针链表存储的 procedure 或 function 程序,按照顺序,在原 Apla 中依次找出 sometype,someaction 或 someadt 所带的参数,用实例化中的参数一一进行替换,然后将 sometype、someaction 或 someadt 关键字及其所带的参数列表一并擦除,并用实例化中新的程序名替换 procedure 或 function 名. 于是此段程序就变成了一段普通过程或函数程序. 最后,将这段普通过程或函数程序直接交给转换器进行自动转换;

(v) 继续扫描原 Apla 程序,如果还有实例化部分,继续第(iii)步工作. Apla 程序转换完成即结束.

2.3 Kleene 算法的实现

Kleene 算法是具有闭半环特性的算法,由它可以生成多种算法,如顶点对最大路径算法,传递闭包算法,最大容量路算法.

下面用 Kleene 算法来演示以上泛型实现方法的效果.

```
program para;
const
  maxnum = 9 999;
  num = 3;
var
  i,j,k,s: integer;
  c1,c3: array [0..num,array [0..num,integer]];
  c2: array [0..num,array [0..num,boolean]];
procedure kleene (sometype elem; someaction ⊙(a,
  b: elem): elem; someaction ⊕(a,b: elem): elem; n:
  integer; c: array [1..num,array [1..num,
    elem]]);
var
  i,j,k: integer;
begin
foreach(k,i,j: 0≤k,i,j<n: c [i,j]: = c [i,j] ⊙
  (c [i,k] ⊕ c [k,j]));
foreach(i: 0≤i<n: foreach(j: 0≤j<n: write(c [i,
  j],","));); writeln;);
end;
someadt = {(integer,min,+),(boolean,∨,∧),
  (integer,max,min)};
procedure floyd: new kleene(integer,min,+);
procedure close_set: new kleene(boolean,∨,∧);
procedure capcity: new kleene(integer,max,min);
begin
  writeln("所有顶点对的最短路径:");
  foreach(i,j: 0≤i,j<num: c1 [i,j]: =9 999;);
  c1 [0,2]: =11; c1 [2,0]: =3; c1 [0,1]: =4;
  c1 [1,0]: =6; c1 [1,2]: =2;
  c1 [0,0]: =0; c1 [1,1]: =0; c1 [2,2]: =0;
  floyd(num,c1);
  writeln("传递闭包");
  c2 [0,0]: =false; c2 [0,1]: =true;
  c2 [0,2]: =false; c2 [1,0]: =true; c2 [1,1]: =
    false; c2 [1,2]: =true;
  c2 [2,0]: =false; c2 [2,1]: =false;
  c2 [2,2]: =false; close_set(num,c2);
  writeln("最大容量路");
  foreach(i,j: 0≤i,j<num: c3 [i,j]: =9999;);
  c3 [0,2]: =11; c3 [2,0]: =3; c3 [0,1]: =4;
  c3 [1,0]: =6; c3 [1,2]: =2;
  c3 [0,0]: =0; c3 [1,1]: =0; c3 [2,2]: =0;
  capcity(num,c3);
end.
```

通过 Apla →Java 转换系统自动转换后,生成的主要 Java 代码如下:

```
import …;
public class  para{
…
public void floyd(int n,int [][] c){
…
  c [i] [j] = MathOp.s_min(c [i] [j],
    (MathOp.s_add(c [i] [k],c [k] [j])));
…}
public void close_set(int n,boolean [][] c){
…
c [i] [j] = MathOp.s_or(c [i] [j],
          (MathOp.s_and(c [i] [k],c [k] [j])));
…}
public void capcity(int n,int [][] c){
```

```
…
  c [i] [j] = MathOp. s_max( c [i] [j],
              ( MathOp. s _ min ( c [i] [k], c [k]
                [j] ) ) );
…}
public void maincall( ) {
…
floyd( num,c1) ;
…
close_set( num,c2) ;
```

图 2　Kleene 算法转换后的 Java 程序

3　结束语

本文首先对 Apla 语言中的 3 种新型泛型语言机制做了详细的介绍，然后提出了一种新的泛型实现方法，并在 Apla →Java 转换系统中进行实现，通过经典的 Kleene 算法演示了实现效果. 除 Kleene 算法外，还编写了许多典型的算法实例对系统进行了测试，如二叉树前中后序遍历、矩阵和积差、堆栈 ADT 等. 测试表明，该实现方法正确有效，转换后的语言简洁高效.

3 种新型泛型语言机制，可以大幅度提高软件开发的效率和软件可靠性，本文提出了一种简单易行的方法实现了这 3 种泛型机制，消除了在 Java 语言中直接添加泛型语言机制带来的复杂性，具有一定的理论和实际应用价值.

初步实现的 region 安全机制，显著提高了 Java 程序的安全性. 用枚举法给出了 region 的取值范围，实现方法简单方便. 事实上还可用谓词定义 region. 这样，泛型参数合理性的判断，涉及定理证明技术，将在后续论文中研究这一问题.

新型泛型语言机制和 region 安全机制的引入，使泛型程序设计变得更加充分具体，软件的可重用性和开发效率得到了显著提高.

```
…
capcity( num,c3) ;}
public static void main( String arg [ ]) {
… } }.
```

在 Apla →Java 生成系统中的运行效果如图 2 所示，在界面的左边导入 Kleene 算法的 Apla 代码，点击“转换”按钮，于是在界面的右边自动生成了 Java 程序，即上面给出的 Java 代码. 运行结果如图 3 所示.

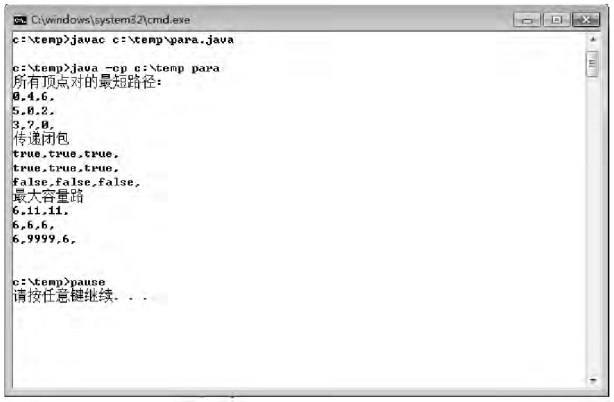

图 3　Kleene 算法运行的结果

4　参考文献

[1] 黄楚新，王丹. “互联网 + ”意味着什么：对“互联网 + ”的深层认识 [J]. 新闻与写作，2015(5) : 5-9.

[2] Musser D R，Stepanov A A. Generic programming [M]. Berlin: Symbolic and Algebraic Computation，1988: 13-25.

[3] Prieto-Díaz R. Status report: software reusability [J]. Software，IEEE，1993，10(3) : 61-66.

[4] Garcia R，Jarvi J，Lumsdaine A，et al. A comparative study of language support for generic programming [J]. OOPSLA，2003，36(3) : 115-134.

[5] Kermarrec Y，Pautet L，Tardieu S. GARLIC: generic Ada reusable library for interpartition communication [C]. ACM，1995: 263-269.

[6] 张玉春，程春英，李海峰. 浅谈 C#泛型和 C + + 模板 [J]. 内蒙古民族大学学报，2008(2) : 51-52.

[7] Ghosh D. Generics in Java and C + + : a comparative model [J]. ACM Siglan Notices，2004，39(5) : 40-47.

[8] 韩志强. 对. NET 平台中泛型技术的探究 [J]. 赤峰学院学报: 自然科学版，2010(11) : 23-24.

[9] Milner R，Tofte M，Harper R，et al. The definition of Standard ML [M]. Massachusetts: MIT Press，1997.

(下转第 92 页)

The Analysis of Procoagulant Constituents of *Eclipta alba* by Thin Layer Chromatography

GUO Qigen, XU Changlong*, Dong Wen, ZHANG Binghuo*
(College of Life Sciences, Jiujiang University, Jiujiang Jiangxi 332000, China)

Abstract: *Eclipta alba*, a kind of herbal medicine with many physiological activities, has various procoagulant components. In this study, the procoagulant components of *E. alba* were extracted by using ethanol and ethyl acetate respectively, and analyzed by suing preparative thin layer chromatography(TLC). There were at least six procoagulant components in metabolites of *E. alba*, and three of them showed strong procoagulant activity.

Key words: *E. alba*; procoagulant activity; thin layer chromatography

(责任编辑: 刘显亮)

(上接第 55 页)

[10] Stroustrup B. The C + + programming language [M]. New York: Pearson Education, 2013.
[11] Gosling J, Joy B, Steele G, et al. The Java language specification [M]. New York: Pearson Education, 2014.
[12] Meyer B. Eiffel: the language [M]. New York: Prentice Hall, 1992.
[13] Hutton G. Programming in Haskell [M]. Cambridge: Cambridge University Press, 2007.
[14] Xue Jinyun. Genericity in PAR platform [J]. Lecture Notes in Computer Science, 2015, 9559: 3-14
[15] Xue Jinyun. PAR method andits supporting platform [A]. Proceeding of International Workshop on Formal method for Devecoping Software [C]. Macao: UNU-IIST, 2006.
[16] 薛锦云. PAR 方法抽象程序设计语言 Apla 技术报告 [R]. 江西师范大学省高性能计算技术重点实验室, 2010.
[17] 朱小征, 薛锦云, 徐华珍. Transaction 在 PAR 平台中的实现方法及应用研究 [J]. 计算机与数字工程, 2015 (10): 1884-1890.

The Research on Implementation Method of Generic Mechanism in Apla →Java Program Generation System

XU Huazhen[1,2], XUE Jinyun[2*], ZHU Xiaozheng[1,2]
(1. College of Computer Information Engineering, Jiangxi Normal University, Nanchang Jiangxi 330022, China;
2. The State Base of Networked Supporting Software of International S/T Cooperation, Jiangxi Normal University, Nanchang Jiangxi 330022, China)

Abstract: A new implementation method of new generic language mechanisms which were implemented in the Apla →Java generation system was presented, and a typical algorithm example was illustrated. The method is simpler than the existing generic methods in Java, C + +, C # and other languages. A large number of successful testing cases demonstrated reliability of the implementation method and correctness of these new generic language mechanisms.

Key words: generic language mechanism; software reliability; safety mechanism; Apla →Java generation system

(责任编辑: 冉小晓)