Journal of Computer Applications
计算机应用,2012,32(8):2333 -2337
ISSN 1001-9081
CODEN JYIIDU
2012-08-01
http://www.joca.cn

文章编号:1001 -9081(2012)08 -2333 -05
doi:10.3724/SP.J.1087.2012.02333

基于 Clight 形式语义的代码功能描述提取

王　涛,陈敏翼*,齐　军
(华南师范大学 计算机学院,广州 510631)
(*通信作者电子邮箱 Chen-minyi@foxmail.com)

摘　要:软件代码的功能提取是功能集成的最基本前提,但软件功能提取普遍存在正确率低的问题。为此,提出基于 Clight 形式语义的代码功能描述提取机制,并用 Clight 代码功能描述算法实现。该机制严格基于 Clight 自然语义推理规则,忽略代码执行的中间细节,只关注执行前后的存储状态,并以此作为代码的功能描述,提高了功能提取的正确率和关键领域软件开发的成功率。

关键词:功能集成;形式语义;Clight;自然语义;代码功能;描述提取
中图分类号:TP311.5　　**文献标志码**:A

Functional description extraction of code based on formal semantics of Clight

WANG Tao, CHEN Min-yi*, QI Jun
(*School of Computer Science, South China Normal University, Guangzhou Guangdong* 510631, *China*)

Abstract: Code function description extraction is the basic premise of functional integration. To address the problem of common low accuracy rate of extraction of code function, the mechanism of extracting code function description based on formal semantic of Clight was proposed, and it was realized by algorithm of Clight code function description. This mechanism was strictly based on Clight natural semantics inference rules, ignoring the details of code execution in the middle, only concerned with the memory states before and after implementation. And as a functional description of the code feature description, it improved the accuracy of functional extraction and the success rate of the key area in software development.

Key words: functional integration; formal semantics; Clight; natural semantic; code function; description extraction

0　引言

要进行软件功能集成,功能查找必不可少;而功能匹配是功能查找的前提,功能提取又是功能匹配的前提[1]。因此本文着重于功能集成的最基本前提——代码功能描述提取。代码功能描述提取属于信息抽取的范畴,当前的信息抽取方法主要有四类[2-3],分别为基于词法分析的抽取、基于语法分析的抽取、基于词法分析和语法分析的结合体[4]以及基于领域本体的抽取。这几类方法在功能描述提取过程中存在查准率和查全率不高,以及自动化程度较低等问题。因此,本文将从程序设计语言形式语义的角度出发,对代码的功能提取进行研究。语义的形式描述尚处于蓬勃发展和多种技术并存的时期,主要的有指称语义、公理语义、操作语义、动作语义和单子语义等。相比其他形式语义,操作语义增加了时序特征,并且考虑了语言的执行过程,特别适合用来描述编程语言的语义。自然语义是操作语义最主要的代表之一,其特点是具有优秀的可推导性和不考虑中间状态的性质,已被用来定义了 SML、Java 等语言的语义[5-6]。本文将对 C 语言的子集 Clglt 进行研究,提出基于 Clight 自然语义的代码功能描述提取机制,并且给出实现此机制的算法。

基于 Clight 语义的代码功能描述提取机制,由于其代码语义的形式性,具有很好的数学基础和严格的语义推导规则,相比于现今多是基于词法、语法关键字匹配或者本体的描述,或者这几者结合的方法,在查准率、查全率以及自动化效率等方面,都有着明显的提高。

1　Clight

Clight 是 C 语言的一个子集,继承了 C 语言的大部分语法,对不纯性(not pure)予以摒弃,保证了 Clight 的表达式无副作用。Clight 自然语义具有优秀的可推导性和不考虑中间状态的特性,已有相关的应用,如程序编译器的验证[7]265,本文选择 Clight 的自然语义作为代码的功能描述。Clight 语义推理规则则是实现代码功能描述提取机制的算法的数学基础。另外,语法决定程序的形式和结构,是进行程序语义描述的前提条件。因此,下面将分别介绍一下 Clight 的形式语法和形式语义[8]。

1.1　Clight 形式语法

Clight 语法支持三种结构:表达式、语句和函数。在 Clight 表达式中,只支持 C 的没有副作用的操作符,不支持赋值操作符(=,+=,++等)和函数调用。赋值和函数调用作为语句来处理,不能出现在表达式中,但语句却可以看成是表达式的集合。Clight 语句支持所有 C 结构的控制语句,但不支持非结构化语句,例如 goto 以及非结构化 switch。由于最小单元的表达式是纯的(pure),即没有副作用,也就使得语句和函数计算结果具有唯一性,保证了应用基于 Clight 的功能描述提取机制提取出来的结果无二义性。

1.2　Clight 形式语义

Clight 语义形式化定义采用自然语义的方式给出,是一系列描述代码执行含义的推导规则,其语义确定,同时还精确描述了 ISO C 标准中未加规范和未加定义的一系列行为,其他诸如去引用空指针或者数组访问越界等 C 中未定义的行

收稿日期:2012-01-09;修回日期:2012-03-21。
作者简介:王涛(1975 -),男,广西合浦人,副教授,博士,主要研究方向:软件工程、形式化方法、信息系统安全;　陈敏翼(1986 -),男,广东潮州人,硕士,主要研究方向:软件工程、形式化方法;　齐军(1987 -),男,河南息县人,硕士,主要研究方向:软件工程、形式化方法。

为全部一致地表示为“出错行为”[9]。对应于 Clight 形式语法支持的三种结构，Clight 自然语义的推理规则也有表达式、语句和函数三个主要模块[10]，是功能描述提取机制及其算法实现的数学理论基础。

Clight 自然语义主要元素包括：表达式（a），变量（id），语句（s），函数（F），存储状态（M），不同未知存储状态（M'、M''），任意存储状态（M -，- 代表任意自然数），全局环境（G），局部环境（E），经过 0 或多步（$\overset{\varepsilon}{\Rightarrow}$），经过 t 步（$\overset{t}{\Rightarrow}$），内存块（b），0 或多个内存块（b^*）。其推理规则包括表达式、语句和函数三个部分，其解释请参看参考文献[7]中第271页的表格说明。

1）Clight 表达式的自然语义推理规则如式（1）~（14）所示。

① 表达式在左值位置：

$$\frac{E(id) = b \text{ 或 } id \notin Dom(E) \text{ 和 } \mathrm{symbol}(G,id) = \lfloor b \rfloor}{G,E \vdash id, M \Leftarrow (b,0)} \tag{1}$$

$$\frac{G,E \vdash a, M \Rightarrow \mathrm{ptr}(l)}{G,E \vdash *a, M \Leftarrow l} \tag{2}$$

$$\frac{\begin{array}{c} G,E \vdash a, M \Leftarrow (b,\delta) \\ \mathrm{type}(a) = \mathrm{struct}(id',\varphi) \end{array}}{\mathrm{field\_offset}(id,\varphi) = \lfloor \delta' \rfloor} \tag{3}$$
$$G,E \vdash a.\,id, M \Leftarrow (b,\delta+\delta')$$

$$\frac{G,E \vdash a, M \Leftarrow l \quad \mathrm{type}(a) = \mathrm{union}(id',\varphi)}{G,E \vdash a.\,id, M \Leftarrow l} \tag{4}$$

② 表达式在右值位置：

$$G,E \vdash n, M \Rightarrow \mathrm{int}(n) \tag{5}$$

$$G,E \vdash f, M \Rightarrow \mathrm{float}(f) \tag{6}$$

$$G,E \vdash \mathrm{sizeof}(\tau), M \Rightarrow \mathrm{int}(\mathrm{sizeof}(\tau)) \tag{7}$$

$$\frac{G,E \vdash a, M \Leftarrow l \quad \mathrm{loadval}(\mathrm{type}(a), M', l) = \lfloor v \rfloor}{G,E \vdash a, M \Rightarrow v} \tag{8}$$

$$\frac{G,E \vdash a, M \Leftarrow l}{G,E \vdash \&a, M \Leftarrow \mathrm{ptr}(l)} \tag{9}$$

$$\frac{G,E \vdash a_1, M \Rightarrow v_1 \quad \mathrm{eval\_unop}(op_1, v_1, \mathrm{type}(a_1)) = \lfloor v \rfloor}{G,E \vdash op_1 a_1, M \Rightarrow v} \tag{10}$$

$$\frac{\begin{array}{c} G,E \vdash a_1, M \Rightarrow v_1 \quad G,E \vdash a_2, M_1 \Rightarrow v_2 \\ \mathrm{eval\_binop}(op_2, v_1, \mathrm{type}(a_1), v_2, \mathrm{type}(a_2)) = \lfloor v \rfloor \end{array}}{G,E \vdash a_1 op_2 a_2, M \Leftarrow v} \tag{11}$$

$$\frac{G,E \vdash a_1, M \Rightarrow v_1 \ \mathrm{is\_true}(v_1, \mathrm{type}(a_1)) \quad G,E \vdash a_2, M \Rightarrow v_2}{G,E \vdash a_1 ? a_2 : a_3, M \Rightarrow v_2} \tag{12}$$

$$\frac{G,E \vdash a_1, M \Rightarrow v_1 \ \mathrm{is\_false}(v_1, \mathrm{type}(a_1)) \quad G,E \vdash a_3, M \Rightarrow v_3}{G,E \vdash a_1 ? a_2 : a_3, M \Rightarrow v_3} \tag{13}$$

$$\frac{G,E \vdash a_1, M \Rightarrow v_1 \ \mathrm{cast}(v_1, \mathrm{type}(a), \tau) = \lfloor v \rfloor}{G,E \vdash (\tau) a, M \Rightarrow v} \tag{14}$$

2）Clight 语句的自然语义推理规则如式（15）~（29）所示。

①除 switch 和循环以外的语句：

$$G,E \vdash \mathrm{skip}, M \overset{\varepsilon}{\Rightarrow} \mathrm{normal}, M \tag{15}$$

$$G,E \vdash \mathrm{break}, M \overset{\varepsilon}{\Rightarrow} \mathrm{break}, M \tag{16}$$

$$G,E \vdash \mathrm{continue}, M \overset{\varepsilon}{\Rightarrow} \mathrm{continue}, M \tag{17}$$

$$G,E \vdash (\mathrm{return}\ \varnothing), M \overset{\varepsilon}{\Rightarrow} \mathrm{return}, M \tag{18}$$

$$\frac{G,E \vdash a, M \Rightarrow v}{G,E \vdash \mathrm{return} \lfloor a \rfloor, M \overset{\varepsilon}{\Rightarrow} \mathrm{return}(v), M} \tag{19}$$

$$\frac{\begin{array}{c} G,E \vdash a_1, M \Leftarrow l \quad G,E \vdash a_2, M_1 \Rightarrow v \\ \mathrm{storeval}(\mathrm{type}(a_1), M, l, v) = \lfloor M' \rfloor \end{array}}{G,E \vdash (a_1 = a_2), M \overset{\varepsilon}{\Rightarrow} \mathrm{normal}, M'} \tag{20}$$

$$\frac{G,E \vdash s_1, M \overset{t_1}{\Rightarrow} \mathrm{normal}, M_1 \quad G,E \vdash s_2, M_1 \overset{t_2}{\Rightarrow} out, M_2}{G,E \vdash (s_1; s_2), M \overset{t_1 \cdot t_2}{\Rightarrow} out, M_q} \tag{21}$$

$$\frac{G,E \vdash s_1, M \overset{t}{\Rightarrow} out, \ M' \ out \neq \mathrm{normal}}{G,E \vdash (s_1; s_2), M \overset{t}{\Rightarrow} out, M'} \tag{22}$$

②while 循环：

$$\frac{G,E \vdash a, M \Rightarrow v \ \mathrm{is\_false}(v, \mathrm{type}(a))}{G,E \vdash (\mathrm{while}(a), s), M \overset{\varepsilon}{\Rightarrow} \mathrm{normal}, M} \tag{23}$$

$$\frac{G,E \vdash a, M \Rightarrow v \ \mathrm{is\_true}(v, \mathrm{type}(a)) \quad G,E \vdash s, M \overset{t}{\Rightarrow} out, M' \ out \overset{\mathrm{loop}}{\sim} out'}{G,E \vdash (\mathrm{while}(a), s), M \overset{t}{\Rightarrow} out', M'} \tag{24}$$

$$\frac{\begin{array}{c} G,E \vdash s, M \overset{t_1}{\Rightarrow} (\mathrm{normal} \mid \mathrm{continue}), M_1 \\ G,E \vdash (\mathrm{while}(a) s), M_1 \overset{t_2}{\Rightarrow} out, M_2 \end{array}}{G,E \vdash (\mathrm{while}(a), s), M \overset{t_1 t_2}{\Rightarrow} out', M_2} \tag{25}$$

③for 循环：

$$\frac{\begin{array}{c} s \neq \mathrm{skip} \quad G,E \vdash s_1, M \overset{t_1}{\Rightarrow} \mathrm{normal}, M_1 \\ G,E \vdash (\mathrm{for}(\mathrm{skip}, a_2, s_3) s), M_1 \overset{t_2}{\Rightarrow} out, M_2 \end{array}}{G,E \vdash (\mathrm{for}(s_1, a_2, s_3) s), M \overset{t_1 t_2}{\Rightarrow} out, M_2} \tag{26}$$

$$\frac{G,E \vdash a_2, M \Rightarrow v \ \mathrm{is\_false}(v, \mathrm{type}(a_2))}{G,E \vdash (\mathrm{for}(\mathrm{skip}, a_2, s_3) s), M \overset{\varepsilon}{\Rightarrow} \mathrm{normal}, M} \tag{27}$$

$$\frac{G,E \vdash a_2, M \Rightarrow v \ \mathrm{is\_true}(v, \mathrm{type}(a_2)) \quad G,E \vdash s, M \overset{t}{\Rightarrow} out_1, M_1 \ out_1 \overset{\mathrm{loop}}{\sim} out}{G,E \vdash (\mathrm{for}(\mathrm{skip}, a_2, s_3) s), M \overset{t}{\Rightarrow} \mathrm{normal}, M_1} \tag{28}$$

$$\frac{\begin{array}{c} G,E \vdash a_2, M \Rightarrow v \ \mathrm{is\_true}(v, \mathrm{type}(a_2)) \\ G,E \vdash s, M \overset{t_1}{\Rightarrow} (\mathrm{normal} \mid \mathrm{continue}), M_1 \\ G,E \vdash s_3, M_1 \overset{t_2}{\Rightarrow} \mathrm{normal}, M_2 \\ G,E \vdash (\mathrm{for}(\mathrm{skip}, a_2, s_3) s), M_2 \overset{t_3}{\Rightarrow} out, M_3 \end{array}}{G,E \vdash (\mathrm{for}(\mathrm{skip}, a_2, s_3) s), M_2 \overset{t_1 \cdot t_2 \cdot t_3}{\Rightarrow} out, M_3} \tag{29}$$

3）Clight 函数的自然语义推理规则如式（30）~（31）所示。

$$\frac{\begin{array}{c} F = \tau, id(dcl_1)\{dcl_2; s\} \\ \mathrm{alloc\_vars}(M, dcl_1 + dcl_2, E) = (M_1, b*) \\ \mathrm{bind\_params}(E, M_1, dcl_1, v_{\mathrm{args}}) = M_2 \\ G,E \vdash s, M_2 \overset{t}{\Rightarrow} out, M_3 \ out, \tau \# v_{\mathrm{res}} \end{array}}{G,E \vdash F(v_{\mathrm{args}}), M \overset{t}{\Rightarrow} v_{\mathrm{res}}, \mathrm{free}(M_3, b*)} \tag{30}$$

$$\frac{Fe = externid, \tau(dcl) \quad v = id(v_{\mathrm{args}}, v_{\mathrm{res}})}{G \vdash Fe(v_{\mathrm{args}}), M \overset{t}{\Rightarrow} v_{\mathrm{res}}, M} \tag{31}$$

2 代码的功能描述提取

2.1 Clight 代码功能描述提取机制

Clight 在描述代码自然语义时，隐藏很多执行的细节，只重点关注代码执行结果以及前、后存储状态的变化，而存储状

态 M 定义块引用到边界及内容的映射,即地址到值的映射($l \mapsto v$)。Clight 代码功能描述提取机制将重点关注存储状态中有关前、后条件地址中内容的变化,将地址到值的映射变化提取出来,并以此作为描述一段代码的功能依据。结构化程序的基本单元是函数,对于任意代码片段进行功能提取意义不大,所以现阶段只以函数为功能的基本单元进行分析提取功能描述,在此提取过程中,函数模块的全局环境 G 和局部环境 E 保持不变。为了用算法实现此机制,定义如下的一些概念:

定义 1　语法单元(Syntax Unit)。即 Clight 支持的三种结构:表达式(a)、语句(s)和函数(Fd)。它们之间依次存在被包含和包含关系,即 $a \subseteq s \subseteq Fd$,表达式通过运算符可以组成不同的复合表达式或者语句,而语句可以通过顺序、分支、循环等结构组成语句块和函数。

定义 2　语法单元树(Syntax Unit Tree)。是一棵含有若干个语法单元节点的树,每个节点又是一棵语法单元树。

定义 3　语义单元(Semantic Unit)。对应语法单元的三种结构,分为三类:1)表达式语义形如 $G,E \vdash aM^? \Rightarrow v_?$,表示在全局环境 G 和局部环境 E 下,表达式 a 在状态 $M^?$ 时求值为 $v_?$,其中 $M^?$ 和 $v_?$ 都是未知量;2)语句语义形如 $G,E \vdash s,M^? \overset{\varepsilon}{\Rightarrow} out,M'^?$,表示在全局环境 G 和局部环境 E 下,语句 s 在状态 $M^?$ 时经过 ε 步(0 或多步)执行后结果为 out,并且此时存储状态达到另一种状态 $M'^?$,同样 $M^?$、$M'^?$ 和 out 都为未知量;3)函数语义形式 $G \vdash F(v_{args},M^? \overset{t}{\Rightarrow} v_{res},\text{free}(M'^?,b^*))$,表示在全局环境 G 下,以 v_{res} 为传入值的函数 F 在状态 $M^?$ 下经过特定步骤运行结果为返回值 v_{res},之后释放存储状态 $M'^?$ 中给传入参数 dcl_1 和局部变量 dcl_2 分配的内存块 b^*。

定义 4　语义单元树(Semantic Unit Tree)。是一棵包含若干个语义单元的树,每个节点又是一个语义单元树。

定义 5　存储状态(Memory State, MS)。包含程序执行到某步时地址到内容的映射,是一个三元组,〈存储状态名,映射,条件〉,分别表示程序执行到当前存储状态标识、所有变量地址到内容的映射,以及产生此存储状态的充分条件,记为 $\langle MSI,MAPPING,CONDITION\rangle$。

定义 6　存储状态集(Memory States Set, MSS)。它是记录描述代码执行过程中所有产生的存储状态的有限集,随着代码的执行而不断更新。

2.2　Clight 代码功能描述提取算法

算法实现基于 Clight 形式语义的代码功能描述提取,步骤如下:

1)Clight 代码规范化:对于目标代码,可能用 C 编写,某些语法 Clight 不支持,对此将其转化为 Clight 支持的相应语法。

2)语法单元解析:按照 Clight 语法拆分代码,将代码分析成函数、语句和表达式,再按三者的包含关系,将整段代码整理成一棵以函数节点为根节点的语法单元树。

3)语法单元树转化为语义单元树:根据 Clight 语义,对应给出每个语法单元代表的语义。

4)Clight 语义推理:严格按照 Clight 语义的推导规则,以树的后根遍历顺序对各个节点进行推导,并更新 MS 和 MSS。

5)提取存储状态的变化:对上述过程所得的 MSS 进行扫描,提取出 MS 中有关前、后条件的部分,并描述其变化,作为这段代码的功能描述。

算法的核心部分在于步骤 4)中的语义推理,其输入为 Clight 语义推理规则和语义单元树,输出为 MSS。在步骤 4)开始的时候,MSS 只包含一个存储状态 $M^\varnothing$,该三元组中另外两个元素 *MAPPING* 和 *CONDITION* 皆为空。首先处理函数中的声明部分,可以分为两步:①$\text{alloc_vars}(M^\varnothing,dcl_1+dcl_2,E)=(M^0+b^*)$,该函数为 Clight 内置函数,为参数列表和局部变量声明中的每个变量分配一个边界从 0 到 sizeof(τ)的内存块 b,此时内存块中包含的内容为 undef。此时为每个变量分配的地址极为 l_{id},更新 MSS,此时存储状态到达 M^0。②$\text{bind_params}(E,M^0,dcl_1,v_{args})=M$,该函数重复调用函数 storeval 给参数列表初始化值,此时存储状态到达 M。声明部分执行完毕,存储状态由 $M^\varnothing$ 更新为 M^0,继而更新到 M,$M^\varnothing$ 时 *MAPPING* 为空,M^0 时参数 id_1 以及局部变量 id_2 分别被分配内存空间 l_{id1} 和 l_{id2},并且此时它们右值求值皆为 undef;到 M 时,初始化参数 id_1,赋予它们相应的值 v_{id1},而地址 l_{id2} 映射到的内容则仍为 undef。接下来,处理函数体中的语句部分,每一个语句节点是一棵语义单元树,从其叶子进行后根遍历推理,直到整个语句执行完成和 MSS 更新完毕,这一部分至少必须经过一次树的遍历,具体的遍历次数与单元树中包含了变量与常量相比较的布尔表达式的节点数正相关,每多一个这样的节点,就需要将遍历次数乘以 2。最后执行的结果是整棵语义单元树依据 Clight 语义规则推理所得的 MSS。

下面,对算法的复杂度进行分析。

对于步骤 1)中的 Clight 代码规范化,由于以函数作为功能描述提取的基本单元,所以时间与空间复杂度与函数中的语句数目相关。一个优秀的函数必定不会过长,所以这个步骤的时间和空间复杂度可以看作是某个不大的常数。

对于步骤 2)中将代码转化为语法单元树,由于表达式、语句和函数具有包含关系,所以,一个函数将拆分为若干个以语句为单元的节点,而一个语句单元节点也将拆分成若干个以表达式为单元的节点,所以可以采用孩子兄弟表示法作为语法单元树的存储结构[11]。而由于规范化之后的代码具有纯洁性(pure),每一个函数和每一个语句总能拆分成有限的孩子和兄弟。另外,优秀的代码段一般来说不会过长,这也保证了转化之后的语法单元树的节点数 n 不会过大。这一部分的时间复杂度和空间复杂度与函数转化为语法单元树之后的节点数 n 相关,均为 $O(n)$。

对于步骤 3)中将语法单元树转化为语义单元树,时空复杂度同样也为 $O(n)$,n 为语法单元树的节点数目。

对于步骤 4)中的语义推理,时间复杂度和语义单元树的初始状态有很大关系,受语义单元树中输入参数和常量进行比较的布尔表达式的节点数量影响很大。最坏的情况是每个节点都包含此类型的布尔表达式,那么就相当于有 2^n 个节点需要独立进行匹配语义规则和推导,耗费的时间为 $(2^n)*t$(t 为每个节点耗费的时间),利用 Drools 散列法后,t 可以看作是常数 1,所以时间复杂度为 $O(2^n)$;而空间复杂度,在具体实现时遇到上述布尔表达式的情况时,采取的是 Java 的 Clone 特性,这是空间换时间的策略,因此最坏情况下,空间复杂度为 $O(n*2^n)$。虽然如此,但在实际情况中,这种最坏情况几乎是不会出现的,因为一段优秀的函数不会允许出现过多的输入参数[12]。

对于步骤 5)中存储状态的变化提取,状态数目不会超过语法单元树的节点数 n,时间和空间复杂度最多为 $O(n)$。

综上所述,实现此功能描述提取机制的算法时间复杂度最坏情况下为 $O(2^n)$,空间复杂度为 $O(n*2^n)$,其中 n 为代码转化为语法单元树的节点个数。

实现基于 Clight 的功能描述提取机制的算法的实验环境是跨平台的,所用到的软件包括 JDK1.6x64,Drools5.1 和

Eclipse 3.6。其中 Drools 是一个带有基于前向推理的规则引擎的业务规则管理系统,其核心是一个能处理大量规则和事实的推理引擎[13-15]。对已有的 18 个(现在只实现 18 个,余者有待扩展)Clight 自然语义推理规则进行封装,作为 Drools 推理引擎规则库;把规范化后的语法单元树作为 Drools 推理引擎的事实,在推理引擎中与规则库通过匹配进行推理。

3 实例验证

实例验证了三段不同的代码:功能分别是 while 循环、for 循环实现的“求 m 的阶乘”,以及“1 到 m 的累加”。在此,由于篇幅关系,仅简单分析 while 循环实现的“求 m 的阶乘”的函数,其他两个例子类似。通过代码规范化、语法单元解析和语法单元转化为语义单元之后,得到一棵语义单元树,如图 1 所示。

```
1)  int factorial1(int m)
2)  { temp = 1;
3)    while(m > 0)
4)    { temp = temp * m;
5)      m = m - 1;
6)    }
7)  return temp;
8)  }
```

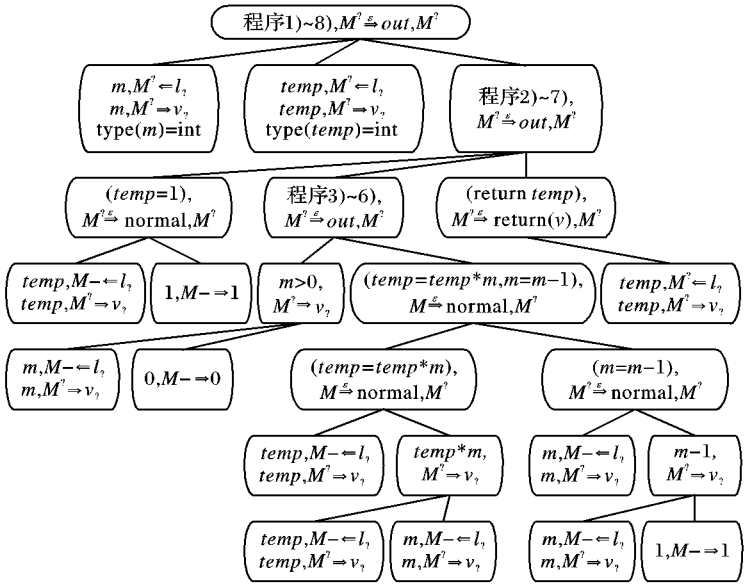

图 1 while 循环实现 m 阶乘的语义单元树

语义单元树按照树的后根遍历顺序进行语义推理,得到一个存储状态集 MSS,MSS 记录了描述代码执行过程中所有产生的存储状态的有限集。在推理过程中,得到的 MSS 为存储状态 $M^{\varnothing}$ 到 M^n 的集合:

$$
\begin{array}{lll}
M^{\varnothing} & \varnothing & \varnothing \\
M^0 & l_m \mapsto \text{undef} \quad l_{temp} \mapsto \text{undef} & \varnothing \\
M & l_m \mapsto v_m \quad l_{temp} \mapsto \text{undef} & \varnothing \\
M^1 & l_m \mapsto v_m \quad l_{temp} \mapsto 1 & \varnothing \\
M^2 & l_m \mapsto v_m \quad l_{temp} \mapsto 1 \times v_m & \varnothing \\
M^3 & l_m \mapsto v_m - 1 \quad l_{temp} \mapsto 1 \times v_m & \varnothing \\
M_1^4 & l_m \mapsto v_m \quad l_{temp} \mapsto 1 & v_m \leqslant 0 \\
M_2^4 & l_m \mapsto v_m - 1 \quad l_{temp} \mapsto 1 \times v_m & v_m > 0 \\
M_2^5 & l_m \mapsto v_m - 1 \quad l_{temp} \mapsto (1 \times v_m) \times (v_m - 1) & v_m > 0 \\
M_2^6 & l_m \mapsto v_m - 2 \quad l_{temp} \mapsto (1 \times v_m) \times (v_m - 1) & v_m > 0 \\
M_{2,1}^7 & l_m \mapsto v_m - 1 \quad l_{temp} \mapsto 1 \times v_m & v_m > 0 \;\&\&\; v_m - 1 \leqslant 0 \\
M_{2,2}^7 & l_m \mapsto v_m - 2 \quad l_{temp} \mapsto (1 \times v_m) \times (v_m - 1) & v_m > 0 \;\&\&\; v_m - 1 > 0 \\
M_{2,2}^8 & l_m \mapsto v_m - 2 \quad l_{temp} \mapsto (1 \times v_m) \times (v_m - 1) \times (v_m - 2) & v_m > 0 \;\&\&\; v_m - 1 > 0 \\
M_{2,2}^9 & l_m \mapsto v_m - 3 \quad l_{temp} \mapsto (1 \times v_m) \times (v_m - 1) \times (v_m - 2) & v_m > 0 \;\&\&\; v_m - 1 > 0 \\
\vdots & & \\
M^n & l_m \mapsto 0 \quad l_{temp} \mapsto (1 \times v_m) \times (v_m - 1) \times \cdots \times 1 & v_m > 0 \;\&\&\; v_m - 1 > 0 \;\&\&\; \cdots \;\&\&\; 1 > 0 \;\&\&\; 0 \leqslant 0
\end{array}
$$

根据此 MSS,可以看出存储状态中有关前、后条件的地址中内容的变化,并得出该函数的功能是求 m 阶乘的结论。在实现的算法中,最终的状态提取将以 XML 文档的形式给出。在此 XML 文档中,包含了 CProgram、CFunction、CStatement 等节点,每一个节点均有其对应的语法元素和语义属性 CSemantics。语法元素描述节点的语法单元,语义属性描述节点的内存状态、运算情况和结果值等,由此可得到一段容易阅读和理解的代码功能描述。

作为代码功能描述,描述代码引起的、运行前状态和代码运行后状态之间的变化值,是描述代码功能最直接的方法。这种方式也是在描述输入、输出以及输入与输出之间的关系。现在软件的功能(或需求)描述,也都是以输入、输出之间的关系来描述代码功能的,这也是进行黑盒测试的基础。而过程在功能描述中是不应该描述的。对同一功能,可以实现这一功能的算法、方案和流程很多,如果这些不同的算法实现,其功能都是一样的,那么其功能描述也应该是一样的。

对 for 循环实现的求 m 阶乘和求 1 到 m 累加两个功能进同样的分析,可以得出 while 循环实现的求 m 阶乘和 for 循环实现的具有相同功能,求 1 到 m 的累加则不具有相同的功能。因为基于 Clight 形式语义的代码功能描述提取机制重点关注的是存储状态中有关前、后条件的地址中的内容变化,将地址到值的映射变化提取出来,并以此作为描述一段代码的功能依据。因此,若两段代码功能不同,所得到的最终语义描述必定不同,即使前后的地址映射可能相同,如例子中两个功能不同的程序块,由于考虑到了所有的取值情况,值的变化也不可能完全相同,否则便是功能一样的代码。综上可得,在暂时不考虑不确定性程序的情况下,通过 Clight 代码功能描述算法,不同代码实现相同的功能,可以得到相同的结果,不同

代码实现不同的功能,将得到不同的结果。这很好地为功能集成的下一个步骤——功能匹配奠定了基础。

4　结语

本文主要针对代码功能描述提取的目标,将变成语言的形式语义应用到代码功能描述中来,提出了基于 Clight 形式语义的代码功能描述提取机制。这种机制建立在 Clight 自然语义的基础上,与基于关键字或者本体的描述提取不同,提取来自于形式化语义的推理,避免了关键字理解意义上的偏差且能实现自动化,并且经过分析已验证了第 3 章中的三段代码是否实现相同的功能。基于 Clight 形式语义的代码功能描述提取研究包含许多方面的工作,由于研究时间和条件的限制,本文只完成了一小部分,还有许多地方有待完善,接下来可以从以下几个方面进行完善:

1)自然语义只能描述正常终止程序的行为,对于非正常终止程序和出错的程序则无法进行区别,这是本文算法的一个局限,有待完善解决;

2)本文提出的算法复杂度跟代码中变量与常量进行比较的布尔表达式的数量正相关,因此对于循环和分支嵌套(其中包含变量与常量比较)的情况处理还有待于改进;

3)现阶段只考虑了一个函数实现的功能,对于函数之间的调用还有待于进一步的研究。

参考文献:

[1] 罗海滨,范玉顺,吴澄. 工作流技术综述[J]. 软件学报,2000,11(7):899 – 907.

[2] 叶彭飞. 一种基于领域本体的程序理解方法研究[D]. 上海:复旦大学,2010.

[3] ABRIAL J-R. B 方法[M]. 裘宗燕,译. 北京:电子工业出版社,2004.

[4] 古辉,马灵春,阳继旭. 一种改进的程序理解信息抽取系统(TSE)设计[J]. 浙江工业大学学报,2008,36(2):174 – 177,203.

[5] 张迎周,张卫丰,钱俊彦. 形式语义描述方法研究进展与评价[J]. 南京邮电大学学报:自然科学版,2006,26(6):86 – 94.

[6] XU BO, HUA BAOJIAN, GAO YING. An imperative formal calculus for Java [C]// ICFCC 2010: The 2nd International Conference on Future Computer and Communication. Piscataway: IEEE, 2010, 1: 168 – 173.

[7] BLAZY S, LEROY X. Mechanized semantics for the Clight subset of the C language [J]. Journal of Automated Reasoning, 2009, 43(3): 263 – 288.

[8] LEROY X, GRALL H. Coinductive big-step operational semantics [J]. Information and Computation, 2007, 207(2): 284 – 304.

[9] IBM. ILog [EB/OL]. [2011 – 12 – 16]. http://www.ibm.com/technologyservices/us/en/?cm_re=masthead-_-itservices-_-more/.

[10] LEROY X. Csem [EB/OL]. [2011 – 12 – 10]. http://compcert.inria.fr/doc-1.6/html/Csem.html.

[11] 严蔚敏,吴伟民. 数据结构:C 语言版[M]. 北京:清华大学出版社,2007.

[12] ISO/IEC 9899: TC2, WG14/N1124 [EB/OL]. [2011 – 11 – 10]. http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1124.pdf.

[13] BALI M. Drools JBoss Rules 5.0 Developer's Guide [M]. [S.l.]: Packt Publishing, 2009.

[14] Drools. JBoss. Community [EB/OL]. [2011 – 12 – 05]. http://www.jboss.org/drools.

[15] IBM. ILog [EB/OL]. [2011 – 12 – 05]. http://www-01.ibm.com/software/websphere/ilog/.

(上接第 2332 页)

今后的工作将利用逻辑 Petri 网的形式化推理与分析功能,完善基于逻辑 Petri 网的 Web 服务簇建模理论和分析技术,进一步研究服务发现、服务组合及动态优化等问题。

参考文献:

[1] BERARDI D, CHEIKH F, de GIACOMO G, *et al*. Automatic service composition via simulation [J]. International Journal of Foundations of Computer Science, 2008, 19(2): 429 – 451.

[2] 徐小良,陈金奎,吴优. 基于聚类优化的 Web 服务发现方法[J]. 计算机工程,2011,37(9):68 – 70.

[3] NAYAK R, LEE B. Web service discovery with additional semantics and clustering[C]// ICWI'07: Proceedings of IEEE/WIC/ACM International Conference on Web Intelligence. Piscataway: IEEE, 2007: 555 – 558.

[4] 孙萍,蒋昌俊. 利用服务聚类优化面向过程模型的语义 Web 服务发现[J]. 计算机学报,2008,31(8):1340 – 1353.

[5] RAJAGOPAL S, THAMARAI S S. Semantic grid service discovery approach using clustering of service ontologies[C]// TENCON'06: Proceedings of 2006 IEEE Region 10 Conference. Piscataway: IEEE, 2006: 1 – 4.

[6] HAN SHUN, WANG HAIYANG, CUI LIZHEN. A user experience-oriented service discovery method with clustering technology[C]// ISCID'09: Proceedings of Second International Symposium on Computational Intelligence and Design. Piscataway: IEEE, 2009: 64 – 67.

[7] 刘譞哲,黄罡,梅宏. 用户驱动的服务聚合方法及其支撑框架[J]. 软件学报,2007,18(8):1883 – 1895.

[8] 刘书雷,刘云翔,张帆,等. 一种服务聚合中 QoS 全局最优服务动态选择算法[J]. 软件学报,2007,18(3):646 – 656.

[9] 杜玉越,蒋昌俊. 基于工作流网的实时协同系统模拟技术[J]. 计算机学报,2004,27(4):471 – 481.

[10] JIANG XIA, WU BING, DU YUYUE. Architecture of dynamic service composition using logic Petri nets[J]. Journal of Software Engineering and Applications, 2011, 4(10): 585 – 589.

[11] DU YUYUE, JIANG CHANGJUN, ZHOU MENGCHU. Modeling and analysis of real-time cooperative systems using Petri nets [J]. IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans, 2007, 37(5): 643 – 654.

[12] LIU WEI, DU YUYUE. Modeling multimedia synchronization using Petri nets [J]. Information Technology Journal, 2009, 8(7): 1054 – 1058.

[13] DU YUYUE, JIANG CHANGJUN. On the design and temporal Petri net verification of grid commerce architecture[J]. Chinese Journal of Electronics, 2008, 17(2): 247 – 251.

[14] MURATA T. Petri nets: properties, analysis and applications[J]. Proceedings of the IEEE, 1989, 77(4): 541 – 580.

[15] 蒋昌俊. Petri 网的行为理论及其应用[M]. 北京:高等教育出版社,2003.

[16] DU YUYUE, QI LIANG, ZHOU MENGCHU. A vector matching method for analyzing logic Petri nets [J]. Enterprise Information Systems, 2011, 5(4): 449 – 468.