软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2011,22(11):2564−2576 [doi: 10.3724/SP.J.1001.2011.03964]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

基于文法分支覆盖的短句子生成算法*

郑黎晓[1,2+], 许智武[1,2], 陈海明[1]

[1](中国科学院 软件研究所 计算机科学国家重点实验室,北京 100190)

[2](中国科学院 研究生院,北京 100049)

Algorithm for Generating Short Sentences from Grammars Based on Branch Coverage Criterion

ZHENG Li-Xiao[1,2+], XU Zhi-Wu[1,2], CHEN Hai-Ming[1]

[1](State Key Laboratory of Computer Science, Institute of Software, The Chinese Academy of Sciences, Beijing 100190, China)

[2](Graduate University, The Chinese Academy of Sciences, Beijing 100049, China)

+ Corresponding author: E-mail: zhenglx@ios.ac.cn

Zheng LX, Xu ZW, Chen HM. Algorithm for generating short sentences from grammars based on branch coverage criterion. *Journal of Software*, 2011,22(11):2564−2576. http://www.jos.org.cn/1000-9825/3964.htm

Abstract: This paper presents a sentence generation algorithm, which takes as input a context-free grammar and produces a set of sentences that achieves branch coverage for the grammar. The algorithm incorporates length control, redundancy elimination, and sentence-set size control strategies into a sentence generation process such that the generated sentences are short and simple, and the sentence set is small with no redundancy. The paper also investigates the application of this algorithm to test data generation for grammar-based systems. Experimental results show that the generated test data not only has high fault detection ability, but can also help testers improve the testing speed.

Key words: context-free grammar; sentence generation; branch coverage; length control

摘 要: 提出一种上下文无关文法的句子生成算法.对于给定文法,算法生成一个满足该文法分支覆盖准则的句子集.结合长度控制、冗余消除和句子集规模控制等策略,使得生成的句子较短、无冗余、句子集规模较小.考察了算法在基于文法的软件系统的测试数据生成方面的应用情况.实验结果表明,该算法生成的测试数据具有较强的程序揭错能力,并且能够帮助测试人员提高测试速度.

关键词: 上下文无关文法;句子生成;分支覆盖;长度控制

中图法分类号: TP301 **文献标识码**: A

文法,尤其是上下文无关文法,在软件开发中具有广泛的应用.例如,编译器、浏览器、形式规约获取工具、代码预处理器等许多软件都密切依赖于文法.这类软件系统被称为基于文法的系统(grammar-based systems)[1]或者文法件(grammarware)[2].文法的句子生成算法在此类系统的测试和验证中起着重要的作用,例如,用于生成句子实例来测试文法本身是否正确定义[3−5],或者为系统的黑盒(功能)测试提供测试数据[6−10]等.

* 基金项目: 国家自然科学基金(61070038, 60573013)
收稿时间: 2010-07-06; 修改时间: 2010-08-13; 定稿时间: 2010-10-26

目前,文法的句子生成主要采用两种方法:随机生成和基于文法覆盖准则的生成.随机生成的基本思想是,从文法开始符号出发,不断地从产生式集合中随机选取产生式,对当前句型中相应的非终极符进行替换,直到句型中没有非终极符出现时为止[10].这种方法实现简单,但是无法确定生成句子的长度、推导深度以及生成过程何时结束.基于文法覆盖准则生成的基本思想是,每次有选择性地选取产生式来替换非终极符,以保证最终生成的句子集达到某种文法覆盖准则[6,7,11].这种方法生成的句子更有代表性,在一些实际应用中尤为重要.

Purdom[6]于 1972 年提出的一种句子生成算法是基于文法覆盖准则的句子生成方法的基础性工作.该算法生成文法的一个句子集,使得文法的每个产生式至少被用到 1 次,称为满足"产生式覆盖"或者"规则覆盖"[3,6].由于 Purdom 算法生成的句子数量较少,并且有一些过于复杂(长)[5,8],不适合用于实际的测试任务中.文献[5]在 Purdom 原始算法中加入一个长度控制策略,生成一组较简单(短)同时仍满足规则覆盖的句子,并通过实验验证了生成的短句子在文法测试方面的有效应用.

但是,规则覆盖是一种相对较弱的覆盖准则,它只是简单地覆盖所有的产生式规则,不能反映出文法内部结构的某些不同.为此,Lämmel 提出了"分支覆盖"(又称"上下文依赖规则覆盖")的概念[3].除了考虑单个的产生式覆盖以外,还考虑了覆盖产生式的上下文环境,因此能够进一步反映文法的内部结构关系,比规则覆盖更精确.文献[11]扩展了 Purdom 算法,给出了一种满足分支覆盖的句子生成算法.其算法保留了 Purdom 算法的特点,即生成的句子数量较少,并且有一些过于复杂.本文提出一种基于分支覆盖的句子生成算法,结合长度控制、冗余消除和句子集规模控制等策略,使得生成的句子:

(1) 满足分支覆盖;

(2) 比较短小简单,并保证最大句子长度最短;

(3) 无冗余,句子集规模较小.

与文献[11]中算法主要区别在于第(2)条和第(3)条,即对句子长度进行控制,并消除冗余.我们通过实验对比了该算法和文献[11]中算法生成的句子特性,以及在软件测试中的应用情况.实验结果表明,本文算法生成的句子较为简单,长度比较均匀,在软件测试中不仅具有较好的程序揭错能力,而且能够帮助测试人员提高测试速度.

本文的主要贡献包括:首先,将文献[5]中的思想扩展到更复杂的文法覆盖准则上;其次,提出了更严格的长度控制策略,保证最短的句子长度,并采用不同的句子生成顺序,消除冗余和控制句子集规模;最后,通过实验考察了生成的短句子在基于文法的软件系统的测试方面的应用情况.

本文第 1 节介绍预备知识.第 2 节给出算法基本思想和算法概述.第 3 节具体描述算法的实现.第 4 节是实验结果分析及讨论.第 5 节介绍相关工作.第 6 节给出结论.

1 预备知识

1.1 上下文无关文法

一个上下文无关文法(context-free grammar,简称 CFG)[12]是一个四元组 $G=(N,T,S,P)$,其中,N 和 T 分别为非终极符集合和终极符集合;$S\in N$ 是文法的开始符;P 是产生式的有限集合,产生式是形如 $A\to\alpha$的推导规则,其中,$A\in N,\alpha\in(N\cup T)^*$.本文中提到的文法如未加说明,均指上下文无关文法.

若存在$\gamma,\delta,\eta\in(N\cup T)^*$和 $A\in N$ 满足$\alpha=\gamma A\eta,\beta=\gamma\delta\eta$,且 $A\to\delta\in P$,则称α一步推导出β,记为$\alpha\Rightarrow\beta$.若存在$\alpha_1,\ldots,\alpha_m$ ($m\geqslant 1$)使得$\alpha=\alpha_1\Rightarrow\ldots\Rightarrow\alpha_m=\beta$,则称$\alpha$推导出$\beta$,记为$\alpha\Rightarrow^*\beta$.如果 $S\Rightarrow^*\alpha$,则称α为 G 的一个句型;如果$\alpha\in T^*$,则称α为 G 的一个句子.G 所定义的语言 $L(G)$即所有句子构成的集合.

从开始符 S 推导出句子 w 的过程可构造一棵树.树的根结点是 S,内部结点是非终极符,叶子结点是终极符.每个非叶子结点和它的子结点分别对应推导中用到的一个产生式的左部和右部.该树称为 w 的推导树.本文假设一个句子只对应一棵推导树,即文法是无二义性的.

定义 1[6]. 定义句子 w 的句子长度为其对应推导树的结点个数.

这个定义取自文献[6].不同于一般意义上的句子长度,它实际上反映了一个句子的推导复杂度.

定义 2. 为论述方便,定义句子集 W 的句子长度为 W 中的句子长度最大值.

1.2 分支覆盖

定义 3[3]. 给定文法 $G=(N,T,S,P)$.如果有 $p:m\to unv\in P$ 和 $q:n\to z\in P$,其中,$m,n\in N,u,v,z\in(N\cup T)^*$,$n$ 在 p 右部第 i 个位置上,则称三元组$\langle p,q,i\rangle$为文法 G 的一个分支.如果有推导 $S\Rightarrow^* xmy\Rightarrow^p xunvy\Rightarrow^q xuzvy\Rightarrow^* w$,其中,$x,y\in(N\cup T)^*$,$i$ 为上述推导中 n 在 p 右部的位置,则称句子 $w\in L(G)$覆盖分支$\langle p,q,i\rangle$.给定句子集 $W\subseteq L(G)$.若对于文法 G 的每个分支 b 都有一个 $w\in W$ 覆盖 b,则称 W 满足文法 G 的上下文依赖规则覆盖,简称分支覆盖(branch coverage).

例 1:图 1 中的示例文法共有 6 个分支:$\langle p_0,p_1,1\rangle$,$\langle p_0,p_2,1\rangle$,$\langle p_1,p_1,1\rangle$,$\langle p_1,p_2,1\rangle$,$\langle p_1,p_3,3\rangle$,$\langle p_2,p_3,1\rangle$.句子 *id+id+id* 覆盖除$\langle p_0,p_2,1\rangle$之外的所有分支,句子长度为 12;句子 *id* 覆盖分支$\langle p_0,p_2,1\rangle$和$\langle p_2,p_3,1\rangle$,句子长度为 4.句子集 $W=\{id+id+id,id\}$满足该文法的分支覆盖,W 的句子长度为 12.

定义 4. 给定文法 G 的两个分支 $b_1=\langle p,q,i\rangle$,$b_2=\langle p',q',i'\rangle$.若满足 $q=p'$,则称 b_1 可直接引入 b_2.定义引入关系为直接引入关系的传递闭包.b_1 引入 b_2 依次经过的分支序列称为 b_1 到 b_2 的路径.

例 2:图 1 示例文法中,分支$\langle p_0,p_1,1\rangle$能够直接引入$\langle p_1,p_1,1\rangle$,$\langle p_1,p_2,1\rangle$和$\langle p_1,p_3,3\rangle$,能够引入分支$\langle p_2,p_3,1\rangle$,引入路径为$\langle p_0,p_1,1\rangle$,$\langle p_1,p_2,1\rangle$,$\langle p_2,p_3,1\rangle$;分支$\langle p_0,p_2,1\rangle$能够直接引入分支$\langle p_2,p_3,1\rangle$,除此之外不能引入其他分支.

p_0: $S\to E$
p_1: $E\to E+T$
p_2: $E\to T$
p_3: $T\to id$

Fig.1 Sample grammar

图 1 示例文法

2 算法基本思想

给定文法 G,设 L 是所有满足 G 分支覆盖句子集的句子长度(见定义 2)集合,则 L 中存在一个最小值.称这个最小值为文法 G 分支覆盖的长度阈值,记为 $BL(G)$.长度阈值在进行句子长度控制时具有重要参考价值.文献[5]利用规则覆盖的长度阈值 $RL(G)$实现长度控制.其基本思想是,在替换非终极之前,首先预测由当前句型能够生成的最短句子长度.若此长度已超过 $RL(G)$,则使用能够到达最短推导的产生式进行替换;否则,仍按照 Purdom 原有策略选择产生式.由于按照 Purdom 策略选择的产生式有可能会引入新的尚未覆盖的产生式(详见文献[6,7]),而这段引入长度在进行长度预测时没有考虑在内,因此,最终句子长度仍有可能超过 $RL(G)$.此外,Purdom 算法本身对产生式顺序比较敏感[6,7].例如,若将图 1 示例文法中的产生式 p_1 和 p_2 的顺序调换,则 Purdom 算法生成的句子集为$\{id,id+id\}$.显然,句子 *id* 是冗余的,因为 *id+id* 已经覆盖了所有的产生式.文献[5]和文献[11]中算法都保留了 Purdom 算法的这种产生式顺序敏感性,因此生成的句子集可能有冗余.

本文将采用更严格的长度控制策略和完全不同于 Purdom 算法的句子生成顺序,在保证满足分支覆盖的前提下,使得句子集的句子长度最短(即恰好等于长度阈值 $BL(G)$),句子集无冗余且规模较小.

为了计算 $BL(G)$,首先给出下面的定理:

定理 1. 给定文法 G,其分支覆盖长度阈值 $BL(G)=\max\{blen[b]|b\in BRAN(G)\}$.其中,$blen[b]$表示覆盖分支 b 的最短句子长度,$BRAN(G)$表示文法 G 的分支集合.

证明:设 L 是所有满足 G 分支覆盖句子集的句子长度集合,只需证明 $\max\{blen\}$是 L 中的元素,并且是最小元素即可(这里,用 $\max\{blen\}$代指 $\max\{blen[b]|b\in BRAN(G)\}$).假设 w_b 是覆盖分支 b 的最短句子,构造句子集 $W=\{w_b|b\in BRAN(G)\}$.显然,W 满足 G 的分支覆盖.而 $\max\{blen\}$是 W 中句子长度最大值,因此在集合 L 中.假设 b' 是满足 $blen[b']=\max\{blen\}$的分支,根据 $blen[b']$的定义,任何覆盖 b'的句子,其长度都将大于等于 $blen[b']$.对于 L 中的任意元素 l,根据 L 的定义可知,l 也必将大于等于 $blen[b']$.因此,$blen[b']$也即 $\max\{blen\}$是 L 中的最小元素.证毕. □

从定理 1 可以看出,为使生成的句子集的句子长度最短,也即句子最大长度恰好等于分支覆盖长度阈值,可以按照 *blen* 值从大到小对分支进行排序,每次选出尚未覆盖且 *blen* 值最大的那个分支,生成覆盖该特定分支的

句子,直到所有分支全部覆盖.算法的基本框架描述如下:

算法框架:

(1) 按照 *blen* 值从大到小对所有分支排序;

(2) 在尚未覆盖的分支中选出 *blen* 值最大的分支;

(3) 生成覆盖该特定分支且长度不超过阈值的句子;

(4) 重复步骤(2)和步骤(3),直到所有分支都被覆盖.

其中,关键问题是如何计算分支的 *blen* 值以及如何生成覆盖特定分支且长度不超过阈值的句子;同时,为了控制句子集规模较小,在生成覆盖特定分支的句子的过程中,如何尽可能多地覆盖新的分支.我们在下面的算法实现中给出具体描述.

3 算法实现

3.1 准备阶段

准备阶段主要收集文法的分支信息,计算各分支的 *blen* 值和长度阈值 *BL*(*G*)(以下简称 *BL*).给定分支⟨*p*,*q*,*i*⟩,设 *q* 左部非终极符为 *A*.显然,覆盖该分支的句子首先必须覆盖产生式 *p*;其次,推导过程中,对 *p* 右部位置 *i* 上的非终极符 *A* 进行替换时,至少有 1 次必须使用产生式 *q*.因此,分支⟨*p*,*q*,*i*⟩的 *blen* 值可通过如下公式计算:

$$blen[\langle p,q,i\rangle]=clen[p]-slen[A]+rlen[q] \tag{1}$$

其中,*clen*[*p*]表示覆盖产生式 *p* 的最短句子长度;*slen*[*A*]表示从非终极符 *A* 能够推导出的最短终极符串长度;*rlen*[*q*]表示当使用产生式 *q* 作为第 1 步推导时,从其左部非终极符 *A* 能够推导出的最短终极符串长度.

以图 1 中的文法为例,考虑分支⟨p_1,p_1,1⟩:

(1) 覆盖产生式 p_1 的最短句子是 *id*+*id*,最左推导序列为 $S\Rightarrow^{p_0}E\Rightarrow^{p_1}E+T\Rightarrow^{p_2}T+T\Rightarrow^{p_3}id+T\Rightarrow^{p_3}id+id$,句子长度为 8;

(2) 从非终极符 *E* 能够推导出的最短终极符串是 *id*,推导序列为 $E\Rightarrow^{p_2}T\Rightarrow^{p_3}id$,长度为 3;

(3) 当使用产生式 p_1 作为第 1 步推导时,从非终极符 *E* 能够推导出的最短终极符串是 *id*+*id*,推导序列为 $E\Rightarrow^{p_1}E+T\Rightarrow^{p_2}T+T\Rightarrow^{p_3}id+T\Rightarrow^{p_3}id+id$,长度为 7.

容易看出,步骤(2)中的推导是步骤(1)中推导的一部分,否则,后者不可能是覆盖 p_1 的最短推导.将该部分替换为步骤(3)中的推导,得到的即是覆盖分支⟨p_1,p_1,1⟩的最短推导,其序列为 $S\Rightarrow^{p_0}E\Rightarrow^{p_1}E+T\Rightarrow^{p_1}E+T+T\Rightarrow^{p_2}T+T+T\Rightarrow^{p_3}id+T+T\Rightarrow^{p_3}id+id+T\Rightarrow^{p_3}id+id+id$,句子长度为 12.*slen*,*rlen* 和 *clen* 的计算分别见文献[6,7]和文献[5],这里不再赘述.

计算出各个分支的 *blen* 值后,选出最大值,即是分支覆盖的长度阈值 *BL*.在下面进行句子生成时,根据该阈值进行长度控制.图 1 中示例文法的长度阈值是 12,对应于分支⟨p_1,p_1,1⟩的 *blen* 值.

3.2 句子生成阶段

句子的生成采用最左推导,使用一个栈来实现.栈中的元素是三元组⟨*p*,*q*,*i*⟩,表示产生式 *p* 右部位置 *i* 上的符号 *s*.初始时⟨*null*,*S*,0⟩入栈.每次从栈中弹出一个含有非终极符 *A* 的元素⟨*p*,*A*,*i*⟩时,选择一个以 *A* 为左部的产生式 *q* 来替换 *A*,同时标记分支⟨*p*,*q*,*i*⟩被覆盖.给定分支 *b*,覆盖该分支的单个句子生成算法描述如下:

算法 1. 单个句子生成算法.

输入:分支 *b*.

输出:覆盖该特定分支 *b* 的单个句子.

```
01: Procedure genOneSentence(b:Branch)
02: stack.push(⟨null,S,0⟩);                    //栈初始化
03: expSL=blen[b];                             //长度预测值初始化
04: while (not stack.empty()) do
```

```
05:  ⟨p,s,i⟩=stack.pop();
06:  if (s is terminal) then print s;
07:  else
08:      (q:s→α)=chooseProduction(b,⟨p,s,i⟩,expSL);          //选择产生式
09:      cover[p,q,i]=true;                                  //标记该分支已被覆盖
10:      for each s∈α in reverse order do                    //替换后的三元组入栈
11:          j=position of s in α;
12:          stack.push(⟨q,s,j⟩);
13:      end for
14:      update(expSL);                                      //更新长度预测值
15:  end if
16: end while
```

该算法使用变量 *expSL* 预测最终生成的句子长度.在进行产生式选择时,根据该预测值和长度阈值的对比来进行长度控制.

3.2.1 产生式选择

产生式选择策略是整个算法中的关键部分.选择策略需要兼顾到以下几个要求:

(1) 覆盖特定的分支;

(2) 最终句子长度不超过长度阈值;

(3) 最终生成的句子集规模较小.

下面首先给出产生式选择算法,然后进行解释.

算法 2. 产生式选择算法.

输入:待覆盖的特定分支 *b*,当前栈中弹出的元素⟨*p*,*A*,*i*⟩,当前句子长度预测值 *expSL*.

输出:用于替换非终极符 *A* 的产生式 *q*.

```
01: Function chooseProduction(b:Branch,⟨p,A,i⟩:StackElement,expSL:Integer)
02: if ∃q st.⟨p,q,i⟩∈path(⟨null,p0,0⟩,b) then return q;          //选择规则(1)
03: else if ∃q st.⟨p,q,i⟩ uncovered and new_expSL≤BL then return q;   //选择规则(2)
04:     else if ∃q st.⟨p,q,i⟩ introduces an uncovered ⟨p′,q′,i′⟩ and ⟨p′,lhs(q′),i′⟩ not in stack
                    and new_expSL≤BL then return q;              //选择规则(3)
05:         else return short[A];                               //选择规则(4)
06:         end if
07:     end if
08: end if
```

假设要生成覆盖分支 *b* 的句子,产生式选择顺序如下:

(1) 若存在 *q*,分支⟨*p*,*q*,*i*⟩位于初始分支⟨*null*,p_0,0⟩到分支 *b* 的最短路径上,则选择该 *q*(这里,p_0 表示以初始符 *S* 为左部的产生式.不失一般性,我们假定以 *S* 为左部的产生式只有 1 个).

(2) 若存在 *q*,分支⟨*p*,*q*,*i*⟩尚未覆盖且使用 *q* 后的句子长度预测值不超过长度阈值,则选择该 *q*.

(3) 若存在 *q*,分支⟨*p*,*q*,*i*⟩能引入一个新的尚未覆盖的分支⟨*p*′,*q*′,*i*′⟩且⟨*p*′,*lhs*(*q*′),*i*′⟩尚未在栈中,并且引入后的句子长度预测值不超过长度阈值,则选择该 *q*.

(4) 选择能够到达最短推导的产生式 *q*.

其中,优先选择规则(1)是为了保证特定分支能被覆盖到.规则(2)和规则(3)借鉴 Purdom 算法中的选择策略,在生成单个句子时尽可能多地覆盖新的分支.当已经没有满足前 3 个规则的产生式时,选择最短推导的那个.容易看出,规则(1)和规则(4)的选择能够确保最终的句子长度在阈值范围之内,规则(2)和规则(3)的选择可能导致

句子长度超出阈值,因此需要加上长度控制.选择规则(1)时,需要用到初始分支到分支 b 的最短路径信息.同样,当选择规则(3)后,为了保证$\langle p',q',i'\rangle$确实能够成功地引进,需要用到$\langle p,q,i\rangle$到$\langle p',q',i'\rangle$的最短路径信息.分支间最短路径的计算在下面小节中结合长度预测的计算进行介绍.规则(4)中的 $short[A]$表示以 A 为左部且使得 A 能够推导出最短终极符串的产生式,其计算方法见 Purdom 原始算法[6,7],此处不再赘述.

3.2.2 长度预测

因为每次生成的句子必将覆盖特定分支 b,因此句子长度预测值初始化为 $expSL=blen[b]$.当按照算法 2 中规则(2)和规则(3)选择产生式 q 时,需对其重新计算.考虑当前栈顶元素$\langle p,A,i\rangle$,新的 $expSL$ 计算公式如下:

- 对于选择规则(2)

$$new\_expSL=expSL-slen[A]+rlen[q] \tag{2}$$

- 对于选择规则(3)

$$new\_expSL=expSL-slen[A]+rlen[q]+pathlen[\langle p,q,i\rangle][\langle p',q',i'\rangle] \tag{3}$$

其中,$pathlen[\langle p,q,i\rangle][\langle p',q',i'\rangle]$表示从分支$\langle p,q,i\rangle$到分支$\langle p',q',i'\rangle$的最短路径长度.以上公式都使用了非终极符 A 的 $slen$ 值和产生式 q 的 $rlen$ 值,计算原理和 $blen$ 的计算原理类似(见第 3.1 节),此处不再解释.与公式(2)相比,公式(3)多了一项路径长度,是因为$\langle p,q,i\rangle$引入$\langle p',q',i'\rangle$会额外增加句子的长度,增加的句子长度即是从分支$\langle p,q,i\rangle$到分支$\langle p',q',i'\rangle$的最短路径长度.为了计算分支间的最短路径以及最短路径长度,首先考虑具有直接引入关系的两个分支.

定义 5. 给定分支 $b_1=\langle p_1,p_2,i_1\rangle$,$b_2=\langle p_2,p_3,i_2\rangle$,$b_1$ 直接引入 b_2.定义 b_1 引入 b_2 后的直接引入长度为同时覆盖 b_1 和 b_2 的最短句子长度与覆盖 b_1 的最短句子长度之差.

参考 $blen$ 的计算方法,同时覆盖 b_1 和 b_2 的最短句子长度计算公式为 $blen[b_1]-slen[A]+rlen[p_3]$,其中,$A$ 是 p_3 左部非终极符.因此,b_1 到 b_2 的直接引入长度为

$$pathlen[\langle p_1,p_2,i_1\rangle][\langle p_2,p_3,i_2\rangle]=rlen[p_3]-slen[A] \tag{4}$$

有了直接引入关系和直接引入长度,根据 Floyd-Warshall 算法[13],即可计算出分支间的引入关系、最短引入路径及最短路径长度.这些计算均可在算法的准备阶段完成.

例 3:以图 1 中的示例文法为例,初始分支$\langle null,p_0,0\rangle$引入分支$\langle p_1,p_1,1\rangle$的最短路径为:$\langle null,p_0,0\rangle$,$\langle p_0,p_1,1\rangle$,$\langle p_1,p_1,1\rangle$,最短路径长度为 8;分支$\langle p_0,p_1,1\rangle$引入分支$\langle p_2,p_3,1\rangle$的最短路径为$\langle p_0,p_1,1\rangle$,$\langle p_1,p_2,1\rangle$,$\langle p_2,p_3,1\rangle$,最短路径长度为 0,因为同时覆盖分支$\langle p_0,p_1,1\rangle$,$\langle p_1,p_2,1\rangle$和$\langle p_2,p_3,1\rangle$的最短句子和仅覆盖$\langle p_0,p_1,1\rangle$或$\langle p_1,p_2,1\rangle$的最短句子相同,都是 $id+id$.

需要指出的是,对于上述产生式选择规则(2)和规则(3),当存在多个满足条件的产生式时,我们采取贪心策略[13],选取使 new_expSL 最短的那个.这样做是为了尽量余下较多的可用长度,以便能够容纳更多新的分支进来,从而减小最终的句子集规模.

例 4:对于图 1 中的示例文法,算法首先生成覆盖 $blen$ 值最大的分支$\langle p_1,p_1,1\rangle$的句子 $id+id+id$.该句子同时覆盖分支$\langle p_0,p_1,1\rangle$,$\langle p_1,p_2,1\rangle$,$\langle p_1,p_3,3\rangle$,$\langle p_2,p_3,1\rangle$.然后生成覆盖分支$\langle p_0,p_2,1\rangle$的句子 id.因此,最终生成的句子集是 $\{id+id+id,id\}$,句子长度分别为 12,4.

3.3 算法分析

本文算法与文献[5]中加入长度控制的规则覆盖算法和文献[11]中基于 Purdom 算法的分支覆盖算法的不同之处具体如下:

(I) 长度控制方面.本文算法对于选择规则(2)和规则(3)分别使用不同的长度预测计算公式(见第 3.2.2 节).在规则(3)中,不仅预测了当前句型能推导出的最短句子长度,而且将可能的“引入长度”也预测在内,从理论上保证了最终句子长度在阈值之内.而文献[5]对于所有的选择规则都使用同一个预测公式,没有考虑可能的“引入长度”.文献[11]则没有长度控制.

(II) 句子生成顺序方面.本文算法按照 $blen$ 长度顺序进行生成,保证每个生成的句子都覆盖一个特定的没有被其他句子覆盖的分支,因此最终句子集中无冗余.文献[5]和文献[11]中的算法则是依照产生式顺

序进行生成,可能出现冗余的情况(见第 2 节中的解释).

(III) 句子集规模控制方面.本文算法采用两种贪心策略:

(a) 每次在尚未覆盖的分支中选择 *blen* 值最大的分支生成句子.直观上,*blen* 值越大,说明推导长度越长,在推导中可能覆盖的分支越多.

(b) 对于选择规则(2)和规则(3),当存在多个满足条件的产生式时,选取使得 *new_expSL* 最短的那个,以便余下较多的可用长度容纳新的分支进来.

这两处策略都是为了在生成覆盖特定分支的单个句子时尽可能多地覆盖其他分支,从而减小最终生成的句子集规模.文献[5]和文献[11]没有这样的控制策略.

从以上分析也可以看出,算法满足预期的 3 个目标:

(1) 显然,算法满足分支覆盖;

(2) 根据定义,长度阈值 *BL* 是任何满足分支覆盖的句子集中最大句子长度的最短值,而长度控制策略保证生成的句子长度都不超出阈值,也即保证最大句子长度最短;

(3) 采取不同的句子生成顺序和句子集规模控制策略,保证无冗余,句子集规模较小.

下面分析算法的时间复杂度.显然,算法的运行时间和文法分支个数有关.假定文法的分支个数为$|B|$.准备阶段计算各分支 *blen* 值和利用 Floyd-Warshall 算法计算分支间路径信息所需时间为 $O(|B|+|B|^3)$,即 $O(|B|^3)$.计算 *blen* 所用到的 *slen*,*rlen* 等数据值的计算方法来自于 Purdom 算法[6,7],但是,Purdom 算法中没有给出相应的复杂度分析.我们在进行实验过程中发现,计算这些数据值所占的时间比重一般小于计算分支间路径信息的时间比重.因此,将准备阶段所需时间计为 $O(|B|^3)$.句子生成阶段运行时间取决于最终生成的句子个数,以及生成单个句子时栈的操作次数和每次选择产生式所用的时间.栈的操作次数正比于句子长度.总的操作次数正比于生成句子的长度总和,最坏情况下为 $O(|B|\times BL)$,其中,*BL* 是文法分支覆盖的长度阈值.选择产生式所花费的时间主要在于选择规则(3),正比于分支$\langle p,q,i\rangle$能引入的分支个数,最坏情况下为$|B|$.因此,算法总的时间复杂度为 $O(|B|^3+|B|^2\times BL)$.文献[11]中的分支覆盖算法也是使用栈进行句子生成,产生式选择规则包括本算法中的规则(2)、规则(3)**和规则(4),但是没有进行长度控制.类似地分析,如果不考虑准备阶段所需时间,其时间复杂度为 $O(|B|^2\times BL)$.由于本文算法加入了长度控制、冗余消除等策略,因此比文献[11]中算法多出了计算分支间路径信息所用的时间.当分支个数小于长度阈值时,两者时间复杂度相同.

4 实验情况

我们考察了本文算法生成的句子特性以及算法在软件测试中的应用情况,并与文献[11]中算法进行了比较.在下文中,分别使用 BC-S 和 BC-P 来指代这两种算法.

4.1 生成的句子特性

我们选择 6 个文法来进行实验,包括:bExpr,布尔表达式文法;polishExpr,逆波兰表达式文法;elemFunc,初等函数文法;miniPascal,简化的类 Pascal 文法;ANSI-C 文法和 Java 文法.其中,前 4 个文法来自形式规约获取系统 SAQ[14],后 2 个来自 compilers ftp(ftp://ftp.iecc.com/pub/file).表 1 左部给出各文法的产生式个数以及分支覆盖长度阈值,右部描述句子生成情况.我们分别统计了两种算法生成的句子个数、最短、最长和平均句子长度,以及句子长度标准差.标准差计算公式为 $\sqrt{\sum_{i=1}^{n}(l_i-\bar{l})^2/n}$, 其中,$l_i$ 和 $\bar{l}$ 分别表示第 i 个句子长度和平均句子长度,n 表示句子个数.标准差在这里用于显示句子长度的均匀程度.

从表 1 可以看出,BC-P 算法生成的句子个数相对文法规模而言比较少,而且长度差异大,最长的句子结构很

** 对于规则(3),文献[11]中使用的"引入"概念是本文定义的"引入"概念的一种受限情况,在句子生成阶段,动态地在 $O(|B|)$时间内计算出.具体参见文献[6,7,11].

复杂.比如,elemFunc 文法含有 38 条产生式,但只生成两个句子:一个很短(长度为 8),另一个很长(长度 505).BC-S 算法则很好地克服了这些缺点:生成的句子较短,且保证最大句子长度最短(恰好等于长度阈值),句子数量相对于文法规模来说也较为合适.除此之外,从最后一列可以看出,BC-S 算法生成的句子还有一个特性:长度很均匀.从本质上来说,两种算法的区别在于,BC-P 算法将文法的分支尽可能多地集中在 1 个或少数几个长句子中覆盖,而本文算法则是将文法的分支均匀分散在多个短句子中覆盖.

Table 1 Results of sentence generation

表 1 句子生成情况

Grammar	Number of rules	*BL*	Number of sentences		Minimum length		Maximum length		Average length		Length standard deviation	
			BC-S	BC-P	BC-S	BC-P	BC-S	BC-P	BC-S	BC-P	BC-S	BC-P
bExpr	9	15	7	2	14	9	15	55	15	32	1.0	23.0
polishExpr	12	17	3	2	15	5	17	32	16	19	0.8	13.5
elemFunc	38	26	32	2	16	8	26	505	25	257	2.2	248.5
miniPascal	81	71	18	4	47	23	71	329	66	191	7.2	120.8
ANSI-C	213	100	144	75	18	8	100	2 131	93	115	16.2	246.0
Java	282	83	361	222	1	1	83	4 889	73	86	16.8	330.6

4.2 在软件测试中的应用

为了检验本文算法在应用中的有效性,我们选择一种基于文法的形式规约语言 LFC[15]来进行实验.LFC 是一种可执行的规约语言,它使用上下文无关文法和文法上的递归函数来分别表示规约的语法和语义.LFC 程序由两部分组成:文法定义和函数定义.在对函数定义进行测试时,可以利用文法的句子生成算法自动获取测试数据.我们设计了两组实验来验证本文算法在 LFC 函数测试中的有效应用:第 1 组实验通过变异测试,检验生成的测试数据的揭错能力;第 2 组实验通过统计完成测试任务的时间来检验对测试速度的提高程度.表 2 简要描述了用于实验的 5 个 LFC 程序.这些程序均来自形式规约获取系统 SAQ[14],程序中用到的文法分别是第 4.1 节中的前 4 个实验文法.这里所指的有效行数不包括文法定义、函数声明、变量声明等语句所在程序行.

Table 2 Experimental programs

表 2 实验程序

Program	Grammar	Description	Line of codes
bEval	bExpr	Evaluation of Boolean expressions	8
pEval	polishExpr	Evaluation of reverse Poland expressions	11
Diff	elemFunc	Formal differentiation of elementary functions	34
Diff2	elemFunc	Formal differentiation of elementary functions plus result reduction	46
idCount	miniPascal	Odentifier counting in Pascal-like programs	60

4.2.1 测试数据的揭错能力

我们采用变异测试[16,17]的技术来检验本文算法生成的测试数据用于 LFC 函数测试时的揭错能力.变异测试是一种衡量测试数据集的揭错能力、评估测试充分性的重要方法.其基本原理是,使用变异算子对被测程序做微小的合乎语法的变动,例如,将二元运算符“+”用“–”替换等,产生大量的新程序,每个新程序称为一个变异体.根据给定的测试数据运行变异体,若变异体的运行结果和原程序的运行结果不同,则称该变异体被杀死.杀死的变异体越多,说明测试数据的揭错能力越强.

变异测试的关键是如何选择变异算子产生变异体.变异算子可以针对不同的语法成分,例如关系运算符、表达式、变量等来进行设计.文献[18]对传统程序中常见的 22 种变异算子进行分析比较,总结出 5 种关键的核心算子,见表 3.本文使用这 5 种核心算子对 LFC 程序进行变异测试,测试过程如图 2 所示.首先,对每个 LFC 程序作用这 5 种变异算子,得到一系列变异体.然后,利用 LFC 编译器对原程序和变异体进行编译,得到可执行程序.根据同一组测试数据,分别运行原程序和变异体,比较两者的输出是否完全相同.若不相同,则标记该变异体被该测试集杀死.

Table 3 Mutation operators used in the experiments

表 3 实验中使用的变异算子

Mutation operator	Description in English	Explanation in Chinese
ABS	Absolute value insertion	将表达式分别替换为0,一个正数和一个负数
AOR	Arithmetic operator replacement	将二元算术运算符分别替换为其他的二元算术运算符
LCR	Logical connector replacement	将二元逻辑运算符分别替换为其他的二元逻辑运算符
ROR	Relational operator replacement	将二元关系运算符分别替换为其他的二元关系运算符
UOI	Unary operator insertion	在表达式前插入一个一元运算符

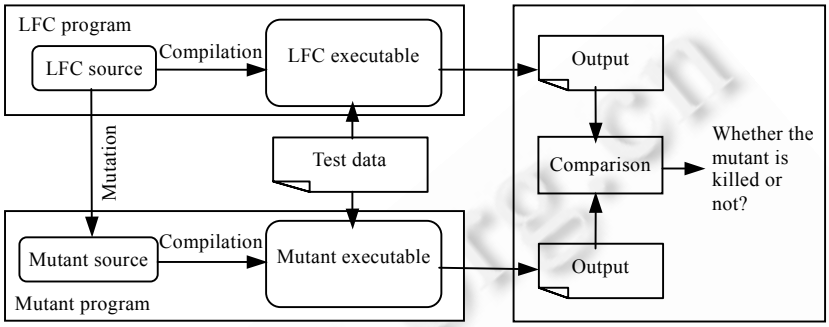


Fig.2 Process of mutation testing

图 2 变异测试过程

表 4 分别给出了各 LFC 程序的变异体个数以及两种算法生成的测试数据所杀死的变异体数目及比率.比率计算公式为:(杀死变异体个数/总变异体个数)×100%.从比率情况来看,两种算法都在 50%以上.对比文献[9]中对基于规则覆盖的测试数据所做的 3 例实验中,杀死变异体的比率分别只有 3.4%,39.7%和 0%.虽然实验程序不同,但是从某种程度上也反映出基于分支覆盖句子集在软件测试中比基于规则覆盖句子集的揭错能力要强.从两种算法的对比情况来看,BC-S 算法生成的测试数据杀死变异体的效果也即揭错能力,要好于 BC-P 算法:除了对于 idCount 程序本文算法效果略低之外,对其他程序都等于或者超过 BC-P 算法.

Table 4 Experimental results of mutation testing

表 4 变异测试的实验结果

Program	Number of mutants	Number of killed mutants		Ratio of killed mutants (%)	
		BC-S	BC-P	BC-S	BC-P
bEval	23	16	9	69.6	39.1
pEval	122	105	104	86.1	85.3
Diff	130	120	120	92.3	92.3
Diff2	273	147	118	53.9	43.2
idCount	650	514	516	79.1	79.4

4.2.2 完成测试任务的时间

对于简单短句子作为测试数据而言,其直观的优点就是,当发现错误时利于错误定位和进行跟踪调试.此外,我们还通过实验验证了其有利于提高测试人员完成测试任务的速度.简单来说,程序测试一般分为以下几步[19]:设计测试用例,包括输入(即测试数据)和预期输出;运行程序计算实际输出;比较实际输出和预期输出.预期输出通常需要人工计算.我们分别统计了使用两种算法进行 LFC 程序测试所花费的时间,如图 3 所示.时间共分两部分:(a) 由人工计算出预期输出的时间.我们共统计了两位测试人员分别计算出预期输出的时间,取其平均值.(b) 由机器运行算法得到测试数据和运行程序计算实际输出并与预期输出相比较的时间.运行环境:CPU 为 Pentium 4,3.20GHz,内存大小为 1GB,操作系统为 Ubuntu 7.10.

由实验可以看出,虽然 BC-S 算法生成的测试数据较多,但是由于每个都比较简单,可以快速地计算出预期输出结果.相反,BS-P 算法则需花费更多的时间,尤其是对于 Diff 和 Diff2 两个程序.两个算法在其他方面所花费

的时间则相差不太大(如图 3 中右图所示),但这些时间在完成整个测试任务中所占的比重不是很大.所以总体来说,与 BC-P 算法相比,本文算法能够帮助测试人员提高测试速度,从而利于降低测试代价.

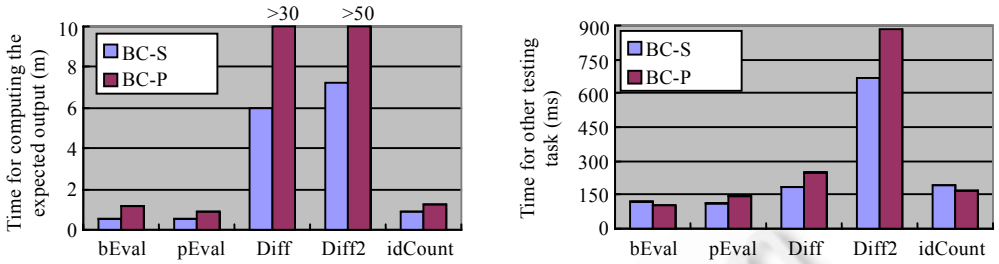


Fig.3 Time consumed by the testing task

图 3 完成测试任务所花费的时间

4.2.3 测试效率综合分析

首先,单独分析 BC-S 算法在揭错能力和测试时间方面相对于 BC-P 算法提高(或降低)的百分比.揭错能力提高百分比根据杀死变异体比率进行计算,测试时间降低百分比根据总测试时间进行计算.对于 Diff 和 Diff2 两个程序,将 BC-P 算法对应的计算预期输出时间分别取为 30min 和 50min.如表 5 第 2 列和第 3 列所示:(1) 揭错能力方面,除 Diff 和 idCount 程序外,其他都有不同程度的提高;(2) 测试时间方面,5 个程序都有显著地降低.

下面将揭错能力和测试时间两方面综合起来分析.记测试时间为 T,测试数据的揭错比率为 E.可以将此直观地理解为测试人员使用该测试数据进行测试时,在时间 T 内可能发现的程序错误的比率为 E.因此,引入一个新的参数 E_T=揭错比率/测试时间,表示单位时间内揭示程序错误的能力,以此来衡量测试数据对测试任务的综合贡献率.两种算法的 E_T 值及综合效率提高百分比见表 5 后 3 列,单位时间为 min.需要指出的是,当测试时间小于 1min 时,直接将揭错比率取为其 E_T 值而不是按照上述公式计算 E_T 值.如表 5 所示,在 5 个实验程序中,综合测试效率少者提高 5.5%,多者可以提高 7 倍多.

Table 5 Improvement ratio of testing efficiency

表 5 测试效率提高百分比

Program	Improvement in fault detection ability (%)	Reduction in testing time (%)	E_T (%)		Improvement of testing efficiency (%)
			BC-S	BC-P	
bEval	78.0	52.5	69.6	29.9	132.8
pEval	0.9	41.6	86.1	81.6	5.5
Diff	0	80.0	14.9	3.1	380.6
Diff2	24.8	84.5	6.9	0.8	762.5
idCount	−0.4	19.3	71.0	57.6	23.3

5 相关工作

文法的句子生成是形式语言理论中一个基本的研究问题,在软件测试数据自动生成方面有着广泛的应用.Harford[10]于 1970 年提出一种句子生成算法,用于生成 PL/I 编译器的测试数据.该算法从 PL/I 文法开始符开始,每次遇到非终极符 A 时,随机选择一种以 A 为左部的产生式替换 A.不断重复该过程,直到替换后的串不含非终极符为止.生成多个句子时,多次重复上面过程.Maurer[20]在随机选择的基础上为每个产生式增加一个权值,根据权值的大小来决定产生式被使用的概率,使得产生式的不同重要性在生成的句子集中得到体现.随机选择产生式的方法实现起来比较简单,生成句子的个数可由用户灵活控制.其缺点是无法控制生成句子的长度、推导深度以及生成过程何时结束.当生成的句子个数较少时,不具有足够的代表性.

Purdom[6]于 1972 年提出一种满足文法规则覆盖准则的句子生成算法.给定文法 G,算法首先收集 G 中非终极符和产生式的相关信息,在句子生成阶段,根据此信息有选择性地选取产生式,使得每个产生式至少被用到 1 次.由于 Purdom 算法解释得比较模糊并且算法本身复杂难以理解,Malloy 等人[7]对其进行了结构化解释并实现

模块化.后续很多工作都是在 Purdom 算法的基础上进行修改和扩充.文献[21]对 Purdom 算法进行了两处扩展:一是提供了最小化和最大化两种产生式选择策略.即当非终极符 A 对应的产生式全部用完以后,每次要么都选用最短的产生式,要么都选用最长的产生式进行替换;二是给每个产生式增加一个自然数作为权值,标识该产生式被覆盖的最少次数.文献[22]扩展 Purdom 算法,使其能够处理一种新的文法——上下文无关参数文法.这种文法是上下文无关文法的扩展[22],能够同时描述程序语言的语法和语义两个方面,因此能够生成不仅合乎语法同时也合乎语义的句子(程序).文献[5]针对 Purdom 算法生成的句子数量较少,并且有一些过于复杂(长)的特点,加入一个长度控制策略,使得生成的满足规则覆盖的句子比较简单(短),并考察了新算法在文法测试方面的应用.文献[11]将 Purdom 算法由规则覆盖扩展到更加复杂的分支覆盖,使得生成的句子更有代表性,但同时也保留了 Purdom 算法原有的特点.本文则实现了一种满足分支覆盖的算法——加入长度控制机制.本文的长度控制比文献[5]中的更严格,能够保证最终的句子长度全部在分支覆盖长度阈值之内,同时消除冗余,并控制句子集规模较小.

Gore 等人[23]提出一种随机提取句子生成算法.给定任意上下文无关文法 G 和自然数 n,算法在多项式时间内随机生成一个长度为 n 的句子.Wang 等人[24]将文法的产生式分为可以导致句子不断加长的核心产生式和使句子推导结束的外围产生式,通过对不同类型产生式的合理选择,生成符合一定长度限制的句子.这两种算法都涉及到了句子的长度控制,但是和随机生成算法一样,不能保证最终生成的句子满足一定的文法覆盖准则.Li 等人[25]分析了句子推导过程中选择产生式所要依据的不同因素,例如产生式期望使用的次数、已经使用的次数等,定义了用于选择产生式的计算公式,在推导过程中,启发式地动态调整推导策略.其方法中考虑到了文法的规则覆盖准则,没有考虑分支覆盖准则.

Härm 等人[26]针对属性文法的句子生成提出了二维相似覆盖的概念,分别考虑了底层上下文无关文法(语法)和属性(语义)两方面的覆盖,并给出了一种满足该覆盖准则的属性文法句子生成算法.

与句子生成相关的工作还有句子枚举[27−29],用于生成文法的全部句子.两者在采取的技术上有所差别.限于篇幅,本文不再赘述.

6 结束语

本文提出一种基于文法分支覆盖的短句子生成算法.分支覆盖是一种比规则覆盖精度更高的文法覆盖准则,基于分支覆盖准则生成的句子集能够更好地反映文法的内部结构,更具有代表性.短句子在实际测试应用中具有便于错误定位和跟踪调试的优点.本文算法结合长度控制、冗余消除和句子集规模控制等策略,使得生成的满足分支覆盖的句子比较短小简单、无冗余并且句子集规模较小.实验结果表明,该算法生成的测试数据具有较强的程序揭错能力,并且能够帮助测试人员提高测试速度,在基于文法的软件系统的测试中有较高的应用价值.

References:

[1] Mernik M, Črepinšek M, Kosar T, Rebernak D, Žumer V. Grammar-Based systems: Definition and examples. Informatica, 2004, 28(3):245−255.

[2] Klint P, Lämmel R, Verhoef C. Towards an engineering discipline for grammarware. ACM Trans. on Software Engineering and Methodology, 2005,14(3):331−380. [doi: 10.1145/1072997.1073000]

[3] Lämmel R. Grammar testing. In: Hussmann H, ed. Proc. of the 4th Int'l Conf. on Fundamental Approaches to Software Engineering. Springer-Verlag, 2001. 201−216. [doi: 10.1007/3-540-45314-8_15]

[4] Zheng LX, Chen HM. A systematic framework for grammar testing. In: Proc. of the 8th IEEE/ACIS Int'l Conf. on Computer and Information Science. Washington: IEEE Computer Society, 2009. 1013−1019. [doi: 10.1109/ICIS.2009.193]

[5] Zheng LX, Wu DY. A sentence generation algorithm for testing grammars. In: Proc. of the 33rd Annual Int'l Computer Software and Applications Conf. Washington: IEEE Computer Society, 2009. 130−135. [doi: 10.1109/COMPSAC.2009.193]

[6] Purdom P. A sentence generator for testing parsers. BIT Numerical Mathematics, 1972,12(3):366−375. [doi: 10.1007/BF01932308]

[7] Malloy BA, Power JF. An interpretation of Purdom's algorithm for automatic generation of test cases. In: Proc. of the 1st Int'l Conf. on Computer and Information Science. 2001.

[8] Woo1 G, Chael HS, Jang H. An intermediate representation approach to reducing test suites for retargeted compilers. In: Abdennahder N, Kordon F, eds. Proc. of the 12th Int'l Conf. on Reliable Software Technologies. Springer-Verlag, 2007. 100−113. [doi: 10.1007/978-3-540-73230-3_8]

[9] Hennessy M, Power JF. Analysing the effectiveness of rule-coverage as a reduction criterion for test suites of grammar-based software. Empirical Software Engineering, 2008,13(4):343−368. [doi: 10.1007/s10664-008-9067-7]

[10] Hanford KV. Automatic generation of test cases. IBM Systems Journal, 1970,9(4):242−257. [doi: 10.1147/sj.94.0242]

[11] Shen Y, Chen HM. Sentence generation based on context-dependent rule coverage. Computer Engineering and Applications, 2005, 41(17):96−100 (in Chinese with English abstract).

[12] Hopcroft JE, Motwani R, Ullman JD. Introduction to Automata Theory, Languages, and Computation. 3rd ed., Addison-Wesley, 2006.

[13] Cormen TH, Leiserson CE, Rivest RL, Stein C. Introduction to Algorithms. 2nd ed., MIT Press and McGraw-Hill, 2001.

[14] Dong YM, Li KD, Hu YQ, Zhang RL, Tang RQ, Wang ZY, Chen ZM. Design and implementation of the formal specification acquisition system SAQ. In: Proc. of the Conf. on Software Theory and Practice, IFIP 16th World Computer Congress. 2000. 201−211.

[15] Chen HM, Dong YM. Facilitating formal specification acquisition by using recursive functions on context-free languages. Knowledge-Based Systems, 2006,19(2):141−151. [doi: 10.1016/j.knosys.2005.10.005]

[16] DeMillo RA, Lipton RJ, Sayward FG. Hints on test data selection: Help for the practicing programmer. IEEE Computer, 1978,11(4):34−41. [doi: 10.1109/C-M.1978.218136]

[17] Hamlet RG. Testing programs with the aid of a compiler. IEEE Trans. on Software Engineering, 1977,SE-3(4):279−290. [doi: 10.1109/ TSE.1977.231145]

[18] Offutt AJ, Lee A, Rothermel G, Untch RH, Zapf C. An experimental determination of sufficient mutant operators. ACM Trans. on Software Engineering and Methodology, 1996,5(2):99−118. [doi: 10.1145/227607.227610]

[19] Myers GJ. The Art of Software Testing. 2nd ed., New Jersey: John Wiley & Sons, Inc., 2004.

[20] Maurer PM. Generating test data with enhanced context-free grammars. IEEE Software, 1990,7(4):50−55. [doi: 10.1109/52.56422]

[21] Celentano A, Crespi-Reghizzo S, Della-Vigna P, Ghezzi C, Granata G, Savoretti F. Compiler testing using a sentence generator. Software Practice and Experience, 1980,10(11):897−918. [doi: 10.1002/spe.4380101104]

[22] Bazzichi F, Spadafora I. An automatic generator for compiler testing. IEEE Trans. on Software Engineering, 1982,SE-8(4): 343−353. [doi: 10.1109/TSE.1982.235428]

[23] Gore V, Jerrum M, Kannan S, Sweedyk Z, Mahaney S. A quasi-polynomial-time algorithm for sampling words from a context-free language. Information and Computation, 1997,134(1):59−74. [doi: 10.1006/inco.1997.2621]

[24] Wang HG, Dong YM. Generating sentences of CFL based on partition of CFG production set. Journal of Software, 2000,11(8): 1030−1034 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/11/1030.htm

[25] Li H, Liu C, Jin MZ, Xu F. Generating sentences of CFG by heuristics. Computer Science, 2007,34(9A):38−41 (in Chinese with English abstract).

[26] Härm J, Lämmel R. Two-Dimensional approximation coverage. Informatica, 2000,24(3):355−369.

[27] Mäkinen E. On lexicographic enumeration of regular and context-free language. Acta Cybernetica, 1997,13(1):55−61.

[28] Dömösi P. Unusual algorithms for lexicographical enumeration. Acta Cybernetica, 2000,14(3):461−468.

[29] Dong YM. Counting and hierarchical lexicographic enumeration of CFL sentences. Science in China (Series E), 2006,36(12): 1375−1413 (in Chinese with English abstract).

附中文参考文献:

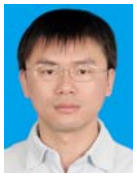
[11] 沈扬,陈海明.基于上下文依赖规则覆盖的句子生成.计算机工程与应用,2005,41(17):96−100.

[24] 王泓皓,董韫美.基于产生式集划分的上下文无关语言句子生成.软件学报,2000,11(8):1030−1034. http://www.jos.org.cn/1000-9825/11/1030.htm

[25] 李虎,刘超,金茂忠,许福.一种上下文无关文法的启发式句子生成方法.计算机科学,2007,34(9A):38−41.

[29] 董韫美.CFL 句子计数和分层词典序枚举.中国科学(E 辑),2006,36(12):1375−1413.

郑黎晓(1983－),女,河南南阳人,博士生,主要研究领域为软件设计方法,形式规约程序语言.



许智武(1983－),男,博士生,CCF 学生会员,主要研究领域为软件设计方法,形式规约程序语言.

陈海明(1966－),男,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为软件设计方法,形式规约,程序语言.