中国科学(E辑)
第32卷 第1期 SCIENCE IN CHINA (Series E) 2002年2月

上下文无关语言上的递归函数*
——Ⅰ. CFPRF及CFRF的定义

董韫美
(中国科学院软件研究所计算机科学开放研究实验室,北京 100080)

摘要 建立上下文无关语言(CFL)上的递归函数理论. 在CFL上定义了函数类CFRF和它的真子类CFPRF,它们可用来十分直接地表述非数值加工算法. 事实上它们分别就是上下文无关语言上的偏递归函数和原始递归函数. 提出了证明CFPRF函数性质的结构归纳法,给出一种枚举CFL句子的方法,定义了极小算子. 基于CFL句子枚举,提出了极小算子的求值方法. 最后,讨论了以CFRF为理论基础的可执行规约语言的设计和实现原则.

关键词 CFRF CFPRF 上下文无关语言的递归函数 CFL分层枚举 结构归纳法

本文假定上下文无关语言(CFL)理论和递归函数论的知识[1,2].

为研究可计算性,20世纪30年代发展了自然数的一般递归函数理论[3]. 随着计算机的发展,20世纪60年代发展了字上的递归函数[4,5]. 文献[5]的工作与Lisp语言紧密相关,Lisp从提出以来的数十年中,对人工智能的研究发展起了巨大的推动作用. 60年代初,字上的递归谓词也被引入到形式文法的研究中,并且用字上的递归函数表达一类形式文法的通用分解判定算法[6].

本文引入CFL上的函数类CFRF及其子类CFPRF,这种函数的定义域和值域是CFL. 于是自变量和函数值具有(可以是不同的)短语结构,同时还包含自然数和字上的函数为其特殊情形. 在以后的工作中,我们将证明,函数类CFRF和CFPRF分别是CFL上的偏递归函数和原始递归函数[1)].

在可计算性理论研究中,自然数的递归函数起了根本的作用,然而字上递归函数的引入仍然是一大进步. 没有它,字上的问题要通过编码等一类手段,化归为自然数的递归函数,以便讨论. 当然也可以不用递归函数,而用正规算法或者图灵机. 但是它们缺乏易构造性,即不像递归函数那样,可以方便地由简单步骤出发,逐步构造出越来越复杂的算法. 因而在实际使用中,还是递归函数来得方便. Lisp语言的广泛使用,亦为字上递归函数用途的明证. 另外,Lisp也可以看成是一种特殊的CFL,即List上定义的递归函数.

如果再行深究,就会发现,计算机上许多字的算法,用CFRF来描述,更为自然简捷. 特别

2000-10-23收稿,2001-08-06收修改稿
* 国家自然科学基金资助项目(批准号: 69873042)
1)董韫美. 上下文无关语言上的递归函数——Ⅱ. CFPRF及CFRF定义的合宜性. 中国科学, E辑(待发表)

是当算法加工的对象本身具有短语结构,即论域是 CFL 时更是如此. 例如图的算法、树的算法等等,都是这种情形. 此时用一般的字上递归函数不能直接地反映出被加工对象的结构,而在加工过程中又不可避免地要涉及到结构,为此必须把一个语法分解算法嵌入函数的定义之中,相当累赘.

提出 CFRF,目的是为计算机上用的非数值算法(特别是为描述形式规约)提供一种理论工具. 已经把它作为一种可执行规约语言的基础,并且对该种语言建立了实验系统,进行了应用研究[1].

1 CFL 句子枚举

CFRF 的定义要用到极小算子,极小算子的求值依赖于 CFL 句子的枚举. 本文提出一种基于 CFL 分层构造的句子枚举,其他枚举方法如平凡语言句子枚举或者字的枚举以及基于特征函数的 CFL 句子枚举,将在以后的工作中述及.

令 Σ 为有穷的非空符号集合,称为字母表. Σ 中的一个串是指 Σ 中的任意符号系列. 与普通语言相类比,符号也称为字母,串称为字. 对任意字母表 Σ,以 Σ^* 表示 Σ 上所有的字组成的集合. 注意 Σ^* 是一个特殊的 CFL,称为 Σ 上的平凡语言.

以下首先给出基础定义,以定义空字、字的长度、连接、端、尾及子字等概念,并给出其基本性质.

1.1 基础定义

字可以看成是由字母表中的字母所组成的串,不含任何字母的串是空字或空串,记为 λ. 串 w 中所含字母的数目就是字的长度,记为 $|w|$. 两个字的连接即组成它们的字母串的连接,将以 uv 表示字 u 和字 v 的连接所得的字. 由于连接满足结合律,所以可以使用多个字的连接记法. 任何 3 个字 u,v,w,如 $w=uv$,则称 u 为 w 的端(如 $v\neq\lambda$,则称 u 为 w 的真端),称 v 为 w 的尾(如 $u\neq\lambda$,则称 v 为 w 的真尾).

对于任意字 z,如存在字 u, v, w,使得 $z=uvw$,则称 v 在 z 中出现,或称 v 为 z 的子字(u,w 不同时为 λ 时,称 v 为 z 的真子字).

空字 λ 具有如下性质:(ⅰ)对于任意的字 u,均有 $u\lambda=\lambda u=u$;(ⅱ)$uv=\lambda$,当且仅当 $u=\lambda$, $v=\lambda$.

字的长度具有如下性质:(ⅰ)空字 λ 的长度为 0,字母的长度为 1;(ⅱ)如 $w=uv$,则 $|w|=|u|+|v|$.

字的连接具有如下性质:(ⅰ)任意 3 个字 u, v, w,如 $uw=vw$ 或 $wu=wv$,则 $u=v$;(ⅱ)如 $u=v$,则 $uw=vw$ 且 $wu=wv$.

对于 CFL,本文采用以下的记号约定:以大写希腊字母 Λ 表示空集. 有穷字母表 $V=V_N\cup V_T$,其中 V_N 是非终极符集合,V_T 是终极符集合,$V_N\cap V_T=\Lambda$. V_N 中元素以大写字母(可能带脚标)$X,Y,Z,\cdots$ 表示. V_T 中元素以小写字母 a, b, $c,\cdots$ 表示. V_T^* 表示由终极符的串的全体组成的集合(包括空串 λ 在内). 其元素以小写字母(可能带脚标)$u,v,w,\cdots$ 表示. 类似地 V^* 表示由 V 中符号的串的全体组成的集合(包括空串 λ 在内),其元素以大写字母(可能带

1) 董韫美, 等. Collection of SAQ Report No 1 ~ 16. Technical Reports ISCAS-LCS-95-09 (August 1995), ISCAS-LCS-96-01 (March 1996). 中国科学院软件研究所计算机科学开放研究实验室

脚标)$P,Q,R,\cdots$表示. 语言 L 是指 V_T^* 的子集,本文只考虑 CFL 的情形. $\mathscr{P}$是产生式集合,即形如 $X\to P$ 的产生式的全体所构成的集合,其中 $X\in V_N$, $P\in V^*$. 记

$$\mathrm{Term}(Y)=\{P\mid Y\to P\in\mathscr{P}\},$$

即以 Y 为其左部的产生式的右部(称为 Y 的项)所组成的集合.

不含非终极符的项称为平凡项,含非终极符的项称为非平凡项. 其右部为(非)平凡项的产生式称为(非)平凡产生式.

设有 $V_N,V_T,\mathscr{P}$, 对 V_N 中每个非终极符 X,都可以定义一个上下文无关文法(CFG) $G_X=(V_N,V_T,X,\mathscr{P})$, 非终极符 X 称为开始符号. 它产生的语言 $L(G_X)$是 V_T^* 的子集[1].

今后为方便起见,有时我们也使用 Backus-Naur 形式(BNF)的 CFG. 这种记法为计算机科学界所习用.

有时需要把语言 L 看成一个良序集. 即对于其中任意两个不同的句子, 都能够确定何者为先(我们把这个关系表为 $\prec$, $u\prec v$ 读作 u 先于 v)且具有性质:(ⅰ) $u\prec v$ 与 $v\prec u$ 不能同时成立;(ⅱ) $u\prec v$, $v\prec w$, 则 $u\prec w$;(ⅲ) L 的每一非空子集都有最小元素, 即先于任何其他元素的元素. 不难看出, 可以对良序集的元素逐一枚举.

在不引致误解的情形下, 我们把不同的序关系都用记号 $\prec$ 来表示.

对任何两个良序集 S_1 和 S_2, 如果它们互不相交(交集为空集合), 那么就可以在 S_1 和 S_2 之间定义序关系, 表为 $S_1\prec S_2$: 对任何 $x\in S_1$, $y\in S_2$, $x\prec y$ 成立. 然后将不同的序关系用于各自适用的局部情形, 我们就可以把原来的 3 个序关系(S_1 中的, S_2 中的, S_1 和 S_2 之间的)扩展到并集 $S_1\cup S_2$ 上, 使 $S_1\cup S_2$ 成为良序集, 并得到良序关系. 同理可以处理多个良序集, 使它们的并集成为良序集.

序 $\prec$ 定义为词典序, 是指通过在字母表中加入一个空白符号来扩展字母表, 它被放在原来字母表的最开头处. 当确定两个字何者为先时, 是通过从左到右逐个字母地对这两个字进行比较, 直到在同一位置上遇到不同的字母, 并以这两个字母在字母表中之先后为相应两字之先后.

1.2 基于 CFL 分层构造的句子枚举

这种枚举方法的思想是:把 CFL 划分为可数无穷个句子集合,称之为层. 每层都是良序的 CFL 句子有穷集合. CFL 即所有各层的并集,当文法无二义时, 各层两两相交为空. 直接从文法本身机械地构造出第 0 层. 当有了第 n 层之后,用已有的各层(从第 0 到第 n 层)机械地构造出第 $n+1$ 层.

当要求枚举某 CFL 的句子时,先构造第 0 层(良序有穷集合),然后依序枚举其中句子. 当第 n 层中的句子已经被枚举完了之后,先构造第 $n+1$ 层(良序有穷集合),然后依序枚举其中句子. 如此继续. 我们称这种枚举方法为 CFL 分层枚举. 现在开始叙述分层的机械构造方法.

设 $G=(V_N,V_T,X,\mathscr{P})$是 CFG. 有 m 个非终极符,即 $V_N=\{X_1,\cdots,X_m\}$, 而 X 是其中之一.

有穷字母表 V_T 和 V_N 中的字母是有序排列的,它们当然是良序集合. 令 $V_T\prec V_N$, 那么如前述可得到它们并集 V 中的良序关系.

考虑以 X_i 为左部的产生式的右部构成的集合(即 X_i 的项构成的集合)

$$\mathrm{Term}_i=\{P\mid X_i\to P\in\mathscr{P}\},\ i=1,\cdots,m.$$

注意项是 $V=V_N\cup V_T$ 中的串. 我们规定 Term_i 中的序为词典序, 它当然是良序有穷集合.

每个非终极符 X_i, 都有一个 CFL 与之对应, 即由文法 $G_i=(V_N, V_T, X_i, \mathscr{P})$所定义的语言 L_i. 设所有 G_i 均为无二义文法. 以下提出 L_i 的分层构造法. 我们将构造 m 个良序有穷集合的系列:

$$H_0^{(1)},\cdots,H_n^{(1)},\cdots$$
$$\vdots$$
$$H_0^{(m)},\cdots,H_n^{(m)},\cdots$$

$H_0^{(i)}$ 的构造方法如下[1]:

$$H_0^{(i)} =_{df}\{P \mid P \text{ 是 } \mathrm{Term}_i \text{ 中的平凡项}\},\quad i=1,\cdots,m.$$

$H_0^{(i)}$ 是 Term_i 的子集, 按规定 $H_0^{(i)}$ 中的序为词典序, $H_0^{(i)}$ 是良序有穷集合.

为叙述 $H_{n+1}^{(i)}$的构造方法, 采用以下的记法:

$$\alpha_n^{(i)} =_{df}\bigcup_{j=0}^{n}H_j^{(i)},\quad i=1,\cdots,m.$$

注意下面的引理 1(ⅱ), 如果 $H_0^{(i)},\cdots,H_n^{(i)}$ 是良序有穷集合, 且令 $H_0^{(i)}\prec\cdots\prec H_n^{(i)}$, 则 $\alpha_n^{(i)}$ 是良序有穷集合, 且得到 $\alpha_n^{(i)}$ 中的良序关系.

设有 Term_i 中的非平凡项

$$P = u_0X_{i_1}u_1\cdots X_{i_r}u_r,$$

其中 $u_0,\cdots,u_r$ 为终极符串(可以是空串), $X_{i_1},\cdots,X_{i_r}$为非终极符. 这时也把该非平凡项写成

$$P = P(X_{i_1},\cdots,X_{i_r}),$$

以指出 P 中出现的各个非终极符.

设有良序有穷句子集合 $\mathscr{S}_1,\cdots,\mathscr{S}_r$, $\mathscr{S}_1\subset L_{i_1},\cdots,\mathscr{S}_r\subset L_{i_r}$, 则定义

$$P(\mathscr{S}_1,\cdots,\mathscr{S}_r) =_{df}\{u_0v_1u_1\cdots v_ru_r \mid v_1\in\mathscr{S}_1,\cdots,v_r\in\mathscr{S}_r\},$$

即在非平凡项 P 中, 用终极符串集合替代非终极符.

令 $P(\mathscr{S}_1,\cdots,\mathscr{S}_r)$中的序为广义词典序, 即对其中任意两个字 $w,\tilde{w}$,

$$w=u_0v_1u_1\cdots v_ru_r,\quad \tilde{w}=u_0\tilde{v}_1u_1\cdots\tilde{v}_ru_r,$$

$w\prec\tilde{w}$, 如果存在 j, 使得 $1\leqslant j\leqslant r$, 且 $v_1=\tilde{v}_1,\cdots,v_{j-1}=\tilde{v}_{j-1}$, 而在 $\mathscr{S}_j$ 中, $v_j\prec\tilde{v}_j$.

令

$$\begin{aligned}P^{(n+1)} =_{df}{}& P(H_n^{(i_1)},\alpha_n^{(i_2)},\cdots,\alpha_n^{(i_{r-1})},\alpha_n^{(i_r)})\cup P(\alpha_{n-1}^{(i_1)},H_n^{(i_2)},\cdots,\alpha_n^{(i_{r-1})},\alpha_n^{(i_r)})\cup\cdots\\ &\cup P(\alpha_{n-1}^{(i_1)},\alpha_{n-1}^{(i_2)},\cdots,H_n^{(i_{r-1})},\alpha_n^{(i_r)})\cup P(\alpha_{n-1}^{(i_1)},\alpha_{n-1}^{(i_2)},\cdots,\alpha_{n-1}^{(i_{r-1})},H_n^{(i_r)}),\end{aligned}$$

易知 $P^{(n+1)}$中的集合 P 两两相交为空. 故令

$$P(\alpha_{n-1}^{(i_1)},\alpha_{n-1}^{(i_2)},\cdots,\alpha_{n-1}^{(i_{r-1})},H_n^{(i_r)})\prec\cdots\prec P(H_n^{(i_1)},\alpha_n^{(i_2)},\cdots,\alpha_n^{(i_{r-1})},\alpha_n^{(i_r)}),$$

则得到良序有穷集合 $P^{(n+1)}$及其中的良序关系.

现在可以构造 $H_{n+1}^{(i)}$了.

设 Term_i 中的非平凡项为 $P_1,\cdots,P_l$, 则

---

1) "$----=_{df}\cdots\cdots$"读作 ---被定义为……

$$H_{n+1}^{(i)} =_{df} \bigcup_{j=1}^{l} P_j^{(n+1)}.$$

易知这些集合 $P_j^{(n+1)}$ 两两相交为空. 由 Term_i 中的序，上式中构造 $H_{n+1}^{(i)}$的每一非终极项 P_j 的序是确定的，故而 $H_{n+1}^{(i)}$是良序有穷集合，并得到其中的良序关系.

引理 1　在 1.2 小节中所述 CFL 的分层构造，具如下性质：

(ⅰ) L_i 为各层的并集，即

$$L_i = \bigcup_{n=0}^{\infty} H_n^{(i)}, \qquad i = 1, \cdots, m.$$

(ⅱ) 在各文法 G_i 无二义的情形下，对同一个 i，不同的层两两相交为空，即

$$H_j^{(i)} \cap H_k^{(i)} = \Lambda, \quad \text{当 } j \neq k,\ i = 1, \cdots, m.$$

(ⅲ) 当文法中各产生式的右部至多出现一个非终极符时，构造第 $n+1$ 层时只用到第 n 层.

2　CFPRF 定义、求值算法和结构归纳法

首先定义 CFL 上的函数类 CFPRF，即上下文无关语言的原始递归函数. 给出 CFPRF 的两个例子，论证联立递归式的合宜性，给出相应的求值算法，以及证明 CFPRF 中函数性质用的结构归纳法.

2.1　CFPRF 的定义

定义　函数 f: $L_1 \times \cdots \times L_n \to L$ 是 CFPRF(上下文无关语言的原始递归函数)，如果 $L_1, \cdots, L_n, L$ 均是 CFL，L_i 由上下文无关文法 $G_i = (V_N, V_T, X_i, \mathscr{P}_i)$产生，$L$ 由 $G = (V_N, V_T, X, \mathscr{P})$产生，所涉及的文法均无二义[1]. f 系由以下步骤生成，而且也只由此生成.

ⅰ. f 是以下基本函数之一：

(ⅰ) 常字函数 $\mathrm{const}_w(x_1, \cdots, x_m) = w$, $w \in L$.

(ⅱ) 投影函数 $U_i^m(x_1, \cdots, x_m) = x_i$, $1 \leqslant i \leqslant m$.

(ⅲ) 连接函数 $\mathrm{concate}(x_1, \cdots, x_m) = x_1 \cdots x_m$.

ⅱ. f 是有穷次使用以下算子得到的：

(ⅰ) 代入算子. 即由 g_i: $L_1 \times \cdots \times L_n \to L^{(i)}$, $i = 1, \cdots, m$ 和 h: $L^{(1)} \times \cdots \times L^{(m)} \to L$，得到 f: $L_1 \times \cdots \times L_n \to L$，其中 $f(x_1, \cdots, x_n) = h(g_1(x_1, \cdots, x_n), \cdots, g_m(x_1, \cdots, x_n))$，各 g_i, h 是 CFPRF 类中已知函数.

(ⅱ) 联立递归式. 设要定义的是 m 个函数 $f_1, \cdots, f_m$: $L_1 \times \cdots \times L_n \to L$，$L_1$ 由 $G_1 = (V_N, V_T, X_1, \mathscr{P}_1)$产生，称此 m 个函数是由联立递归式所定义的，如果对每个 $P \in \mathrm{Term}(X_1)$，存在 CFPRF 类中的 m 个已知函数 h_{Pi}, $i = 1, \cdots, m$, 使得

1) 当 P 是平凡项，即 P 中不含非终极符时，有规则组

$$f_i(P, y_2, \cdots, y_n) =_{df} h_{Pi}(y_2, \cdots, y_n), \quad i = 1, \cdots, m; \tag{1}$$

2) 当 P 是非平凡项，即

$$P = u_0 Z_1 u_1 \cdots Z_r u_r,$$

其中 $u_0, \cdots, u_r$ 为终极符串(可以是空串)，$Z_1, \cdots, Z_r$ 为非终极符，则有规则组

---

1) 有时我们写 f: $G_1 \times \cdots \times G_n \to G$ 以强调 f 的定义域、值域的文法结构

$$f_i(P,y_2,\cdots,y_n) =_{df} h_{Pi}(Z_1,\cdots,Z_r,y_2,\cdots,y_n,\cdots,f_k(Z_j,y_2,\cdots,y_n),\cdots),$$
$$i,k = 1,\cdots,m;\ 1 \leqslant j \leqslant r. \tag{2}$$

上式 h_{Pi}中，我们把非终极符视为与自变元等同，并在由该非终极符代表的 CFL 中取值. 注意同一个非终极符可以多次出现，各个出现均被认为不同. h_{Pi}式中的未知函数f_k 亦可不出现，但如出现，其首一自变元位置对应的必须是 X_1. 对所有 $P\in \mathrm{Term}(X_1)$，分别按以上情形取规则组(1)或(2)的全体，便是联立递归式. 只有一个待定义未知函数的联立递归式，即 $m=1$ 的情形，称为原始递归算子.

简言之，CFPRF 是包含基本函数 const_w，U_i^m，concate，并在代入算子和联立递归式算子下封闭的全函数的最小类. 与此等价的说法是，CFPRF 是可由基本函数出发，有穷次应用代入和联立递归式算子而构建的全函数的类.

2.1.1 联立递归式的求值规则

联立递归式规定了计算函数 $f_1,\cdots,f_m$ 的求值规则. 设

$$w \in L_1,\ y_2 \in L_2,\cdots,y_n \in L_n.$$

(ⅰ) 当 w 是某产生式 $X_1\to P$ 的右部时(此时 $w=P$，其中不含非终极符)，按该产生式决定的联立递归式中的规则组((1)式)

$$f_i(P,y_2,\cdots,y_n) =_{df} h_{Pi}(y_2,\cdots,y_n),\quad i=1,\cdots,m,$$

定义函数值为

$$f_i(w,y_2,\cdots,y_n) = h_{Pi}(y_2,\cdots,y_n),\quad i=1,\cdots,m.$$

(ⅱ) 设 $X_1\overset{*}{\Rightarrow}w$，即当有产生式

$$X_1\to P,\ P = u_0Z_1u_1\cdots Z_ru_r,$$

其中 $u_0,\cdots,u_r$ 为终极符串(可以是空串)，$Z_1,\cdots,Z_r$ 为非终极符. 且存在 $v_1,\cdots,v_r$，使得

$$Z_1\overset{*}{\Rightarrow}v_1,\cdots,Z_r\overset{*}{\Rightarrow}v_r;\quad w = u_0v_1u_1\cdots v_ru_r.$$

此时联立递归式中有由 $X_1\to P$ 所决定的规则组((2)式)

$$\begin{cases} f_i(P,y_2,\cdots,y_n) =_{df} h_{Pi}(Z_1,\cdots,Z_r,y_2,\cdots,y_n,\cdots,f_k(Z_j,y_2,\cdots,y_n),\cdots), \\ i,k = 1,\cdots,m;\quad 1\leqslant j\leqslant r, \end{cases}$$

则定义函数值为

$$\begin{cases} f_i(w,y_2,\cdots,y_n) = h_{Pi}(v_1,\cdots,v_r,y_2,\cdots,y_n,\cdots,f_k(v_j,y_2,\cdots,y_n),\cdots), \\ i,k = 1,\cdots,m;\quad 1\leqslant j\leqslant r. \end{cases}$$

即把函数值 $f_i(w,y_2,\cdots,y_n)$的计算问题，化归为更简单的 $f_k(v_j,y_2,\cdots,y_n)$的计算问题.

注意，对于有二义的文法，句子的分解不惟一. 由此产生多种归约可能，导致多值函数. 这种情形是我们希望避免的.

2.2 CFPRF 的例子

例 1 栈操作

$$\langle\mathrm{stack}\rangle :: = \lambda \mid \langle\mathrm{stack}\rangle\langle\mathrm{data}\rangle,$$

这一文法定义了一个栈 stack. 在这里没有规定组成栈的元素 data 是什么，可以再由其他文法定义. 栈的两个操作 push(下推)和 pop(上托)可定义如下：

push: stack × data→stack

pop: stack→data

push(λ, data) = data　　(投影函数)

push(stack data1, data) = stack data1 data　　(连接函数)

pop(λ) = λ　　(常字函数)

pop(stack data) = data　　(投影函数)

例 2　二叉树的遍历

⟨binary_tree⟩∷ = λ | ⟨node⟩↦⟨binary_tree⟩**R**⟨binary_tree⟩,

⟨node_list⟩∷ = λ | ⟨node_list⟩⟨node⟩.

这一文法定义的二叉树，或为空，或为一个结点(node)指向两个二叉子树(左子树前没有标记，右子树前有一标记 **R**). 定义中对结点(node)未作规定，可以再由其他文法定义.

还定义了结点表(node_list)，即结点的串(可以为空).

前序、中序及后序遍历算法定义如下：

preorder, inorder, postorder: binary_tree→node_list

preorder(λ) = λ

preorder (node↦lbt **R** rbt) = node preorder(lbt) preorder(rbt)

inorder(λ) = λ

inorder (node↦lbt **R** rbt) = inorder(lbt) node inorder(rbt)

postorder(λ) = λ

postorder (node↦lbt **R** rbt) = postorder(lbt) postorder(rbt) node

二叉树为空时，前序遍历(preorder)结果为空串. 二叉树不空时，前序遍历结果为：先得到根结点，然后先对左子树进行前序遍历，再对右子树进行前序遍历. 这正是二叉树前序遍历定义本身的直接表述.

中序(inorder)及后序遍历(postorder)的情形与此相似.

2.3　联立递归式的合宜性

CFPRF 可认为是通过一个函数序列来定义的. 序列中的每一函数，或者是基本函数，或者是对在它之前的函数使用了代入算子、联立递归式算子而得到的. 注意到任何常字可以单独构成 CFL，或被包含在一个 CFL 中，于是，常字函数作为基本函数是合理的. CFL 类对于连接操作是封闭的，故连接函数作为基本函数也是合理的. 投影函数显然是必要与合理的. 代入算子是定义函数的常规和必要手段，在 CFL 类中不导致任何困难. 联立递归式以未知函数间的关系来定义函数，关于其合宜性，有以下的定理：

定理 1(联立递归式的合宜性)　设 L_1 的文法无二义，则对于任何 $w \in L_1$ 和相应的 $y_2, \cdots, y_n$，联立递归式惟一地决定了函数组的值：$f_1(w, y_2, \cdots, y_n), \cdots, f_m(w, y_2, \cdots, y_n)$.

证　施归纳于句子 w 推导的长度 l. 句子推导的长度是指：从文法的开始符号出发，到获得该句子为止，使用产生式的次数.

基始. $l = 1$. 即 w 是一步得到的，于是只可能是直接使用产生式 $X_1 \to P$ 的情形，而且 $P = w$. L_1 的文法无二义，故适用于此的产生式是惟一的.

w 是句子，故 P 中不含非终极符，于是

$$f_i(w, y_2, \cdots, y_n) = h_{Pi}(y_2, \cdots, y_n) \quad (i = 1, \cdots, m)$$

是已知函数.

归纳. 设 $l \leqslant s$ 的情形已证,即 w 的推导步数不超过 s 时,$f_1(w, y_2, \cdots, y_n), \cdots, f_m(w, y_2, \cdots, y_n)$的值均已确定.

今当 $l = s+1$, 设有长度为 $s+1$ 步的推导

$$X_1 \Rightarrow P \overset{*}{\Rightarrow} w,$$

其中 $X_1 \to P$ 是一个产生式,$w \in L_1$. L_1 的文法无二义,故适用于此的产生式是惟一的.

不妨设

$$P = u_0 Y_1 u_1 \cdots Y_r u_r,$$

易知从推导

$$P \overset{*}{\Rightarrow} w$$

中,对 P 中每一非终极符 Y_j,可以析出 w 的子字 w_j,使得

$$w = u_0 w_1 u_1 \cdots w_r u_r$$

和

$$Y_j \overset{*}{\Rightarrow} w_j, \ j = 1, \cdots, r,$$

且各 w_j 的推导长度不超过 s 步.

由于 L_1 的文法无二义, w 的分解结果是惟一的. 依照 2.1.1 小节中所述的联立递归式求值规则,可导出

$$f_i(w, y_2, \cdots, y_n) = h_{Pi}(w_1, \cdots, w_r, y_2, \cdots, y_n, \cdots, f_k(w_j, y_2, \cdots, y_n), \cdots), i = 1, \cdots, m.$$

上式中,函数 f_k 如出现,则其首一变元 Y_j 其实就是 X_1. 此时

$$Y_j \overset{*}{\Rightarrow} w_j, \quad 1 \leqslant j \leqslant r,$$

也就是

$$X_1 \overset{*}{\Rightarrow} w_j, \quad 1 \leqslant j \leqslant r.$$

推导长度不超过 s 步. 由归纳假定,

$$f_k(w_j, y_2, \cdots, y_n) \quad (k = 1, \cdots, m; \ j = 1, \cdots, r)$$

的值已经确定, h_{Pi}是已知函数, 故

$$f_i(w, y_2, \cdots, y_n) = h_{Pi}(w_1, \cdots, w_r, y_2, \cdots, y_n, \cdots, f_k(w_j, y_2, \cdots, y_n), \cdots)$$

为已知. 即对 $l = s+1$ 的情形定理亦成立,归纳步骤得证.

推论 1 以上定理对原始递归算子也成立.

2.4 实现联立递归式的算法过程

以上联立递归式合宜性定理的证明,提示了通过联立递归式所定义函数的计算步骤,可以作为函数施用的实现算法的基础. 以下具体描述有关的算法过程. 该算法假定有通用的语法分解算法,能够从合法句子得出其推导树. 这一假定当然不成为问题,例如可以用 Earley 算法[7],并且它可在计算机上使用. 董韫美[1)]定义的函数 parse_Y 也是通用的求推导树的函数. 但它本身是通过联立递归式来定义的.

1) 见 103 页脚注

以下叙述联立递归式的实现算法.

问题　对任意 $w\in L_1$ 和给定的 $y_2,\cdots,y_n$，如何计算由联立递归式定义的各个函数：

$$f_i(w,y_2,\cdots,y_n),\qquad i=1,\cdots,m.$$

算法过程　观察推导 $X_1\overset{*}{\Rightarrow}w$ 及相应的推导树 T. 注意到树 T 的根结点是 X_1，每一个叶结点都是终极符串，连接起来就得到 w. T 的每一个非叶结点是一个非终极符(某个产生式的左部). 以它为根的子树的第一层结点(即根直接指向的结点)，或为终极符串，或为非终极符，连接起来就是该产生式的右部. 下文称此产生式为该子树的根本产生式.

以下叙述如何对树 T 进行归约加工，来计算函数 $f_i(w,y_2,\cdots,y_n)$ $(i=1,\cdots,m)$.

在树 T 的归约加工过程中，旧有的叶结点不断地被剪除，非叶结点(连同以它为根的子树)则不断转化成新的叶结点. 新形成的叶结点仍然是原来的非终极符，但为它建立了指针.

当非终极符是 X_1 时，就为它建立两种指针：一种指针是变元指针(一个)，指向该非终极符所对应的变元值(终极符的串)；另一种指针是函数指针(m 个)，分别指向作为函数 f_i 的值的终极符串. 函数关系是由以该非终极符为根的子树的根本产生式所决定的，函数的自变元取值是被归约的子树各非终极符叶结点对应的变元值. 当非终极符不是 X_1 时，只为它建立一种指针，即变元指针(一个)，指向该非终极符所对应的变元值(即该非终极符所推出的终极符串).

树 T 的归约加工过程是寻找这样的非叶结点 Y，以 Y 为根的子树结点(除 Y 以外)均为叶结点，即子树除根结点 Y 以外只有第一层结点. 归约加工即剪除叶结点，只留下子树的根，并且为它建立指针. 在推导树 T 最终被归约为一个结点 X_1 之前，这种子树是必定存在的. 子树分 4 种情形：

(ⅰ) Y 是 X_1，子树叶结点是终极符串. 这时子树的根本产生式是 $X_1\to P$ 且 P 中不含非终极符. 所进行的归约加工是，由根本产生式和要计算的各个函数 f_i，决定已知函数 h_{pi}，且按照

$$f_i(P,y_2,\cdots,y_n)=h_{Pi}(y_2,\cdots,y_n)\quad(i=1,\cdots,m)$$

计算出各 f_i 值. 子树叶结点随即被剪除，子树根 X_1 转化为叶结点，建立变元指针和函数指针. 其变元值是终极符串 P，函数值是求得的各 f_i 值.

(ⅱ) Y 不是 X_1，子树叶结点是终极符串. 这时子树的根本产生式是 $Y\to P$ 且 P 中不含非终极符. 所进行的归约加工是，剪除叶结点，子树根 Y 转化为叶结点，建立变元指针，其变元值是终极符串 P.

(ⅲ) Y 是 X_1，子树有非终极符的叶结点. 此时子树的根本产生式是 $X_1\to P$，设

$$P=u_0Z_1u_1\cdots Z_ru_r,$$

其中 $u_0,\cdots,u_r$ 为终极符串(可以是空串)，$Z_1,\cdots,Z_r$ 为非终极符. Z_j 的变元值(经由相应的指针得到)为 v_j. 当 Z_j 是 X_1 时，函数值(经由相应的指针得到)为

$$f_i(v_j,y_2,\cdots,y_n),\quad i=1,\cdots,m,j=1,\cdots,r.$$

所进行的归约加工是，连接子树各叶结点的变元值 $v_1,\cdots,v_r$ 和终极符串叶结点 $u_0,\cdots,u_r$，便得到子树根结点 X_1 的变元值，设为 v $(v=u_0v_1u_1\cdots v_ru_r)$. 根据产生式 $X_1\to P$ 和要计算的各个函数 f_i，决定已知函数 h_{pi}，且按照

$$f_i(v,y_2,\cdots,y_n)=h_{Pi}(v_1,\cdots,v_r,y_2,\cdots,y_n,\cdots,f_k(v_j,y_2,\cdots,y_n),\cdots),$$
$$k=1,\cdots,m,\quad j=1,\cdots,r,$$

求出各 $f_i(v,y_2,\cdots,y_n)$作为子树根结点 X_1 的函数值. 随后剪除以 X_1 为根的子树的各个叶结点,X_1 转化成为新的叶结点, 并为它建立变元指针和函数指针.

(ⅳ) Y 不是 X_1, 子树有非终极符的叶结点. 此时子树的根本产生式是 $Y\to P$. 所进行的归约加工是,连接子树各非终极符叶结点的变元值和终极符串叶结点,便得到结点 Y 的变元值. 剪除叶结点,子树根 Y 转化为叶结点,建立变元指针,此过程继续进行,用归纳法容易证明,推导树 T 最终将归约为一个结点 X_1. 此时,其变元值是 w,函数值是

$$f_i(w,y_2,\cdots,y_n),\quad i=1,\cdots,m.$$

这就结束了一组联立递归定义的函数的计算.

上述算法是用树的语言来表达的,没有规定严格的加工顺序,为计算机上的具体实现方案留下了很大的余地. 按照实际采用的变元值 w 的语法分析过程,可以把算法再具体化, 包括进行某些优化. 例如,在上述归约过程的(ⅳ)中,可能遇到的一种情形是,被剪除的叶结点中有非终极符 X_1. 这时以前的归约过程中针对它进行的函数值计算实际上没有用处. 但是在树 T 建立之后,不难在其中发现这种非终极符结点,对于它只建立变元指针,而不建立函数指针.

2.5 联立递归式的结构归纳法

联立递归式刻画了所要定义的一组函数之间的相互关系, 因此保证了在所定义的该组函数之间存在着给定的关系式. 其实这也就是刻画了被定义函数所具备的某种性质, 这些性质密切联系于函数定义域的语法构造. 有时我们希望确认所定义的该组函数具备另外的一些性质,对于用联立递归式定义的函数组, 可以使用以下叙述的结构归纳法证明函数性质.

结构归纳法定理在 CFPRF 函数性质有关命题论证方面的作用地位, 与数学归纳法之于自然数函数性质有关的命题相当.

设函数 $f_1,\cdots,f_m$: $L_1\times\cdots\times L_n\to L$ 是由联立递归式(1)和(2)定义的,将用 n 元关系 R 来刻画函数$f_1,\cdots,f_m$ 间的关系, 用记号 $R(f_1(y_1,\cdots,y_n),\cdots,f_m(y_1,\cdots,y_n))$表示.

定理 2 (结构归纳法) 给定 $y_2,\cdots,y_n$, 如果以下的基始步和归纳步都已经被证明成立, 则对给定的 $y_2,\cdots,y_n$ 和任意 $y_1\in L_1$,

$$R(f_1(y_1,\cdots,y_n),\cdots,f_m(y_1,\cdots,y_n))\text{ 成立.}$$

(ⅰ) 基始步. 对语言 L_1 的文法 G_1 中的每个平凡产生式 $X_1\to P$, 中联立递归式规则组(1)右部的 $h_{Pi}(y_2,\cdots,y_n)$, $i=1,\cdots,m$, 均使得

$$R(h_{P1}(y_2,\cdots,y_n),\cdots,h_{Pm}(y_2,\cdots,y_n))\text{ 成立.}$$

(ⅱ) 归纳步. 对文法 G_1 中的每个非平凡产生式 $X_1\to P$, 对于 $y_1\in L_1$, 按(2)式决定的函数组 $f_1,\cdots,f_m$ 求值规则.

1) 如 P 不含非终极符 X_1,则对任意可以由推导 $X_1\Rightarrow P\overset{*}{\Rightarrow}y_1$ 导出的 y_1,

$$R(f_1(y_1,\cdots,y_n),\cdots,f_m(y_1,\cdots,y_n))\text{ 成立.}$$

2) 如 P 含非终极符 X_1,设在 y_1 关于产生式 $X_1\to P$ 的分解式中, x_1 对应于 X_1. 注意 X_1 可在 P 中多次出现, 故 y_1 可以包含多个这样的 x_1. 对于所有这样的 x_1, 从 $R(f_1(x_1,y_2,\cdots,y_n),\cdots,f_m(x_1,y_2,\cdots,y_n))$ 成立, 可以推出 $R(f_1(y_1,\cdots,y_n),\cdots,f_m(y_1,\cdots,y_n))$ 成立.

证 施归纳于句子 y_1 的推导长度 l.

基始. $l=1$. 即 y_1 是一步得到的,于是只可能是直接使用平凡产生式 $X_1\to P$ 的情形, 而

且 $P = y_1$．于是由联立递归式(1)决定的函数组 $f_1,\cdots,f_m$ 求值规则，

$$f_i(y_1,y_2,\cdots,y_n) = h_{Pi}(y_2,\cdots,y_n),\quad i = 1,\cdots,m.$$

由定理的基始步，

$$R(f_1(y_1,\cdots,y_n),\cdots,f_m(y_1,\cdots,y_n)) = R(h_{P1}(y_2,\cdots,y_n),\cdots,h_{Pm}(y_2,\cdots,y_n))\ \text{成立}.$$

归纳．设 $l \leqslant s$ 的情形已证，即对任意推导步数不超过 s 的句子 $x_1 \in L_1$，$R(f_1(x_1,y_2,\cdots,y_n),\cdots,f_m(x_1,y_2,\cdots,y_n))$ 成立．今当 $l = s+1$，设有长度为 $s+1$ 步的推导：

$$X_1 \Rightarrow P \overset{*}{\Rightarrow} y_1,$$

其中 $X_1 \to P$ 是一个非平凡产生式，$y_1 \in L_1$．

(ⅰ) 如 P 不含非终极符 X_1．由本定理的归纳步 1)，有

$$R(f_1(y_1,\cdots,y_n),\cdots,f_m(y_1,\cdots,y_n))\ \text{成立}.$$

(ⅱ) 如 P 含非终极符 X_1．此时特别有某些 $v_j \in L_1$，每一个都是 y_1 的子字，且推导式 $X_1 \overset{*}{\Rightarrow} v_j$ 的推导步数不超过 s．由归纳假定，

$$R(f_1(v_j,y_2,\cdots,y_n),\cdots,f_m(v_j,y_2,\cdots,y_n))\ \text{成立}.$$

由本定理的归纳步 2)，可推出

$$R(f_1(y_1,\cdots,y_n),\cdots,f_m(y_1,\cdots,y_n))\ \text{成立}.$$

即对 $l = s+1$ 的情形定理亦成立，归纳步骤得证．结构归纳法定理证毕．

3　极小算子和 CFRF 的定义

在本节中我们定义极小算子和 CFL 上的函数类 CFRF，即上下文无关语言的递归函数．

3.1　极小算子

定义　设有函数 f：$L_0 \times L_1 \times \cdots \times L_n \to L$，$\lambda \in L$，则极小算子($\mu$ 算子)$\mu_y[f]$求得的值定义为集合

$$\{y \mid y \in L_0,\ f(y,x_1,\cdots,x_n) = \lambda\}$$

中长度最短或者构造最简的 y．即按规定的枚举方法，首先被举出并且满足相应条件的 y．如果集合为空，则该值无定义．在枚举 y 的过程中，在遇满足要求的 y 之前，对所有被枚举的 y，$f(y,x_1,\cdots,x_n)$必须都有定义．否则 $\mu_y[f]$无定义．

这样的 y 决定于 f，并且是 $x_1,\cdots,x_n$ 的函数，即 $\mu_y[f(y,x_1,\cdots,x_n)]$定义了一个函数

$$\phi:\ L_1 \times \cdots \times L_n \to L_0,\quad \phi(x_1,\cdots,x_n) = \mu_y[f(y,x_1,\cdots,x_n)].$$

确切地说，即

$$\phi(x_1,\cdots,x_n) = \begin{cases} \text{Firstof}\{y \mid f(y,x_1,\cdots,x_n) = \lambda\}, \text{使得(ⅰ)对所有 } \tilde{y} < y,\ f(\tilde{y},x_1,\cdots,x_n) \\ \qquad \text{有定义，且(ⅱ) } f(y,x_1,\cdots,x_n) = \lambda, & \text{如果这种 } y \text{ 存在}, \\ \text{无定义} & \text{这种 } y \text{ 不存在}. \end{cases}$$

Firstof$\{y\}$是按照规定方法逐个枚举 L_0 的句子时，集合$\{y\}$中首先被举出的元素．$<$是枚举的顺序关系．

3.2　CFRF 定义

定义　函数 f：$L_1 \times \cdots \times L_n \to L$ 是 CFRF(上下文无关语言的递归函数)，如果满足 2.1 小

节 CFPRF 定义中的条件,此外在 f 的构造过程中,还允许使用极小算子 $\mu_y[f]$.

注意,通过使用极小算子,可能定义出偏函数,即不是处处都有定义的函数.而 CFPRF 中的函数是全函数,处处都有定义.

3.3 极小算子求值方法

极小算子的求值方法强烈地依赖于 CFL 句子的枚举方法. 1.2 小节中提出的 CFL 句子枚举方法,适用于一般 CFL 上函数的情形.它在时间上是高效的,但在空间上有高要求.在一般情况下,它要求记住所有已经被枚举过的句子.

注意在下文中,求值过程不是把自变元值化归为更简单的情形,而是继续枚举,直至找到满足条件的 y 为止.这一过程可能无终止地进行下去,即不存在满足条件的 y,这时 $\mu_y[f]$是偏函数.然而,一个函数是否偏函数,是不可判定问题.就是说在过程进行中,我们不知道它是否会在以后的某个时刻终止.

对作为谓词 f 第 1 个自变元的定义域的 CFL,从第 0 层开始,逐层构造并枚举其元素.设当时举出的句子为 y,并且满足

$$f(y,x_1,\cdots,x_n)=\lambda,$$

则 y 即为所求.否则继续枚举下一个句子,并进行检验.如此继续,或是在某一时刻找到满足条件的句子,或是无终止地继续这一过程.

一般情况下,CFL 分层枚举要求记住所有已经被枚举过的句子.换言之,这种方法在运行中要求越来越大的记忆量.在现实的运行中,系统可能因不堪记忆量的重负,在遇到满足条件的句子之前先已崩溃.

然而从引理 1(ⅲ)知,当文法中各产生式的右部至多出现一个非终极符时,构造第 $n+1$ 层时只用到第 n 层,记忆负担较轻.例如,由文法

$$X::=ab\mid aXb$$

定义的语言 L 具有如下分层:

$$L=\bigcup_{n=0}^{\infty}H_n,\quad H_0=\{ab\},\quad H_{n+1}=aH_nb=\{\underbrace{a\cdots a}_{n+2}\ \underbrace{b\cdots b}_{n+2}\},\ n=0,1,\cdots.$$

每层由一个句子构成,构造第 $n+1$ 层时只用到第 n 层.

4 讨论

本文旨在建立上下文无关语言上的递归函数理论.在 CFL 上定义了函数类 CFRF 和它的真子类 CFPRF,它们可以用来十分直接地表述非数值加工算法.

已有的与此有关的工作是代数语义学中抽象数据类型(ADT)的研究.ADT 可以看成是一种特殊形式的 CFRF,采用等式系来定义,得到的是一般递归函数.看来不易从中区分出原始递归的情形.比起在 ADT 实现中极为重要的项重写技术,CFRF 在原始递归的情形下避免了至关重要的 Church-Rosser 性质的讨论.而通过实际工作发现,原始递归具有足够强的描述能力,大多数情形下,CFPRF 就已经可以满足需要了.

CFRF 是为了研究计算机上使用的算法而提出的.以它为理论基础,可以设计一种可执行规约语言,能够在机器上直接执行,得到结果.

这种语言的基本数据类型应包括:自然数、逻辑值、字和 CFL.并应该提供基本类型间的

转换手段，由已经有的基本类型构造复合类型的手段. 从效率的角度考虑，不应当把自然数看成字来处理.

这种语言的基本函数类型应包括：CFRF，自然数、字上的函数和谓词，并应该

提供构造由复合类型到复合类型的函数的手段.

提供在计算机上直接高效率实现的常用类型和强力函数库.

提供定义新函数用的强力算子，以及某些特殊操作：CFL 句子的识别、分解、枚举、串运算和串匹配.

在创建新函数时，应能在计算机上用人机交互方法和机器辅助手段来保证按照规定来定义函数.

作为原型速成工具，这种语言应能用来写具有复杂语法结构的对象的加工算法.

CFRF 计算实现的问题，应是相当直截了当，最关键的是在系统中嵌入了通用的 CFL 分解算法.

参　考　文　献

1 Salomaa A. Formal Languages. New York: Academic Press, 1973

2 Cutland N. Computability: An Introduction to Recursive Function Theory. Cambridge: Cambridge University Press, 1980

3 Kleene S C. Introduction to Metamathematics. Princeton and North-Holland: Van Nostrand, 1985

4 胡世华. 递归算法论. 数学学报, 1960, 10(1): 66 ~ 103

5 McCarthy J. Recursive functions of symbolic expressions. Comm ACM, 1960, 3(4): 184 ~ 195

6 董韫美，李开德. 一类形式语法及其判定、分解问题. 数学进展, 1965, 8(1): 1 ~ 33

7 Earley J. An efficient context-free parsing algorithm. Comm ACM, 1970, 13(2): 94 ~ 102