多类型递归函数

献给徐利治教授 95 华诞

董楹美

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

E-mail: dym@ios.ac.cn

收稿日期: 2015-01-28; 接受日期: 2015-06-01

国家自然科学基金 (批准号: 61472405) 资助项目

摘要 本文旨在建立多类型递归函数 (MTRF), 目的是, 为在计算机上实现的函数式语言提供理论原型. MTRF 函数是可计算函数, 其定义域和值域的数据类型包括: 自然数、逻辑值、有穷字母表上的字、上下文无关语言 (CFL) 句子、CFL 句子的分析树. 本文指出 MTRF 函数与自然数递归函数是等价的, 构造了可在计算机上实现的 MTRF 函数 eval, 用来对任意 MTRF 函数求值.

关键词 多类型递归函数 (MTRF) 多类型原始递归函数 (MTPRF) 通用求值函数

MSC (2010) 主题分类 03D20, 68Q45

1 引言

本文假定上下文无关语言 (CFL) 理论和递归函数论的知识, 读者可分别参见文献 [1, 2].

文献 [3] 曾引入上下文无关语言 (CFL) 上的函数类上下文无关语言递归函数 (CFRF) 及其子类上下文无关语言原始递归函数 (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, 它还包含自然数、逻辑值和字上的函数为其特殊情形. 文献 [4] 证明了函数类 CFRF 和 CFPRF 分别是 CFL 上的偏递归函数和原始递归函数.

在可计算性理论研究中, 自然数的递归函数起了根本作用. CFRF 的提出意不在此, 而是为在计算机上使用的非数值算法, 特别是为描述形式规约, 提供一种理论工具. 我们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.

本文的 MTRF 又进了一步, 变元和函数值允许是自然数、真假值、字符串、CFL 句子和句子的分析树 (句子文法结构本身的树形表示), 函数的各个变元有自己独立的数据类型, 为书写算法提供更多便利.

本文分为 5 节. 第 1 节为引言, 叙述本文使用的记号和约定; 第 2 节提出函数类 MTPRF 和 MTRF 的定义和求值方法; 第 3 节讨论它们与其他论域上定义的递归函数的关系, 证明 MTPRF 与自然数上的原始递归函数等价, MTRF 与自然数递归函数等价; 第 4 节占篇幅最大, 叙述一个特殊的 MTRF 函数 eval 的构造方法, 即 MTRF 通用求值函数, 所述方法可在计算机上实现; 第 5 节讨论进一步应关注的问题.

记号、定义和约定

为行文方便, 我们使用下面的定义、记号和约定.

记号 1 “ $=_{df}$ ”读作“被定义为”. $XXX =_{abbr} yyy$ 表示, XXX 是 yyy 的缩写. 空集记为 $\emptyset$. $f_\emptyset$ 表示无处定义函数 (定义域和值域均为空集). 记号 $\simeq$ 用于并非总是有定义的情况, 即 $f(x) \simeq g(x)$, 如果对任何 x , $f(x)$ 和 $g(x)$ 同有定义或同无定义, 在有定义时二者相等. $/\ast \cdots \ast/$ 所括内容是注释.

定义 1 (字母表、字、空字) (1) 字母表为有限的非空符号集合, 符号也称为字母; (2) 字是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 ε .

定义 2 (上下文无关文法) 上下文无关文法 (CFG) $G =_{df} (V_N, V_T, \mathcal{P}, X)$, 其中 V_N 是变元的有限集合, V_T 是终极符的有限集合, $\mathcal{P}$ 是产生式的有限集合, $X \in V_N$ 是开始变元.

这里 $V_N \cap V_T = \emptyset$, $V = V_N \cup V_T$. 变元以大写字母 X, Y, Z (可能带脚标) 表示; 终极符以小写字母 $a, b, c, \dots$ 表示; V_T^* 表示由终极符的串的全体组成的集合 (包括空串 ε 在内), 其元素以小写字母 (可能带脚标) $u, v, w, \dots$ 表示; V^* 表示由 V 中符号的串的全体组成的集合 (包括空串 ε 在内), 其元素以大写字母 (可能带脚标) $Q, R, S, \dots$ 表示; 产生式形如 $Y \rightarrow Q$, 其中 $Y \in V_N$ 称为产生式的左部, $Q \in V^*$ 称为产生式的右部.

定义 3 (项、常项、变项、常项产生式、变项产生式) 产生式的右部称为项, 不含变元的项称为常项, 含变元的项称为变项. 右部为常项的产生式称为常项产生式, 右部为变项的产生式称为变项产生式.

定义 4 (推导式、句型、句子、派生符号、上下文无关语言、分析树、路径、分析树高度)

(1) 对任意 $QYR \in V^*$ 和产生式 $Y \rightarrow S \in \mathcal{P}$, 可写 $QYR \Rightarrow QSR$, 读为 (应用产生式 $Y \rightarrow S$) 从 QYR 推出 QSR . 若有 $Q_1, \dots, Q_n \in V^*$, 满足 $Q_1 \Rightarrow Q_2 \Rightarrow \cdots \Rightarrow Q_n$ (称为从 Q_1 到 Q_n 的推导式), 则可写 $Q_1 \xRightarrow{*} Q_n$ (包括 $n = 1$ 的情形, 即对任何 $Q \in V^*$, 可写 $Q \xRightarrow{*} Q$).

将以 $Y \xRightarrow{*} w$ 表示从 Y 到 w ($w \in V_T^*$) 的推导式. 用 $\Rightarrow$ 表示推导式中的一步, 用 $\xRightarrow{*}$ 表示推导式中的 0 步或多步. 若 $X \xRightarrow{*} Q$, $Q \in V^*$, 即 Q 是由开始符号 X 推出的, 则称 Q 是该文法的一个句型. 若句型 Q 中不含任何变元, 则称 Q 为句子.

设有文法 $G_1 = (V_N, V_T, X_1, \mathcal{P}_1)$, 则集合 $DS(X_1) = \{X \mid X_1 \xRightarrow{*} X, X \in V_N\}$ 称为 X_1 的派生符号 (derived symbol) 集合. 易知, $DS(X_1)$ 可机械地求得, 且 $X_1 \in DS(X_1)$.

(2) $L(G) =_{df} \{w \mid X \xRightarrow{*} w, w \in V_T^*\}$, 即 CFG G 所定义的语言, 是由开始变元 X 推出的 V_T^* 中的字 (句子) 的集合, $L(G)$ 称为上下文无关语言 (简称为 CFL).

(3) 称 $X \xRightarrow{*} w$ 的展开表示为句子 w 的推导式. 推导式可以用一个树 (称为分析树) 表示. 树的根节点是开始变元 X , 它指向所应用的产生式右部的各个字符 (当右部是空字时, 被当作一个代表空的字符). 然后再以其中的变元为子树的树根, 如此继续, 直到所指向的全是终极符为止. 分析树的全体叶节点组成该树所推出 (生成) 的句子. 下文中将以 $\overset{X}{\downarrow}_w^*$ 表示相应的分析树 (根为 X , 叶节点构成 w).

(4) 从树根到叶节点, 所经历的变元系列称为 (从树根到该叶节点的) 路径. 路径的长度即路径上变元的个数. 分析树所有路径长度的最大值是该分析树的高度.

语言 $L(G)$ 句子的分析树的集合 $\mathbb{L}(G) =_{df} \{\overset{X}{\downarrow}_w^* \mid w \in L(G)\}$.

注意, $X \xRightarrow{*} w$ 的推导式唯一地决定了一个分析树 $\overset{X}{\downarrow}_w^*$, 反之不真.

约定 1 本文中凡是谈到 CFL, 总关联于生成它的那个具体的 CFG.

为方便起见, 本文第 4 节使用一种紧凑形式的 CFG (也称为 BNF (Backus Naur form) 记法): 具有相同左部的产生式被写成一行 (称为元语言公式), 即左部, “ $::=$ ” 相当于箭头 $\rightarrow$, 然后是各个右部.

各右部之间, 以竖线 “|” 隔开. 全体左部构成集合 V_N , 右部中不在 V_N 中出现的全体字符构成集合 V_T . 这些公式构成一组 CFG, 选定一个左部作为开始符号, 就得到一个 CFG.

2 多类型递归函数定义

多类型原始递归函数 (MTPRF) 是包含基本函数: 首元函数、后继函数、投影函数, 在代入算子和原始递归式下封闭的全函数的最小类. 如果还允许使用极小算子, 便得到多类型递归函数 (MTRF, 可能是偏函数).

MTRF 函数各变元的定义域 D 和值域 R 相互独立地可以是: 自然数集合 $\mathbb{N}$, 逻辑值集合 $\mathbb{B} = \{\text{true}, \text{false}\}$, 有穷非空字母表 $\Sigma = \{o_1, \dots, o_m\}$ 上的平凡语言 L_Σ , 任意上下文无关文法 G 定义的语言 $L(G)$ 与上下文无关文法 G 的句子分析树集合 $\mathbb{L}(G)$.

L_Σ 由 Σ 上的所有字组成, 即由文法 $X ::= \varepsilon | Xo_1 | \dots | Xo_m$ (用 BNF 表示的产生式) 所定义, 故平凡语言 L_Σ 是 CFL 的特例, 不必专门讨论. $\mathbb{N}$ 和 $\mathbb{B}$ 涉及经典的递归函数和递归谓词, 所以, 我们只着重叙述 CFL 句子集合 $L(G)$ 和分析树集合 $\mathbb{L}(G)$ 的情形.

2.1 原始递归式和 MTPRF 的定义

定义 5 (基本函数和代入算子) 基本函数包括:

(1) 首元函数: $\text{Start} : \rightarrow R$, $\text{Start}() =_{df} R$ 的首元 (首元和后继涉及论域 ($\mathbb{B}$ 除外) 中的序关系, 见定义 9 和定理 1 的证明);

(2) 后继函数: $\text{Next} : D \rightarrow D$, $\text{Next}(x) =_{df} x$ 的后继元素;

(3) 投影函数: $U_i^m : D_1 \times \dots \times D_m \rightarrow D_i$, $1 \leq i \leq m$, $U_i^m(x_1, \dots, x_m) =_{df} x_i$.

代入算子: 由已知函数 $g_i : D_1 \times \dots \times D_n \rightarrow D^{(i)}$ ($i = 1, \dots, m$) 和 $h : D^{(1)} \times \dots \times D^{(m)} \rightarrow R$ 得到 $f : D_1 \times \dots \times D_n \rightarrow R$, $f(x_1, \dots, x_n) =_{df} h(g_1(x_1, \dots, x_n), \dots, g_m(x_1, \dots, x_n))$.

2.1.1 原始递归式

定义 6 (原始递归式 ($\mathbb{N}$)) 设函数 $f : \mathbb{N} \times D \rightarrow R$, 如果

$$f(0, y) =_{df} h(y), \quad f(x+1, y) =_{df} g(x, y, f(x, y)),$$

则称 f 是用函数 h 和 g 的原始递归式定义的, $h : D \rightarrow R$, $g : \mathbb{N} \times D \times R \rightarrow R$.

定义 7 (原始递归式 ($L(G)$ 和 $\mathbb{L}(G)$)) 设要定义函数 $f : D_1 \times D_2 \rightarrow R$, $D_1 = L(G)(\mathbb{L}(G))$ 由文法 $G = (V_N, V_T, P, X_1)$ 产生, 称此函数 f 是由原始递归式所定义的, 如果对于 X_1 的每个派生符号 X 和每个产生式 $X \rightarrow P$, 有已知函数 h_P , 使得

(1) 当 P 是常项, 即 P 中不含非终极符时, 有规则 (适用于 f 的首一自变元值是由相应常项产生式推出的情形)

$$f(P, y) =_{df} h_P(y); \quad (1)$$

(2) 当 P 是变项, 即 $P = u_0 Z_1 u_1 \dots Z_r u_r$, 其中 $u_0, \dots, u_r$ 为终极符串 (可以是空串), $Z_1, \dots, Z_r$ 为非终极符, 有规则 (适用于 f 的首一自变元值是由相应变项产生式推出的情形)

$$f(P, y) =_{df} h_P(Z_1, \dots, Z_r, y, \dots, f(Z_j, y), \dots), \quad 1 \leq j \leq r. \quad (2)$$

上式中, h_P 式中的函数 f 亦可不出现, 但如出现, f 的首一自变元位置对应的必须是 X_1 或其相应的派生符号. 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 D_1 中取值. 注意, 每一个非终极符 Z_j 均被视为不同.

对所有产生式 $X \rightarrow P$, 分别按以上情形取规则组的全体. 规则的适用场合见下文 (第 2.3.1、4.2 和 4.8 小节).

定义 8 (MTPRF 函数) 多类型原始递归函数 MTPRF 是由基本函数出发、有穷次应用代入算子、原始递归式而构建的函数的类.

2.2 极小算子和 MTRF 的定义

定义 9 (极小算子) 设有函数 $g: D_1 \times D_2 \rightarrow \mathbb{B}$, 则极小算子 (μ 算子) $\mu_y[g(y, x)]$ 求得的值定义为集合 $\{y \mid y \in D_1, \text{使得 } g(y, x) = \text{true}\}$ 中“最小”的 y , 即从 $\text{Start}()$ 开始, 逐次用 Next 函数枚举, 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 , $g(y, x)$ 必须都有定义, 否则 $\mu_y[g]$ 无定义. 枚举的顺序关系是 $\prec$. 在 $\mathbb{N}$ 中, $\prec$ 是小于 ($<$) 关系, 在 CFL 和 CFG 分析树集合中, 是分层词典序^[5,6].

这样的 y 决定于 g , 是 x 的函数, 即 $\mu_y[g(y, x)]$ 定义了一个函数 $f: D_2 \rightarrow D_1$, $f(x) =_{df} \mu_y[g(y, x)]$.

定义 10 (MTRF 函数) 多类型递归函数 MTRF 是由基本函数出发、有穷次应用代入、原始递归式和极小算子而构建的函数的类.

2.3 原始递归式和极小算子的求值

MTRF 可认为是通过一个有穷的函数序列来定义的. 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、原始递归式、极小算子而得到的.

以下将对如此的有穷序列的函数定义, 描述函数的求值方法.

假定我们要对函数 $f(x)$ 求值, 自然我们知道 f 的定义和所涉及的文法. 这里只需要叙述两种情形.

2.3.1 原始递归式 ($L(G)$ 和 $\mathbb{L}(G)$ 情形)

在函数 f 的定义中, 应该包含各个有关已知函数 h 的定义在函数定义序列中何处的信息, 以及产生式信息.

对 $f(x, y)$ 的求值需取其自变元的值 x 和 y . 考察 x 的分析树 τ (在 $L(G)$ 的情形需要分析树, 否则 x 本身就是分析树).

如果分析树 τ 的高度是 1, 那么, $f(x, y)$ 的计算与 f 的其他函数值无关, 从分析树 τ , 直接查悉 f 的定义中相关的产生式及相应的已知函数 h_P 的定义, 按 $f(x, y) = h_P(y)$ 求值即可.

如果分析树 τ 的高度 > 1 , 那么, $f(x, y)$ 的计算依赖于 f 的较低自变元值位上的函数值. 从分析树 τ , 直接查悉 f 的定义中相关的产生式及相应的已知函数 h_P 的定义, 求值问题转化为先对若干个 $f(x_j, y)$ 求值, 然后再把它们作为自变元的值来对已知函数 h_P 求值. 这些 x_j 是 x 按变项 P 分解式的分量, 其分析树都是 τ 的真子树, 树的高度较 τ 为低. 这样继续进行下去, 树的高度逐次降低, 最终归结为对有穷个高度为 1 的分析树来计算函数 f , 中间穿插着再把它们作为自变元的值, 对若干个已知函数 h_P 求值, 最后得到 $f(x, y)$ 的值.

2.3.2 极小算子

f 是利用已知函数 g 经由极小算子 $\mu_y[g(y, x)]$ 定义的. 在函数 f 的定义中, 应该包含有关已知函数 g 的定义在函数定义序列中何处的信息, 以及文法信息. 依下式计算 $f(x) \simeq \phi(\text{Start}(), x)$,

$$\phi(y, x) =_{df} \begin{cases} y, & \text{如果 } g(y, x) = \underline{\text{true}}, \\ \phi(\text{Next}(y), x), & \text{否则.} \end{cases}$$

注意, 函数 ϕ 在自变元某值位上的值, 不是依赖于该函数在较低值位上的值, 而是依赖于该函数在较高值位上的值. 可能不存在 y 值能使函数 $g(y, x)$ 取值 $\underline{\text{true}}$. 此情形下计算便无终结之时, 函数 $f(x)$ 无定义.

3 与其他论域上定义的递归函数的关系

定理 1 自然数递归函数、字上递归函数、CFL 上定义的递归函数、CFG 分析树的递归函数、多类型递归函数, 彼此是等价的. 上面陈述中, 把“递归函数”换成“原始递归函数”, 结论仍然成立.

证明 不提供完整的形式证明, 只给出非形式的证明思路陈述.

设有论域 D , 又有互为逆映射的 $\alpha_D : D \rightarrow \mathbb{N}$ 和 $\alpha_D^{-1} : \mathbb{N} \rightarrow D$. 通过它们在 D 和 $\mathbb{N}$ 之间建立了 1-1 对应关系, 即对任何 $d \in D, n \in \mathbb{N}$, 恒有 $\alpha_D^{-1}(\alpha_D(d)) = d, \alpha_D(\alpha_D^{-1}(n)) = n$.

如果函数 $f : D_1 \times \cdots \times D_n \rightarrow R, f^* : \mathbb{N} \times \cdots \times \mathbb{N} \rightarrow \mathbb{N}$, 满足

$$f^*(x_1, \dots, x_n) = \alpha_R(f(\alpha_{D_1}^{-1}(x_1), \dots, \alpha_{D_n}^{-1}(x_n))),$$

则称 f 与 f^* 等价 (参见文献 [2]).

当 D 是 $L(G)$ 和 $\mathbb{L}(G)$ 的情形, 文献 [5, 6] 在集合 D 中定义了分层词典序, 给出以下函数 (用程序语言写的, 可以认为是原始递归的):

- (1) $\text{Start} : \rightarrow D, \text{Start}()$ 是集合 D 的首元;
- (2) $\text{Next} : D \rightarrow D, \text{Next}(x)$ 是 D 中 x 的后继元素;
- (3) $\text{L2N} : D \rightarrow \mathbb{N}, \text{L2N}(x)$ 是 D 中 x 的序号;
- (4) $\text{N2L} : \mathbb{N} \rightarrow D, \text{N2L}(n)$ 是 D 中第 n 个元素.

通过它们, 建立了集合 D 与非负整数集合 $\mathbb{N}$ 间的保序对应. 使得 $\text{Start}()$ 与 0 对应, $\text{Next}(x)$ 与 $\text{L2N}(x)+1$ 对应. 函数 Start 和 Next 可以用作枚举 D 中元素的手段. 函数 L2N 和 N2L 分别就是上面要求的 α_D 和 α_D^{-1} .

字上递归函数的情形. 第 2 节开头处已指出, 可归为 CFL, 无需再进行讨论.

关于自然数集合上的递归函数, 我们采用 Gödel-Kleene 的定义 (参见文献 [2]), 即原始递归函数类是包含三种基本函数: 零函数 $O(x) = 0$, 后继函数 $S(x) = x + 1$, 投影函数 $U_i^m(x_1, \dots, x_m) = x_i$, 并且封闭于代入算子和原始递归算子的最小函数类; 递归函数类是用定义原始递归函数类的各种手段和极小算子而得到的.

在由函数 f 构造函数 f^* 的过程中, L2N 和 N2L 可用来建立集合 D 与 $\mathbb{N}$ 之间的 1-1 对应关系. 函数 Start 和零函数, Next 和后继函数互为对偶, 两组投影函数和代入算子的对应非常直接. MTPRF 的原始递归式转写到 $\mathbb{N}$ 上函数类中时, 变成串值递归式. 根据文献 [7], 在自然数递归函数类中, 串值递归式最终可以化归为原始递归式. 不同论域中枚举的手段和极小算子的对应是直接的.

函数构造的各个手段不难得到检验, 据此得到结论: 相关函数类彼此等价. $\square$

4 通用求值函数的构造

由第 3 节的论证, 在自然数递归函数理论中的结果在 MTRF 中相应地也都成立, 例如, 存在 MTRF 的通用求值函数. 因为研究 MTRF 的初衷是希望在计算机上应用, 我们不能满足于由等价性处理得到的通用求值函数, 宁可直接考察相关问题. 按照本节中叙述的方法, 应该能够构造出函数 `eval` 的定义式. 依其计算, 结果是实现求值意图, 即定理 2.

定理 2 有 MTRF 函数 `eval`, 使得当 f 是 MTPRF 函数时, $\text{eval}(f, x) = f(x)$; 当 f 是 MTRF 函数时, $\text{eval}(f, x) \simeq f(x)$.

本节将通过给出函数 `eval` 的定义式来证明此定理, 本节其余部分用来讨论 `eval` 定义式的构造问题 (`eval` 工作时要用到两个常量 `FnDefList` 和 `CfgDefList`, 分别包含函数 f 涉及的所有函数定义及文法定义. `eval` 作为 MTRF 函数式语言的解释器在计算机上实现时, 应考虑到 `FnDefList` 和 `CfgDefList` 中未出现 `eval` 的首变元 f 的可能性. 此时可将待求值函数 f 视为无处定义函数 $f_\emptyset$ 来处置, 此情形可被 `eval` 检测出来).

4.1 相等关系和分情形定义

下面要用一些辅助函数和简略记法, 即相等关系和分情形定义.

在文献 [7] 中曾指出, $a = b$ 是一个原始递归关系, 即可以定义原始递归函数 $\text{EQ} : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{B}$, 当 $x = y$ 时, $\text{EQ}(x, y) = \text{true}$; 当 $x \neq y$ 时, $\text{EQ}(x, y) = \text{false}$.

据此可定义更广论域 D 上的相等关系 eq , $\text{eq} : D \times D \rightarrow \mathbb{B}$, $\text{eq}(x, y) =_{df} \text{EQ}(\text{L2N}(x), \text{L2N}(y))$.

现在, 给出写分情形定义的表示法. 设 $\text{if} : \mathbb{B} \times D \times D \rightarrow D$,

$$\text{if}(\text{true}, y, z) =_{df} y, \quad \text{if}(\text{false}, y, z) =_{df} z.$$

令 $(x \rightarrow y, z) =_{abbr} \text{if}(x, y, z)$,

$$(x_1 \rightarrow y_1, \dots, x_n \rightarrow y_n, z) =_{abbr} \text{if}(x_1, y_1, \text{if}(x_2, y_2, \dots, \text{if}(x_n, y_n, z) \dots)),$$

即分情形定义, 表示依次检查条件 $x_1, \dots, x_n$ 选择最先被满足者. 如果能保证这 n 个条件至少有一个被满足, 即 $x_1 \vee \dots \vee x_n = \text{true}$, 则 z 不必出现, 可以写为 $(x_1 \rightarrow y_1, \dots, x_n \rightarrow y_n)$.

4.2 `eval` 的求值意图

`eval(Start, x)` = 相关文法的首元,

`eval(Next, x)` = x 的后继,

`eval( $U_i^m, x_1, \dots, x_m$ )` = x_i ,

`eval( $f, x$ )` = `eval(h(eval( $g_1, x$ ), ..., eval( $g_m, x$ )))`. /* 对已知函数 $h, g_1, \dots, g_m$ 施用代入算子 */
/* $\mathbb{N}$ 中原始递归的情形就不列举了 */

对于 f 适用 $L(G)$ 和 $\mathbb{L}(G)$ 的原始递归式定义的情形, 则分别对 Z 的分析树高度为 1 (对应的 P 是常项), 或 Z 的分析树高度大于 1 (对应的 P 是变项) 的情形, 有

`eval( $f, Z, y$ )` = `eval( $h_P, y$ )`, /* 常项情形 */

`eval( $f, Z, y$ )` = `eval( $h_P, Z_1, \dots, Z_r, y, \dots, \text{eval}(f, Z_j, y), \dots$ )`, /* 变项情形 */

`eval( $f, x$ )` $\simeq$ `eval( $\phi, \text{Start}(), x$ )`. /* f 由极小算子定义. $f(x) \simeq \phi(\text{Start}(), x)$ */

/* $f(x) =_{df} \mu_y [g(y, x)]$, $\phi(y, x) =_{df} (g(y, x) \rightarrow y, \phi(\text{Next}(y), x))$ */

4.3 函数定义及有关信息的获得

为规定 eval 所依赖的函数定义和 CFG 定义的结构, 我们使用 BNF 记法来给出相应的文法. 它们具有直观的解释, 与定义 4 中句子的推导实质上是一致的. 例如,

$\langle \text{数字} \rangle ::= 0|1|2|3|4|5|6|7|8|9$

$\langle \text{整数} \rangle ::= \langle \text{数字} \rangle | \langle \text{整数} \rangle \langle \text{数字} \rangle$

第 1 条元语言公式是说, $0, 1, \dots, 9$ 是 $\langle \text{数字} \rangle$ 的元素, 此外无其他元素. 第 2 条公式是说, 集合 $\langle \text{数字} \rangle$ 的元素都是集合 $\langle \text{整数} \rangle$ 的元素, 整数后面再随以数字也是整数, 此外集合 $\langle \text{整数} \rangle$ 无其他元素.

下文也对所列元语言公式的含义给出由 $/\ast$ 和 $\ast/$ 所括的注释.

4.3.1 函数定义

$\langle \text{函数定义表} \rangle ::= \langle \text{函数定义} \rangle | \langle \text{函数定义表} \rangle ; \langle \text{函数定义} \rangle \quad /\ast \text{一串用 ; 隔开的函数定义} \ast/$

$\langle \text{函数定义} \rangle ::= \langle \text{函数名} \rangle : \langle \text{变元表} \rangle \rightarrow \langle \text{值名} \rangle \{ \langle \text{定义体} \rangle \}$

$\quad /\ast f: D_1 \times \dots \times D_n \rightarrow R \{ \text{定义体} \} \ast/$

$\langle \text{函数名表} \rangle ::= \langle \text{函数名} \rangle | \langle \text{函数名} \rangle , \langle \text{函数名表} \rangle \quad /\ast \text{一串用 , 隔开的函数名} \ast/$

$\langle \text{函数名} \rangle ::= \langle \text{英文字母串} \rangle \quad /\ast \text{函数名称} \ast/$

$\langle \text{变元表} \rangle ::= \langle \text{变元名} \rangle | \langle \text{变元表} \rangle \times \langle \text{变元名} \rangle \quad /\ast D_1 \times \dots \times D_n \ast/$

$\langle \text{变元名} \rangle ::= \mathbb{N} | \mathbb{L}(\langle \text{CFG 名} \rangle) | \mathbb{L}(\langle \text{CFG 名} \rangle) | \langle \text{平凡语言名} \rangle | \mathbb{B} \quad /\ast \text{定义域 } D \ast/$

$\langle \text{值名} \rangle ::= \langle \text{变元名} \rangle \quad /\ast \text{值域 } R \ast/$

$\langle \text{定义体} \rangle ::= \text{Zero} \quad /\ast \text{开始函数} \ast/$

$\text{Succ} \quad /\ast \text{后继函数} \ast/$

$\text{Proj}, \langle \text{变元总数} \rangle, \langle \text{投影位标} \rangle \quad /\ast \text{投影函数, } m, i \ast/$

$\text{Substi}, \langle \text{函数名} \rangle, \langle \text{函数名表} \rangle \quad /\ast \text{代入算子, } h, g_1, \dots, g_m \ast/$

$\text{Rec}, \langle \text{常项规则表} \rangle; \langle \text{变项规则表} \rangle \quad /\ast \text{原始递归, 常项规则表; 变项规则表} \ast/$

$\text{Mu}, \langle \text{函数名} \rangle \quad /\ast \text{极小算子, } g \ast/$

$\langle \text{变元总数} \rangle ::= \langle \text{整数} \rangle \quad /\ast m \ast/$

$\langle \text{投影位标} \rangle ::= \langle \text{整数} \rangle \quad /\ast i \ast/$

有些不重要的繁琐细节, 有选择地从略. 如 $\langle \text{平凡语言名} \rangle$ 未予定义.

4.3.2 CFG 定义

$\langle \text{CFG 定义表} \rangle ::= \langle \text{CFG 定义} \rangle | \langle \text{CFG 定义表} \rangle ; \langle \text{CFG 定义} \rangle \quad /\ast \text{一串用 ; 隔开的 CFG 定义} \ast/$

$\langle \text{CFG 定义} \rangle ::= \langle \text{CFG 名表} \rangle \{ \langle \text{终极符表} \rangle; \langle \text{产生式表} \rangle \} \quad /\ast V_N \{ V_T; \text{产生式表} \} \ast/$

$\langle \text{CFG 名表} \rangle ::= \langle \text{CFG 名} \rangle | \langle \text{CFG 名表} \rangle, \langle \text{CFG 名} \rangle \quad /\ast \text{以 , 隔开的非终极符表} \ast/$

$\langle \text{CFG 名} \rangle ::= \langle \text{以特殊字符开头以空格结束的英文字母串} \rangle \quad /\ast \text{文法名字, 也是非终极符} \ast/$

$\langle \text{终极符表} \rangle ::= \langle \text{终极符} \rangle | \langle \text{终极符表} \rangle, \langle \text{终极符} \rangle \quad /\ast \text{以 , 隔开的终极符, 设不含 , 和 ;} \ast/$

$\langle \text{产生式表} \rangle ::= \langle \text{产生式} \rangle | \langle \text{产生式表} \rangle, \langle \text{产生式} \rangle$

$\langle \text{产生式} \rangle ::= \langle \text{左部名} \rangle \rightarrow \langle \text{右部} \rangle$

$\langle \text{左部名} \rangle ::= \langle \text{CFG 名} \rangle$

$\langle \text{右部} \rangle ::= \varepsilon | \langle \text{终极符与非终极符串} \rangle \quad /\ast \text{项} \ast/$

假定通用求值函数 $\text{eval}(f, x)$ 工作时, 涉及的所有函数定义均已包含在函数定义表 **FnDefList** 中; 所有 CFG 定义均已包含在 CFG 定义表 **CfgDefList** 中, 其结构由上列元语言公式所规定.

至于如何从 **FnDefList** 中找到并且抽出需要的函数定义来, 从 **CfgDefList** 中找到并且抽出需要的 CFG 定义来, 又如何区分出是什么函数, 以及不同的函数如何求值等问题, 是后面各小节的内容.

注意, 每个非终极符对应一个 CFG. 它所定义的句子集合以 “L (CFG 名)” 为名, 所定义的分析树集合以 “ $\mathbb{L}$ (CFG 名)” 为名.

通用求值函数 $\text{eval}(f, x)$ 中, f 是待求值的函数名, x 是自变元的值本身.

基本函数有固定的名字, 定义域可以不同. 可随意定义的函数是用它们构造出来的.

4.3.3 从函数定义表中抽取所要的定义

辅助函数

$\text{head} : \langle \text{函数定义表} \rangle \rightarrow \langle \text{函数定义表} \rangle$ /* 在函数定义表中去掉末尾的函数定义 */

$\text{last} : \langle \text{函数定义表} \rangle \rightarrow \langle \text{函数定义} \rangle$ /* 在函数定义表中取末尾的函数定义 */

$\text{head}(\langle \text{函数定义} \rangle) =_{df} \langle \text{函数定义} \rangle$

$\text{head}(\langle \text{函数定义表} \rangle; \langle \text{函数定义} \rangle) =_{df} \langle \text{函数定义表} \rangle$ /* 下加横线表示是一整体, 其中的分隔符不是用来隔开函数自变元的 */

$\text{last}(\langle \text{函数定义} \rangle) =_{df} \langle \text{函数定义} \rangle$

$\text{last}(\langle \text{函数定义表} \rangle; \langle \text{函数定义} \rangle) =_{df} \langle \text{函数定义} \rangle$

4.3.4 从函数定义中分离出函数名和定义体

辅助函数

$\text{NameOf} : \langle \text{函数定义} \rangle \rightarrow \langle \text{函数名} \rangle$ /* 在函数定义中抽取函数名 */

$\text{findOut} : \langle \text{函数名} \rangle \times \langle \text{函数定义表} \rangle \rightarrow \langle \text{函数定义} \rangle$ /* 从函数定义表取函数名所指函数定义 */

$\text{NameOf}(\langle \text{函数名} \rangle : \langle \text{变元表} \rangle \rightarrow \langle \text{值名} \rangle \{ \langle \text{定义体} \rangle \}) =_{df} \langle \text{函数名} \rangle$

$\text{findOut}(\langle \text{函数名} \rangle, \langle \text{函数定义表} \rangle) =_{df} (\text{eq}(\langle \text{函数名} \rangle, \text{NameOf}(\text{last}(\langle \text{函数定义表} \rangle)))$
 $\rightarrow \text{last}(\langle \text{函数定义表} \rangle),$
 $\text{findOut}(\langle \text{函数名} \rangle, \text{head}(\langle \text{函数定义表} \rangle)))$.

为行文简洁, 下面将使用缩写记号. 令

$\text{Def_of_f} =_{abbr} \text{findOut}(f, \text{FnDefList}).$ /* 以 f 为函数名的函数定义 */

现在引入函数 $\text{FnDefBody} : \langle \text{函数定义} \rangle \rightarrow \langle \text{定义体} \rangle$ /* 从函数定义中抽取定义体 */

$\text{FnDefBody}(\langle \text{函数名} \rangle : \langle \text{变元表} \rangle \rightarrow \langle \text{值名} \rangle \{ \langle \text{定义体} \rangle \}) =_{df} \langle \text{定义体} \rangle$

于是, $\text{FnDefBody}(\text{Def_of_f})$ 是 f 的函数定义体, 同样地, 令

$\text{Fn_f_DefBody} =_{abbr} \text{FnDefBody}(\text{Def_of_f}).$

现在引入函数 $\text{FnCh} : \langle \text{定义体} \rangle \rightarrow \langle \text{保留字} \rangle$ /* 从定义体中抽取函数的特征 */

$\text{FnCh}(\text{Zero}) =_{df} \text{Zero}$

$\text{FnCh}(\text{Succ}) =_{df} \text{Succ}$

$\text{FnCh}(\text{Proj}, \langle \text{变元总数} \rangle, \langle \text{投影位标} \rangle) =_{df} \text{Proj}$

$\text{FnCh}(\text{Substi}, \langle \text{函数名} \rangle, \langle \text{函数名表} \rangle) =_{df} \text{Substi}$

$\text{FnCh}(\text{Rec}, \langle \text{常项规则表} \rangle; \langle \text{变项规则表} \rangle) =_{df} \text{Rec}$

$\text{FnCh}(\text{Mu}, \langle \text{函数名} \rangle) =_{df} \text{Mu}$

下文中令 $\text{Ch_f} =_{abbr} \text{FnCh}(\text{Fn_f_DefBody}).$ /* f 的函数特征 */

现在可以写出函数 eval 的定义式,

$$\begin{aligned} \text{eval}(f, x) =_{df} & (\text{eq}(\text{Ch}_f, \text{Zero}) \rightarrow \text{CaseOfStart}, \\ & \text{eq}(\text{Ch}_f, \text{Succ}) \rightarrow \text{CaseOfNext}, \\ & \text{eq}(\text{Ch}_f, \text{Proj}) \rightarrow \text{CaseOfProj}, \\ & \text{eq}(\text{Ch}_f, \text{Substi}) \rightarrow \text{CaseOfSubsti}, \\ & \text{eq}(\text{Ch}_f, \text{Rec}) \rightarrow \text{CaseOfRec}, \\ & \text{eq}(\text{Ch}_f, \text{Mu}) \rightarrow \text{CaseOfMu}). \end{aligned}$$

所列 6 种缩写 **CaseOf**· 是适用相应情形的求值定义式, 在以后给出, 这就完成了函数 **eval** 的定义.

4.4 开始函数求值的定义式

对于开始函数的情形, 即 $f(x) =_{df} \text{Start}()$, 此时, x 无关紧要, 函数值则依 f 的函数定义 **Def_of_f** 的值名而定. 该值名可以用以下函数来计算,

$$\begin{aligned} \text{Range} : \langle \text{函数定义} \rangle & \rightarrow \langle \text{值名} \rangle \quad /* \text{从函数定义中抽取值域名} */ \\ \text{Range}(\langle \text{函数名} \rangle : \langle \text{变元表} \rangle & \rightarrow \langle \text{值名} \rangle \{ \langle \text{定义体} \rangle \}) =_{df} \langle \text{值名} \rangle \\ \text{令 } \text{Range_of_f} =_{abbr} & \text{Range}(\text{Def_of_f}). \quad /* \text{即函数 } f \text{ 的值域名} */ \\ \text{CaseOfStart} =_{abbr} & (\text{eq}(\text{Range_of_f}, \mathbb{N}) \rightarrow 0, \\ & \text{eq}(\text{Range_of_f}, \mathbb{L}(\langle \text{CFG 名} \rangle)) \rightarrow \text{相应 CFL 的首句}, \\ & \quad /* \text{从 CfgDefList 中抽出相关 CFG 定义, 用文献 [5] 中算法求首句} */ \\ & \text{eq}(\text{Range_of_f}, \mathbb{L}(\langle \text{CFG 名} \rangle)) \rightarrow \text{相应文法的首一分析树}, \\ & \quad /* \text{从 CfgDefList 中抽出相关 CFG 定义, 用文献 [6] 中算法求首树} */ \\ & \text{eq}(\text{Range_of_f}, \text{平凡语言名}) \rightarrow \text{相应平凡语言的首句}). \\ & \quad /* \text{从 CfgDefList 中抽出相关 CFG 定义, 用文献 [3] 中算法求首句} */ \end{aligned}$$

4.5 后继函数求值的定义式

对于后继函数的情形, 即 $f(x) =_{df} \text{Next}(x)$, 此时, f 的定义域与值域相同, 根据 f 的函数定义 **Def_of_f** 的值名 **Range(Def_of_f)** (缩写记号 **Range_of_f**), 利用相关文献中提供的算法, 来求得 x 的后继.

$$\begin{aligned} \text{CaseOfNext} =_{abbr} & (\text{eq}(\text{Range_of_f}, \mathbb{N}) \rightarrow x + 1, \\ & \text{eq}(\text{Range_of_f}, \mathbb{L}(\langle \text{CFG 名} \rangle)) \rightarrow \text{Next}(x), \quad /* \text{相应文法中 } x \text{ 的后继句子} */ \\ & \quad /* \text{从 CfgDefList 中抽出相关 CFG 定义, 用文献 [5] 中算法求后继句} */ \\ & \text{eq}(\text{Range_of_f}, \mathbb{L}(\langle \text{CFG 名} \rangle)) \rightarrow \text{Next}(x), \quad /* \text{相应 CFL 中 } x \text{ 的后继分析树} */ \\ & \quad /* \text{从 CfgDefList 中抽出相关 CFG 定义, 用文献 [6] 中算法求后继树} */ \\ & \text{eq}(\text{Range_of_f}, \text{平凡语言名}) \rightarrow \text{Next}(x)). \quad /* \text{相应平凡语言中 } x \text{ 的后继句子} */ \\ & \quad /* \text{从 CfgDefList 中抽出相关 CFG 定义, 用文献 [3] 中算法求后继句} */ \end{aligned}$$

4.6 投影函数求值的定义式

求得其定义体后, 从中分离出投影位标 i ; 在变元值 x 中, 选中并分离出第 i 个分量 x_i , 即所求函数值.

自变元值 x 具有如下文法结构:

$\langle \text{值} \rangle ::= \langle \text{值元} \rangle | \langle \text{值元} \rangle, \langle \text{值} \rangle$

$\langle \text{值元} \rangle ::= \langle \text{值名} \rangle$

定义函数 $\text{EraseLeft} : \mathbb{N} \times \langle \text{值} \rangle \rightarrow \langle \text{值} \rangle$ /* 在值中抹去若干个最左值元 */

$\text{EraseLeft}(0, \langle \text{值元} \rangle) =_{df} \langle \text{值元} \rangle$ /* 抹去 0 个 */

$\text{EraseLeft}(0, \langle \text{值元} \rangle, \langle \text{值} \rangle) =_{df} \langle \text{值元} \rangle, \langle \text{值} \rangle$ /* 抹去 0 个 */

$\text{EraseLeft}(n+1, \langle \text{值元} \rangle) =_{df} \langle \text{值元} \rangle$ /* 只剩 1 个则不抹 */

$\text{EraseLeft}(n+1, \langle \text{值元} \rangle, \langle \text{值} \rangle) =_{df} \text{EraseLeft}(n, \langle \text{值} \rangle)$ /* 抹去 $n+1$ 个 */

定义函数 $\text{LeftValue} : \langle \text{值} \rangle \rightarrow \langle \text{值元} \rangle$ /* 在值中取最左值元 */

$\text{LeftValue}(\langle \text{值元} \rangle) =_{df} \langle \text{值元} \rangle$

$\text{LeftValue}(\langle \text{值元} \rangle, \langle \text{值} \rangle) =_{df} \langle \text{值元} \rangle$

定义函数 $\text{ProjIndex} : \langle \text{定义体} \rangle \rightarrow \mathbb{N}$ /* 在投影函数定义体中取投影位标 */

$\text{ProjIndex}(\mathbf{Proj}, \langle \text{变元总数} \rangle, \langle \text{投影位标} \rangle) =_{df} \langle \text{投影位标} \rangle$

$\text{ProjIndex}(\text{其他情形}) =_{df}$ 无所谓 /* 原始递归的缩减定义方式. 有时只对自变元取值的某些情形感兴趣, 对其余情形不一列举 */

对于投影函数的情形, 即

$$f(x) =_{df} f(x_1, \dots, x_m) = \mathbf{U}_i^m(x_1, \dots, x_m) = x_i.$$

令 $\mathbf{Index} =_{abbr} \text{ProjIndex}(\mathbf{Fn_f_DefBody})$ /* 即投影位标 i */

$\mathbf{CaseOfProj} =_{abbr} \text{LeftValue}(\text{EraseLeft}(\mathbf{Index} - 1, x)).$

4.7 代入算子求值的定义式

对于施用代入算子的情形, 即 $f(x) =_{df} h(g_1(x), \dots, g_m(x)).$

定义辅助函数, 以便从定义体中抽取函数 f 的其他有关信息.

$\text{SubstiFn} : \langle \text{定义体} \rangle \rightarrow \langle \text{函数名} \rangle$

$\text{SubstiFn}(\mathbf{Substi}, \langle \text{函数名} \rangle, \langle \text{函数名表} \rangle) =_{df} \langle \text{函数名} \rangle$

$\text{SubstiFn}(\text{其他情形}) =_{df}$ 无所谓

$\text{SubstiFnList} : \langle \text{定义体} \rangle \rightarrow \langle \text{函数名表} \rangle$

$\text{SubstiFnList}(\mathbf{Substi}, \langle \text{函数名} \rangle, \langle \text{函数名表} \rangle) =_{df} \langle \text{函数名表} \rangle$

$\text{SubstiFnList}(\text{其他情形}) =_{df}$ 无所谓

显然, $\text{SubstiFn}(\mathbf{Fn_f_DefBody})$ 就是 h . 下文中令 $h =_{abbr} \text{SubstiFn}(\mathbf{Fn_f_DefBody}).$

同样, $\text{SubstiFnList}(\mathbf{Fn_f_DefBody})$ 就是函数名表 $g_1, \dots, g_m.$

下文中令 $\mathbf{NmList_f} =_{abbr} \text{SubstiFnList}(\mathbf{Fn_f_DefBody}).$

函数 $\text{NrFnList} : \langle \text{函数名表} \rangle \rightarrow \mathbb{N}$ /* 计算函数名表中函数名的个数 */

$\text{NrFnList}(\langle \text{函数名} \rangle) =_{df} 1$

$\text{NrFnList}(\langle \langle \text{函数名} \rangle, \langle \text{函数名表} \rangle \rangle) =_{df} \text{Next}(\text{NrFnList}(\langle \text{函数名表} \rangle))$

显然, $\text{NrFnList}(\mathbf{NmList_f})$ 就是 $g_1, \dots, g_m$ 的个数 m . 下文中令 $\mathbf{TotalNrg} =_{abbr} \text{NrFnList}(\mathbf{NmList_f}).$

下面的函数用于从函数名表中逐个取函数名,

$\text{CurrentNm} : \langle \text{函数名表} \rangle \rightarrow \langle \text{函数名} \rangle$

$\text{RestList} : \langle \text{函数名表} \rangle \rightarrow \langle \text{函数名表} \rangle$
 $\text{CurrentNm}(\langle \text{函数名} \rangle) =_{df} \langle \text{函数名} \rangle$
 $\text{CurrentNm}(\langle \text{函数名} \rangle, \langle \text{函数名表} \rangle) =_{df} \langle \text{函数名} \rangle$ /* 取表中第 1 个函数名 */
 $\text{RestList}(\langle \text{函数名} \rangle) =_{df} \langle \text{函数名} \rangle$
 $\text{RestList}(\langle \text{函数名} \rangle, \langle \text{函数名表} \rangle) =_{df} \langle \text{函数名表} \rangle$ /* 丢弃表中第 1 个函数名 */
 $\text{CurrentNm}(\text{NmList_f})$ 就是 g_1 , 令 $g_1 =_{abbr} \text{CurrentNm}(\text{NmList_f})$.
 $\text{RestList}(\text{NmList_f})$ 就是 $g_2, \dots, g_m$, 令 $g_2 \cdots =_{abbr} \text{RestList}(\text{NmList_f})$.
 依次地, $\text{CurrentNm}(g_2 \cdots)$ 就是 g_2 , 令 $g_2 =_{abbr} \text{CurrentNm}(g_2 \cdots)$.
 $\text{RestList}(g_2 \cdots)$ 就是 $g_3, \dots, g_m$, 令 $g_3 \cdots =_{abbr} \text{RestList}(g_2 \cdots)$. 如此继续 **TotalNrg** 步, 最后得
 $\text{RestList}(g^{\text{TotalNrg}} \cdots)$ 就是 g_m , 令 $g_m \cdots =_{abbr} \text{RestList}(g^{\text{TotalNrg}} \cdots)$.
 $\text{CurrentNm}(g_m \cdots)$ 就是 g_m , 令 $g_m =_{abbr} \text{CurrentNm}(g_m \cdots)$.
 $\text{CaseOfSubsti} =_{abbr} \text{eval}(h, (\text{eval}(g_1, x), \dots, \text{eval}(g_m, x)))$.

4.8 原始递归求值的定义式

对于原始递归的情形: 对第一变元递归, 根据该变元值, 分别择取相应常项规则或者变项规则, 得到 **eval** 的递归算式, 有 $\mathbb{N}, L(G), \mathbb{L}(G)$ 的不同选择 (平凡语言的情形可以归入 $L(G)$).

变元 x 由若干分量组成, 即 $x = (z, y)$. 考虑第一分量 z 的定义域是 $L(G)$ 和 $\mathbb{L}(G)$ 的情形, 按照规定, 求值算法意图可表为

$$\begin{aligned}
 f(z, y) = & (\text{Suited}(z, \text{常项}_1) \rightarrow h_1(y), \dots, \text{Suited}(z, \text{常项}_m) \rightarrow h_m(y), \\
 & \text{Suited}(z, \text{变项}_1) \rightarrow \tilde{h}_1(Z_1, \dots, Z_r, y, f(Z_j, y), \dots), \dots, \\
 & \text{Suited}(z, \text{变项}_n) \rightarrow \tilde{h}_n(Z_1, \dots, Z_r, y, f(Z_j, y), \dots))
 \end{aligned}$$

函数组 $h_1, \dots, h_m, \tilde{h}_1, \dots, \tilde{h}_n$ 中每个函数的定义, 自应包含在函数定义表 **FnDefList** 中.

函数 f 定义体的常项规则表含如下形式的信息:

$$\text{常项}_1 : \text{函数名 } h_1, \dots, \text{常项}_m : \text{函数名 } h_m$$

即

$$\langle \text{常项规则表} \rangle ::= \langle \text{常项信息} \rangle, \langle \text{常项规则表} \rangle$$

$$\langle \text{常项信息} \rangle ::= \langle \text{终极符号串} \rangle : \langle \text{函数名} \rangle$$

函数 f 定义体的变项规则表含如下形式的信息:

$$\text{变项}_1 : \text{函数名 } \tilde{h}_1 : \langle \text{变元规则表} \rangle, \dots, \text{变项}_n : \text{函数名 } \tilde{h}_n : \langle \text{变元规则表} \rangle$$

即

$$\langle \text{变项规则表} \rangle ::= \langle \text{变项信息} \rangle; \langle \text{变项规则表} \rangle$$

$$\langle \text{变项信息} \rangle ::= \langle \text{非终极与终极符号串} \rangle : \langle \text{函数名} \rangle : \langle \text{变元规则表} \rangle$$

$$\langle \text{变元规则表} \rangle ::= \langle \text{变元规则} \rangle | \langle \text{变元规则} \rangle, \langle \text{变元规则表} \rangle$$

$$\langle \text{变元规则} \rangle ::= \langle \text{整数} \rangle | \star \langle \text{整数} \rangle$$

函数 **Suited** 的含义如下. 检查 z 的分析树, 考察推导首步所用产生式. 如果是常项产生式, 则相应的 **Suited**(z , 常项) 被满足, 随后的函数 h 计算不涉及 z 值.

如果是变项产生式 $Z \rightarrow u_0 Z_1 u_1 \cdots Z_r u_r$, 则该分析树形如

$$\begin{array}{c}
 Z \\
 \downarrow \\
 u_0 Z_1 u_1 \cdots Z_r u_r, \\
 \downarrow \quad \downarrow \\
 v_1 \quad v_r
 \end{array}$$

推出句子

$$z = u_0 v_1 u_1 \dots v_r u_r.$$

相应的 $\text{Suited}(z, \text{变项})$ 被满足. 随后的函数 $\tilde{h}$ 计算, 自变元的值不仅涉及 y , 还会涉及 z 按该变项的分解式和被定义函数 f 在其自变元的较低值位处的值. 我们通过变元规则表来标识, 变元规则表元素个数与相应函数 $\tilde{h}$ 自变元数目相同, 用来指出函数 $\tilde{h}$ 自变元的取值方式. 具体地, 0 表示该处用 y 代入; 正整数 j 表示在该处用 v_j 代入; $\star j$ 表示在该处要用函数 $f(v_j, y)$ 的值代入.

关于求值函数 $\text{eval}(f, x)$ 在原始递归式情形的定义方式, 因所涉及的技巧前面已有展示和篇幅限制, 我们将只讲做什么, 不讲如何做的细节.

由前所述, 我们已经得到函数 f 的定义 $\text{Def_of_}f$ 、定义体 $\text{Fn_}f_ \text{DefBody}$ 和函数特征 $\text{Ch_}f$. 由函数特征是 Rec , 识别出函数 f 是用原始递归式定义的.

接下来需要把变元 x 分拆成 z 和 y . 从函数 f 的定义 $\text{Def_of_}f$ 中的变元表, 查第一个变元名, 以获知 z 的定义域, 以决定是何种原始递归. $\mathbb{N}$ 中的原始递归不在此讨论, 平凡语言也不讨论, 只留下 $\mathbb{L}(\mathbb{G})$ 和 $\mathbb{L}(\mathbb{G})$, 两者的处理是类似的.

它们都需要用函数 Suited 来检查 z 的分析树, 获知推导首步所用产生式, 抽出分量 $v_1, \dots, v_r$ 或者子树

$$\begin{array}{c} Z_1 \quad \quad \quad Z_r \\ \Downarrow \star, \dots, \Downarrow \star, \\ v_1 \quad \quad \quad v_r \end{array}$$

这里的关键之处在于如何从句子求分析树和求分量. 完成此任务的原始递归函数参见文献 [4]. 据此可以构造出函数 Suited 和有关的辅助函数的定义来.

上文中的 $\text{Suited}(z, \text{常项}_i) \rightarrow h_i(y)$ 只是算法意图的描述, 在函数 eval 的定义中, 实际要产生的却是定义式 $\text{Suited}(z, \text{常项}_i) \rightarrow \text{eval}(h_i, y)$.

相应地, 对应算法意图描述

$$\text{Suited}(z, \text{变项}_k) \rightarrow \tilde{h}_k(Z_1, \dots, Z_r, y, f(Z_j, y), \dots),$$

实际要的是定义式

$$\text{Suited}(z, \text{变项}_k) \rightarrow \text{eval}(\tilde{h}_k, v_1, \dots, v_r, y, \text{eval}(f, v_j, y), \dots),$$

其中涉及的 z 、常项 _{i} 、变项 _{j} 和 $h_i, \tilde{h}_j, y, v_1, \dots, v_r$ 都是 (本文未写出的) 辅助函数的缩写. 其后, 令 $\text{CaseOfRec} =_{\text{abbr}} (\text{Suited}(z, \text{常项}_1) \rightarrow \text{eval}(h_1, y), \dots,$

$$\text{Suited}(z, \text{变项}_1) \rightarrow \text{eval}(\tilde{h}_1, v_1, \dots, v_r, y, \text{eval}(f, v_j, y), \dots), \dots).$$

4.9 极小算子求值的定义式

对于极小算子的情形, 对第一变元枚举求最先满足条件者, 该变元值有 $\mathbb{N}, \mathbb{L}(\mathbb{G}), \mathbb{L}(\mathbb{G})$ 的不同选择 (平凡语言不专门讨论).

当获知函数 f 是用极小算子定义的情形时, 我们已经得到 f 的函数定义

$$\text{Def_of_}f = f : D_2 \rightarrow D_1 \{\text{Mu}, \text{函数名}g\}, \quad f(x) =_{df} \mu_y [g(y, x)].$$

自然, 在函数定义表 FnDefList 中有函数 g 的定义, 并且与函数 f 的情形类似, 可以获得它的函数定义 $\text{Def_of_}g$. 注意, 此时必须有

$$\text{Def_of_}g = g : D_1 \times D_2 \rightarrow \mathbb{B}\{\text{定义体}\},$$

即函数 g 和 f 的定义域与值域有必须满足的约束性关联.

为对 $f(x)$ 求值, 要做的事是构造一个新函数 $\phi: D_1 \times D_2 \rightarrow D_1$,

$$\phi(y, x) =_{df} (g(y, x) \rightarrow y, \phi(\text{Next}(y), x)),$$

并且把函数 ϕ 的定义放入函数定义表 **FnDefList** 中,

$$f(x) \simeq \phi(\text{Start}(), x)$$

$$\text{CaseOfMu} =_{abbr} \text{eval}(\phi, \text{Start}(), x)$$

对函数 ϕ 求值时用求后继元素的函数 **Next**. 对于定义域 $\mathbb{N}$, 就是加 1 运算; 对于 CFL 句子和 CFG 分析树, 文献 [5, 6] 分别提供了算法 (都是原始递归的). 平凡语言是 CFL 的特例, 不需要专门讨论.

至此, 全部六种情形的定义式构造完毕.

5 讨论

可计算性研究丰富的理论成果, 是以潜在无穷的时间、空间为前提的. 对于可解的问题, 只要照给定的算法一直做下去, 最后一定能得到结果. 然而, 不论技术如何进步, 对人和计算机, 时间和空间都是有穷的. 理论成果指明了方向: 这个问题是可解的! 实践者要找出办法: 在计算机上该怎么做?

作者试从实践者的角度来观察和思考. 下面列出本文中未涉及的某些可供思考的问题.

如何能更简捷地定义新函数? 如何定义出合乎我们需求的函数 (程序正确性问题)? 如何使复杂的函数定义容易理解? 如何能高效地对函数求值?

致谢 周巢尘研究员认真审读了本文并提出问题, 使本文的叙述得以改善.

参考文献

- 1 Hopcroft J E, Motwani R, Ullman J D. Introduction to Automata Theory, Languages, and Computation. 3rd ed. Beijing: China Machine Press, 2007
- 2 Cutland N. Computability: An Introduction to Recursive Function Theory. Cambridge: Cambridge University Press, 1980
- 3 Dong Y M. Recursive functions of context free languages (I) — the definitions of CFPRF and CFRF. Sci China Ser F-Inf Sci, 2002, 45: 25–39
- 4 Dong Y M. Recursive functions of context free languages (II) — validity of CFPRF and CFRF definitions. Sci China Ser F-Inf Sci, 2002, 45: 81–102
- 5 董韫美. CFL 句子计数和分层词典序枚举. 中国科学 E 辑, 2006, 36: 1375–1413
- 6 Dong Y M. Linear algorithm for lexicographic enumeration of CFG parse trees. Sci China Ser F-Inf Sci, 2009, 52: 1177–1202
- 7 Péter R. Recursive Functions. New York: Academic Press, 1967

Multi-type recursive functions

DONG YunMei

Abstract In this paper, we intend to establish the theory of multi-type recursive functions (MTRF). The aim is to present a theoretical prototype of functional language to be implemented on computer. MTRF functions

are computable, whose domain and range include: natural numbers, Boolean values, words on finite alphabet, context free languages, and parse trees of context free grammars. It is indicated that the natural number recursive functions and the MTRF functions are equivalent. Finally, an MTRF function `eval` is constructed, which is an evaluator to compute arbitrary MTRF functions, and can be implemented on the computer.

Keywords multi-type recursive function (MTRF), multi-type primitive recursive function (MTPRF), function evaluator

MSC(2010) 03D20, 68Q45

doi: 10.1360/N012015-00053