方程式语言及其实现

陆汝占 韩启龙 林凯 奚宏伟 孙永强

(上海交通大学计算机系, 200030)

EQUATIONAL PROGRAMMING LANGUAGE AND ITS IMPLEMENTATION

Lu Ruzhan, Han Qilong, Lin Kai, Xi Hongwei and Sun Yongqiang

(Shanghai Jiaotong University Computer Department, 200030)

ABSTRACT

Equational Programming Language (EP) is a novel intelligence language. This paper describes our EP system based on equational logic. Its execution mechanism is pattern matching. The paper focuses the discussion on the improvement to bottom-up tree pattern matching. Our system has quick response time and strong descriptive power.

摘 要

方程式语言是一种新型智能语言. 本文介绍该语言的一个实现系统. 该系统是以方程逻辑为语义基础, 模式匹配为执行机制. 本文讨论了自底向上模式匹配方法及其改进, 整个系统具有时间响应快、描述能力强的特点.

§1. 引 言

1990 年 5 月 23 日收到, 1990 年 7 月 28 日定稿. 本文受国家自然科学基金、高科技基金资助. 陆汝占, 上海交通大学教授, 目前主要从事定理证明、方程式语言、计算理论方面的研究工作. 韩启龙, 1988 年毕业于上海交通大学, 现任该校讲师, 主要从事方程式逻辑语言方面的研究工作. 林凯, 1987 年在上海交通大学获硕士学位, 现任该校讲师, 主要从事定理证明、重写技术及理论方面的研究工作. 奚宏伟, 25 岁, 1988 年在南京大学获硕士学位, 现任上海交通大学讲师, 主要从事计算复杂性、方程式语言、数理逻辑方面的研究工作. 孙永强, 现为上海交通大学计算机系教授, 博士导师, 目前主要从前计算机理论、新型语言方面的研究。

1.1两类语言 FP 、 LP

目前新型智能语言可概括为函数型与逻辑型两大类．两者异同列于表 1(并以求解最大公约数为例).

表 1

		FP	LP
相异	理论基础	λ– 演算，递归方程理论	一阶谓词演算
	数据、函数定义	可含高阶抽象算子、无穷流 显式表示	谓词 隐式表示，便于描述
	输入、输出关系	确定的显式关系 待求解为显式	不确定的隐式关系 待求解为谓词隐式
	求解机制	匹配与归约 有较高的归约化简效率	合一与归结 有较强的推理能力
	求解结果	函数表达式最终归约结果 若归约终止，则范式唯一	归结法求出隐式解 可能存在多个解
相同		描述性语言；无副作用； 尽管执行机制不同，本质上均可看成归约演绎.	
例		(DEFINE GCD(X Y) (COND((ZEROP Y)X) (T GCD (Y MOD(X Y))))) (GCD(105, 60)) (待求解) 15 (归约结果)	gcd(x, 0, x):–! gcd(x, y, Z):–Ris x mod y, gcd(y, R, Z) ?–gcd(105, 60, u) (待求解 u=?) u=15 (归结结果)

1.2 发展趋向

近年来，两类语言在保持核心主体前提下，相互影响、补充、混合、嵌入，力求自身的完善 [3,4], 但往往很难保持一个统一的语义理论．因此，产生的新倾向是寻找一种同时具有强计算能力与强推理能力的、本身又有统一语义理论基础的新型语言．由 C. Hoffman 与 M. J. O'Donnell 在 1981 年研制的方程式语言及其实现系统正是这类语言的基础．1985 年新版本——左顺序式方程解释系统 [5] 在执行效率上有显著提高.

本课题组受国家自然科学基金及高科技基金资助，在国内首次尝试研制方程式语言解释器．本文介绍实现情况.

§ 2. 方程式语言 EP

程序的主要形式是方程．方程可用来定义函数和规则，并有递归方程的显式表示能力和带等词逻辑的隐式表示能力．程序易验证、排错，并具并行计算的潜力.

该语言语义基于方程逻辑，用统一的等式逻辑理论来描述．执行机制是模式匹配与归约，归约策略丰富，只要程序存在解，最终必求得，多解时可产生所有解．采用惰性计算，可减少冗余计算．随着现代重写技术的完善，有望获得较高的执行效率.

例 3(EP 型程序)(表 2)

表 2

Symbols

```
  gcd: 2;
  mod: 2;
  if: 3;
  less: 2;
  sub: 2;
for all a, b;
  gcd[a, 0]=a;
  mod[a, b]=if [less [a, b], a, mod [sub[a, b], b]];
  gcd[a, b]=gcd[b, mod[a, b]];
```

(注：其中 if, less, sub 是内部函数)

求解过程：欲求 gcd[105, 60], 则须将程序输入至预处理器，产生解释程序，再启动该解释程序，由用户输入待求解项 gcd[105, 60], 则系统显示归约结果 15, 然后退出.

2.1 语法

方程式语言的 BNF 语法列于表 3.

表 3

```
<program>::=Symbols<symbol description list>
            For all <variable list>:begin<equation list> end
<symbol description list>::=<symbol description>; ·····;
                                <symbol description>;
<symbol description>::=<symbol>:<arity> |include<symbol class>
<symbol class>::=atomicsymbols|integernumber|truthvalue|
                 listsymbols
<symbol list>::=<symbol>, ·····, <symbol>
<symbol>::=<string of nonblank alphanumeric characters,
            starting with alphabetic>
<arity>::=<number>
<variable list>::=<variable>, ·····, <variable>
<variable>::=<string of nonblank alphanumeric characters,
             starting with alphabetic>
<equation list>::=<equation>; ·····; <equation>;
<equation>::=<term>=<term> |
             <term >=<term>where<qualification>endwhere|
             include<equation class list>
<qualification>::=<qualification item list>
<qualification item list>::=<qualification item>, ·····,
                                <qualification item>
<qualification item>::=<variable>is<qualified term> |
                        <variable list>are<qualified term>
<qualified term>::=in<symbol class> |
                   <term> |
                   <qualified term>where<qualification>endwhere|
                   either<qualified term list>endor
<qualified term list>::=<qualified term>or·····or
                        <qualified term>
<equation class list>::=<equation class> |
                        <equation class><equation class list>
```

<equation class>::=addint|subint|mulint|divint|modint|
lessint|equint

Symbol 表示程序中预定义的函数符及其参数维数，维数为零的函数符是常数. Symbol class 表示预定义符所属类别.

关于函数表达形式，不管用户采取何种形式，系统内均采用显式一致的内部形式 (Curry 形式). 例如 f(a, b, c) 的 Curry 形式即为 Apply(Apply(Apply(f, a), b), c). 解释器的输入输出都采用此形式，优点是该形式以串为基础的语法，可简化语义处理，便于移植. 用户采用前缀、中缀、后缀形式，均由系统自动转换之.

2.2 语义

2.2.1 方程的指称语义

给定有限函数符号集 Σ 及其上的项集 $\Sigma_{\#}$ 与维数函数 ρ.

定义 1: Σ 的一个解释是 I=(D, ψ), 其中 D 为论域, Ψ 是 Σ 的赋值, $\Psi:\Sigma\to\bigcup_{n\in N}(D^n\to D)$, 对任一 $s\in\Sigma$, 则 $\Psi s\in D^{\rho s}\to D$. 很自然地，可将对函数符的赋值扩展到对函数填式 (复合项) 的赋值 $\psi':\Sigma_{\#}\to D$.

定义 2: 由上述 I 导出的 $\Sigma_{\#}$ 的赋值 $\Psi':\Sigma_{\#}\to D$, 对于任一 $s\in\Sigma_{\#}$, 且 s 呈 $f(A_1,A_2,\cdots A_n)$ 时, $\Psi' s=(\Psi f)(\Psi' A_1,\cdots\Psi' A_n)$. 特别当 $\rho A_i=0$ 时, $\Psi' A_i=\Psi A_i$. 不妨将 Ψ' 仍记为 Ψ.

定义 3: 若 $\Psi A=\Psi B$, 则称解释 I=(D, Ψ) 使 A=B 满足，记为 I⊨A=B. 设 J 是 Σ 的一个解释集，则 J⊨A=B iff $\forall_I\in J$, I⊨A=B. 设 E 为方程集，则 I⊨E iff $\forall_{A=B\in E}$, I⊨A=B, 于是, J⊨E iff $\forall_{I\in J}$, I⊨E. E⊨A=B iff {I|I⊨E}⊨A=B. 此记为 Ψ. 于是 I=(D, Ψ), 若 $\Psi A=\Psi B$, 则称 I 使 A=B 满足，记为 I⊨A=B.

2.2.2 方程的操作语义

方程逻辑的推理规则： (i) 自反 A=A; (ii) 对称 $\frac{A=B}{B=A}$; (iii) 传递 $\frac{A=B, B=C}{A=C}$; (iv) 替代 $\frac{A=B}{C(X\leftarrow A)=C(X\leftarrow B)}$.

定义 5: 设 E 为方程集，由 E 导出方程 A=B 的一个证明 D 是指存在有限序列 $\{C_i=D_i(1\le i\le n)\}$, 其中 C_iD_i 满足： (i) $A=C_n$, $B=D_n$; (ii) 或者 $C_i=D_i\in E$; (iii) 或者 $C_i=D_i$ 由 $C_j=D_j$ $C_k=D_k(1\le j,k<i\le n)$ 由上述推理规则得到. 证明 D 也可记为 E⊢A=B.

2.2.3 语义的一致性

定理 1: E⊨A=B iff E⊢A=B

本定理即是方程逻辑的正确性 (逆向) 与完备性 (正向). 定理证明：前半部由定义 3 、 5 及规则显见；后半部可由 A 、 B 的构造归纳证之，也可由通常的代数规范语言的完备性证明得出. 详略.

2.3 方程的语义限制

为使方程解释程序所用的归约策略正确适用，须从语义上对方程加上一些限制：

1. 方程的左端，变量不许重复. 如 if(x, y, y)=y 是禁止的，因解释器对左端变量不作相等性检查.

2. 在方程右端出现的每个变量必须亦在左端出现.

3. 两个不同的左端表达式不可与同一表达式相匹配，如方程组 g(0, x)=0; g(x, 1)=1

是禁止的，因为与两方程都匹配的项 g(0, 1), 归约结果会不同.

4. 当两个左端表达式（未必不同）与同一表达式的两个不同部分匹配时，这两部分不可重迭，如方程组 first(pred(x))=predfunc; pred(succ(x))=x 是禁止的，因二者在 first(pred
(succ(0))) 这部分是重迭的，此时，归约可得出不同的结果，取决于所使用的匹配方程.

满足上述限制的方程组称为“正则”方程组，它能保证 C-R 性质和范式是唯一的（仅为一充分条件），并保证最外层的归约策略能求出所有的范式. 当方程组不满足限制 3 与 4 且不含相交的子模式时，若有解，解释器可输出一解.

§ 3. 实现技术

方程式语言系统已在 Micro-VAXII 机上用 C 语言实现.

3.1 总体

实现系统的总体结构以方便用户和便于修改扩充为原则，采用层次结构，核心是预处理器和解释器（图 4).

预处理器的功能是将用户定义的方程程序中的函数表达式经语法分析后转换成统一的内部语法 Curry 形式，再进行语义预处理，包括语义限制检查，子模式集和匹配集的产生，各函数匹配表的生成等.

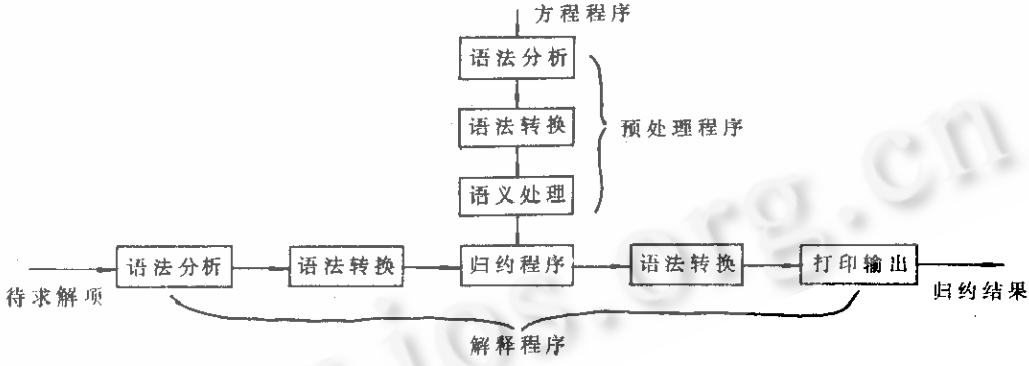

图 4

解释器的功能是将待求解项经语法分析后转换成 Curry 形式，再根据各函数的匹配表进行匹配，按归约策略选取 redex 进行归约化简，重复这二步直至不能归约为止，即产生范式，最后将范式的 Curry 形式还原成用户的表示式输出.

外层是交互式用户界面，便于用户编写、修改和运行程序. 各层次内部采用模块结构，相互关系清晰、易维护.

3.2 数据表示

系统中所有的项，包括待匹配项、方程的左端和右端表达式都被包含在 DAG 中，节点包含符号名、符号类型与子节点表，用杂凑法可快速找到该节点.

如有待求解项 X(e, (i(A), A)) 和方程组 X(i(A), A)=e; X(e, A)=A, 它们在 DAG 中的存贮如图 5.

3.3 树模式匹配

树模式匹配技术通常是归约的关键，有各种实现方法，旨在寻求高效低开销的算法.

树模式匹配问题是指给定有限模式集 p 与有限字母集 Σ 上的项集中的一个待匹配项 t, 求 $\langle n, i\rangle$ 的集合，其中 n 是 t 中的节点且 p_i 匹配 n.

例 4: 给定模式集 p={p_1, p_2} 与待匹配树 t, 其中 p_1=A(A(*, *), b); p_2=A(b, *), A 是函数符， b, c 是常量符， * 是变量符，其匹配问题解答如图 6 所示.

树模式匹配方法主要有三类：自底向上、自顶向下、左顺序的串匹配.

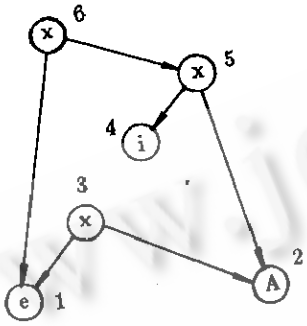

图 5

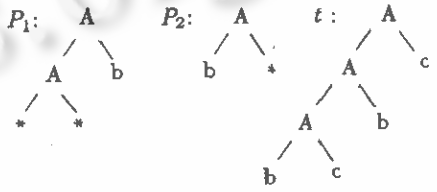

图 6

匹配集:

M1 = {*}
M2 = {b, *}
M3 = {A(*, *), *}
M4 = {A(b, *), A(*, *), *}
M5 = {A(A(*, *), b), A(*, *), *}

A 的匹配表

左面	右面 1	2	3	4	5
	1	3	3	3	3
	2	4	4	4	4
	3	3	5	3	3
	4	3	5	3	3
	5	3	5	3	3

b 的匹配表: b:2
c 的匹配表: c:1
结果

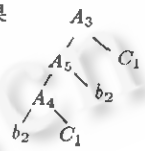

图 7

自底向上的匹配是从待匹配树的底部开始寻找匹配的模式，典型步骤是：输入模式集 P⟶ 产生 P 的模式林 PF⟶ 产生匹配集 ⟶ 生成各函数的匹配表 ⟶ 根据各函数的匹配表求匹配输入树各节点的模式集合. 例 4 的自底向上匹配过程如图 7 所示.

该法的时间复杂性是 O(subsize+match), 空间复杂性是 O(subsize+patsize$^{\text{rank}}$×sym). 其中 subsize 是待匹配树的节点数， match 是找到的匹配节点数， patsize 是输入模式的模式林节点数， rank 是 Σ 中各函数符号的最大维数值， sym 是 Σ 中的符号个数.

自顶向下的匹配是从树的根部起将匹配问题转换成串匹配问题，然后用自动机来解. 典型步骤：输入模式集 P⟶ 产生模式的路径串树 ⟶ 生成匹配自动机 ⟶ 根据自动机求匹配输入树各节点的模式集合. 该法的时间复杂性是 O(subsize× patno), 空间复杂性是 O(subsize×patno+patsize). 其中 patno 是输入模式的个数.

左顺序的串匹配法是将树模式略去变量压偏成预排序串，用 Aho-Corasick 算法能产生识别这些串的有限自动机，它具有线性的时间复杂性和较少的空间开销，但该法要求用户定义的方程组不仅是正则，而且是左顺序的.

综上所述，自底向上法从速度上讲是最快的，当然空间开销过大，匹配表所占的空间有可能使任何机器不能容忍，必须设法解决；自顶向下法速度慢；左顺序的串匹配法时空效率较好，但适用范围有限制.

如何减少自底向上法空间开销，可用以下策略：

1. 由匹配集的生成算法[2]，可知匹配集的个数随输入模式集的个数成指数增长，这是空间开销大的原因之一.

设 t 是有限字母集 Σ 上的树，能匹配 t 的模式集为 $M_i(t)=\{m_{i,1}, m_{i,2}\cdots m_{i,n_i}\}$ 且可设法改造（证略），使得其中诸元素依偏序 $\geq_\theta$ 排列，$m_{i,k} \geq m_{i,k+1}$ 即 $\exists_{\theta_k}(m_{i,k} = m_{i,k+1}\theta_k)(\theta_k$ 为一代换）. 对每一模式 p 均可找到一 $M_i(t)$，使得 $M_{i,1}$=p，则两个模式 p 与 p' 间的关系，也即是 $M_i(p=m_{i,1})$ 与 $M_j(p'=M_{j,1})$ 的关系可分成四类：

(1) 独立　不存在 $M_i(t)$ 使得 $p\in M_i(t)$ $p'\in M_j(t)$ 成立.

(2) 相交　存在 $M_i(t_1)$, $M_j(t_2)$ 且 $M_{i,1}$=p, $M_{j,1}$=p' 使得 $\exists_{\theta_1\theta_2}(M_{i,1}\theta_1=M_{j,1}\theta_2)$ 成立. (θ_i 是代换）.

(3) 包含　即通常集合意义下相应的 $M_i(t)$ 间包含关系.

(4) 真包含　类 (3).

当输入模式集中不包含相交的子模式（可称之为简单模式集）时，其生成的匹配集总数便大大减少，可证，此时匹配集个数等于输入模式的子树数. 因此通过将匹配问题中的输入模式集限制为简单模式集可以解决匹配集个数的指数增长，同时，对输入的非简单模式集可以通过转换成简单模式集来解决.

2. 匹配表的尺寸与 $|R|^{arity(a)}$ 成正比 (|R| 是匹配集中的元素数)，虽然典型函数符的维数是 2–3 维，但仍可使机器不能忍受. 当将输入模式集限制为简单模式集时，因模式间的包含关系，其匹配表就存在大量的相同元素，可采用表与栈相结合方法来进行压缩. 例 4 中 A 的匹配表，常规要 25 个单元存贮，现仅 5 个. （表 8）根据典型实测，当匹配表的个数 |R| 是 25 时，一个三维函数匹配表一般需单元数是 25^3 = 15,000 个，而现为 <50 个单元.

表 8

L	R	Ta
*	*	3
2	*	4
3	2	5
4	2	5
5	2	5

* 表示 {1, 2, 3, 4, 5}

由上所述，建立新的自底向上树模式匹配算法如表 9，算法 1~5 建立 PF 中各模式的包含关系，6~10 生成各匹配表. PF 的生成与根据各函数符的匹配表进行匹配的算法按常规.

表 9

输入：简单模式集的子树集 PF.

输出：用于自底向上树模式匹配的匹配表.

算法：

1. 把 PF 中的树按高度升序排列.
2. 初始化 $\overline{Gs}$[注] 中各结点且去掉边.
3. For each p=a($p_1,\cdots,p_n$) n> 0 且高度是 h，按高度的升序 do.
4. For each p' in PF 且高度 ≤ h do（按高度的降序）
5. if p'=* or p'=a($p'_1,\cdots,p'_m$)
where for $1 \leq i \leq m$ $p_i \rightarrow p'_i$ 在 $\overline{Gs}$ 中

then Add $p \rightarrow p'$ to $\overline{Gs}$(即加入有向边) go to 步骤 3.

6. 初始化所有匹配表 Ta 为 *

7. For each Σ 中元素 do

8. For each 模式 $p=a(p_1,\cdots,p_n)$ 按高度的升序 do

9. For each m–taple $(p'_1,\cdots,p'_m)$ 且 for $1 \leq j \leq m$ $p'_j \rightarrow p_j$ do

10. if Ta $[p'_1,\cdots,p'_m] \neq p$

then Add $Ta[p'_1,\cdots,p'_m]=p$

(" $\longrightarrow$ " 表示有向边)

[注] $\overline{Gs}$ 是简单模式集中各模式的包含关系图，结点表示各模式，根据简单模式集的定义，可以得知这一 $\overline{Gs}$ 图必存在.

3.4 归约策略

归约策略包括 redex 的选取和归约化简.

redex 的选取涉及不同的归约路径、时空开销和结果. 本系统采用了最外层的 redex 的选取策略，旨在忠实于方程的逻辑语义，进行惰性计值以避无限的冗余计值.

§ 4. 结果与讨论

方程式语言系统在 Micro–VAXII 机上运行典型实例的时间分析如表 10 所示. 可以看出，该系统的响应时间是较快的，从适应的方程范围看，对于满足 4 条限制的方程，可得出范式的唯一解；对不满足方程限制 3 与 4 的，当其有解时 (即存在一解或多解的情况下), 可以产生其中的一解，进一步，亦可得到所有的解.

表 10

	运行时间 (秒)
皇后问题 (n=5)	8
斐布纳契数	2
阶乘 (n=10)	1
最大公约数	1

进一步研究将对方程进行分类分析，通过开发方程程序固有的并行性，来实现并行处理.

参考文献

[1] C. Hoffman, O'Donnell, M:, " Programming with Equations ", ACM Trans. on Programming Language and Systems (1982), 83–112.

[2] David R. Chase, " An Improvement to Bottom-up Tree Pattern Matching ", Conf. Record of ACM SIGACT/SIGPLAN Symposium on POPL (1987), 168–177.

[3] J. Darlington, A. J. Field, H. Pull, " The Unification of Functional and Logic Language ", In: Logic Programming, Functions, Relations and Equations (1986).

[4] J. A. Goguen, " EQLOG: Equality, Types and Generic Modules for Logic Programming ", In: Logic Programming, Functions, Relations and Equation (1986).

[5] O'Donnell, M. J., Equational Logic as a Programming Language, MIT Press Cambridge Mass (1985).