第4卷第4期
1993年8月

软 件 学 报
JOURNAL OF SOFTWARE

Vol. 4, No. 4
Aug. 1993

一种扩展图重写模型下函数语言惰性模式匹配的实现方法

周光明　　王鼎兴

（清华大学计算机系，北京 100084）

A LAZY PATTERN MATCHING IMPLEMENTING METHOD OF FUNCTIONAL LANGUAGE BASED ON EXTENDED GRAPH REWRITING MODEL

Zhou Guangming and Wang Dingxing

(*Department of Computer Science, Tsinghua University, Beijing* 100084)

Abstract　A lazy pattern matching implementation method based on extended graph rewriting model is proposed, including a lazy pattern matching compiling algorithm, a sometime—matching argument eliminating and pattern flattening algorithm. The first algorithm has solved the problem of efficient implementing lazy pattern matching which remained unsolved in [6]. The later algorithm is simpler and more efficient than that proposed in [7].

摘要　本文给出惰性模式匹配基于扩展图重写模型的实现方法，包括模式匹配编译算法、参量一致化和模式平坦化算法. 前一算法较好地实现了[6]中尚未解决的惰性模式匹配问题；而后一算法较[7]中提出的算法简单、高效.

§1. 引　言

函数语言中引入模式匹配，使得程序形式简洁、可读性好且有利于程序的正确性证明[2]，所以已被函数语言广泛采用，如ML、HOPE、Miranda等都引入了这一机制.

同时越来越多函数语言采用了惰性计算方式(Lazy Evaluation)，其好处是保证程序的停机性，支持无限数据结构. 在惰性计算方式下模式匹配的实现需要解决这样一个问题，即函数参量在模式匹配过程中，只驱动刚好能使匹配完成的计算部分. 文献[2,3]提出的模式匹配的编译算法中没有解决这一问题. 文献[6]只从理论上解决了惰性模式匹配算法的存在性和可计算性问题，也未给出实现算法.

扩展图重写模型是笔者参加研制的并行图归约系统中支持多种说明性语言的计算模型[5]，CIL(Compiler Intermediate Language)是该模型的描述语言. 文献[7]详细讨论了

本文1990年12月7日收到，1991年5月3日定稿. 作者**周光明**，1991年博士毕业于清华大学，主要研究领域为计算机系统结构，并行处理，编译技术. **王鼎兴**，教授，主要研究领域为并行、分布处理与智能计算机系统.

重写系统对函数语言实现的支持,并提出了函数定义加标记算法,但该算法由三个部分组成,其中函数参量一致化和模式平坦化的算法较为繁琐,编译效率不高.

本文在Laville提出的概念MEP的基础上,给出一个惰性模式匹配编译算法,较好地解决了并行惰性模式匹配的实现问题.随之提出的参量一致化和模式平坦化的一步转换算法,较[7]中的算法简单高效.

§2. 惰性模式匹配和扩展图重写模型

2.1 EML语言和惰性模式匹配

ML语言是爱丁堡大学定理证明系统LCF的元语言.SML是在ML基础上吸取其他函数语言优点制定的标准版本.EML语言是笔者设计的为并行图归约系统使用的一种SML的变种语言[2].EML程序由一组定义和一个表达式组成,其中函数定义可使用模式匹配,形式如下:

$$\begin{aligned} &\text{fun } p_1 = e_1 \\ &\mid\ \ p_2 = e_2 \\ &\cdots\cdots \\ &\mid\ \ p_n = e_n \end{aligned} \tag{1}$$

其中$p_i(i=1,\cdots,n)$为模式,即由函数名、数据构造子和变量组成的表达式,也称为项.其中函数名f必定在且只在每一等式最外层位置.计算项t:$f[e_1\ e_2\cdots e_n]$时,找出函数定义左部与t匹配的等式,计算相应的右部表达式.如果不止一个等式左部与t匹配,ML类语言选用第一个模式.为后文阐述方便,下面给出一些定义和记号.

下文中$\sum$为所有构造子和表示未定义的符号Ω组成的集合.

定义 1:由$\sum$中的元素构造的项称为部分项.

定义 2:项上的偏序$\leqslant$定义如下:

1. 对于项N:$\Omega \leqslant N$
2. $F[N_1\cdots N_n] \leqslant F[M_1\cdots M_n]$,当且仅当$N_i \leqslant M_i(i=1,\cdots,n)$

在下文中我们将使用以下记号:

- $M \uparrow N$ 表示M和N有公共上界(也称M和N相容(Compatible));
- $M \# N$ 表示M和N无公共上界(不相容);
- $M \vee N$ 和 $M \wedge N$ 分别表示M和N的最小上界和最大下界.

定义 3:用树表示一个项,树中任一子树表示该项的一个子项.整数表$O=[o_1,\cdots,o_s]$称为项T的一个位置(Occurrence),用来表示树中特定的子项.例如,[2,3]表示第二个子项的第三个子项.O(M)表示项M中的所有位置,O(M)表示M中所有非Ω位置,O_Ω(M)表示M中所有为Ω的位置.M(o)表示项M在位置o处的符号.

定义 4:设$O_1=[a_1,a_2,\cdots,a_k]$,$O_2=[b_1,\cdots,b_s]$,如果 ①$k < s$ 或 ②$k=s$ 且 $a_1=b_1,\cdots,a_i=b_i,a_{i+1}<b_{i+1}$,则称$O_1$小于$O_2$,记作$O_1 < O_2$.

下面对(1)式定义的函数模式匹配下一定义.tt和ff表示逻辑真与假.

定义 5:对$i\in\{1,\cdots,n\}$,$match_i(M)=tt$当且仅当①和②成立:① $p_i \leqslant M$;② 对于$j<i$,$p_j \# M$. $match_n(M)=tt$当且仅当存在$i\in\{1,\cdots,n\}$,$match_i(M)=tt$.显然,$match_i(M)$

$=tt$ 表示 M 已经足够确定，能与 p_i 匹配，且任一优先级高的模式都匹配不成功. $match_{\Pi}(M)=tt$ 表示 M 已经足够确定，可以完成模式匹配的计算. 可以证明：

引理 1：如果规定 $ff<tt$，则 $match_i$ 和 $match_{\Pi}$ 都是递增函数.

2.2 扩展图重写模型和 CIL 语言

扩展图重写模型是我们设计的一种支持多种说明性语言的计算模型. CIL 是该模型的描述语言，是体系结构与高级说明性语言的一个界面. 我们构造的二个编译系统分别把函数和逻辑程序编译成 CIL 代码.

2.2.1 函数语言的图归约计算方法

组合子图归约方法是从 λ 表达式的图归约计算方法演变来的. 函数语言程序可以用 λ 表达式来表示. λ 表达式的计算是通过不断应用 α、β、γ 三条转换规则对表达式进行变换，直到不能进一步变换为止，这时的表达式形式称为正规式. 用 β 规则对 λ 表达式进行转换的过程称为归约. 对于一个表达式中多个可归约表达式 redex(REDucible EXpression)，选择其中哪一个进行归约，即计算序有两种：选择最左 redex 归约的正规序归约(Normal Order)和选择最右 redex 归约的作用序归约(Applicative Order). Church—Rosser 定理已证明如下结论[2]：

- 若对一表达式采用不同序归约都能得到正规式，它们必定相同；
- 若表达式能归约为正规式，一定可用正规序归约得到该正规式.

采用正规序语义的函数语言有良好的停机性、能避免无用计算以及支持无穷数据结构等，所以较基于作用式的语言有更强的能力.

归约的执行的另一个问题是如何有效地进行变量替代，它包括 λ 表达式的表示方法和替代形式两个问题. 图归约计算中表达式是一个图，通过图的共享有效地避免参量的重复计算和无用的参量计算，无需管理大量的表格. 这一方法的主要问题是函数体的拷贝. 通过引入组合子的概念可以解决这一问题[8]. 组合子实际就是一个不含自由变量的 λ 表达式，其一般形式为：

$$C: (\lambda x1.(\lambda x2\cdots\cdots(\lambda xn.E))).$$

根据正规序归约的原则，在 C 的所有参量到齐以前，C 是不会有任何改变的，所以 C 的归约规则可以写成：

$$C\ E1\ E2\ \cdots\cdots En \Rightarrow E[E1/x1, E2/x2, \cdots\cdots, En/xn].$$

在早期的理论研究中，Curry 证明了可计算函数都可以用两个最基本的函数组合而成：S 和 K. SK 组合子归约易于实现且效率较高. 但其计算粒度比较小，不适于并行计算[5]，为了增大计算粒度，我们采用了基于用户定义函数进行无冗余计算的超组合子归约方法[5].

2.2.2 描述语言 CIL 及其并行执行

扩展图重写计算模型是图归约计算模型的扩充，模型中计算对象表示为有向图的形式：每个数据对象为一张有向图，包括结点、有向弧和结点的标记. 有向图的结构(弧)用于描述表达式之间的关系，而结点的标记用于记录说明性成份，如标量常数，数据结构，原始函数和组合子. 每个结点可以带有如下控制标记：

'*' 表示结点处于活动状态，可以进行重写尝试；

'#n' 表示结点处于挂起状态，等待 n 个参量子图计算结果；

‘’空标记表示结点目前无计算必要.

每条弧可带如下控制标记:

‘^’表示结点正在等待此弧所指向的结点的计算结果;

‘’空标记表示结点没有等待此弧所指结点的计算结果.

CIL 的重写规则描述模型中图变换操作,其形式为:

pattern :: condition → NodeDef,Activate,Redirect,Update

→左部是扩展模式,用来规定选择规则的条件.其中 Pattern 是简单模式,是一个不带标记,以可重写结点为根的图,其子图只能是变量结点或结构结点,子图的参量只能是变量;Condition 描述简单限制条件.右部是图变换操作内容:NodeDef 用于新结点的构造;Activate 用于标记计算子图中要标记为‘*’或‘#’的结点;Redirect 描述图结构的变换;Update 描述对某结点内容的更新操作,如结点标记的修改等.

CIL 程序由重写规则和一条初始规则组成.程序的执行从初始规则开始,不断找出带‘*’的可重写图用规则进行图变换.当一个结点计算完毕,沿所有指向该结点的带‘^’弧发出相应信息.当带‘#n’的结点收到某‘^’弧发回子图计算完毕信息,则 n 减 1(‘#0’等价于‘*’).

§3. 模式匹配树的生成算法

3.1 惰性模式匹配算法的存在性和可计算性

A. Laville 在[6]中提出了最小扩展模式的概念:

定义 6:项 T 为最小扩展模式(mep),如果 T 满足 $match_{\Pi}(T)=tt$ 且对于任一项 $T'\leqslant T$,$match_{\Pi}(T')=ff$.

下面我们用 MEP 表示所有的最小扩展模式构成的集合.

定义 7:设 match 为 $match_i$ 或 $match_{\Pi}$,M 是部分项,$u\in O_n(M)$,对于 $N\geqslant M$,若 $match(N)=tt$,那么 $N(u)\neq\Omega$,则称 u 为 M 的索引(index).

match 对 M 是顺序的当且仅当以下两点成立时则 M 中有 match 的索引:1. $match(M)=ff$;2. 存在 $N\geqslant M$ $match(N)=tt$

定义 8:把任一部分项与 Π 匹配的确定的算法称为模式匹配算法.若一模式匹配算法不驱动无用计算,则称为惰性模式匹配算法(LPM).

A. Laville 进一步证明了下述结论.

定理 1:·若 $\sum$ 是有限集,则 MEP 是有限的且可从模式表 Π 计算出来;

·对于一个用模式定义的函数,存在相应的惰性模式匹配算法,当且仅当 $match_{\Pi}$ 对任一部分项都是顺序的;

·若 MEP 的两个元素是相容的,则不存在惰性模式匹配算法;

·若 MEP 任两元素都不相容,则存在惰性模式匹配算法;若存在惰性模式匹配算法,则可以机械地从初始模式建立.

如果 MEP 中任意两项都不相容,$match_{\pi}$ 的顺序性很容易检查,只要检查它是否对 MEP 中任一项都是顺序的即可.如果得到肯定的结果,则存在 LPM.这两个目标都可以通过建立一个匹配树来完成.匹配树的结构如下:树结点含有一个项和一个索引 I,该结点引出的每条弧上含有一个构造子 C,表示该弧所指的子结点上的项在 I 处为 C.叶子结

点上都是MEP,表示一种成功的匹配.树根的项为Ω.

从匹配树很容易构造LPM:从根开始驱动索引计算,根据计算结果选择子结点再往下直到叶子结点.如找不到对应子结点,则匹配失败.

3.2 一个实际的LPM构造方法

在[6]中,MEP的计算只作了如下的描述:MEP中任一元素的位置都在集合$\bigcup O(p_i)$中,且$\sum$是有限的.即MEP的构造可通过每一位置用$\sum$中所有的符号来试.但当$\sum$和位置增长时,要计算的部分项将指数增长,所以这对编译来说是不能接受的.改进的方法是构造MEP时充分利用初始模式中所含的信息,并且利用函数语言类型检查的结果,如对于模式p_i中任一位置u上的符号$p_i(u)$必然是某类型的构造子,而不必在$\sum$中寻找.下面给出的MEP构造算法就是这样的,算法如下:

算法 3.1:MEP的生成

1. 对于(1)中定义的函数,p_1是mep.

2. 对于$p_i(i=2\cdots n)$,若对于$j<i$有$p_i \# p_j$,则p_i是mep;否则设$q_1,q_2,\cdots,q_k$是与p_i有二义性且排在p_i前面的初始模式,则集合M_k即为与p_i匹配的所有mep:

① $M_0=\{p_i\}$, $j=1$.

② $M=\{t \mid t\in M_{j-1}$且$t \# q_i\}$.

③ 对于$t'\in M_{j-1}$且$t' \uparrow q_j$,显然$O=O(q_j)\cap O_\Omega(t')$非空,对于O中每一位置u,设$q_j$在u上符号为$C_1$,类型相同的构造子还有$C_2,\cdots,C_s$,构造集合:$\{t \mid t$为$t'$中位置u上的符号用$C_n$替代所得的项,$n=2,\cdots,s\}$,MM为这些集合的并.

④ $M_j=M\cup MM$,如果$j=k$,则结束;否则$j=j+1$并转②.

定理 2:$t\in M_k$,则t是与p_i匹配的mep.

证明:设$t\in M_k$,则由构造过程可知$match_i(t)=tt$,设$t'<t$,则

①对于任一$j\leqslant i$,$t' \wedge p_j<p_j$,即$match_\Pi(t')=ff$,或

②存在$j\leqslant i$,$t'\geqslant p_j$,

如果$j<i$,则$match_i(t')=ff$.设$match_\Pi(t')=tt$,由引理1知$match_i(t')=tt$,$match_\Pi(t')=tt$不成立.

如果$t'\geqslant p_i$且$t' \wedge p_i<p_i(j<i)$,不妨设t'与t唯一只有在位置u上符号不同.由M_k的构造过程知,若$p_i[u]\neq\Omega$,则$match_i(t')=ff$;若$p_i[u]=\Omega$,则$t[u]$是在步骤3中变为非Ω的,存在$p_j \uparrow t'(j<i)$.所以$match_\Pi(t')=ff$.所以t是mep且$match_i(t)=tt$. ■

由于构造M_k的过程穷尽了数据类型的所有构造子,所以M_k中包含了所有与p_i匹配的mep.MEP构造出来以后,可由算法3.2生成匹配树.

算法 3.2:匹配树的生成

建立树根R:$f[x_1\cdots x_n]$;当前结点P为树根;

$S=\{t \mid t$为mep且$t \uparrow R\}$;

create_pmt(S,P):

1. 若存在$t\in S$的某非Ω位置没在匹配树的生成中使用,则转2,否则本(子)树已构造完毕,返回;

2. 找S中所有的mep都非Ω的最小位置u并存入结点P,并转3;如找不到,则不存

在LPM,返回;

3.设项R的在位置u的符号可为构造子$C_1,\cdots,C_k$(根据类型),生成R各个子结点$N_1,\cdots,N_k$,对应各弧记下$C_1,\cdots,C_k$;结点N_j中的项为$R[u\leftarrow C_j]$,且如C_j有参量则都为Ω;$S_j=\{t|t\in S$ 且 $t(u)=C_j\}$;

4.对P的各个子树$N_j(j=1,\cdots,k)$递归调用 create_pmt(S_j,N_j).

§4. 目标代码的生成

4.1 加标记算法及其实现

匹配树转换成重写规则,加上并行控制标记,就得到CIL代码了.Kennaway在[7]中给出了一个加标记算法,这个算法要求:(1)重写规则必须是无二义性的;(2)重写规则中的模式必须是简单模式;(3)函数某重写规则的左部为项T,T的一个位置l,则该函数各规则T(I)存在且同为变元或构造子.

通过匹配树的建立,可以保证第一条要求满足.Kennaway又提出了两个转换算法,使不满足2和3的重写规则转换成满足要求的重写规则.但这两个转换算法采用等价类划分的方法,算法复杂且效率低.本节将提出一个算法,该算法直接从匹配树生成满足2和3的重写规则.该算法较[7]中的算法要简洁,编译速度快,代码效率高.算法如下:

算法 4.1:匹配树到重写规则的编译

1.从根结点开始,I={},集合N包含根的所有儿子,Name存原函数名;

2.若N中所有结点的索引都为X,则转3,否则转5;

3.设$X=[x_1,\cdots,x_n]$,若存在$T\in I$,$T=[t_1,\cdots,t_m]$,满足$m<n$,且$t_1=x_1,\cdots,t_m=x_m$,则转5;

4.$I_1=I\cup\{X\}$,$N_1=\{n|$存在$m\in N$,n为m的儿子$\}$,分别以I_1,N_1代替I和N,转2;

5.按下列方法定义一组组合子重写规则:

1)组合子名为Name中所存放标识符;

2)对于N中各个结点n,产生一重写规则:左部为结点中的项;若n为叶子结点,右部为原函数定义相应等式的右部表达式;否则,右部为项$F[x_1\cdots x_n]$,其中F为新的函数名,并转2定义新函数F:即Name存F,N包含本结点所有子结点,I不变;

6.结束.

可以证明,算法4.1所生成的重写规则满足条件2和3.算法只对匹配树进行一遍搜索,其复杂度为O(n).Kennaway在[7]中提出的参量一致化算法和模式平坦化算法中,分别对函数定义中所有等式按不同关系划分等价类,原函数中同一等价类的等式合并为一个等式,右部为对新函数的调用.新函数的等式对应等价类中各等式.所以两个算法对函数定义一共至少搜索两遍.而新函数可能又含有参量的不一致和模式不平坦,需要进一步转换.所以算法4.1比[7]中算法效率高.

完成上述转换,算法4.3可以对重写规则加标记,生成CIL代码.

算法 4.2:加标记算法

1.若右部为一构造子,加上'*';

2.若右部根为一个变元,加上'*';

3.若右部根为函数符号F,且F的定义中参量都为变量,则加上'*';

4. 若右部根为一个函数符号 F，但 F 的参量中有 n 个构造子，则 F 加 n 个‘#’，且在指向这些参量的弧上加上‘^’；

5. 对函数或构造子的参量递归地进行 1 到 4 加标记算法.

结　语：本文给出了一个惰性模式匹配的编译实现算法、函数定义参量一致化和模式平坦化算法. 前一算法提供了一个构造 MEP 的有效途径，解决了惰性模式匹配编译实现效率低的问题. 后一算法利用了现有的结果，直接从匹配树构造重写规则，避免了对函数定义的复杂计算，更是大大提高了编译效率.

上述两种算法已在并行图归约系统的 EML 编译器上实现，效果良好.

参考文献

1 R. Milner & Mitchell, Introduction to Standard ML, Computer Science Dept., Univ. of Edinburgh, UK.

2 王鼎兴，周光明，模式匹配在函数语言中的作用及其编译算法，《计算机学报》，1989，11.

3 L. Augustsson, Compiling Pattern Matching, LNCS 201, Nancy, France, 1985.

4 L. Cardelli, Compilling a Functional Language, Proc. of the 1984 LISP and Functional Programming Conference.

5 王鼎兴，郑纬民，杜晓黎，函数语言的图归约实现方法与技术，《智能技术与系统基础》，高庆狮主编，北京大学出版社.

6 A. Laville, Lazy Pattern Matching in the ML Language, Proc. of the 7th Conf. on Foundations of Software & Comp. Sci., Pune(India), Dec. 1987, LNCS.

7 J. R. Kennaway, Implementing Term Rewrite Language in Dactl, CAAP'88.