平行最外模式匹配*

沈 理 林 凯 孙永强

(上海交通大学计算机系 上海 200030)

摘要 平行最外策略是归约系统中适用范围非常广的策略,平行最外模式匹配研究适用于该策略的高效模式匹配方法.本文在生成自适应模式匹配自动机APMA的基础上,充分利用匹配失败状态的部分匹配信息,构造了平行最外模式匹配自动机POPMA.利用POPMA进行模式匹配,在维持空间开销不大的前提下,降低了朴素思想的2大时间开销,并将模式匹配和归约策略结合起来,几乎一遍扫描即可找出所有平行最外匹配子项,具有很高的时空效率.借助于POPMA,还可以对平行最外策略进行改进.

关键词 模式匹配,归约策略,自动机,项.

模式匹配和归约策略是归约系统研究中密切相关的2大主题,广泛应用于非过程化程序语言、重写系统、定理证明和代码优化等领域的理论和实践中.要实现高效的归约系统,需将两者结合起来考虑.较之函数式语言中最左最外策略[1]和强顺序正则系统中的按需调用策略[2],平行最外策略是归约系统中一种适用范围更广的策略[3,4],实际工作中非常值得研究.我们在本文中提出平行最外模式匹配问题,研究适用于平行最外归约策略的高效模式匹配方法.

平行最外模式匹配问题要寻找目标项的一组平行最外模式匹配子项,目前的模式匹配方法还无法很好地解决它.归约系统中的项一般用树表示,我们研究的模式匹配技术也是基于树.朴素的树模式匹配思想尝试用模式集里每个模式去匹配目标项的每个子项,带来2大开销:1)目标项中逐个子项的匹配;2)模式集中逐个模式的匹配.目前的方法针对这2大问题进行了改进:文献[5,6]的方法较好地解决了问题1,文献[1,6~8]和文献[5]中自底向上方法较好地解决了问题2.虽然文献[6]和文献[5]中自底向上方法克服了2大开销,但前者将树匹配转化为串匹配,导致了替代操作的不便利,不适于用树或有向无环图(DAG)表示的实际系统,而后者找到的第1个匹配是最左最内的,对于最左最外和平行最外策略来说是很低效的.

我们的方法克服了目前方法的缺点,在维持空间开销不大的前提下,降低了朴素思想的2大时间开销,还具有不必读完目标项的平行最外特征,最好情况下,一遍扫描即可找出所

---

* 作者沈理,1971年生,硕士生,主要研究领域为函数式语言和重写系统.林凯,1963年生,副教授,主要研究领域为软件技术.孙永强,1931年生,教授,主要研究领域为计算和科学理论,并行处理,程序语言.

本文通讯联系人:沈理,上海200030,上海交通大学计算机系

本文1995-09-22收到修改稿

有平行最外匹配子项. 我们首先预处理模式集,构造自适应模式匹配自动机 APMA[7],避免逐个模式匹配目标项. 由于要求的是目标项的一组平行最外模式匹配子项,对 APMA 作进一步的处理,将每个匹配失败状态与一组特定操作联系起来. 这组操作充分利用部分匹配信息,避免不可能成功的匹配尝试,指出下一组可能匹配的平行最外子项并将之与相应的自动机中间状态相联系,可大大降低冗余搜索的可能性,也降低了开销 1. 改进的 APMA 称为平行最外模式匹配自动机 POPMA,根据 POPMA,即可自顶向下进行平行最外模式匹配. 匹配的时间复杂度为 O(emdrt * subsz/usert),空间复杂度为 O(subsz+autsz),其中 emdrt * subsz 为匹配完成时目标项被搜索部分规模,usert 为信息利用率,subsz 为目标项规模,autsz 为 POPMA 规模.

1 概念和表示

假定读者熟悉项(Term)和位置(Position)的概念. $P(t)$ 表示项 t 所有出现位置的集合,若 $p\in P(t)$,则 t/p 表示项 t 在位置 p 处的子项(Subterm),$t[p<-s]$ 表示由 s 替代子项 t/p 所得新项. 从变量到项的映射叫做替代(Substitution),项 t 的例示(Instance)$t\beta$ 是将替代 β 作用到 t 中每个变量后所得新项. 若 t 为 u 的例示,则称 u 为 t 的前缀(Prefix),记 $u\leqslant t$. u 的边缘(Fringe)是 u 中所有变量位置的集合. 若项 t 和 s 拥有一个共同的例示,称其能合一(Unify). 前缀 u 的匹配集(Match Set)L_u 是指能和 u 合一的模式集. 我们用 $f,g,a,b,\cdots$ 表示函数,$x,y,\cdots$ 表示变量.

下面的定义中,$L=\{l_1,\cdots,l_n\}$ 为模式集,t 为项. 若存在 $l\in L, l\leqslant t$,称 t 有匹配模式 l,若存在 $l\in L$,l 能与 t 合一,称 t 有合一模式 l. 显然,匹配模式必为合一模式,而合一模式不一定为匹配模式.

定义 1.1. 若 t 的子项 t' 有匹配模式 $l\in L$,则称 t' 为 t 在 L 上的模式匹配子项(Pattern Matching Subterm),简称 *Masbt*.

定义 1.2. 若 t 的子项 t' 有合一模式 $l\in L$,则称 t' 为 t 在 L 上的模式合一子项(Pattern Unifying Subterm),简称 *Unsbt*.

定义 1.3(**平行最外 *Masbt* 集**). 设 $tset=\{t/p\mid p\in P(t), t/p$ 为 t 在 L 上的 *Masbt*,且不存在 $p'<p$,有 t/p' 为 t 在 L 上的 *Masbt*$\}$,则称 $tset$ 为 t 在 L 上的平行最外 *Masbt* 集.

定义 1.4(**平行最外 *Unsbt* 集**). 设 $tset=\{t/p\mid p\in P(t), t/p$ 为 t 在 L 上的 *Unsbt*,且不存在 $p'<p$,有 t/p' 为 t 在 L 上的 *Unsbt*$\}$,则称 $tset$ 为 t 在 L 上的平行最外 *Unsbt* 集.

定义 1.5(**平行最外模式匹配问题**). 寻找一组 t 在 L 上的平行最外 *Masbt* 集$\{(t_1,l_{k1}),\cdots,(t_n,l_{kn})\}$,$l_{ki}$ 为 t_i 的匹配模式.

定义 1.6(**平行最外模式合一问题**). 寻找一组 t 在 L 上的平行最外 *Unsbt* 集$\{(t_1,l_{k1}),\cdots,(t_n,l_{kn})\}$,$l_{ki}$ 为 t_i 的合一模式.

给定归约系统,对项 t 进行归约,平行最外策略的思想是平行地归约 t 的所有最外的 *Redex*(可归约子表达式). 若将归约系统的规则左部的表达式集合 L 看作模式集,则 t 在 L 上的 *Masbt* 和 t 的 *Redex* 是一一对应的. 那么,通过平行最外模式匹配即可得到 t 的平行最外 *Redex* 集,实施平行最外策略归约. 平行最外模式匹配是本文探讨的主要问题,平行最外模式合一则是我们构造匹配自动机中涉及的一个子问题.

例 1.7:设模式集 $L=\{l_1:f(a,\_),l_2:g(b,\_,c)\}$,则对于项 $t=f(f(a,b),g(g(b,a,c),g(x,y,z),c))$,其在 L 上的平行最外 *Masbt* 集为$\{(t/1,l_1),(t/2.1,l_2)\}$,其在 L 上的平行最外 *Unsbt* 集为$\{(t/1,l_1),(t/2.1,l_2),(t/2.2,l_2)\}$,其中 $t/1=f(a,b)$,$t/2.1=g(b,a,c)$,$t/2.2=g(x,y,z)$.

2　构造自适应模式匹配自动机

将模式集预处理成一个类似 *DFA* 的自动机可以快速地得到匹配于目标项(仅在其根部)的模式.这种匹配自动机的优点在于一遍扫描即可区分出所有模式,匹配时间不依赖于模式数.文献[7]中引入自适应遍历概念所构造的自动机,虽然解决的是模式匹配的一个子问题,但鉴于其规模小、匹配效率高的特点,作为我们进一步构造平行最外模式匹配自动机的基础却是非常适合的.下面通过增加匹配失败状态,并用 *DAG* 最小自动机规模,来构造我们的自适应模式匹配自动机 *APMA*.

APMA 状态有 3 种类型:Ⅰ)部分匹配状态,Ⅱ)匹配成功状态,Ⅲ)匹配失败状态.状态 v 指代三元组$(u,pos[v],Mthp[v])$,u 为到达 v 时读入的前缀,$pos[v]$为下一自适应匹配位置,$Mthp[v]$为匹配模式编码或匹配失败编码.Ⅱ,Ⅲ型状态无 *pos* 域;Ⅰ型状态无 *Mthp* 域.初态 s_0 指代三元组$(x,\Lambda,null)$,*null* 为空;Ⅱ,Ⅲ型状态统称为终态.*APMA* 状态转换关系则由算法 2.1 构造得到.

算法 2.1(构造 *APMA*). 设 v 为 *APMA* 状态,u 为到达 v 状态时读入的前缀,L_u 为前缀 u 的匹配集,*failstate* 为部分模式集 L_u 匹配失败状态,即 L_u 中任一模式无法得到匹配时应转入的状态,则构造过程如下.

```
Procedure Build APMA(v,u,Lu,failstate)
1. if Lu 中存在匹配于 u 的模式 m then
2.    Mthp[v]:=m /* v 为匹配成功状态 */
3. else
4.    p:=select(u,Lu)
          /* select 为选择下一个自适应匹配位置的函
             数 */
5.    pos[v]:=p
6.    if ∃ l∈Lu with root(l/p)为变量 then
          /* root 取项根部符号 */
7.       建立新结点 v'
8.       建立新边(v,v',≠)
          /*(v,v',≠)表示从 v 到 v'标注有≠符号的边
             */
9.       L':={l|l∈Lu 且 root(l/p)为变量}
10.      BuildAPMA(v',u,L',failstate)
11.      failstate:=v'
12.   else
13.      建立新边(v,failstate,≠)
14.   for 每个 c∈{非变量符 root(l/p)|l∈Lu} do
15.      建立新结点 v'
16.      建立新边(v,v',c)
17.      u':=u[p<-c(y1,…,yrank(c))]
             /* rank 指出函数符号的元数 */
18.      L':={l|l∈Lu 且 root(l/p)=c}
19.      BuildAPMA(v',u',L',failstate)
```

BuildAPMA 由 *BuildAPMA*$(s_0,x,L,dismatched)$初始激活,其中 s_0 为自动机初态,L 为模式集,*dismatched* 为匹配失败状态,表示目标项无法匹配 L 中任一模式.

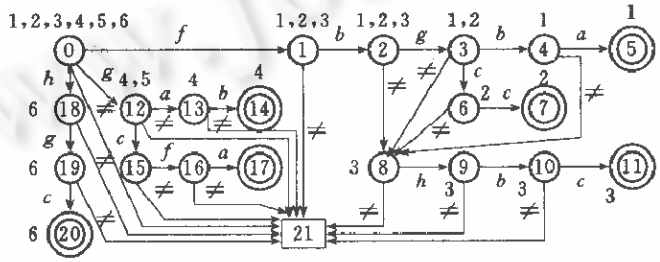
对 L_u 中每个模式在 p 位置出现的不同函数符号,14～19 行为其建立不同的状态转换关系.6～13 行对≠情况进行了处理:当存在 p 位置为变量的模式时,遇≠仍可匹配下去,7～10 行为其建立状态转换关系;否则无法继续匹配部分集 L_u,13 行将其转入相应的部分模式集 L_u 匹配失败状态 *failstate*.11 行对 *failstate* 的调整,是为 v 的子状态在部分模式集上匹配失败所作的准备.13 行语句体现了用 *DAG* 最小化 *APMA* 自动机规模的思想.

自适应遍历提高效率的思想体现在第 4 行 *select*(u,L_u)函数上.*select* 函数根据前缀 u

和其匹配集 L_u，选择下一个自适应匹配位置. 文献[7]中介绍了 5 种贪心法及 *Index* 策略，并证明了后者在自动机规模及匹配时间上均是最优的.

Index 是指前缀 u 边缘上的某个位置，当 u 的匹配集 L_u 中任一模式被匹配时，该位置均要被检测. 事实上就是 u 边缘上的位置 p，对任意 $l\in L_u$，$l(p)$为函数(常数为零元函数). 考虑到前缀的 *Index* 并非每个归约系统所必需，在找不到 *Index* 时需要采用贪心策略，因此我们的自适应遍历策略结合了 *Index* 和贪心思想. 在该策略中，前缀 u 的下一个遍历位置 p 依次满足下列 2 个条件：1)设 $s_1=\{l|l\in L_u$ 且 $l(p)$为函数$\}$，$|s_1|$最大；2)设 $s_2=\{l(p)|l\in L_u$ 且 $l(p)$为函数$\}$，$|s_2|$最小. 不难看出，对于在 p 位置出现为变量的 L_u 中的模式，对 p 位置所做的检测其实是不必要的. 因此，条件 1 不仅能将 *Index* 找出，而且具有最小化检测不必要位置模式数的贪心思想. 在条件 1 的基础上，条件 2 最小化自动机的宽度.

例 2.2：设模式集 $L=\{l_1:f(\_,g(a,\_,b),b),l_2:f(\_,g(\_,c,c),b),l_3:f(h(b,c),\_,b),l_4:g(\_,a,b),l_5:g(f(\_,\_,a),c,\_),l_6:h(g(\_,\_,\_),c)\}$，构造 L 的 *APMA* 如图 1.



Build APMA 生成模式集 L 的 *APMA*. ○，◎，□，分别为Ⅰ，Ⅱ，Ⅲ型状态，L_u 显示在每个状态 u 上，对应状态信息如下：$pos[0]=\Lambda$，$pos[1]=3$，$pos[2]=2$，$pos[3]=2.3$，$pos[4]=2.1$，$pos[6]=2.2$，$pos[8]=1$，$pos[9]=1.1$，$pos[10]=1.2$，$pos[12]=2$，$pos[13]=3$，$pos[15]=1$，$pos[16]=1.3$，$pos[18]=1$，$pos[19]=2$，$Mthp[5]=1$，$Mthp[7]=2$，$Mthp[11]=3$，$Mthp[14]=4$，$Mthp[17]=5$，$Mthp[20]=6$，$Mthp[21]=dismatched$.

图 1

匹配由状态 0 开始，此时 u 为空，在 $pos[0]=\Lambda$ 位置读入符号，若为 f,g,h，可继续匹配，分别进入状态 1，12 或 18，否则进入失败状态 21. 状态 1 对应的 $u=f(\_,\_,\_)$，由自适应要求，需在 $pos[1]=3$ 位置读入符号，若为 b，进入状态 2，否则进入失败状态 21. 当到达状态 5 时，$u=f(\_,g(a,\_,b),b)$，l_1 得到匹配. 可见，通过 *APMA* 状态所保存的部分匹配信息 u 和自适应匹配位置以及状态间关系，即可进行自适应模式匹配.

3 构造平行最外模式匹配自动机

APMA 解决了模式匹配的一个子问题，判定目标项 t 本身是否为一个 *Masbt*，而平行最外模式匹配问题，是要寻找一组 t 在模式集 L 上的平行最外 *Masbt* 集. 我们在 *APMA* 的基础上，充分利用中间状态的部分匹配信息，一方面避免不可能成功的匹配尝试，另一方面将可能匹配成功的子项的部分匹配信息与自动机中间状态相联系，来构造平行最外模式匹配自动机 *POPMA*，实现几乎一遍扫描的高效模式匹配思想.

APMA 仅有一个匹配失败状态 *dismatched*，但是在不同位置处匹配失败所保留的部分匹配信息 u 却不同，因此有必要分裂 *APMA* 中的状态来构造 *POPMA*. 在 *POPMA* 的每一匹配失败状态上，当前项 t 无法匹配下去，但充分利用部分匹配信息 u，可得到一组有匹配可

能的 t 的平行最外子项,使之继续匹配下去.

于是,*POPMA* 需与集合 *Mthngs* 协同工作. *Mthngs* 中存放欲匹配的一组平行最外子项,每个子项 t/p 带有一个部分匹配信息,以 *POPMA* 中状态 s 表示,则 *Mthngs* 中元素为二元组 (p,s). 设 t 为目标项,t/p 为当前匹配子项,则可将 *POPMA* 的每一匹配失败状态 v 对应于集合 $Nxts[v]=\{(p_1,s_1),\cdots,(p_n,s_n)\}$,表示将元组 $(p.p_1,s_1),\cdots,(p.p_n,s_n)$ 依次加入 *Mthngs* 的一组操作,由 *Mthngs* 来传递需继续匹配的平行最外子项信息,其中 p_i 为相对于 t/p 的位置信息,s_i 为 *POPMA* 状态. 若 u 为到达 v 时读入的 t/p 之前缀,再设 $\{(u/p_1,l_{k1},\cdots,(u/p_n,l_{kn})\}$ 为 u 在 L 上的平行最外 *Unsbt* 集,则 s_i 应为 $t/(p.p_i)$ 读入前缀 u/p_i 后所能到达的状态. 可见,如何充分利用部分匹配信息,可转化为平行最外模式合一问题.

基于上述思想,*POPMA* 状态继承了 *APMA* 的 3 种类型:Ⅰ)部分匹配状态;Ⅱ)匹配成功状态;Ⅲ)匹配失败状态. 不过,*POPMA* 状态 v 指代四元组 $(u,pos[v],Mthp[v],Nxts[v])$,$u$ 和 $pos[v]$ 与 *APMA* 类似,但 $Mthp[v]$ 在此仅指出匹配模式编码,而 $Nxts[v]$ 则指出继续进行匹配的一组平行最外子项. Ⅰ和Ⅲ型状态无 *pos* 域和 *Mthp* 域,Ⅰ和Ⅱ型状态无 *Nxts* 域. 初态 s_0 指代四元组 $(x,\Lambda,null,null)$,*null* 为空;Ⅰ,Ⅲ型状态统称为 *POPMA* 终态. *POPMA* 状态转换关系则由算法 3.1 构造得到.

算法 3.1(构造 POPMA). *BuildPOPMA* 为构造主过程,*ReBuildAPMA* 为改造 *APMA* 过程,*AppendUnsbt* 为求得平行最外 *Unsbt* 集过程,*Unified* 为合一性检查过程.

```
Procedure BuildPOPMA
1. BuildAPMA(s0,x,L,dismatched)
2. ReBuildAPMA(s0,x)
3. 删除 dismatched 状态
Procedure ReBuildAPMA(v,u)
/* v 为 APMA 状态,u 为到达 v 时读入的前缀 */
1. if v 已被访问或 v 为匹配成功状态 then return
2. 标记 v 被访问过
3. v'i:=nextstate(v,≠)        /* nextstate 为状态转换关系函数,此处指出 v 的≠边指向状态 */
4. if v'=dismatched then
5.    删除边(v,v',≠)
6.    建立新结点 v''
7.    建立新边(v,v'',≠)
8.    if v=s0 then
9.       Nxts[v'']i:=sonset     /* sonset 指示要将当前匹配子项的每个儿子加入 Mthngs */
10.   else
11.      Nxts[v'']i:=Φ
12.      AppendUnsbt(u,Λ,Nxts[v''])
13. else
14.   ReBuildAPMA(v',u)
15. for 每个 v'i:=nextstate(v,c) with c 不为≠ do
16.   ReBuildAPMA(v',u[pos[v]<-c(y1,…,yrank(c))])
Procedure AppendUnsbt(t,p,set)
/* 将 t/p 的平行最外 Unsbt 集加入 set */
1. if t/p 为变量 then seti:=set∪{(p,s0)}
2. else
3.    si:=s0
4.    if Unified(t/p,s,s0) then
5.       seti:=set∪{(p,s)} /* t/p 有合一模式 */
6.    else      /* t/p 无合一模式 */
7.       for ii:=1 to rank(root(t/p)) do
8.       AppendUnsbt(t,p.i,set)
Function Unified(t,s,v):boolean
/* t 为欲与 L 合一的项,s 为 t 在 L 的 APMA 上可达的部分匹配状态,v 为合一性检查的下一个位置所在状态 */
1. if v 为匹配成功状态 then return true
2. else
3.    if v 为匹配失败状态 then return false
4.    else
5.       if t(pos(v))为函数 then /* 合一性检查 */
6.          if v=s then
7.             si:=nextstate(v,t(pos(v)))   /* s 随 v 可继续匹配 */
8.             return Unified(t,s,s)
9.          else
10.            returnUnified(t,s,nextstate(v,t(pos(v))))
11.      else /* t 中 pos(v)位置可任意 */
12.         for 每个 v'i:=nextstate(v,c) do
13.            if Unified(t,s,v') then
14.               return true
15. return false
```

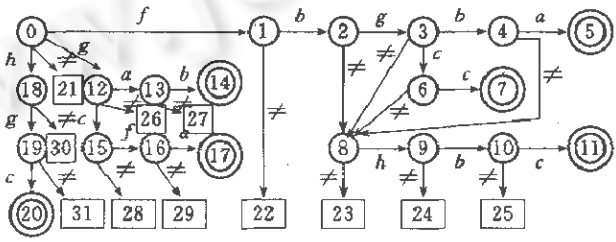
BuildPOPMA 通过改造 *APMA* 得到 *POPMA*,分裂匹配失败状态的过程 *ReBuildAPMA* 是关键之一.

ReBuildAPMA 在分裂 *APMA* 匹配失败状态的同时，赋以对应的继续匹配子项集 *Nxts*. 1～2 行是针对 *APMA* 为 *DAG* 时所进行的处理；3～7 行是分裂匹配失败状态操作；3，13～16 行是对 *APMA* 进行广度优先遍历. 8～12 行是求 *Nxts*[*v*]的操作，主要通过找到 *u* 在 *L* 上的平行最外 *Unsbt* 集来实现，体现在过程 *AppendUnsbt* 上.

AppendUnsbt(*t*,*p*,*set*)功能是将 *t*/*p* 的平行最外 *Unsbt* 集加入 *set*. 此处 *Unsbt* 集中元素(*p*,*s*)与第 1 节定义形式稍有出入，*s* 表示项 *t*/*p* 作为需进一步匹配的前缀在 *APMA* 中所能达到的状态. 4 行 *Unified*(*t*/*p*,*s*,s_0)检测 *t*/*p* 是否有合一模式；1，5，8 行执行并入操作.

布尔型函数 *Unified*(*t*,*s*,*v*)返回 *t* 是否能与 *L* 合一的真假值，变量参数 *s* 记录当 *Unified* 为真时 *t* 在 *APMA* 上可达的部分匹配状态，*v* 为合一性检查的下一个位置所在状态. 合一性检查过程与利用 *APMA* 进行模式匹配过程极为相似，区别在于 11～14 行对 *t*(*pos*(*v*))为变量的处理上. 不是进入≠边指向状态，而是对 *v* 的每一子状态作合一性检查，有一个合一即可返回真，这是合一问题不同于匹配问题的关键所在.

例 3.2：构造例 2.2 中模式集 *L* 的 *POPMA* 如图 2.



BuildPOPMA 生成模式集 *L* 的 *POPMA*. ○，◎，□，分别为Ⅰ，Ⅱ，Ⅲ型状态，对应状态信息如下：*pos*，*Mthp* 域参见图 1，*Nxts*[21]=*sonset*，*Nxts*[22]=*Nxts*[26]={(1,0),(2,0),(3,0)}，*Nxts*[23]=*Nxts*[30]={(1,0),(2,0)}，*Nxts*[24]={(1,18),(2,0)}，*Nxts*[25]={(1.2,0),(2,0)}，*Nxts*[27]=*Nxts*[28]={(1,0),(3,0)}，*Nxts*[29]={(1,1),(3,0)}，*Nxts*[31]={(1,12),(2,0)}.

图 2

Ⅰ，Ⅱ型状态的生成和工作原理与 *APMA* 类似，Ⅲ型状态带有的 *Nxts* 信息是其与 *APMA* 的主要区别. 状态 0 在∧位置读入符号不为 *f*，*g*，*h* 时，进入Ⅲ型状态 21，*Nxts*[21]=*sonset* 表示以后的匹配应对当前项的 1，2，…位置处子项进行. 状态 9 的 *u*=*f*(*h*(_,_),_,*b*)，此时若 *pos*[9]=1.1 位置不为 *b*，则进入Ⅲ型状态 24，*Nxts*[24]={(1,18),(2,0)}表示以后的匹配应对当前项的位置 1 和 2 处子项进行. 匹配位置 1 处子项时，由于已读入 *h*(_,_)，故可直接从状态 18 开始匹配. 具体匹配方法详见下节. 另外，对于 *Nxts* 相同的匹配失败状态可将之合并以减小 *POPMA* 规模，易于实现便不再赘述. 上例中合并后的匹配失败状态应为 21，{22,26}，{23,30}，24，25，{27,28}，29，31.

4 平行最外模式匹配

我们用 *Mthngs* 和 *Mtheds* 2 个集合与 *POPMA* 协同工作来进行平行最外模式匹配. *Mthngs* 为欲匹配的平行最外子项集，子项上附带有部分匹配的 *POPMA* 状态 *s*；*Mtheds* 记录已匹配成功的平行最外子项集，子项上附带有对应的匹配模式. 利用 *POPMA* 不同状态带有的不同信息，匹配可按如下过程循环执行直至 *Mthngs* 为空为止：从 *Mthngs* 中取一子项 *t'*，用 *POPMA* 匹配至终态 *v*，若 *v* 为Ⅱ型，则将 *t'* 连同 *Mthp*[*v*]加入 *Mtheds*，若终态 *v* 为

Ⅱ型,则根据 $Nxts[v]$ 将需继续匹配的子项加入 *Mthngs*. 所得的平行最外 *Unsbt* 集最终存放在 *Mtheds* 中,详细过程见算法 4.1.

算法 4.1(平行最外模式匹配). t 为目标项,*Mthngs* 和 *Mtheds* 如上述,匹配过程如下.

```
Procedure match(t,Mthngs,Mtheds)
1. if Mthngs 为空 then return
2. else
3.    取出 Mthngs 的第一元放入二元组(p,s)
4.    v1:=s      /* s 为部分匹配状态 */
5.    while v 为 I 型状态 do      /* 用 POPMA 匹配至终
                                    态 v */
6.       v1:=nextstate(v,t(p.pos[v]))
7.    if v 为 Ⅱ 型状态 then      /* 匹配成功 */
8.       Mtheds1:=Mtheds∪{(p,Mthp[v])}
9.    else      /* 匹配失败 */
10.      if Nxts[v]=sonset then
11.         Mthngs1:=Mthngs∪{(p,1,s0),…,(p.rank
            (root(t/p)),s0)}
12.      else
13.         for 每个(p',s')∈Nxts[v] do
14.            Mthngs1:=Mthngs∪{(p.p',s')}.
15. match(t,Mthngs,Mtheds)
```

平行最外模式匹配以 *match*(t,*Mthngs*,*Mtheds*)初始激活,其中 t 为目标项,*Mthngs*=$\{(\Lambda, s_0)\}$,*Mtheds*=Φ. 5～6 行为按 *POPMA* 匹配,7～8 行对匹配成功进行处理,9～14 行对匹配失败进行处理,15 行递归进行剩余匹配.

例 4.2:利用例 3.2 的 *POPMA* 对目标项 t 进行平行最外模式匹配,其中 $t=f(h(g(f(g(a,b,c),h(a,b),a),c,f(a,b,c)),a),g(c,a,b),b)$.

匹配由 *Mthngs*=$\{(\Lambda,0)\}$开始,取出$(\Lambda,0)$,则被测子项为 t,初始状态为 0,在 *POPMA* 上读入 $f(h(g(_,_,_),_),g(c,_,b),b)$后到达终态 24. 根据 $Nxts[24]$,*Mthngs* 变为 $\{(1,18),(2,0)\}$. 再从 *Mthngs* 中取出$(1,18)$,则被测子项为 $t/1$,初始状态为 18,在 *POPMA* 上读入 $h(g(_,_,_),a)$后到达终态 31. 这样,借助 *Mthngs*,*Mtheds* 和 *POPMA*,最终可将匹配完成,得 *Mtheds*=$\{(1.1,5),(2,4)\}$,即$\{(t/1.1,l5),(t/2,l4)\}$为 t 在 L 上的平行最外 *Masbt* 集. 完整匹配步骤如表 1 所示.

表 1

步骤	*Mthngs*	*Mtheds*	被测子项	终止状态	读入符号	有用符号
1	$(\Lambda,0)$	Φ	t	24	$f(h(g(\_,\_,\_),\_),g(c,\_,b),b)$	$f(h(\_,\_),\_,b)$
2	$(1,18),(2,0)$	Φ	$t/1$	31	$h(g(\_,\_,\_),a)$	$h(g(\_,\_,\_),a)$
3	$(1.1,12),(1.2,0),(2,0)$	Φ	$t/1.1$	17	$g(f(\_,\_,a),c,\_)$	$g(f(\_,\_,a),c,\_)$
4	$(1.2,0),(2,0)$	$(1.1,5)$	$t/1.2$	21	a	a
5	$(2,0)$	$(1.1,5)$	$t/2$	14	$g(\_,a,b)g(\_,a,b)$	
结果	Φ	$(1.1,5),(2,4)$				

在给出算法之后,我们对匹配的时空复杂度进行一定的分析. 目标项和 *POPMA* 是匹配时所需存储的主要对象,*Mthngs* 和 *Mtheds* 的空间开销相对于目标项开销可忽略不计,故空间复杂度应为 $O(subsz+autsz)$,*subsz* 为目标项规模,*autsz* 为自动机规模. 显然,*subsz* 可以用目标项所含的符号数作为衡量标准.

为了测度 *autsz*,我们将Ⅱ型状态结点的费用计入其父结点(必为 I 型)中,标有函数符号的边的费用计入其终止结点(必为 I 型或Ⅱ型)中,≠边的费用计入其起始结点(必为 I 型)中. 于是,每个计值单元以一个 I 型或Ⅱ型结点为代表,最多再包含一个Ⅱ型结点,一条导入自身的函数符号边和一条自身导出的≠边. 设 *POPMA* 共有 *untnu* 个计值单元,最大

计值单元规模等价于 *mlusz* 个符号空间，则 $autsz \leqslant untnu * mlusz$. 由于 I 型和 II 型状态对应于按某一自适应序搜索过程中的一组两两不同的模式前缀，因此其状态总数 $untnu \leqslant patsz$，*patsz* 为模式集所含符号总数，故 $autsz \leqslant patsz * mlusz$. 而计值单元规模的测度可用 II 型结点 *Nxts* 域规模来衡量，因为其它的费用易知均为常数级. 设最大的 II 型结点 *Nxts* 集合有 *mnssz* 个元素，则存在常数 k，$mlusz \leqslant k * mnssz$. 设 *mptsz* 为模式集中规模最大的模式所含符号数，则 $mnssz < mptsz$，故算法空间复杂度不超过 $O(subsz + patsz * mptsz * k)$. 由于 *POPMA* 以 *DAG* 存放，尤其是一般系统中 II 型结点共享的可能性颇大，而且我们的估算均按最坏情况进行，故实际的空间开销要小得多.

时间复杂度可用匹配完成时读入的信息量作为测度. 设 *emdrt*＝(匹配完成时已搜索部分规模/*subsz*)，信息利用率 *usert*＝(被利用的信息量/读入的信息量)，由于 *emdrt* * *subsz* 为匹配完成时被利用的信息量，故时间复杂度为 $O(emdrt * subsz/usert)$. 例 4.2 中，*emdrt*＝11/23，*subsz*＝23，*usert*＝11/15，所以匹配时间为 15. 最好情况下 *usert*＝1，读入信息均被充分利用，则一遍扫描即可得到目标项平行最外 *Masbt* 集.

5 结束语

本文在 APMA 的基础上，构造了 POPMA. 利用 POPMA 来进行平行最外模式匹配，具有 O(subsz＋ patsz * mptsz * k)的最坏情况空间复杂度和 O(emdrt * subsz/usert)的时间复杂度，不仅限制了自动机规模，还优化了朴素匹配思想的 2 大时间开销，并将匹配过程与归约策略有机地结合了起来，时空效率很高.

我们的讨论是基于线性模式的，将其扩展至非线性模式也不难实现. 只需将非线性模式对应的 II 型状态 v 改为 I 型，并增设其 2 个子状态 v_1，v_2，若一致性检查成功则转入 II 型子状态 v_1，否则转入 III 型子状态 v_2.

结合文献[2]中按需调用思想，利用 POPMA 在寻找平行最外 Masbt 集过程中，可以发现一些必须要归约的位置，先对其进行归约消除冗余归约可能性，以此来改进平行最外归约策略进一步提高归约效率，我们目前正在做这方面的工作.

参考文献

1 Peyton Jones S L. The implementation of functional programming languages. Prentice Hall, 1987.

2 Klop Jan Willem, Middeldrop Aart. Sequentiality in orthogonal term rewriting systems. JSC, 1991, **12**: 161～195.

3 Bergstra J A, Klop J W. Conditional rewrite rules: confluence and termination. JCSS, 1986, **32**.

4 陆朝俊. 重写系统研究[博士论文]. 上海交通大学, 1995.

5 Hoffmann C M, O'Donnell M J. Pattern matching in trees. JACM, 1982, **29**(1): 68～95.

6 Ramesh R, Ramakrishnan I V. Nonlinear pattern matching in trees. LNCS, July 1988, **317**: 473～488.

7 Sekar R C, Ramesh R, Ramakrishnan I V. Adaptive pattern matching. LNCS, 1992, **623**: 247～260.

8 Ramesh R, Ramakrishnan I V. Incremental techniques for efficient normalization of nonlinear rewrite systems. LNCS, 1991, **488**: 335～347.

PARALLEL—OUTERMOST PATTERN MATCHING

Shen Li　Lin Kai　Sun Yongqiang

(*Department of Computer Science　Shanghai Jiaotong University　Shanghai*　200030)

Abstract　Parallel—outermost strategy is widely used in reduction systems. Parallel—outermost pattern matching studies the pattern matching method adapted to the strategy. By preprocessing the patterns into an adaptive pattern matching automaton APMA, this paper takes advantages of local matching information of unsuccessful matching states to build parallel—outermost pattern matching automaton POPMA. By use of POPMA, all of parallel—outermost pattern matching subterms can be gotten through a single scan. The method limits the size of the automaton, avoids the overheads in the naive algorithm and makes the good relation between the pattern matching and the reduction strategy. POPMA also can be used to improve parallel—outermost strategy.

Key words　Pattern matching, reduction strategy, automaton, term.