高效率重写型程序的设计*)

冯 速
(北京师范大学信息科学院 北京100875)

摘 要 本文考虑如何设计高效率(即重写步数较少的)重写型程序。文中以计算Fibonacci数列的程序为例,比较具有相同功能的重写型程序,展示编写高效率重写型程序的可能性。介绍利用动态项重写计算编写高效率重写型程序的直观、简洁的方法。其中,动态项重写计算是项重写系统的元计算模型,其计算同样基于项重写。

关键词 项重写系统,效率,动态项重写计算,重写控制

How to Write Effective Rewrite Programs

FENG Su
(College of Information Science, Beijing Normal University, Beijing 100875)

Abstract This paper considers how to design effective, that is, computes with less rewritingsteps, rewriting programs. We use programs computing Fibonacci numbers as examples, compare the effectiveness of these programs, illustrate the possibility of designing effective rewritingprograms, and show how to use dynamic term rewriting calculus, a meta computation model of term rewriting systems whose computation is also based on term rewriting, to design efective programs naturally.

Keywords Term rewriting systems, Effectiveness, Dynamic term rewriting calculus, Rewrite controlling

1 引言

项重写系统(TRS)是一种简明的计算模型,是函数型语言的基础,本身也具有可计算性,可以认为是程序设计语言,有其自身的编译器和解释器。然而,对TRS的研究大多局限于理论和应用基础的研究,作为程序设计语言的TRS研究并不活跃。我们认为其原因主要有两个:一是,TRS是一种平坦的计算模型,本身不具备模块化结构,这导致难以编写出结构性较强的程序,不符合对现代程序设计语言的要求;二是,一般认为TRS的效率较低,难以设计出高效率的程序。这里,高效率程序指的是重写步数较少的程序。

动态项重写计算(DTRC)[1]是TRS的元计算模型,同时也可以看成是TRS的拓展,其计算同样基于项重写。DTRC的层次化结构使其具有高度的模块化结构,因而克服了TRS的平坦性的弱点。

本文讨论重写型计算模型的高效程序设计问题。文中以计算Fibonacci数列的程序为例,通过对具有相同功能的重写型程序的对比,考察重写型程序的效率问题,指出利用DTRC的层次化结构及其相关的重写策略,我们可以较容易地设计出效率较高的程序。我们的设计思路非常简单:通过重写策略的控制,让希望先计算的部分先进行计算。

以下假定读者熟悉TRS和DTRC的相关基本知识,参见文[1,2],对于DTRC的应用,参见文[3,4]。为使文中的表述具有一致性,同时使所述程序可以在同一个平台上进行运行比较,这里使用DTRC系统的形式表示TRS。所有程序均可在DTRC的运行平台下运行。运行平台及相关说明以及本文的程序可从 http://www.cs.bnu.edu.cn/.fengs/d/DRTC.rar 获取。

2 Fibonacci数列

Fibonacci数列 $fib: N \to N$ 的递归定义如下:

$fib(0) = 1$

$fib(1) = 1$

$fib(m+2) = f(m) + f(m+1)$

下面的TRS *forg* 计算 *fib*,其中的 *add* 对应于自然数加法。$forg(f\ (\overbrace{s(\cdots(s}^{m\uparrow}(0)\cdots))))$计算 $fib(m)$。相对于 m,无论是否考虑加运算的时间,*forg* 都在指数时间(步骤)计算 *fib*。参见表1和表2。

```
forg =[ ; m; i; j :
        f(0)          →s(0);
        f(s(0))       →s(0);
        f(s(s(m)))    → add(f(s(m)); f(m));
        add(i; 0)     →i;
        add(i; s(j))  → s(add(i; j)) ]
```

Toyama[5]给出了一个忽略加运算时以线性时间计算 *fib* 的TRS *ftoy*(实际上,*ftoy* 只是在函数型语言的意义下是线性的。注意,其中的 *pair* 不是被定义函数。):

```
ftoy =[ ; m; i; j :
        f(0)          → s(0);
        f(s(0))       → s(0);
        f(s(s(m)))    → p(h(g(m)));
        h(m)          → pair(add(p(m); q(m)); p(m));
        p(pair(i; j))  →i;
        q(pair(i; j))  → j;
        g(0)          → pair(s(0); s(0));
        g(s(m))       → h(g(m));
        add(i; 0)     → i;
        add(i; s(j))  → s(add(i; j))]
```

*)国家自然科学基金资助项目(60273015),教育部留学回国人员科研启动基金资助项目。冯 速 博士,副教授,主要研究方向为符号计算与自动证明。

表1 各TRS的运行步骤(其中,*hi* 表示使用最内优先重写策略(程序 dtrc.hi)时的运行步骤,*ho* 表示使用最外优先重写策略(程序 dtrc.ho)时的运行步骤)

m	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
forg; *hi*	1	1	5	9	18	31	57	99	171	293	500	850	1441	2437	4113	5658
ftoy; *hi*	1	1	9	16	24	33	44	58	77	104	144	205	300	450	689	1072
ftoy; *ho*	1	1	9	19	37	66	115	196	331	555	927	1544	2567	4262	7069	11715
ftab; *hi*	1	1	4	7	11	16	23	33	48	71	107	164	255	401	636	1015
ftab; *ho*	1	1	4	7	13	22	38	65	112	193	333	574	988	1697	2908	4971

表2 不考虑加运算时各TRS的重写步骤(*hi* 和 *ho* 的含义同表1)

m	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
forg; *hi*	1	1	3	5	9	15	25	41	68	109	177	287	465	753	1219	1973
ftoy; *hi*	1	1	7	12	17	22	27	32	37	42	47	52	60	68	74	84
ftoy; *ho*	1	1	7	15	28	49	83	138	227	371	604	981	1591	2578	4175	6759
ftab; *hi*	1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
ftab; *ho*	1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

ftoy 的设计本身需要细致的观察和分析(需要所谓的神谕)。而实际上,这是常见的列表法的一种形式。通过以下简单观察可知,一般情况下,*fib* 的中间运行结果具有 $ifib(m+1)+jfib(m)$ 的形式:

$$fib(m+2)=fib(m+1)+fib(m)$$
$$fib(m+3)=fib(m+2)+fib(m+1)$$
$$=2fib(m+1)+fib(m)$$

而且有

$$ifib(m+2)+jfib(m+1)=(i+j)fib(m+1)+ifib(m)$$

由此,使用 $fm(i,j,m)$ 表示 $ifib(m+1)+jfib(m)$,易得如下 TRS *ftab*:

```
ftab =[ ; m; i; j :
        f(0)          → s(0);
        f(s(0))       → s(0);
        f(s(s(m)))    → fm(s(0); s(0);m);
        fm(i; j; 0)   → add(i; j);
        fm(i; j; s(m)) → fm(add(i; j); i;m);
        add(i; 0)     → i;
        add(i; s(j))  → s(add(i; j)) ]
```

ftab 也在线形时间计算 *fib*。然而,在最坏情况下,它需要更长的时间(表1)。这是因为在 *fm* 的重写中对 $add(x,y)$ 的项进行了大量的复制。忽略加运算时则总为线性时间复杂度(表2)。

由此可见,如果在使用左部为 $fm(x,y,z)$ 的规则之前计算所有能够计算的 $add(x,y)$ 的话,那么就可以避免对其复制,从而达到加快计算的目的。也就是说,我们希望左部为 $add(x,y)$ 的规则优先于左部为 $fm(x,y,z)$ 的规则。这可以通过 DTRC 的层次结构及关于层次的重写策略轻而易举地实现(对于此例,TRS 的最内优先策略也可以达到同样的目的,但是,一般情况并非如此)。

3 动态项重写计算与上层优先重写策略

DTRC 是 TRS 的元计算模型,也是 TRS 的拓展,其计算基于项重写。DTRC 的典型项的形式为

$[F; V : R]t$

其中,F 为函数声明,V 为变量声明,R 为规则声明,$[F; V : R]$ 的部分称为系统,t 为被重写项。

DTRC 具有高度的层次化结构,R 和 t 均可含典型项为子项(为简单起见,本文只考虑特殊的 DTRC 项:F 总为空,R 中不含典型子项,而且系统间不共享变量名或变量名的共享不影响计算)。DTRC 程序(项)中的系统呈树形结构,可以按层次设置重写策略,这样的重写策略可以和被重写项的选择策略(如与 TRS 相同的最内优先、最外优先策略)组合在一起,形成多种重写策略。

这里,我们只考虑上层优先策略,即,上层优先最内策略和上层优先最外策略。上层优先最内策略优先使用上层的系统进行重写,而对被重写项则是从左到右、从内到外地搜索重写位置。上层优先最外策略同样优先使用上层的系统进行重写,但对被重写项则是从左到右、从外到内地搜索重写位置。

为了给出两种重写策略的严格定义,下面先定义左内匹配序偶和左外匹配序偶两个概念。

定义1(左内匹配序偶) 对于系统$[; V: l_1\to r_1,\cdots,l_n\to r_n]$和项 t,

1. 当 $t=f(t_1,\cdots,t_k)$时,

(a) 若 t_i 有左内匹配序偶$\langle C[\![\ ]\!],s\rangle$;且 $t_1,\cdots,t_{i-1}$无左内匹配序偶,则 t 有左内匹配序偶$\langle f(t_1,\cdots,t_{i-1},C[\![\ ]\!],t_{i+1},\cdots,t_k),s\rangle$;否则,

(b) 若存在规则 $l_i\to r_i$ 和替换 σ $(dom(\sigma)\subseteq V)$ 使得 $l_i\sigma=t$,则 t 有左内匹配序偶$\langle\square,t\rangle$;否则 t 无左内匹配序偶;

2. 当 $t=[; V': l_1\to r_1',\cdots,l_m'\to r_m']t'$时,

(a) 若 t'有左内匹配序偶$\langle C[\![\ ]\!],s\rangle$,则 t 有左内匹配序偶$\langle[; V': l_1\to r_1',\cdots,l_m'\to r_m']C[\![\ ]\!],s\rangle$;否则,

(b) 若 $l_1,r_1',\cdots,l_{i-1},r_{i-1}'$均无左内匹配序偶(相应地,$l_i'$也无左内匹配序偶),且 l_i'有左内匹配序偶$\langle C[\![\ ]\!];s\rangle$(相应地,$r_i'$ 有左内匹配序偶$\langle C[\![\ ]\!],s\rangle$),则 t 有左内匹配序偶$\langle[;V': l_1\to r_1',\cdots,l_{i-1}'\to r_{i-1}',C[\![\ ]\!]\to r_i',\cdots,l_m'\to r_m']t',s\rangle$(相应地,$t$ 有左内匹配序偶$\langle[; V': l_1\to r_1',\cdots,l_{i-1}'\to r_{i-1}',l_i'\to C[\![\ ]\!],\cdots,l_m'\to r_m']t',s\rangle$);否则 t 无左内匹配序偶。

定义2(左外匹配序偶) 对于系统$[; V: l_1\to r_1,\cdots,l_n\to r_n]$和项 t,

1. 当 $t=f(t_1,\cdots,t_k)$时,

(a)若存在规则 $l_i\to r_i$ 和替换 $\sigma(dom(\sigma)\subseteq V)$使得 $l_i\sigma=t$,则 t 有左外匹配序偶$\langle\square,t\rangle$;否则,

(b)若 t_i 有左外匹配序偶$\langle C[\![\ ]\!],s\rangle$且 $t_1,\cdots,t_{i-1}$无左外匹配序偶,则 t 有左外匹配序偶$\langle f(t_1,\cdots,t_{i-1},C[\![\ ]\!],t_{i+1},\cdots,t_k),s\rangle$;否则 t 无左外匹配序偶;

2. 当 $t=[;V': l_1\to r_1',\cdots,l_m'\to r_m']t'$时,

(a)若 t'有左外匹配序偶$\langle C[\![\ ]\!],s\rangle$,则 t 有左外匹配序偶$\langle[;V': l_1\to r_1',\cdots,l_m'\to r_m']C[\![\ ]\!],s\rangle$;否则,

(b)若 $l_1,r_1',\cdots,l_{i-1},r_{i-1}'$均无左外匹配序偶(相应地,$l_i'$ 也

无左外匹配序偶),且 l_i 有左外匹配序偶$\langle C[\![\ ]\!],s\rangle$(相应地,$r_i$ 有左外匹配序偶$\langle C[\![\ ]\!],s\rangle$),则 t 有左外匹配序偶$\langle[;V':l_1\to r_1,\cdots,l_{i-1}\to r_{i-1},C[\![\ ]\!]\to r_i,\cdots,l_m\to r_m]t',s\rangle$(相应地,$t$ 有左外匹配序偶$\langle[;V':l_1\to r_1,\cdots,l_{i-1}\to r_{i-1},l_i\to C[\![\ ]\!],\cdots,l_m\to r_m]t',s\rangle$);否则 t 无左外匹配序偶。

定义3(上层优先最内策略) 对于任意的 DTRC 项 T,

1. 当 $T=[;V:R]t$ 时,其中 $R=l_1\to r_1,l_2\to r_2,\cdots,l_n\to r_n$,

(a)对于系统 $[;V:R]$ 和项 t,若 t 有左内匹配序偶$\langle C[\![\ ]\!],s\rangle$,则存在 l_i 和替换 σ 使得 $l_i\sigma=s$,这时有 $T\to_{hi}[;V:R]C[\![r_i\sigma]\!]$;否则,

(b)若 $t\to_{hi}s$,则 $T\to_{hi}[;V:R]s$;否则 T 为 hi-范式;

2. 当 $T=f(t_1,\cdots,t_n)$ 时,若 $t_i\to_{hi}s$ 而且 $t_1,\cdots,t_{i-1}$ 为 hi-范式,那么 $T\to_{hi}f(t_1,\cdots,t_{i-1},s,t_{i+1},\cdots,t_n)$;否则 T 为 hi-范式。

定义4(上层优先最外策略) 对于任意的 DTRC 项 T,

1. 当 $T=[;V:R]t$ 时,其中 $R=l_1\to r_1,l_2\to r_2,\cdots,l_n\to r_n$,

(a)对于系统 $[;V:R]$ 和项 t,若 t 有左外匹配序偶$\langle C[\![\ ]\!],s\rangle$,则存在 l_i 和替换 σ 使得 $l_i\sigma=s$,这时有 $T\to_{ho}[;V:R]C[\![r_i\sigma]\!]$;否则,

(b)若 $t\to_{ho}s$,则 $T\to_{ho}[;V:R]s$;否则 T 为 ho-范式;

2. 当 $T=f(t_1,\cdots,t_n)$ 时,若 $t_i\to_{ho}s$ 而且 $t_1,\cdots,t_{i-1}$ 为 ho-范式,那么 $T\to_{ho}f(t_1,\cdots,t_{i-1},s,t_{i+1},\cdots,t_n)$;否则 T 为 ho-范式。

利用列表法和上层优先重写策略快速计算 $fib(m)$ 的 DTRC 程序 $ftabl$ 是

$$Add(Ftab(\overbrace{f\ (s(\cdots(s}^{m\uparrow s}(0))\cdots))))$$

其中,
$$Add=[\ ;i,j:\ add(i;0)\to i,\ add(i,s(j))\to s(add(i,j))\ ]$$
$$Ftab=[\ ;m,k,l:\ f(0)\to s(0),\ f(s(0))\to s(0),\ f(s(s(m)))\to fm(s(0),s(0),m),\ fm(k,l,0)\to add(k,l),\ fm(k,l,s(m))\to fm(add(k,l),k,m)]$$

无论是在下层优先最内策略还是下层优先最外策略下,$ftabl$ 的效率都与 $ftab$ 在最内优先策略下的效率相同:层次结构和上层优先策略避免了 add 的复制。

事实上,对 $ftoy$ 做同样的处理并进行简单的展开,可以达到同样的效果,获得高效率程序 $Add(Ftoy(f(s(\cdots(0)\cdots))))$。其中,系统 Add 同上,$Ftoy$ 如下。该程序的效率见表3。

$$Ftoy=[\ ;m,k,l:\ h(pair(k,l))\to pair(add(k,l),k),\ g(0)\to pair(s(0),s(0)),\ g(s(k))\to h(g(k)),\ p(pair(k,l))\to k,\ f(0)\to s(0),\ f(s(0))\to s(0),\ f(s(s(m)))\to p(h(g(m)))]$$

这一程序表明,可以通过对已有的 TRS 做简单的处理来显著提高程序的效率。

表3 带层次结构程序的运行步骤(hi 和 ho 分别表示上层优先最内策略和上层优先最外策略,un 表示展开。最上行是比较用的 $forg$,最下行给出了 $fib(m)$ 的取值)

m	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$forg,hi$	1	1	5	9	18	31	57	99	171	293	500	850	1441	2437	4113	5658
$ftoy,hi$	1	1	9	16	24	33	44	58	77	104	144	205	300	450	689	1072
$ftoy,ho,un$	1	1	6	10	15	21	29	40	56	80	117	175	267	414	650	1030
$fsiml,hi$	2	2	6	10	15	21	29	40	56	80	117	175	267	414	650	1030
$fsiml,ho$	2	2	6	10	15	21	29	40	56	80	117	175	267	414	650	1030
fib	1	1	2	3	5	8	13	21	34	55	89	144	233	377	610	987

最后,我们展示一个简单、自然的高效率程序 $fsiml$。使用的系统如下所示,其中 $t(i,f(m))$ 表示 $ifib(m)$。

$$S1=[;i,j,m:\ add(add(t(i,f(s(m))),t(i,f(m))),t(j,f(s(m))))\to add(t(add(i,j),f(s(m))),t(i,f(m)))\ ]$$

$$S2=[;i,m:\ t(i,f(s(s(m))))\to add(t(i,f(s(m))),t(i,f(m))),\ t(i,f(0))\to i,\ t(i,f(s(0)))\to i\ ]$$

$$S3=[\ ;m:\ f(m)\to t(s(0),f(m))\ ]$$

首先,为了有一个统一的表示,我们使用 $S3$ 中的规则将 $fib(m)$ 转换成1$fib(m)$)的形式。这一转换只在最初使用一次,因此,我们将 $S3$ 放在最下层。

然后,反复利用 $S2$ 的规则将 $ifib(m+2)$ 展开成 $ifib(m+1)+ifib(m)$ 的形式,利用 $S1$ 的规则合并 $fib(m)$ 的系数,利用 Add 求系数的值,直至得到 $ifib(1)+jfib(0)$ 的形式。

最后,用 $S2$ 中的规则计算 $fib(1)$ 和 $fib(0)$,利用 Add 计算最终结果。

所得程序为 $Add(S1(S2(S3(f(\overbrace{s(\cdots(s}^{m\uparrow s}(0)\cdots\ ))))))))$,其中,系统 $S1,S2,S3$ 如上。程序的效率见表3。

正是由于 $fsiml$ 的层次结构和上层优先重写策略,$S3$ 的规则只能在最初使用一次,而且 $S2$ 中的规则也不会失去控制,不会改写 $ifib(m+1)+jfib(m)$ 中的 $jfib(m)$,因而可以实现我们的高效计算的基本思路:利用重写策略,优先计算需要优先计算的部分。

$fsiml$ 还可以进一步改进。

结束语 本文以计算 Fibonacci 数列的重写型程序为例,展示了设计高效率重写型程序的可能性,揭示出利用动态项重写计算的层次结构和关于层次的重写策略设计高效程序的直观、自然的方法。其思路非常简单:利用重写策略,优先计算需要优先计算的部分。优先重写系统[6](PRS)在某种程度上也可以实现我们的思路,但对重写规则的控制较复杂,没有良好的操作语义,使其不可计算。因此,DTRC 是重写型模型实用化的一个适当候选。

参考文献

1 冯速. 动态项重写计算. 计算机科学,2002,29(8):13~14

2 Huet G. Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems. J. ACM,1980,27: 797~821

3 冯速. 项重写系统的等价性的归纳证明. 计算机科学,2000,27(8):5~7

4 冯速. 项重写系统弱基终止性的归纳证明. 计算机科学,2001,28(7):105~108

5 Toyama Y. How to prove equivalence of term rewriting systems without induction. Theoret. Comput. Sci.,1991,90:369~390

6 Mohan C K. Priority Rewriting: Semantics, Confluence, and Conditionals. Lecture Notes in Computer Science,355:278~291