Вісник Харківського національного університету
Серія «Математичне моделювання. Інформаційні технології. Автоматизовані системи управління»
№ 847, 2009, с.213-220

УДК 004.41,004.51

Оптимизация переписывающей машины системы алгебраического программирования APS

А. А. Летичевский, А. А. Летичевский (мл.), В. С. Песчаненко
Институт кибернетики имени В.М. Глушкова НАН Украины, Украина,
Научно-исследовательский институт информационных технологий, Херсонский государственный университет, Украина

The main idea of algebraic programming system APS was described in the article. The main differences between all versions of algebraic programming system APS were considered. Optimization of rewriting algorithm in APS was examined. Numerical experiment for rewriting of APS was discussed.

1. Введение

APS – система алгебраического программирования была разработана в середине 80-х в отделах 100,105 Института кибернетики НАН Украины[1]. Исторически **APS** – это первая система переписывания термов, которая разделила понятия стратегий и систем правил переписывания (с.п.п.).

Система алгебраического программирования **APS** - система для прототипирования алгоритмов. Однако, ее вторая версия [2] успешно применяется в целом ряде коммерческих продуктов: **VRS**(**V**erification for **R**equirement **S**pecification)[3], **MathLog** (**Math**ematic **Log**ic for Universities)[4], **TerM** (**Ter**ra **M**athematics)[5], что позволяет авторам **APS** позиционировать ее как коммерческую систему.

Разработка конечного продукта с использованием **APS** проходит ряд этапов: разработка прототипа на языке **APLAN** (**A**lgebraic **P**rogramming **Lan**guage) системы **APS**, анализ прототипа и его оптимизация на языке **APLAN**, реализация окончательной версии на языке C++.

Всем разработчикам, которые работают с **APS**, известен факт, что перенос кода с языка **APLAN** на C++ дает ускорение на порядок. Это факт был подтвержден многочисленными независимыми опытами и разными разработчиками, поэтому не подвергается сомнению. Отметим лишь то, что при реализации на C++ не использовалась переписывающая машина **APS**.

Однако, реализация некоторых задач на C++ без использования переписывающей машины приводит к фактической реализации алгоритма слияния для конкретной задачи, что не всегда можно сделать быстро. Классическим примером такой задачи может быть функция *subs*[2] -функция подстановки (на вход поступают равенства ,перечисленные через запятую, и сам терм – куда будет выполнена подстановка). В общем случае, в левых частях равенств могут быть произвольные термы, а это значит, что существуют два варианта реализации: хеш-таблица с использованием строк в качестве ключа (медленный вариант) и нагруженное дерево (требует достаточно много времени на реализацию). Использование переписывающей машины на языке C++ при реализации подобных задач существенно ускорит время разработки программ.

Таким образом, возникают важные вопросы:

1. Какими недостатками обладает переписывающая машина **APS**?
2. Почему ее лучше не использовать в коде ,написанном на C++?
3. Как можно устранить недостатки?
4. Насколько лучше окажется переписывающая машина **APS** и на каких задачах это улучшение будет ощутимо?

Ответам на эти вопросы посвящена данная статья.

2. Переписывающая машина APS

Переписывающая машина **REM**(**Re**writing **M**achine)[6] – это алгоритм, который применяет систему переписывающих правил к данному терму. В **APS** системы переписывающих правил имеют следующий вид:

<система переписывающих правил>::= *rs*(<список переменных>)
(<список правил разделенных запятой>)
<правило>::=<простое правило> | <условное правило>
<простое правило>::=<алгебраическое выражение> =
<алгебраическое выражение>
<условное правило>::=<условие>->(<простое правило>)
<переменная>::=<идентификатор>

Для того, чтобы применить систему переписывающих правил к некоему терму, используются стратегии (специальные функции обхода дерева, которые по определенному алгоритму применяют к вершине дерева систему переписывающих правил). Стратегии **APS** базируются на двух основных стратегиях: *applr*(*t,s*), *appls*(*t,s*), где *t* - терм, *s* – система переписывающих правил, *applr* – применяет некоторое правило из *s* к *t* (если в *s* есть такие правила), *appls* – применяет некоторое правило из *s* к *t* до тех пор ,пока такое правило существует.

Алгоритм применения одного правила из системы переписывающих правил выглядит так: поочередно, сверху вниз сопоставляется терм с левыми частями правил, если сопоставление успешно (если это условное правило, и условие не равно 0) , то происходит замена правой частью правила и подстановка переменных.

Таким образом, переписывающая машина – это функция *applr*. Поскольку через нее определяются все остальные стратегии, то эффективность ее реализации имеет первоочередное значение.

Эффективная реализация стратегии *applr* использует следующие соображения:

Если левые части нескольких соседних правил идентичны, то метчинг можно осуществлять только одни раз (в некоторых случаях можно использовать этот вид оптимизации и для не подряд идущих правил).

Можно не рассматривать те правила, главная операция в левой части которых отличается от главной операции обрабатываемого терма (т.е. переписывающие правила можно сгруппировать по главной операции их левой части). Аналогичные соображения применимы и к главным операциям подтермов левых частей правил.

Для реализации указанных соображений **REM** использует специальный синтаксис для представления систем переписывающих правил, входной язык **REM**. Более детально см. [6].

Описанная выше переписывающая машина была реализована в **APS v1**, **APS v2**. Рассмотрим небольшой пример:

```
left:=rs(x,y)(
  (x,y) = x,
  (x;y) = x
);
```

Во входном языке **REM** это правило будет иметь вид:

```
left:=array(_,_)(
  match(var(1),var(2))rewrite(var(1))          (1)
  +
  match(var(1);var(2))rewrite(var(1))          (2)
);
```

При применении данного правила переписывающая машина сначала будет пытаться выполнить слияние для первой записи (1), если оно будет не успешно, то перейдет на вторую запись.

В самом плохом случае, при n различных главных знаках в левых частях правил, для того применить последние правило **REM** машина последовательно выполнит n слияний и только последние будет успешно.

Таким образом, основным недостатком переписывающей машины **в APS v2** – последовательная проверка альтернатив, что не всегда проходит быстро. Особенно это становится заметно на очень больших системах переписывающих правил, и чем больше в ней правил с различными главными знаками, тем медленнее переписывающая машина найдет нужное правило.

В переписывающей машине, реализованной в **APS v3**, мы устранили этот недостаток, используя хеш-таблицы. Данное предложение является расширением предложения 2, описанного выше. Т.е. нужно не только группировать правила по главной операции их левой части, а и уметь быстро определять, к какой группе относится данных подтерм.

Более детально о различиях **APS v1,v2,v3** можно будет ознакомиться в следующих публикациях авторов.

3. Эксперимент с переписывающей машиной APS v3

Создадим систему переписывающих правил R_3, которая состоит из 5 правил (3- средняя глубина поиска), в левых частях которых термы имеют различную главную операцию:

```
R3:=rs(x,y)(
  x+y=1,
  x-y=2,
  x*y=3,
  x/y=4,
  x^y=5
);
```

Отметим тот факт, что средняя глубина поиска нужного правила в с.п.п. будет равна 3 (мы будем использовать этот факт для более сложных с.п.п.).

На следующем шаге вычислим разницу во времени k применений системы переписывающих правил к входящему терму: a+b, a^b (для первого и последнего правила) для переписывающих машин **APS v3** и **APS v2**(время применения для второго терма вычтем от времени применения для первого). Повторим эксперимент для R_4, R_5, R_9 (n={3,4,6,9}). Результаты экспериментов занесем в таблицу (Таблица 1).

Табл. 1. Скорость применения систем переписывающих правил

k, n	**APS v2**				**APS v3**			
	R_3	R_4	R_6	R_9	R_3	R_4	R_6	R_9
10	0	0	0	0,001	0	0	0	0
100	0,003	0,008	0,011	0,020	0,000	0,000	0,000	0,000
1000	0,047	0,063	0,108	0,219	0,002	0,001	0,007	0,009
10000	0,453	0,749	1,312	1,999	0,002	0,006	0,008	0,009
50000	2,438	3,703	6,306	10,266	0,009	0,041	0,042	0,017
100000	4,153	6,641	11,437	19,666	0,109	0,048	0,095	0,063
500000	23,173	42,956	76,254	103,439	0,006	0,109	0,735	0,219

Составим таблицу разностей между временем работы переписывающей машины **APS v3** и переписывающей машины **APS v2** (Таблица 2).

Табл. 2. Разность времени работы переписывающих машин

k,n	3	4	6	9
10	0	0	0	0,001
100	0,003	0,008	0,011	0,02
1000	0,045	0,062	0,101	0,21
10000	0,451	0,743	1,304	1,99
50000	2,429	3,662	6,264	10,249
100000	4,044	6,593	11,342	19,603
500000	23,167	42,847	75,519	103,22

Построим зависимость между временем работы, количеством применений каждой системы переписывающих правил и количеством различных главных операций в левых частях правил. (Рис. 1)

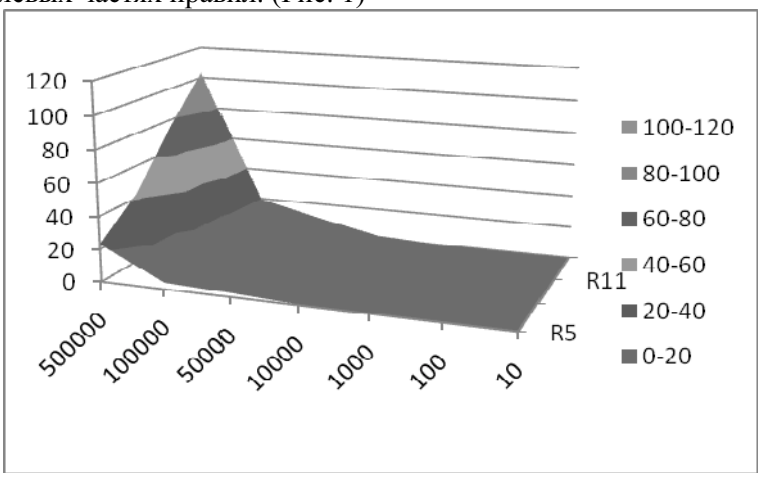

Рис. 1. Зависимость времени применения с.п.п.

Таким образом, мы подтвердили экспериментально тот факт, что использование переписывающей машины **APS v2** для систем переписывающих правил, в которых есть большое количество разных главных операция в левых частях правил, приводило к значительному замедлению работы программы при достаточно долгой работе системы.

Не сложно заметить, что рассматривая произведения k*n ,мы получим возрастающую последовательность значений (Таблица 3).

*Табл. 3. Разность времени работы с.п.п зависящая от k*n.*

k*n	Время	k*n	Время	k*n	Время	k*n	Время
30	0	$9*10^2$	0,02	$6*10^4$	1,304	$45*10^4$	10,249
40	0	$3*10^3$	0,045	$9*10^4$	1,990	$6*10^5$	11,342
60	0	$4*10^3$	0,062	$15*10^4$	2,429	$9*10^5$	19,603
90	0	$6*10^3$	0,101	$20*10^4$	3,662	$15*10^5$	23,167
$3*10^2$	0,003	$9*10^3$	0,210	$30*10^4$	4,044	$20*10^5$	42,847
$4*10^2$	0,004	$3*10^4$	0,451	$30*10^4$	6,264	$30*10^5$	75,519
$6*10^2$	0,011	$4*10^4$	0,743	$40*10^4$	6,593	$45*10^5$	103,220

Построим эти точки на плоскости, соединив их отрезками (рис. 2).

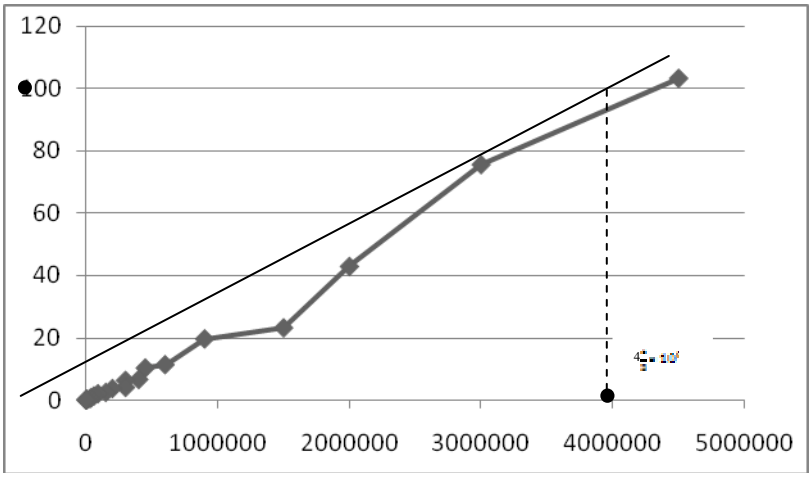

*Рис. 2. Зависимость времени от k*n.*

Если сложить все время работы новой и старой переписывающих машин для всех экспериментов соответственно, то мы получим, что новая переписывающая машина работала в 285,5 раза быстрей.

4. Анализ примера нахождения дизъюнктивной нормальной формы (ДНФ) из APS user manuale[2].

Алгоритм построения ДНФ в алгебре логики первого порядка, написанный на языке **APLAN** системы алгебраического программирования **APS** ,выглядит следующим образом:

```
dnf:=proc(x)(
      ntb(x,eliminate),
      can_ord(x,dnf_dn,dnf_up),
      return(x)
);
eliminate:=rs(x,y)(
       (x <=>  y) = ((x -> y) & (y -> x)),
       (x  ->  y) = (~(x) |/ y),
       ~( ~(x)  ) = x,
       ~(x |/  y) = (~(x) & ~(y)),
       ~(x  &  y) = (~(x) |/ ~(y)),
       ~(x <=> y) = (~(x -> y) |/ ~(y -> x)),
       ~(x  -> y) = (x & ~(y))
);
dnf_dn:=rs(x,y,z,u,v)(
   ( x |/ y) & (z |/ u) |/ v =   x & u |/  y & z  |/ y & u |/ x & z |/ v,
   ((x |/ y) & z) |/ u      =   y & z |/  x & z  |/ u,
    x & (y |/ z) |/ u       =   x & z |/  x & y  |/ u,
   (x |/ y) & (z  |/ u)     =  (x & z |/  y & z) |/ x & u |/ y & u ,
   (x |/ y) &  z            = ((x & z)|/ (y & z)),
    x & (y |/  z)           =   x & y |/  x & z
);
dnf_up:=rs(x,y,z)(
   (x |/ y) & z      = (x & z) |/ (y & z) ,
    x      &(y |/ z) = (x & y) |/ (x & z),
   (x  & y) & z      =  x & y  & z
);
```

Тут, *ntb* – стратегия, которая сверху вниз к каждому подтерму применяет с.п.п. до тех пор, пока она применима, *can_ord* – стратегия, которая сверху вниз применяет первую с.п.п., потом снизу вверх – вторую, если вторая с.п.п. была применима к текущему подтерму, то начинаем снова применять сверху вниз первую с.п.п, а снизу вверх вторую и т.д. Более детально с этим примером можно ознакомиться в [1].

Рассчитаем глубину поиска для каждого правила из с.п.п., которое стоит в середине **REM** языка переписывающей машины (*k*):

1. Для *eliminate* таким будет 2 правило, таким образом $R_{e\lim inate} = 2$.
2. Для *dnf_dn* таким будет 2 или 5 правило, поэтому $R_{dnf\_dn} = 2$.
3. Для *dnf_up* таким будет 1 или 3 правило, значит $R_{dnf\_up} = 1$.

Рассчитаем время, которое мы можем выиграть при использовании переписывающей машины *APS v3*, если на вход алгоритма построения ДНФ будет поступать терм:

$$\left(A_6 \leftrightarrow \left(A_6 \wedge A_5 \wedge \overline{A_3} \vee A_4 \wedge A_5 \wedge A_3 \vee \overline{A_2} \wedge \overline{A_5} \wedge \overline{A_3} \vee A_1 \wedge A_2 \wedge \overline{A_3} \vee A_1 \wedge \overline{A_2} \wedge A_3\right)\right) \wedge$$
$$\left(A_6 \leftrightarrow \left(A_6 \wedge A_5 \wedge \overline{A_3} \vee A_4 \wedge A_5 \wedge A_3 \vee \overline{A_2} \wedge \overline{A_5} \wedge \overline{A_3} \vee A_1 \wedge A_2 \wedge \overline{A_3} \vee A_1 \wedge \overline{A_2} \wedge A_3\right)\right) \wedge$$
$$\left(A_6 \leftrightarrow \left(A_6 \wedge A_5 \wedge \overline{A_3} \vee A_4 \wedge A_5 \wedge A_3 \vee \overline{A_2} \wedge \overline{A_5} \wedge \overline{A_3} \vee A_1 \wedge A_2 \wedge \overline{A_3} \vee A_1 \wedge \overline{A_2} \wedge A_3\right)\right) \wedge$$
$$\left(A_6 \leftrightarrow \left(A_6 \wedge A_5 \wedge \overline{A_3} \vee A_4 \wedge A_5 \wedge A_3 \vee \overline{A_2} \wedge \overline{A_5} \wedge \overline{A_3} \vee A_1 \wedge A_2 \wedge \overline{A_3} \vee A_1 \wedge \overline{A_2} \wedge A_3\right)\right) \wedge$$
$$\left(A_6 \leftrightarrow \left(A_6 \wedge A_5 \wedge \overline{A_3} \vee A_4 \wedge A_5 \wedge A_3 \vee \overline{A_2} \wedge \overline{A_5} \wedge \overline{A_3} \vee A_1 \wedge A_2 \wedge \overline{A_3} \vee A_1 \wedge \overline{A_2} \wedge A_3\right)\right) \wedge$$
$$\left(A_6 \leftrightarrow \left(A_6 \wedge A_5 \wedge \overline{A_3} \vee A_4 \wedge A_5 \wedge A_3 \vee \overline{A_2} \wedge \overline{A_5} \wedge \overline{A_3} \vee A_1 \wedge A_2 \wedge \overline{A_3} \vee A_1 \wedge \overline{A_2} \wedge A_3\right)\right) \wedge$$
$$\left(A_6 \leftrightarrow \left(A_6 \wedge A_5 \wedge \overline{A_3} \vee A_4 \wedge A_5 \wedge A_3 \vee \overline{A_2} \wedge \overline{A_5} \wedge \overline{A_3} \vee A_1 \wedge A_2 \wedge \overline{A_3} \vee A_1 \wedge \overline{A_2} \wedge A_3\right)\right) \wedge$$

Подсчитаем количество применения для каждой из с.п.п., написав специализированную стратегию *appls_i* и *can_ord_i* с подсчетом количества применений каждой с.п.п.

```
appls_i:=proc(t,R)(
     appls_count:=appls_count+1;
    applr(t,R);
    yes->(
        dowhile(
     appls_count:=appls_count+1;
            applr(t,R)
        )yes;
        yes:=1;
        return 1
    );
    return 1
);
can_ord_i:=proc(t,R1,R2)loc(s,i)(
    t:=can(t);
    appls_count:=0;
    appls_i(t,R1);
    R1_count:=R1_count+appls_count;
    for(i:=1,i<=ART(t),i:=i+1,
        can_ord_i(arg(t,i),R1,R2)
    );
    can_up_i(t,R2)
);
can_up_i:=proc(t,R)loc(s,i)(
    appls_i(t,R);
    while(yes,
        for(i:=1,i<=ART(t),i:=i+1,
            can_up(arg(t,i),R)
        );
          appls_count:=0;
```

```
        appls(t,R);
          R2_count:=R2_count+appls_count
      );
      t:=mrg(can(t))
  );
```

Итак, для данного терма стратегия *ntb*(t,eliminate) применит с.п.п. *eliminate* 509 раз, а стратегия *can_ord* применит с.п.п. *dnf_dn u cnf_up* 66497 и 332375 раз соответственно. Таким образом, разница работы во времени переписывающей машины **APS v2** и **APS v3** составляет:

$$k * n = 2 * 509 + 2 * 61955 + 1 * 264770 = 389968$$

Запустив **APS v2** и **APS v3** на этот пример, мы получим разницу во времени $t_{v2} - t_{v3} = 2{,}998\,сек$, а значит, ошибка вычислений составляет 5,860 сек.

Намного большее ускорение можно получить в **APS v3** по сравнению с **APS v2** при использовании функции *subs*[2], однако примеры с этой специализированной стратегией будут описаны в следующих публикациях авторов.

7. Выводы

Таким образом, полученная оптимизация переписывающей машины **APS v3** дает возможность намного эффективнее использовать большие с.п.п. в прототипах программ, что позволяет эффективно использовать системы переписывающих правил для хеширования произвольных термов и существенно расширяет области применения **APS**.

ЛИТЕРАТУРА

1. A. A. Letichevsky, J.V. Kapitonova, S.V. Kanozenko, Algebraic programming system APS-1. In: O.M.Tammepuu, Informatics'-89, Proc. of the Soviet-Franch symp. Tallin, 1989, p.46-55.
2. Letichevsky A.A., Kapitonova J.V., Konozenko S.V. Computations in APS, Theoretical Computer Science 119, 1993, p.145-171.
3. А.А. Летичевский, Ю.В.Капитонова и др. Сертификация систем с помощью базовых протоколов//Кибернетика и системный анализ, №4, 2005, p.256-268.
4. M. Lvov, V. Peschanenko The Program Environment of Support of the Practical Training in the Course of Mathematical Logic // Трансфер технологій: погляд через призсу якості: Третя Міжнародна конференція "Нові інформаційні технології в освіті для всіх: система електронної освіти", Київ, 1-3 жовтня 2008 р.- К.:Академперіодика, 2008.-С. 383-391.
5. Львов М.С., Львова Н.М. Алгебра з комп’ютером: Навч. посіб.–К.: Шкільний світ, 2008. –128 с.
6. A.A. Letichevsky, V.V. Khomenko A Rewriting Machine and Optimization of Strategies of Term Rewriting// Cybernetics and Systems Analysis, №5, 2002, p. 637-649.

Надійшла у першій редакції 12.03.2009, в останній - 07.04.2009.