Переписывающая машина и оптимизация стратегий переписывания термов

А.А.Летичевский, В.В.Хоменко

Институт Кибернетики им. В.М.Глушкова
Национальной Академии Наук Украины

` Аннотация

В статье рассматривается машина переписывания термов (REM), базовый алгоритм для поддержки алгебраического программирования. Алгоритм представлен в виде верифицированной программы на языке алгебраического программирования APLAN и использован в качестве спецификации для реализации переписывающей машины в новой версии системы APS. Такой подход позволил быстро получить достаточно надежную реализацию переписывающей машины и возможность дальнейших модификаций для реализации различных стратегий переписывания.

1 Введение

Парадигма алгебраического программирования в последнне время становится одной из важнейших парадигм декларативного программирования. Техника переписывания термов [1], на которой базируется алгебраическое программирование, широко используется как самостоятельно, так и в составе систем программирования, основанных на других парадигмах (функциональной, логической, в объектно-ориентированном и агентном программировании, в компьютерной алгебре и системах доказательств теорем). Среди систем алгебраического программирования, появившихся в последние годы, следует отметить Maude, Elan, Cafe-obj, APS [2, 3, 4, 5, 6, 7]. Теория систем переписывания детально разработана и представлена, например, в [8, 9].

Система алгебраического программирования APS была разработана в Институте Кибернетики им. В.М.Глушкова НАН Украины в начале 90-х годов. Система представляет собой профессионально-ориентированное инструментальное средство для разработки прикладных систем, основанное на алгебраических и логических моделях предметных областей. Основная используемая техника программирования — системы переписывающих правил. Язык алгебраического

программирования APLAN, используемый в системе, интегрирует основные парадигмы программирования и легко расширяется семантически с помощью метапрограммирования, основанного на переписывании.

Быстрое переписывание термов обусловлено использованием эффективного параллельного метчинга для реализации переписывающей машины. Соответствующая программа была реализована С.В.Конозенко на языке C в первой версии APS. В статье рассматривается новый вариант переписывающей машины, разработанный для новой версии системы APS. Машина представлена в языке APLAN, используемом в качестве языка исполнимых спецификаций. Такое представление позволяет построить доказательство корректности соответствующей программы, быстро разработать надежную реализацию на базовом языке программирования (C++) и получить новые модификации машины специализированные для различных стратегий переписывания.

2 Система APS

Главной особенностью APS является то обстоятельство, что системы переписывающих правил могут применяться с использованием различных стратегий переписывания. Этот подход позволяет рассматривать не только канонические (конфлюентные и нетеровы), но и любые другие системы переписывающих правил и алгебраические программы могут быть построены путем сочетания систем правил с различными стратегиями их применения.

Другой особенностью является возможность сочетания императивного и алгебраического методов программирования. Входным языком системы является APLAN. Этот язык позволяет описывать не только алгебраическую среду (компоненты, операции и тождества алгебры данных), но также писать процедуры и вызывать их из алгебраических программ (систем переписывающих правил). Удобные средства для представления процедур позволяют пользователю манипулировать как стандартными, так и разработанными им самим эффективными стратегиями для применения переписывающих правил. Стратегии переписывания могут использовать различные свойства алгебры данных, рассматривая алгебраические выражения (термы) с точностью до некоторых отношений конгруентности. Для этой цели используется механизм канонических (нормальных) форм алгебраических выражений. Этот механизм работает всегда, когда применяется некоторое переписывающее правило. Выбор канонических форм определяется требованиями предметной области и управляется пользователем. Более полное описание применения систем правил переписывания и вычислительных стратегий можно найти в [5, 6, 7].

3 Системы переписывающих правил.

В APS системы переписывающих правил имеют следующий синтаксис:

```
<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>
```

Стратегии переписывания в APS базируются на двух основных внутренних процедурах - `applr` и `appls`. Оператор `applr`(t, R) пытается применить одно из правил системы R к терму t. Если применимых правил нет, то переменная `yes` получает значение 0; в противном случае применяется первое применимое правило и переменная `yes` устанавливается в 1. Применение простого правила обычное: терм сопоставляется с левой частью правила, и если сопоставление успешно, то происходит замена терма правой частью правила и подстановка переменных. Затем полученный терм приводится к канонической форме с помощью правил, подобных правилам вычисления значения терма, но без подстановки значений имен. Применение условных правил осуществляется следующим образом: сначала осуществляется сопоставление (matching). Если оно успешно, то условие приводится к базовой канонической форме путем вычисления функции `CAN`. Если в результате получается 1, то применение правила продолжается обычным образом, в противном случае — отменяется.

В новой версии APS семантика применения условных правил переписывания несколько изменена: теперь при вычислении условия переменные мэтчинга защищаются от приведения к канонической форме. Как показала практика, такая семантика более удобна, а согласования со старой семантикой можно добиться явным вызовом функции `CAN`.

4 Стратегии переписывания.

Стратегия может быть определена как множество допустимых историй переписывания для заданного множества правил. Для того, чтобы стратегия переписывания была конструктивной, она должна быть определена с помощью алгоритма, который распознает или порождает допустимые истории переписывания. Возможность использования широкого спектра различных видов стратегий делает язык алгебраического программирования гибким, удобным и выразительным. Большой интерес для практического применения имеют локальные вычислительные стратегии, которые могут быть определены с помощью ответов на следующие вопросы:

- В каком порядке подвыражения данного выражения обозреваются и проверяются на применимость правил?
- В каком порядке обозреваются правила?

- Как продолжается процесс после успешного завершения предыдущего шага переписывания?
- Каковы условия окончания переписывания?

На первый вопрос имеются два основных ответа: поиск сверху-вниз (самое внешнее вхождение) и поиск снизу-вверх (самое внутреннее вхождение). Эти ответы соответствуют вызову по имени и вызову по значению в функциональном программировании.

Простейшим ответом на второй вопрос является просмотр соотношений в соответствии с некоторым порядком; применение следующего соотношения находится в зависимости от предыдущего и от того, можно ли его применить вообще. После успешного применения некоторого соотношения можно применять его к тому же самому подвыражению до тех пор, пока это возможно, а затем продолжать поиск в том же самом или противоположном направлении. Другая возможность — начинать с самого начала всякий раз, когда некоторое правило может быть применено (как в алгоритмах Маркова).

После завершения полного обхода выражения переписывающий процесс может быть остановлен (однократное применение) или продолжен в том же самом или противоположном направлении (многократное применение). Выражение называется нормализованным, если к его подвыражениям не применимо ни одно из правил. Стратегия называется нормализующей, если она останавливается только при достижении нормализованного выражения. Широкий класс локальных стратегий может быть определен заданием рекурсивных программ в языке APLAN системы APS. Вот некоторые из основных стратегий переписывания этой системы. Все они в конечном счете сводятся к повторным применениям в различных контекстах базовой стратегии applr.

```
appls:=proc(t, p)loc(Yes)(
    applr(t, p);
    Yes:=yes;
    while(yes,
        applr(t, p)
    );
    yes: = Yes
);

ntb:=proc(t, R)loc(s, i)(
    appls(t, R);
    forall(s=arg(t, i),
       ntb (s, R)
    );
    t:=can (t)
);
```

```
nbt:=proc(t, R)loc (s, i)(
   forall(s=arg (t, i),
       nbt (s, R)
   );
   appls(t, R);
   t:=can(t)
);

can_ord:=proc(t, R1, R2)loc(s, i)(
    t:= can(t);
    appls(t, R1);
    forall(s=arg(t, i),
         can_ord(s, R1, R2)
    );    can_up(t, R2)
);

can_up:=proc(t, R)loc(s, i)(
    appls(t, R);
    while(yes,
       forall(s=arg(t, i),
           can_up(s, R)
       );
       appls (t, R)
    );
    t:=can (t);
    merge (t)
);
```

Процедура `appls` применяет систему R к терму t, пока это возможно, и выдает результат `yes` $= 1$, если применение успешно; `nbt` — однократная снизу вверх стратегия, `ntb` — однократная сверху вниз стратегия. Стратегия `can_ord`(t, R, S) обходит дерево, представляющее терм, сверху вниз и слева направо. При попадании в вершину первый раз сверху, она применяет систему R, после полного обхода всех дочерних вершин и возвращении в данную вершину, применяется система S. Если на некотором шаге применение S было успешным, то S применяется ко всем дочерним вершинам повторно. После полной обработки каждой вершины применяется упорядочение относительно ассоциативно-коммутативных операций (функция `merge` осуществляет слияние уже упорядоченных термов).

5 Переписывающая машина.

Переписывающая машина (REM, rewriting machine) реализует стратегию переписывания `applr`. Эта стратегия является базовой и через нее определяются

все остальные стратегии. Поэтому эффективность ее реализации имеет первостепенное значение. Подставляя стратегию `applr` в другие стратегии и выполняя подходящие преобразования, можно получить специализированные переписывающие машины для этих стратегий.

Эффективная реализация стратегии `applr` использует следующие соображения:

- Если левые части нескольких соседних правил идентичны, то мэтчинг можно осуществлять только один раз (в некоторых случаях можно использовать этот вид оптимизации и для не подряд идущих правил; условия, при которых два правила можно поменять местами, будут рассмотрены далее).

- 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.

5.1 Синтаксис языка REM.

Синтаксис языка переписывающей машины выражается следующей параметрической грамматикой:

```
<замкнутая REM-программа>::=
     <заголовок программы><программа ранга 1>
<заголовок программы>::=Rs array(<список пропусков>)
<список пропусков>::= _ | _ , <список пропусков>
<REM-программа>::=<программа ранга k>
<программа ранга k>::=<защищенная программа ранга k> |
     <защищенная программа ранга k> + <программа ранга k>
<защищенная программа ранга 0>::= rewrite(<term>) |
     If(<term>,rewrite(<term>))
<защищенная программа ранга n>::=match(<терм>)
     <программа ранга n-1> |
     test(<m-тип>)<программа ранга n+m-1>
```

Здесь m, n — положительные целые числа, k — неотрицательное целое, <терм> — терм (алгебраическое выражение) языка APLAN, $<$ m-тип$>$ — терм вида $\omega((), \ldots, ())$, где ω — отметка арности m. Термы вида `var`(i), где i — положительные целое, не превышающее количества пропусков в заголовке программы, и только они, рассматриваются как переменные замкнутой

программы. Программы ранга k представляют собой части полностью сформированной программы, которая имеет ранг 1. Ранги соответствующих частей должны быть сбалансированы подобно тому, как балансируются части терма в бесскобочной записи. Ниже будет показано, что каждая замкнутая REM-программа однозначно определяет некоторую систему переписывающих правил языка APLAN (с точностью до переименования переменных).

5.2 Алгебра REM-программ.

Пусть $T_\Omega(Z)$ есть базовая алгебра термов некоторой алгебраической программы. Множество Ω есть сигнатура операций этой алгебры, а Z — порождающее множество термов арности 0. REM-программы образуют многосортную алгебру $R = (R_k)_{k\in N}$ над $T_\Omega(Z \cup V)$, где V — множество переменных этих программ. Здесь R_k — множество программ ранга k. Операциями этой алгебры являются следующие:

1. $+ : R_k \times R_k \to R_k,\ k > 0$;
2. $\mathtt{test}(\omega((), \ldots, ())) : R_{n+m-1} \to R_n, m > 0, n > 0, \omega \in \Omega, m = \mathtt{ART}(\omega)$;
3. $\mathtt{match}(t) : R_n \to R_{n+1}, n \geq 0, t \in T_\Omega(Z \cup V)$;
4. $\mathtt{rewrite}(t) \in R_0, t \in T_\Omega(Z \cup V)$;
5. $\mathtt{If}(u, \mathtt{rewrite}(t)) \in R_0, u, t \in T_\Omega(Z \cup V)$.

Операции вида 4) и 5) имеют нулевую арность и являются порождающими элементами алгебры REM-программ. Как следует из определений, всякая программа ранга k может быть представлена в виде выражения алгебры REM-программ.

Определение 5.1 *Назовем REM-программу* **match**-*программой, если в ней не используется операция* $\mathtt{test}(\omega((), \ldots, ()))$, *и* **элементарной**, *если в ней не используется операция* $+$.

Элементарные match-программы ранга 1 имеют вид

$$\mathtt{match}\ t\ \mathtt{rewrite}\ t'$$

или

$$\mathtt{match}\ t\ \mathtt{If}(s, \mathtt{rewrite}\ t'),$$

Пусть $P = \mathtt{Rs}\ S_m\ Q$ есть замкнутая REM-программа, где S_m есть массив из m пропусков, а Q есть элементарная match-программа ранга 1 и $V = (\mathtt{var}(1), \ldots, \mathtt{var}(m))$. Тогда в первом случае программа представляет систему $\mathtt{rs}\ V(t = t')$, а во втором - систему $\mathtt{rs}\ V(s- > (t = t'))$. Произвольная match-программа ранга 1 есть "сумма" элементарных программ. Если Q

есть произвольная match-программ ранга 1, то P должна быть эквивалентной системе переписывающих правил соответствующих элементарным программам, составляющим программу Q. Принимая такое соответствие, произвольную систему переписывающих правил можно легко транслировать в REM-программу с помощью следующего простого транслятора (перевод заголовков предполагается уже сделанным):

```
equ2match:=rs(u, s, t, p, q, r)(
   (s = t) = match(s)rewrite(t),
   (u ->(s = t)) = match(s)If(u, rewrite(t)),
   (p, q) = equ2match p + equ2match q
);
```

Выражения

$$\texttt{match}(t)$$

$$\texttt{rewrite}(t)$$

$$\texttt{If}(u, \texttt{rewrite}(t))$$

$$\texttt{test}(\omega((), \ldots, ()))$$

образуют систему команд переписывающей машины. Смысл этих команд состоит в следующем. Команда `match` сопоставляет входной терм с образцом t, команда `rewrite` осуществляет переписывание, а команда `If` условное переписывание входного терма в новый терм t. Команда `test` проверят, является ли главная операция входного терма операцией ω.

5.3 Эквивалентность REM-программ.

Рассмотрим следующую систему тождеств на множестве REM-программ:

$$\begin{aligned}
(p+q)+r &= p+(q+r) && (1)\\
\texttt{match}(t)(p+q) &= \texttt{match}(t)p+\texttt{match}(t)q && (2)\\
\texttt{test}(t)(p+q) &= \texttt{test}(t)p+\texttt{test}(t)q && (3)\\
\texttt{test}(\omega((), \ldots, ()))\texttt{match}(t_1)\ldots\texttt{match}(t_m)p &= \texttt{match}(\omega(t_1, \ldots, t_m))p, && (4)
\end{aligned}$$

где p, q, r в каждом из тождеств представляют сбой произвольные программы одного и того же сорта, $m = \texttt{ART}(\omega(\ldots))$. Тождества (1-4) определяют отношение конгруентности $\sim$ на множестве REM-программ, которое мы будем называть синтаксической эквивалентностью. Если тождества (1-4) рассматривать как систему переписывающих правил, то можно заметить, что она обладает следующими важными свойствами:

- Все правила системы являются леволинейными, и отсутствуют критические пары, т.е. система является регулярной, и, следовательно, конфлюэнтной.

- Система правил переписывания является нетеровой.

Из этих свойств и теоремы Кнута-Бендикса непосредственно следует существование и единственность канонической формы относительно этих тождеств для любой REM-программы. Теперь мы в состоянии доказать следующий результат:

Теорема 5.1 *Пусть p — некоторая REM-программа ранга k, и p' — ее каноническая форма. Тогда p' имеет вид $\sum_{i=1}^{n} p_i$, где p_i — элементарные* `match`*-программы ранга k, $i = 1, \ldots, n$.*

Для доказательства теоремы замечаем, что любую REM-программу можно преобразовать с помощью равенств (1) — (3) в сумму элементарных программ. Затем с помощью соотношения (4) из каждой элементарной программы можно устранить все вхождения операции `test`, отслеживая сохранение ранга. Алгоритм приведения REM-программы к канонической форме, записанный на языке APLAN, имеет следующий вид:

```
NAMES t2m_up, test2match;
test2match:=rs(t, p, q)(
   match(t)p = t2m_up match(t) test2match p,
   test(t)p = t2m_up test(t) test2match p,
   p + q = t2m_up (test2match p + test2match q)
);NAME Starg;
t2m_up:=rs(t, p, q, r)(
   (p+q)+r         = t2m_up p+t2m_up(q + r),
   match(t)(p + q) = match(t)p+t2m_up match(t)q,
   test(t) (p + q) = Starg(ART(t), 1, t, p)+t2m_up test(t)q,
   test(t)  p      = Starg(ART(t), 1, t, p)
);

Starg:=proc(x)(
    appls(x,Starg_rs);
    return x
);
Starg_rs:= rs(m, i, t, p, q)(
   (m, m, t, match(p)q) = match(starg(t, m, p))q,
   (m, i, t, match(p)q) = (m, i + 1, starg(t, i, p), q)
);
```

Функция `starg`(t, i, p) устанавливает терм p i-ым аргументом главной операции терма t.

5.4 Алгоритм REM

Как следует из теоремы 5.1 каждой замкнутой REM-программе можно поставить в соответствие систему переписывающих правил. Отсюда вытекает формальное требование к алгоритму работы переписывающей машины. Этот алгоритм должен вычислять функцию двух аргументов. Первым аргументом является терм, который должен быть переписан. Вторым аргументом является программа переписывания (REM-программа). Результатом должно быть переписывание исходного терма путем применения к нему соответствующей системы переписывающих правил. Иными словами, требуется написать процедуру `napplr`(t, p), которая преобразует терм t так же, как и `applr`(t, R), где R есть система переписывания, соответствующая программе p. Верхний уровень программы `napplr` имеет следующий вид.

```
NAME napplr;NAME appl;
napplr:=proc(t,p)loc(pr,Yes,s)(
         let(p, Rs pr p);
         s->(p,t Nil,pr)+Nil;
         appls(s, appl);
         let(s,1:s);
         yes-> t:= s
);
```

Основная часть представлена системой переписывающих правил `appl`, которая итеративно применяется к состоянию переписывающей машины s. Начальное состояние содержит программу p ранга 1, исходный терм t аппликированный с константой `Nil` и массив pr, состоящий из пропусков. Это есть предусловие, определяющее требования к оператору `appls`$(s,$ `appl`$)$. Постусловие состоит в следующем. Если система R, соответствующая программе p, применима к терму t, то после остановки $s = (1 : R(t))$. В противном случае, после остановки $s = 0$. Рассмотрим систему правил appl.

```
NAME perform;
appl:=rs(p,q,r,t,pr)(
         (1:     t    ) + r = (1:t),
         (p + q, t, pr) + r = (p,t,new(pr)) + (q,t,pr) + r,
         (p,     t, pr) + r = perform(p,t,pr)+ r
);
NAMES vsc, do_match, check_match, concline;
perform:=rs(p,q,s,t,pr)(
         (rewrite(q),  t,pr) = (1:vsc(q,pr)),
         (match(p)q, s t,pr) = check_match(q,t,do_match(p,s,pr)),
         (test(p)q,  s t,pr) = is_type(p,s)&(q,concline(s,t),pr),
         (If(p,q),     t,pr) = (vsc(p,pr)==1)&(q,t,pr)
);
```

```
check_match:=rs(p,t)(
    (p, t, 0) = 0
);
```

Алгебраическая программа `appl` использует следующие вспомагательные функции.

- Функция `do_match`(p, s, pr). Аргументы: p есть терм, который может зависеть от переменных (`var`$(1), \ldots,$ `var`$(\mathtt{m})$), s есть константный терм, pr есть m-мерный массив уже определенных значений переменных. Если значение переменной не определено, на соответствующем месте стоит пропуск. Через $Sub(p, pr)$ обозначим результат подстановки уже определенных значений переменных в p. Осуществляется мэтчинг, терм s сопоставляется с образцом $Sub(p, pr)$. В случае успеха возвращается новое (дополненное) значение pr, в случае неудачи возвращается 0.

- Функция `is_type`(p, s) определяет совпадают ли главные операции термов p и s.

- Процедура `vsc`(p, pr) осуществляет приведение терма p к базовой канонической форме с помощью функции `can` вместе с подстановкой в него значений переменных из массива pr. Это есть заключительная часть переписывания и она должна быть реализована в соответствии с требованиями, сформулированными в [7]. В частности, необходимо учитывать семантику аппликации и операции ‘(), которая отменяет применение канонизации.

- Функция `concline`$(\omega(t_1, \ldots, t_m), t)$ возвращает терм $(t_1 t_2 \ldots t_m t)$.

Для доказательства корректности оператора `appls`$(s,$ `appl`$)$, где

$$s = (p, t\ \mathtt{Nil}, pr)$$

удовлетворяет предусловию для начального состояния, воспользуемся теоремой 5.1 и проведем доказательство индукцией по числу шагов переписывания, которое необходимо сделать для приведения программы p к канонической форме.

Базис индукции легко проверяется. Пусть $p \sim p'$, преобразование p в p' происходит за один шаг и для p' корректность доказана. Рассматривая разные случаи правил переписывания, а также случаи равенства нулю результата вычисления функции `do_match`, получаем, что через конечное число шагов результат переписывания состояния $s = (p, t\,\mathtt{Nil}, pr)$ системой `appl` совпадает с результатом переписывания состояния $s = (p', t\ \mathtt{Nil}, pr)$, для которого справедливо предположение индукции. Замечая, что из этого утверждения получается также завершимость процесса переписывания, получаем доказательство полной корректности оператора `appls`$(s,$ `appl`$)$.

Приведем без обоснования реализацию в языке APLAN функций, которые используются в алгоритме переписывающей машины.

```
NAME Pr;
Pr:=Nil;
vsc:=proc(t,pr)loc(opr)(
         opr:=Pr;
         Pr->new(pr);
         t->vsc_rec(t);
         Pr->opr;
         return t
);
vsc_rec:=proc(t)loc(i,j,r,s,pr,p)(
         let(t,var(i));
         yes->return Pr(i);
         is_type(t, ‘(‘(Nil)) )->(
                  t->vs arg(t,1);
                  return(t)
         );
         let(t,p s);
         yes->(
                  p->is_rs(p);
                  ~(equ(p,0))->(
                                    t->vsc_rec(s);
                                    napplr(t,p);
                                    return(t)
                  )
         );
         t->new(t);
         for(i:=1,i<=ART(t),i:=i+1,
                  s->arg(t,i);
                  let(s,var(j));
                  yes->arg(t,i)->Pr(j)
                  else arg(t,i)->vsc_rec(s)
         );
         t->can(t);
         return(t)
);
vs:=proc(p)loc(i,s)(
          let(p,var(i));
          yes->return Pr(i);
          p->new(p);
          for(i:=1,i<=ART(p),i:=i+1,
                         s->arg(p,i);
                         is_var(s)->
                              arg(p,i)->arg(Pr,arg(s,2))
                         else arg(p,i)->vs(s)
```

```
          );
          return(p)
);
is_rs:=proc(t)(
         let(t,Rs _);
         yes->return t;
         isname(t)->return is_rs vl(t);
         return 0
);
do_match:=proc(l,t,pr)loc(opr)(
         opr:=Pr;
         Pr->new(pr);
         match_rec(l,t)->(
                  pr->Pr;
                  Pr->opr
         )else(
                  pr->0;
                  Pr->opr
         );
         return pr
);
match_rec:=proc(l,t)loc(i,npr)(
         let(l,var(i));
         yes->(
                  equ(Pr(i), _ )->(
                                Pr(i)->t;
                                return 1
                  );
                  return equ(Pr(i),t)
         );
         is_type(t,l)->(
                  for(i:=1,i<=ART(l),i:=i+1,
                                npr->match_rec(arg(l,i),arg(t,i));
                                equ(npr,0)->return 0
                  );
                  return 1
         );
         return 0
);
NAME Con;
concline:=proc(s,p)loc(m,i)(
         m:=ART(s);
         for(i:=m,i>0,i:=i-1,
                  p->Con(arg(s,i),p)
```

```
        );
        return p
);
Con:=rs(x,y)(
        (x,y) = `(x y)
);
```

6 Оптимизация REM-программ.

Одна и та же система правил переписывания может быть представлена многими различными способами в виде REM-программы. Поэтому возникает задача улучшения кода, т.е. построения REM-программы, имеющей по возможности меньшее время выполнения.

Различные программы имеют различное время выполнения, которое складывается из времен выполнения команд. При этом команда `test`(...) имеет ограниченную временную сложность, а команда `match`(t) имеет сложность пропорциональную объему терма t. К сожалению, каноническая форма REM-программы часто не является оптимальной в смысле времени выполнения. Для примера рассмотрим систему правил переписывания реализующую быстрое возведение в степень:

```
pw:=rs(x,n)(
    x^0=1,
    x^1=x,
    x^2=x*x,
(isnum(n)&((n mod 2)==0))->(
    x^n=pw0(pw0(x^(n/2))^2)
),
isnum(n)->(
    x^n=x*pw0(x^(n-1))
)
);
```

После трансляции с помощью `aplan2rem` получим следующую REM-программу:

```
pw:=Rs array(_,_)(
    match(var(1)^0)rewrite(1)
   +match(var(1)^1)rewrite(var(1))
   +match(var(1)^2)rewrite(var(1)*var(1))
   +match(var(1)^var(2))
      If(isint(var(2))&(var(2) mod 2 == 0),
        rewrite(pw(pw(var(1)^(var(2)/2))^2)))
   +match(var(1)^var(2))
      If(isint(var(2)),
```

```
        rewrite(var(1)*pw(var(1)^(var(2)-1))))
);
```

Нетрудно видеть, что эта программа находится в канонической форме (REM-программы, получающиеся в результате работы `aplan2rem`, всегда находятся в канонической форме!). Она выполняет последовательный просмотр (до первого применения) всех правил системы и является неэффективной. Оптимизированный вариант этой программы выглядит следующим образом:

```
pw:=Rs array(_,_)(
  test(NIL ^ NIL)
    match(var(1))(
      match(0)rewrite(1)
     +match(1)rewrite(var(1))
     +match(2)rewrite(var(1)*var(1))
     +match(var(2))(
        If(isnum(var(2))&(var(2) mod 2 == 0),
             rewrite(pw0(pw0(var(1)^(var(2)/2))^2)))
       +If(isnum var(2),
             rewrite(var(1)*pw0(var(1)^(var(2)-1))))
      )
    )
);
```

Для получения этой программы был использован оптимизатор, реализованный в виде следующей APLAN-программы:

```
NAMES split_atom,split_type,split_arg,m2t;
match2test:=rs(p,q,i,s)(
    match(var(i))p + q = m2t split_atom (match(var(i))p + q),
(ART(‘(s))==0)->(
    match(s)p + q = m2t split_atom (match(s)p + q)
),
    match(s)p + q = m2t split_type (match(s)p + q)
);

m2t:=rs(s,p,q)(
    p+q = m2t p + match2test q,
    match(s)p = match(s)match2test p,
    test (s)p = test (s)match2test p
);

split_atom:=rs(p,q,r,s)(
  match(s)p + match(s)q + r =
    split_atom(match(s)turn_right(p + q) + r),
```

```
  match(s)p + match(s)q =
    match(s)turn_right(p + q)
);

NAME turn_back;
split_type:=rs(p,q,r,s,t,u,v,i)(
/* match */
  match(s)p + match(s)q + r =
    split_type(match(s)turn_right(p+q) + r),
  match(s)p + match(s)q =
    split_type(match(s)turn_right(p+q)),

  match(s)p + match(var(i))q + r =
    match(s)p + match(var(i))q + r,
  match(s)p + match(var(i))q =
    match(s)p + match(var(i))q,

is_type(‘(t),‘(s))->(
  match(s)p + match(t)q + r =
    split_type(test(type(s))split_arg(s,p)+
    match(t)q + r)
),
is_type(‘(t),‘(s))->(
  match(s)p + match(t)q =
    split_type(test(type(s))split_arg(s,p)+
    match(t)q)
),

  match(s)p + match(t)q + r =
    turn_back(match(t)q+split_type(match(s)p + r)),

/* test */
  test(s)p + match(var(i))q + r =
    test(s)p + match(var(i))q + r,
  test(s)p + match(var(i))q =
    test(s)p + match(var(i))q,
is_type(‘(t),‘(s))->(
  test(s)p + match(t)q + r =
    split_type(test(s)turn_right(p+
    split_arg(t,q))+r)
),
is_type(‘(t),‘(s))->(
  test(s)p + match(t)q =
    test(s)turn_right(p+split_arg (t,q))
```

```
),

  test(s)p + match(t)q + r =
    turn_back(match(t)q + split_type(test(s)p + r))
);

turn_back:=rs(t,p,q,r,s)(
  match(t)p+test(s)q+r = test(s)q+match(t)p+r,
  match(t)p+test(s)q = test(s)q+match(t)p,
  match(t)p+match(s)q+r = match(s)q+match(t)p+r,
  match(t)p+match(s)q = match(s)q+match(t)p
);
```

Функция `split_arg`$(f(t_1,\ldots,t_m),p)$ возвращает $\texttt{match}(t_1)\ldots\texttt{match}(t_m)p$.

Для обоснования корректности оптимизатора нам понадобится расширить отношение синтаксической эквивалентности $\sim$, определенное в разделе 5.3, следующими дополнительными тождествами.

- Операция + ассоциативна:

$$(p1+p2)+p3 \sim p1+(p2+p3) \tag{5}$$

- $\texttt{match}(t)$ и $\texttt{test}(t)$ — аддитивные операции:

$$\begin{aligned} \texttt{match}(s)p+\texttt{match}(s)q &\sim \texttt{match}(s)(p+q) \\ \texttt{test}(s)p+\texttt{test}(s)q &\sim \texttt{test}(s)(p+q) \end{aligned} \tag{6}$$

- Правило разложения:

$$\texttt{match}(\omega(t_1,\ldots,t_m))p \sim \texttt{test}(\omega((),\ldots,()))\texttt{match}(t_1)\ldots\texttt{match}(t_m)p, \tag{7}$$

  где ω — m-арная операция алгебры $T_\Omega(Z)$.

- Правило коммутативности: если t и s неунифицируемы, то

$$\begin{aligned} \texttt{match}(t)p+\texttt{match}(s)q &\sim \texttt{match}(s)q+\texttt{match}(t)p \\ \texttt{match}(t)p+\texttt{test}(s)q &\sim \texttt{test}(s)q+\texttt{match}(t)p \\ \texttt{test}(t)p+\texttt{match}(s)q &\sim \texttt{match}(s)q+\texttt{test}(t)p \\ \texttt{test}(t)p+\texttt{test}(s)q &\sim \texttt{test}(s)q+\texttt{test}(t)p \end{aligned} \tag{8}$$

Примечание 6.1 *Из ассоциативности + следует, что*

$$p1+p2 \sim \texttt{turn\_right}(p1+p2),$$

где функция `turn_right` *(в данном случае) выравнивает скобки для верхних сумм вправо.*

Примечание 6.2 *Соотношения (6) и (7)совпадают с правилами синтаксической эквивалентности. Из них вытекает следующее утверждение.*

Следствие 6.1 *Если* $\texttt{ART}(t) \neq 0$, *то*

$$\texttt{match}(t)p \sim \texttt{test}(\texttt{type}(t))\texttt{split\_arg}(t, p).$$

Примечание 6.3 *В последних трех случаях правила коммутативности проверка унифицируемости просто сводится к вызову функции* `is_type`, *которая проверяет равенство главных отметок своих аргументов.*

Теперь для обоснования правильности оптимизатора достаточно показать, что результирующая программа синтаксически эквивалентна входной. Это делается простой проверкой того, что каждое переписывающее правило сохраняет синтаксическую эквивалентность. При этом нужно учитывать, что:

- аппликация в правой части будет интерпретироваться только когда ее первый аргумент есть имя системы правил переписывания;
- операция + нигде не интерпретируется.

Доказательство конечности алгоритма проводится индукцией по построению терма REM-программы и не представляет затруднений.

7 Оптимизация стратегии `appls`.

Стратегия `appls` представляет собой итерацию стратегии `applr` и может быть определена следующей процедурой:

```
appls:= proc(t, p)loc (Yes)(
    applr (t, p);
    Yes:= yes;
    while(yes,
        applr(t, p)
    );
    yes: = Yes
);
```

Описанная выше оптимизация `applr` автоматически повышает эффективность `appls`, но остаются и дополнительные возможности оптимизации. Действительно, после каждой итерации остается некоторая информация о структуре обрабатываемого терма, которую можно использовать на последующих итерациях.

Пример 7.1 *Моделирование работы конечного автомата, распознающего запись натурального числа.*

Система переписывающих правил имеет вид:

```
AUT:=rs(q,h,t,x)(
  (q0, "+", t) = (q1, t),
  (q0, "-", t) = (q1, t),
isnum(h)->
  ((q0, h, t)  = (q2, t)),
isnum(h)->
  ((q0, h)    = 1),
isnum(h)->
  ((q1, h, t)  = (q2, t)),
isnum(h)->
  ((q1, h)    = 1),
isnum(h)->
  ((q2, h, t)  = (q2, t)),
isnum(h)->
  ((q2, h)    = 1),
  (q, x)      = 0
);
```

Если на некоторой итерации применилось седьмое правило, то на следующей итерации первые шесть правил заведомо не смогут примениться. Это наблюдение можно обобщить следующим образом: если на некоторой итерации применилось правило r, то на следующей итерации могут примениться лишь те правила, левые части которых унифицируются (в конечных термах) с правой частью r.

Если правая часть r содержит интерпретированные операции, то, в общем случае, вид результирующего терма может отличаться от ожидаемого. Поэтому перед унификацией все подтермы в правой части r, имеющие вид $f(t_1, \ldots, t_n)$, где f — интерпретированная операция, следует заменять на новую (свободную) переменную унификации.

Идея оптимизации состоит в следующем:

1. система правил переписывания модифицируется таким образом, чтоб после применения правила возвращался не только результирующий терм, но и номер этого правила;

2. строятся системы R_i, $i = 1, \ldots, n$, которые состоят в точности из правил, которые могут примениться после применения i-го правила;

3. после каждой итерации определяется номер i применившегося правила, и на следующей итерации применяется система R_i.

Так, для приведенной выше системы, моделирующей работу конечного автомата, совокупность систем будет следующей:

```
/* R1, R2 */
rs(q,h,t,x)(
  (q1,h,t)   = (5,(q2,t)),
isnum(h)->(
  (q1,h)     = (6,1)
),
  (q,x)      = (9,0)
);

/* R3, R5, R7 */
rs(q,h,t,x) (
isnum(h)->
  (q2,h,t)   = (7,(q2,t)),
isnum(h)->(
  (q2,h)     = (8,1)
),
  (q,x)      = (9,0)
);

/* R4, R6, R8, R9 */
rs(q,h,t,x) ();
```

Теперь процедура `appls` должна сначала применить полную систему правил переписывания, а затем, в зависимости от номера последнего применившегося правила, сокращенные системы.

Примечание 7.1 *Многие базовые стратегии (*`ntb`, `nbt`, `can_ord` *и др.) используют* `appls`*, поэтому скорость их работы тоже увеличится.*

Аналогичную оптимизацию можно провести для стратегии `ntb`. Только унификацию необходимо производить не с правыми частями, а с их подтермами.

8 Заключительные замечания.

Возможности оптимизации в системах, основанных на переписывающих правилах, далеко не исчерпываются подходом, рассмотренным в работе. Более глубокая оптимизация может быть проведена с использованием идеи смешанных вычислений, как это сделано в работе [10].

`Список литературы

[1] Bergstra J.A., Hearing J., and Klint P., editors. *Algebraic Specification.* ACM Press and Addison-Wesley, Reading, MA, 1989.

[2] P. Lincoln M. Clavel, S. Eker and J. Meseguer. Principles of maude. In *J. Meseguer, editor, Proceedings of the 1rst international workshop on rewriting logic, volume 4*, Asilomar (California), September 1996. Electronic Notes in Theoretical Computer Science.

[3] H. Kirchner P.-E. Moreau P. Borovanskiy, C. Kirchner and M. Vittek. Elan : A logical framework based on computational systems. In *J. Meseguer, editor, Proceedings of the 1rst international workshop on rewriting logic, volume 4*, Asilomar (California), September 1996. Electronic Notes in Theoretical Computer Science.

[4] K. Futatsugi and T. Sawada. Cafe as an extensible specification environment. In *Proc. of the Kunming International CASE Symposium*, Kunming, China, November, 1994.

[5] Letichevsky A.A. and Kapitonova J.V. Algebraic programming in APS system. In *Proc. ISSAC '90 Tokyo, Japan*, pages 68–75, New York, 1990. ACM.

[6] Letichevsky A.A., Kapitonova J.V., and Konozenko S.V. Optimization of algebraic programs. In *Proc. ISSAC '91 Bonn, Germany*, pages 370–376, New York, 1991. ACM.

[7] Letichevsky A.A., Kapitonova J.V., and Konozenko S.V. Computations in APS. *Theoretical Computer Science*, (119):145–171, 1993.

[8] Jouannaud J.-P. and Kaplan S., editors. *Proc. Intern. Workshop on Conditional Term Rewriting Systems*, Berlin, 1988. Springer.

[9] Lescanne P., editor. *Rewriting Techniques and Applications, Lecture Notes in Computer Science*, volume 256, Berlin, 1987. Springer.

[10] А. А. Летичевский. Смешанные вычисления и оптимизация программ. *Программирование*, (1):69–76, 1990.