УДК 004.41, 681.142.2

А.А. Летичевский, А.А. Летичевский (мл.), В.С. Песчаненко

# Алгоритм трансляции *APLAN* кода

Рассмотрен алгоритм трансляции *APLAN*-кода в системе алгебраического программирования *APS* с использованием методов инсерционного моделирования.

An algorithm of translation of the APLAN-code in the APS algebraic programming system with the help of the methods of insertion modelling is considered.

Розглянуто алгоритм трансляції *APLAN*-коду в системі алгебраїчного програмування *APS* з використанням засобів інсерційного моделювання.

**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, отделившая понятия стратегий от систем правил переписывания.

*APS* – система для прототипирования алгоритмов. Ее вторая и третья версии [2] успешно применяются в ряде коммерческих продуктов: *VRS* (*Verification for Requirement Specification*) [3], *MathLog* (*Mathematic Logic for Universities*) [4], *TerM* [5], что позволяет авторам *APS* позиционировать ее как коммерческую систему. Такое использование системы *APS*, конечно, невозможно без некоторого промежуточного языкового уровня, связывающего язык реализации системы *C++* с языком самой системы *APLAN*. Такой язык называется *APLANC* [6].

Разработка различного программного обеспечения (ПО) с использованием системы алгебраического программирования *APS* проходит следующие этапы разработки:

1. Создание прототипа всех основных алгоритмов, которые будут необходимы ПО.

2. Анализ прототипа: его доработка и оптимизация.

3. Реализация алгоритмов на языке программирования ПО (пока только *C++*).

Основные математические и алгоритмические проблемы решаются на первом и втором этапах разработки, поэтому написание с «чистого листа» этих алгоритмов на языке программирования ПО не вызывает осложнений. Однако такой подход требует значительных затрат времени и ресурсов. Таким образом, возникает вопрос: как можно автоматически или полуавтоматически перейти от второго этапа к третьему?

Всемирно известные системы переписывания термов *MAUDE* [7], *ELAN* [8], *STRATEGO* [9] решают проблему автоматической генерации исходного кода по его прототипу, однако все эти конверторы поддерживают лишь небольшую часть функциональных возможностей системы и результат генерации – не только некоторый исходный код, а и его компиляция в выполняемый файл. В настоящее время нет информации об использовании этих систем в коммерческих продуктах, что позволяет позиционировать их как системы для исследовательских проектов исключительно. С другой стороны, система алгебраического программирования *APS* представляет собой обобщение всех этих систем, поскольку большинство их функциональных возможностей включены в *APS*. Кроме того, в третьей версии *APS* существенно улучшены алгоритмы переписывания и языковые возможности системы. Решена также задача эффективного «хеширования» термов. Более детально об этих результатах, а также о сравнительной характеристике, языковых и функ-

циональных возможностях систем переписывания термов можно будет ознакомиться в статьях, которые готовятся к печати в Вестнике Харьковского национального университета, серия «Математическое моделирование. Информационные технологии. Автоматизированные системы управления».

Наш опыт разработки коммерческого ПО свидетельствует о том, что подход к генерации исходного кода алгоритмов ПО с использованием его прототипа, при котором создается запускаемый файл, не применим для больших коммерческих продуктов, поскольку такие генераторы ориентированы на получение запускаемого файла. Такие генераторы поддерживают только небольшую часть языковых возможностей системы.

В больших коммерческих продуктах при переходе от второго к третьему этапу совсем не обязательно получать запускаемый файл, а достаточно получить исходный код конкретного алгоритма, который потом необходимо вставить в большой программный продукт.

Таким образом, проблема генерации исходного кода по его прототипу в системах переписывания термов – актуальна.

Данная статья посвящена проблеме генерации *C++* кода по его прототипу в системе алгебраического программирования *APS* так, чтобы этот генератор поддерживал все функциональные возможности языка, и полученный исходный код можно было использовать в больших программных продуктах (*C++* – язык реализации *APS*).

### *APS* – инструмент для создания прототипов

*Алгебраическое программирование* – это программирование, основанное на переписывании. Оно часто рассматривается как расширение функционального программирования и применяется при решении задач компьютерной алгебры (таких, как проблема слов в конечно определенных алгебрах, алгоритмы пополнения Кнута–Бендикса или Бухбергера), а также задач, связанных с операционной семантикой языков программирования (исполняемые алгебраические спецификации компонентов ПО, определение операционных семантик языков программирования, разработка интерпретаторов и прототипов компонентов ПО и др.).

В отличие от традиционного подхода, ориентированного на использование канонических систем правил переписывания с «очевидной» стратегией их применения, в *APS* возможно сочетание любых систем правил переписывания и разнообразных стратегий переписывания. Такой подход значительно расширяет возможности техник переписывания, поскольку возрастает их гибкость и выразительность. *APS* интегрирует четыре основные парадигмы программирования таким образом, что основная часть программы может быть написана в виде систем переписывания, императивное и функциональное программирование используются для определения стратегий, логическая парадигма реализуется на базе переписывания, использующего встроенную процедуру унификации.

*Инсерционное моделирование*, развиваемое в настоящее время в Институте кибернетики им. В.М. Глушкова, продолжает традиции алгебраического подхода в прикладной теории алгоритмов, программировании и разработке кибернетических систем, основы которого были заложены В.М. Глушковым и его школой. В алгебре алгоритмов Глушкова программа рассматривается как алгебраически определенное преобразование множества состояний информационной среды (память, база данных, многоуровневые распределенные структуры данных и др.). В инсерционном программировании этот взгляд обобщается на объекты, обладающие поведением. Вместо пассивной среды, такой, например, как память, рассматривается активная среда, в которой преобразование информации системой взаимодействующих агентов определяет поведение этой среды, наблюдаемое пользователями информации, производимой внутри среды. Программа в инсерционном программировании рассматривается как агент, обладающий поведением. Его погружение в среду изменяет поведение этой среды. Переход от программ как преобразователей состояний к программам как преобразователям поведений можно сравнить с переходом от точечных к функциональным пространствам в математике.

Инсерционное программирование – это программирование на базе модели поведения агентов в средах [10]. В основе модели – понятие размеченной транзиционной системы, т.е. системы, определенной так же, как и автомат, множеством состояний и множеством переходов (пары состояний), размеченных действиями или событиями. Формально понятие транзиционной системы совпадает с понятием недетерминированного частично определенного автомата, однако, в отличие от теории автоматов отношение эквивалентности транзиционных систем более сильное [2].

В настоящее время система алгебраического программирования рассматривается ее авторами как базовая система для создания системы «Инсерционного моделирования», работы по созданию которой активно ведутся в Институте кибернетики им. В.М. Глушкова НАН Украины и Херсонском государственном университете МОН Украины. Более детальную информацию об этих работах можно будет получить в дальнейших публикациях авторов.

Входным языком системы является *APLAN*, позволяющий описывать не только алгебраическую среду (компоненты, операции и тождества алгебры данных), но и процедуры и вызывать их из алгебраических программ (систем переписывающих правил). Удобные средства для представления процедур позволяют пользователю манипулировать как стандартными, так и собственными разработанными эффективными стратегиями для применения переписывающих правил. Стратегии переписывания могут использовать различные свойства алгебры данных, рассматривая алгебраические выражения (термы) с точностью до некоторых отношений конгруэнтности. 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 [2].

Основная часть программы на *APS* может быть разделена на две части:

- процедурную и
- часть программы с системами переписывающих правил.

В языке *APLAN* процедуры определяются следующим образом:

<процедура>::=*proc(*<список формальных параметров, разделенных запятой>)

[*loc*<список локальных параметров, разделенных запятой>]

<список операторов процедуры, разделенных точкой с запятой >

<формальный параметр>::=<идентификатор>

<локальный параметр>::=<идентификатор>

Операторы в *APS* могут быть трех типов:

1. Внутренние операторы – канонизаторы отметок, реализованные как *C++* функции, функции и процедуры, которые определены в бинарных файлах дистрибутива *APS*.

2. Внешние операторы – канонизаторы отметок, реализованные как системы переписывающих правил, процедуры и функции, описанные на языке *APLAN*.

3. Операторы, определенные пользователем – все языковые конструкции, описанные в специальной системе переписывающих правил *compile*, которая вызывается, если данный оператор не был определен как внутренний или внешний.

Системы переписывающих правил на языке *APLAN* определяются так:

<система переписывающих правил>::= *rs*(<список переменных, разделенных запятой >)

(<список правил, разделенных запятой>)

<правило>::=<простое правило> | <условное правило>

<простое правило>::=<алгебраическое выражение> ==<алгебраическое выражение>

<условное правило>::=<условие>–>(<простое правило>)

<переменная>::=<идентификатор>

Отметим, что в алгебраическом выражении могут быть только операторы первого и второго типов, т.е. внутренние и внешние операторы.

При первом применении системы переписывающих правил *APS* выполнит ее трансляцию в *REM* (*Rewriting Machine Language*) – язык команд переписывающей машины.

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*>::=*hash*(<защищенная программа ранга *k*>) |
*hash*(<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 ω((), . . . , ()), где ω – отметка арности *m*. Термы вида *var*(*i*), где *i* – положительное целое, не превышающее количества пропусков в заголовке программы, и только *var*(*i*) 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. Далее будет показано, что каждая замкнутая *REM*-программа однозначно определяет некоторую систему переписывающих правил языка *APLAN* (с точностью до переименования переменных). Оператор *REM*-языка *hash* был реализован в третьей версии *APS* для эффективного поиска соответствующей «защищенной программы ранга *n*» [11].

Поскольку поддерживается идеология частичной конвертации кода из языка *APLAN* в *C*++, то входными данными для нашего конвертора будет не программа, написанная на языке *APLAN*, а имя процедуры или системы переписывающих правил *C*++, код которой необходимо получить. Исходя из практического опыта работы авторов с *APS*, такой подход наиболее востребован при переходе от прототипа к коммерческой версии. Рассмотрим более детально каждую из проблем, возникающих при трансляции *C*++ кода из *APLAN* языка.

**Проблемы трансляции**

Система алгебраического программирования *APS* поддерживает самомодификацию программы на языке *APLAN* в процессе ее выполнения. Это означает, что в процессе выполнения *APLAN*-программ могут создаваться динамически новые процедуры и системы переписывающих правил (с.п.п.), дописываться (удаляться) операторы в процедуре и с.п.п., что в некоторых случаях значительно упрощает процесс написания кода.

Наиболее распространенным примером, реализация которого упрощается с помощью динамического создания с.п.п., есть задача замены подтермов в заданном терме. Для ее решения нужно создать с.п.п. и применить её к заданному терму с помощью определенных в системе стратегий [2]. Отметим, что среди наиболее близких систем переписывания термов такой особенностью обладает только *APS*.

При трансляции процедур, описанных на языке *APLAN,* необходимо учитывать следующее:

1. Все формальные параметры *APLAN*-процедур воспринимаются как один терм, при передаче параметров происходит процесс унификации терма формальных параметров с входным термом, при этом выбирается максимальная последовательность формальных параметров слева направо, при которых процесс унификации будет успешным, а лишним значениям формальных параметров присваивается значение «*Nil*».

Например:

*Процедура*1:=*proc*(*x*,*y*,*z*)…
…
*Процедура*1(1,2);
…

Отметим, что *APS* поддерживает русский язык. При вызове процедуры будет происходить унификация термов 1,2 и *x*,*y* (максимальная подпоследовательность формальных параметров), после чего *x* будет равен единице, *y* – двум, а *z* – *Nil*.

2. Внутренние операторы уже реализованы на *C*++ и их необходимо корректно вызывать.

3. Внешние операторы реализованы на языке *APLAN* и их следует не только вызывать, но и транслировать корректно.

4. Операторы пользователя определены в с.п.п. *compile*, поэтому конвертировать нужно не собственно оператор, а результат применения с.п.п. *compile* к нему.

Для трансляции с.п.п. следует решить такие проблемы:

1. Как эффективно выполнить унификацию алгебраических выражений в левых частях правил системы?

2. Как эффективно найти нужное правило с с.п.п. или определить, что подходящего правила нет?

Транслятор языка *APLAN* в *C*++ в большой степени решает все описанные проблемы.

**Транслятор кода**

Рассмотрим положение транслятора кода в модели интерпретатора (рисунок).

Традиционно к транслятору кода предъявляются жесткие требования. Получаемый код должен быть корректным и высококачественным, что означает эффективное использование целевой машины. Кроме того, эффективно должен работать и генератор кода.

Математически проблема генерации оптимального кода – неразрешима. На практике приходится довольствоваться эвристическими технологиями, дающими хороший, но не обязательно оптимальный код. Выбор эвристики очень важен, так как тщательно разработанный алгоритм генерации кода может давать код, работающий в несколько раз быстрее кода, полученного с помощью недостаточно продуманного алгоритма [12].

**Реализация транслятора**

Для реализации алгоритма транслятора, по мнению авторов, лучше всего использовать идеи инсерционного моделирования. В этом случае будем рассматривать код функции и с.п.п., написанный на *APLAN* ,как некий определенный агент с поведением (операторы подпрограммы), в качестве среды возьмем стандарты языка *C*++ и библиотеку *C*++ типов *APS*, тогда функцией погружения будет алгоритм транслятора.

В качестве системы для реализации возьмем систему алгебраического программирования *APS*, поскольку**:**

• *APS* уже имеет синтаксический и лексический анализы для *APLAN*-программ (Парсер, Таблица символов (см. рисунок)).

• *APS* – мощный инструмент для прототипирования алгоритмов.

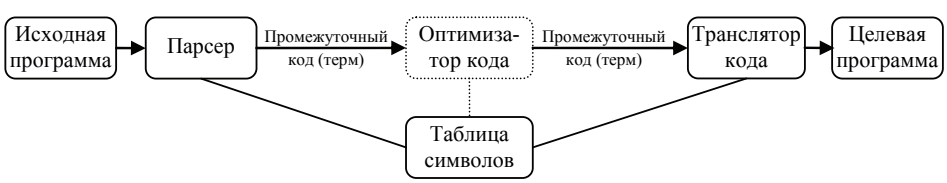
• *APS* – базовая система для инсерционного моделирования.

Итак, рассмотрим принятые решения для проблем трансляции *APLAN*-кода в *C*++. Для кода, который может модифицироваться динамически, создан промежуточный язык *APLANC*, позволяющий использовать все языковые возможности *APLAN* на *C*++. Более детально с этим языком можно ознакомиться в [6].

Для использования внутренних операторов *APS* создан специальный интерфейс и библиотека с минимальным набором таких операторов. Однако в данном случае транслятор только сообщает список операторов, которые не входят в этот минимальный набор, но используются, и при включении полученного кода пользователь должен самостоятельно проверить наличие этих операторов в своей программе.

Поскольку основной языковой «сахар» в *APLAN* достигается с помощью с.п.п. *compile*, транслируются не сами эти операторы, а результат применения *compile*. После решения описанных проблем трансляция *APLAN*-процедур не представляет сложности.

Для трансляции с.п.п. используем «оптимизатор кода» (см. рисунок), который переводит с.п.п. в *REM*-язык. После этого основная сложность состоит в реализации алгоритма унификации для команды *REM*-языка *match*. Сейчас в трансляторе существуют две возможности трансляции с.п.п.: с использованием языка *AP-*




Положение транслятора кода в модели *APS*

*LANC* и без него (*APLANC* делает код более понятным, а его отсутствие может затруднить понимание кода даже у опытных разработчиков).

**Пример**

В качестве примера рассмотрим одну из с.п.п. *R* (описание находится в руководстве пользователя [2, с. 25]). Итак, имеем следующий *APLAN*-код:

```
R :=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))
);
```

Используя переписывающую машину *APS* в качестве оптимизатора кода, переведем эту с.п.п. в *REM*-язык:

```
R=Rs ( array(_,_))
rem_hash(
      ()–>():match(var(0)–>var(1))rewrite(~(var(0))|
            /var(1),2),
      ~(()) : rem_hash(
            ()–>() : match(var(0)–>var(1))rewrite(var(0)&
                  ~(var(1)),7),
            ~(()) : match(~(var(0)))rewrite(var(0),3),
      ()&() : match(var(0)&var(1))rewrite(~( var(0))|/
            ~( var(1)),5),
            ()|/() : match(var(0)|/var(1))rewrite(~( var(0))&
                  ~( var(1)),4),
            ()<=>() : match(var(0)<=>var(1))
            rewrite(~(( var(0))–>( var(1)))|/ ~(( var(1))–>
                  ( var(0))),6))
      ) ,
      ()<=>() : ( match(var(0)<=>var(1))rewrite((var(0)
            –>var(1))&(var(1)–>var(0)),1)
);
```

Здесь *Rs* означает, что с.п.п. переведена в *REM*-язык. *array*(_,_) – массив из двух элементов, используемый для сохранения значений параметров после успешного процесса унификации, также он указывает на то, что используется два параметра, а в теле с.п.п. заменен *x* на *var*(0), а *y* на *var*(1). *rem_hash* – команда *REM*-языка, которая по отметке входящего терма выбирает соответствующие команды *REM*-языка без последовательного перебора (как это было во второй версии *APS*). *match* – команда *REM*-языка, выполняющая процесс унификации, *rewrite* – подставляет вместо исходного терма, терм, который стоит в аргументах (значения *var*(0), *var*(1) есть всегда, так как по умолчанию они равны *Nil*). Второй аргумент команды используется для трассировки с.п.п. в *APS*, а также для оптимизации некоторых внутренних стратегий системы.

Результат трансляции *REM*-языка будет выглядеть так (с использованием *APLANC*):

```
int eliminate(intr_ptr &fpl,node_ptr &t){
nodes_ptr tmp;
node_ptr tmp_t=t;
node_ptr x,y;
if (t–>get_mark()==fpl–>m_Marks[MARK_IFF]){
      if (fpl–>let("eliminate1 ",get_term_val(fpl,tmp_t),"
            (ac_h <=> ac_h)",tmp){
             get_term_value(fpl,tmp_t);
             y=tmp[1];x=tmp[0];
             /*((x <=> y) ) = (((x –> y) &  (y –> x)) ) */
      t=fpl–>make_formula("((() –>  ()) &  (() –>  ()))",4 ,
         *fpl–>new_o(x),*fpl–>new_o(y),*fpl–>new_o(y),
             *fpl–>new_o(x));
      return 1;
      }
…
}
…
  return 0;
}
```

Здесь *intr_ptr* – класс интерпретатора языка *APLANC*, *node_ptr* – класс узла дерева, *nodes_ptr* – класс массива узлов дерева, функция *let* – реализовывает алгоритм унификации, в качестве переменных для этого алгоритма используется массив *tmp* и ключевое имя *ac_h*, функция *get_term_value* – подготавливает терм для дальнейшего использования, а *get_term_val* – возвращает текущий терм для процесса унификации, *make_formula* – создает терм по шаблону. Более детально эти функции описаны в [6].

**Заключение**. Транслятор кода *APLAN* в *C++* решает три основные проблемы, возникающие при переносе кода с языка *APLAN* в *C++*:

1. Время получения *C++* кода существенно сокращается.

2. Минимизируются ошибки, связанные с опечатками при переносе кода (параметры при вызове процедуры поменялись местами, не те параметры передаются, где-то теряется значение и пр.). Некоторые из таких ошибок очень трудно локализовать.

3. В *C++* коде создаются *APLAN*-комментарии, что делает *C++* код более читаемым и позволяет при небольших изменениях *APLAN*-кода вносить качественно изменения в *C++* код.

К недостаткам описанной трансляции можно отнести тот факт, что код, написанный с использованием языка *APLANC*, будет всегда медленней работать, чем код на *C++* по понятным причинам. Однако трансляция без использования *APLANC* эллименирует этот недостаток.

Таким образом, система алгебраического программирования *APS* – мощный инструмент не только для прототипирования алгоритмов, а и для создания окончательных версий на языке *C++*. Предложенные решения и опыт использования *APS* на ряде коммерческих продуктов позволяет позиционировать ее как наиболее мощную систему переписывания термов, которую можно эффективно использовать не только для исследовательских, но и для коммерческих продуктов.


1. [*http*://www.icyb.kiev.ua]
2. *Algebraic* programming system *APS*-1. Informatics'89 / A.A. Letichevsky, Ju.V. Kapitonova, S.V. Konozenko et al. // Proc. of the Soviet-Franch symp. Tallin, 1989. – P. 46–55.
3. *Летичевский А.А.*, *Капитонова Ю.В.*, *Волков В.А.* Сертификация систем с помощью базовых протоколов // Кибернетика и системный анализ. – 2005. – № 4. – С. 256–268.
4. *The Program* Environment of Support of the Practical Training in the Course of Mathematical Logic. – http://www.kspu.edu/About/Downloads/LabRVPPZ/MathLg.htm
5. *Программно*-методический комплекс Терм VII поддержки практической математической деятельности. – http://www.kspu.edu/About/Downloads/LabRVPPZ/Term.htm
6. *Letichevsky A.*, *Letichevsky A.* jr, *Peschanenko V.* APS C++ User Library // Проблемы программирования. – 2008. – № 2–3. – С. 299–304.
7. *Maude* System. – http://maude.cs.uiuc.edu
8. *Elan* System. – http://elan.loria.fr
9. *Stratego* System. – http://www.program-transformation.org/Stratego/WebHome
10. *Letichevsky A.A.*, *Gilbert D.R.* A general theory of action languages // Кибернетика и системный анализ. – 1998. – № 1. – С. 16–36.
11. *Letichevsky A.A.*, *Khomenko V.V.* A Rewriting Machine and Optimization of Strategies of Term Rewriting // Cybernetics and Systems Analysis. – 2002. – № 5. – P. 637–649.
12. *Ахо А.*, *Рави Сети*, *Ульман Дж.* Компиляторы: принципы, технолгии и инструменты. – М.: Вильямс, 2003. – 768 с.



Поступила 11.05.2010


Тел. для справок: (044) 526-0058 (Киев)



●