не менее, скорость работы предложенного алгоритма даже в самом сложном случае более чем в два раза превышает показатели sat4j. График также достаточно наглядно подтверждает, что алгоритм сохраняет линейные показатели эффективности при обработке схем с условиями и подсхемами.

Заключение

Предложена теоретическая база и реализация алгоритма, предназначенного для синтеза ветвящихся программ и программ с подпрограммами. Экспериментально показано, что предложенная стратегия вывода на базе С-моделей сохраняет эффективность и линейные показатели сложности при планировании на схемах с вариантной частью и подсхемами. Использование стратегии динамической развертки не ухудшает показателей скорости вычислений, но, очевидно, использует значительно меньший объем вычислительных ресурсов при компиляции. В последующих работах будут приведены результаты исследований в области синтеза рекурсивных программ.

СПИСОК ЛИТЕРАТУРЫ

1. Новосельцев В.Б., Пинжин А.Е. Реализация эффективного алгоритма синтеза линейных функциональных программ // Известия Томского политехнического университета. – 2008. – Т. 312. – № 5. – С. 32–35.
2. Новосельцев В.Б. Синтез рекурсивных программ в системах управления пакетами прикладных программ: Дис. ... канд. техн. наук. – Институт теоретической астрономии академии наук СССР. – Л., 1985. – 50 с.
3. Коваленко Д.А., Новосельцев В.Б. Стратегия установления выводимости формул в структурных функциональных моделях // Известия Томского политехнического университета. – 2006. – Т. 309. – № 7. – С. 126–130.
4. Новосельцев В.Б. Теория структурных функциональных моделей // Сибирский математический журнал. – 2006. – Т. 47. – № 5. – С. 1014–1030.
5. SAT-Race 2006: Runtime comparison of all SAT-Race solvers [Электронный ресурс]. – 2006. – Режим доступа: http://fmv.jku.at/sat-race-2006/analysis.html.

Поступила 03.09.2008 г.

УДК 004.89

АЛГОРИТМ СИНТЕЗА ПРОГРАММ С ЯВНОЙ И НЕЯВНОЙ РЕКУРСИЕЙ

А.Е. Пинжин

Томский политехнический университет
E-mail: alex_pinjin@tpu.ru

Теория С-моделей расширяется понятием рекурсивных подсхем. Предлагается алгоритм синтеза рекурсивных программ, где затраты на вывод характеризуются полиномиальной функцией третьей степени. Приведены теоретические и экспериментальные результаты оценки эффективности алгоритма.

Ключевые слова:
Функциональная связь, алгоритмы, логический вывод, синтез программ, функциональное программирование, подпрограммы, условия, рекурсия.

Ранее, в [1] был предложен подход к синтезу программ, содержащих условия и подпрограммы. Предполагалось, что схема не может включать в себя атрибуты сложного типа, объявленные на этой же схеме. Такая ситуация при выполнении доказательства существования решения приводит к бесконечному повторению вывода на одной и той же схеме (зацикливание). В настоящей статье рассматриваются правила введения рекурсивных конструкций в описание схем и предлагаются алгоритмы вывода на рекурсивных подсхемах без использования статической развертки. При этом учитывается случай неявной рекурсии, когда рекурсивный вызов достигается посредством одной или нескольких промежуточных подсхем. Приводятся экспериментальные оценки эффективности предложенного алгоритма.

Правила описания рекурсивных схем

Напомним общий синтаксис описания схемы [2, 3]:

T=(a_0:S_{00}, a_1:S_{01}, ..., a_n:S_{0n}

if $p_1(...)\supset a_{10}$:S_{10}, a_{11}:S_{11}...| $f\_set_1\square...\square p_k(...)\supset a_{k0}$:$S_{k0}$, a_{k1}:S_{k1}... *endif*|f_set_k.

|f_set).

Введение рекурсивных выражений порождает в описаниях схем конструкции следующего вида [4]: T_1=(..., t_2:T_2, ...), T_2=(..., t_3:T_3, ...), ... , T_k=(..., t_1:T_1, ...), где T_1, T_2, T_3, ..., T_k – схемы С-модели М. Будем называть рекурсию *явной*, если k=1, т. е. T_1=(..., t_1:T_1, ...), и *неявной* или *опосредованной*, если k>1. Соответственно, вхождение атрибута t_1:T_1 в схему T_k при k=1, будем называть *явным*, а при k>1 *неявным* или *опосредованным вхождением рекурсивной подсхемы*.

Введем следующее ограничение для схем, допускающих рекурсивные выражения: для каждой $T_i \in M$ явные или опосредованные вхождения подсхем вида $t{:}T_i$, могут появляться только в одной из двух альтернативных ветвей схемы T_i.

Синтаксически замкнутые С-модели, включающие схемы, удовлетворяющие правилам, описанным выше, будем называть *рекурсивными детерминированными* С-моделями (РДС-модели). Отметим, что введенное здесь определение РДС-модели идентично по смыслу с определением, приведенным в [4], однако избегает использования понятия полной развертки схемы.

Вывод на РДС-модели

Для наглядного описания предлагаемого алгоритма вывода представим РДС-модель в виде схемы, рис. 1.

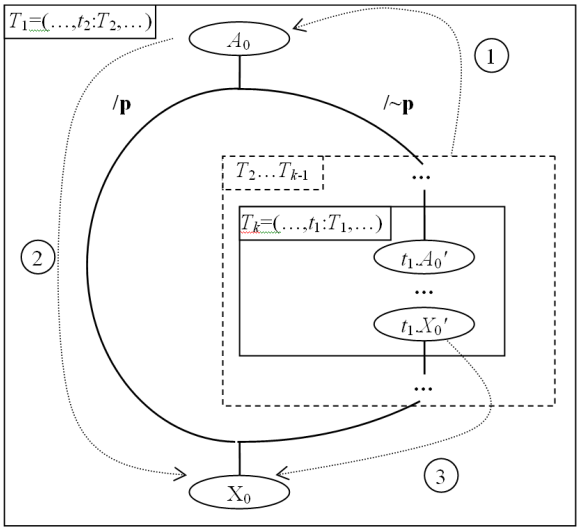

Рис. 1. *Структура РДС-модели*

На представленной схеме прямоугольниками изображены схемы, входящие в РДС-модель, в левом верхнем углу каждой схемы отмечено ее имя и имя атрибута, порождающего рекурсивный вызов подсхемы. Эллипсами отображены исходные (A_0) и целевые (X_0) атрибуты из постановки задачи на схеме. Жирными линиями отображается последовательность функциональных связей (ФС) схемы (*f_set*), дуга в левой части отображает *f_set* альтернативной ветви *p*, в правой – *f_set* ветви ~*p*. Из схемы видно, что в ветви ~*p* содержится ряд вызывающих друг друга промежуточных подсхем ($T_2 \ldots T_{k-1}$) и, наконец, схема T_k, содержащая обращение $t_1{:}T_1$ к головной схеме T_1. Стрелками (1), (2), (3) обозначена предлагаемая последовательность этапов обработки ФС при осуществлении вывода. Рассмотрим эти этапы.

Первый этап включает в себя *идентификацию рекурсии* – определение является ли вхождение подсхемы рекурсивным, а также построение *транзитивного замыкания* – определение множества аргументов рекурсивной процедуры. Предположим, что наличие в T_1 явного или неявного вхождения подсхемы T_k уже установлено и известна последовательность (T_1, $T_2 \ldots T_{k-1}$, T_k). Будем называть вызов головной подпрограммы T_1 *внешним вызовом*, а вызов подпрограммы, соответствующий вхождению $t_1{:}T_1$, *внутренним вызовом*. Описание того, каким образом может быть обнаружено потенциально рекурсивное вхождение $t_1{:}T_1$, приведено в последующих разделах статьи, посвященных реализации системы. При первом достижении потенциально рекурсивной подсхемы T_k определяется множество достижимых атрибутов $t_1.A_0'$ поступающих на вход вызова $t_1{:}T_1$. Если $t_1.A_0 \subseteq t_1.A_0'$, то подсхема идентифицируется как рекурсивная и выполняется переход ко второму шагу. Иначе, если $t_1.A_0 \cap t_1.A_0' \neq \varnothing$, то вызов не является рекурсивным – вывод на T_k осуществляется в обычном режиме (хотя не исключено обнаружение рекурсии при дальнейшем выводе на $t_1.A_0'$). Самая сложная ситуация возникает, когда зафиксировано *сужение множества аргументов*, т. е. $t_1.A_0 \cap t_1.A_0' \neq \varnothing$, $t_1.A_0 \not\subset t_1.A_0'$. В этом случае выполняется присвоение $t_1.A_0 = t_1.A_0'$ и осуществляется повторный вывод по ветви ~*p* схемы T_1 с целью получения нового множества $t_1.A_0'$ (этап (1) на рис. 1). Эта операция выполняется до тех пор, пока не будет выполнено одно из условий: $t_1.A_0 \cap t_1.A_0' = \varnothing$, или $t_1.A_0 \subseteq t_1.A_0'$. В первом случае делается заключение о том, что обращение к подсхеме не является рекурсивным, во втором – фиксирует обнаружение рекурсивного вызова и выполняется переход на следующий этап.

Говоря неформально, введенное правило означает, что если на вход внутреннего вызова подается множество аргументов, полностью отличное от аргументов внешней вызывающей процедуры, то вызов не является рекурсивным – по нему формируется независимая подпрограмма (на основе ФС этой же схемы). Если аргументы внутреннего вызова включают все аргументы внешней процедуры – вызов является рекурсивным. Если же на внутренний вызов подается лишь часть аргументов внешнего вызова, необходимо определить множество аргументов внутреннего вызова при сокращенном множестве аргументов внешнего вызова и так до тех пор, пока не будет достигнуто устойчивое множество входных атрибутов и аргументов внутреннего рекурсивного вызова, либо не будет получено пустое или непересекающееся множество аргументов внутренней процедуры.

Второй и третий этапы вывода осуществляются, если подсхема идентифицирована как рекурсивная. Вывод на рекурсивной подсхеме основан на принципе структурной индукции – устанавливается выводимость ($t{:}T_1$, A_0, X_0/p) (база индукции), затем, исходя из предположения, что ($t_k{:}T_k$, $t_1.A_0'$, $t_1.X_0'$) доказуемо (гипотеза индукции), выполняется доказательство теоремы ($t_1{:}T_1$, $t_1.X_0'$, $X_0/\sim p$).

Предлагаемый подход позволяет синтезировать программы, в которых предусмотрен выход из рекурсии по одной из ветвей условия. Завершаемость

таких программ зависит от интерпретации РДС-модели, однако в настоящей статье этот вопрос не рассматривается.

Реализация системы вывода на РДС-модели

Подготовка (компиляция) схем РДС-модели не требует каких-либо дополнительных действий в связи с введением рекурсивных конструкций. Выявление и обработка рекурсивных вхождений выполняется на стадии вывода. Для этих целей в структуры данных добавляются (рис. 2):

- ссылка внешнийВызовПодпрограммы – указывает на вызов подпрограммы, который инициировал данный вызов. Для вызова подпрограммы, на котором поставлена задача вывода, эта ссылка не определена;
- свойство логического типа содержитРекурсивныйВызов объекта Условие – служит для выделения рекурсивной ветви вариантной части.

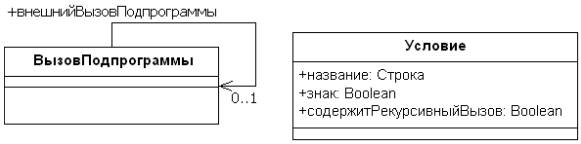

Рис. 2. *Модификация модели исходных данных*

Алгоритм вывода модифицируется следующим образом: при получении очередного вызова подпрограммы из стека свойству внешнийВызовПодпрограммы назначается текущий вызов подпрограммы. Далее, производится восходящее перечисление всех внешних вызовов и проверяется идентичность соответствующих им схем той схеме, которая обрабатывается для текущего вызова. При обнаружении такого совпадения делается вывод об обнаружении потенциальной рекурсии и проверяется пересечение списков аргументов. Если такое пересечение имеет место и обнаружено сужение списка аргументов, то внешний и внутренний вызовы передаются в процедуру построения замыкания. После этого делается окончательный вывод о наличии рекурсивного вызова и выполняется его обработка.

Фрагмент псевдокода, соответствующий описанному алгоритму, приведен ниже. Жирным шрифтом выделена модифицированная часть алгоритма.

```
...
if (счетчик необработанных атрибутов and стек вызовов подпрограмм не пустой) then
  Получить вызов подпрограммы из стека, зафиксировать его как внутренний вызов подпрограммы
  Установить внешний вызов этой подпрограммы равным текущему
  Зафиксировать внутренний вызов в качестве очередного
  while (у очередного вызова подпрограммы существует внешний вызов) begin
  получить внешний вызов подпрограммы
    if (имена схем текущего и внешнего вызова совпадают and
    список аргументов текущего и внешнего вызова пересекаются) then
      зафиксировать обнаружение рекурсивного вызова
      зафиксировать найденный внешний вызов подпрограммы
    end_if
      зафиксировать внешний вызов в качестве очередного
  end_while

  if (обнаружен рекурсивный вызов and сужение множества аргументов) then
    запустить процедуру построения замыкания
    if (кол-во аргументов в результате построения замыкания равно 0) then
      отменить обнаружение рекурсивного вызова
    end_if
  end_if

  if (обнаружен рекурсивный вызов) then
    связать внутренний вызов подпрограммы с подпрограммой, соответствующей внешнему вызову
    скопировать множество достижимых атрибутов-целей внешнего вызова подпрограммы во внутренний вызов подпрограммы
  else
    рекурсивно запустить процедуру вывода на внутреннем вызове подпрограммы
  end_if
  получить достижимые атрибуты-результаты вызова подпрограммы
  создать шаг доказательства и поместить его в список шагов текущей подпрограммы
  увеличить счетчик необработанных шагов на 1
end_if
...
```

Приведенный алгоритм использует процедуру построения замыкания, которая получает на вход две ссылки – на внутренний и на внешний вызов подпрограммы и возвращает количество аргументов внутренней подпрограммы в результате построения замыкания. Алгоритм построения замыкания выглядит следующим образом:

```
begin
while (обнаружено сужение множества атрибутов) begin
  заменить множество фактических и формальных параметров внешнего вызова параметрами внутреннего вызова
  получить начальный список исключенных атрибутов, т. е. аргументов внешнего вызова, отсутствующих во внутреннем вызове
  запустить процедуру очистки на внешнем вызове подпрограмме
end_while
вернуть количество аргументов внутренней подпрограммы
end
```

Этот алгоритм в свою очередь использует процедуру очистки, которая принимает на вход ссылку на внешний вызов подпрограммы, внутренний вызов потенциально рекурсивной подпрограммы и список исключенных атрибутов:

```
begin
поместить формальные параметры подпрограммы в список достижимых атрибутов
while (существует очередной шаг доказательства) begin
  получить из текущего вызова подпрограммы очередной шаг доказательства
  if (доставляющее ПВ является ФС and
    (ФС является безусловной or условная ветвь ФС является рекурсивной) and среди ФС, в которых недостижимые атрибутов участвуют в качестве аргументов содержится доставляющая ФС текущего шага доказательства) then
      добавить результат ФС в список исключенных атрибутов
    if (ФС доставляет результат вызова текущей подпрограммы) then
      добавить соответствующий фактический параметр - результат вызова подпрограммы в список недостижимых атрибутов
    end_if
    if (результат ФС является аргументом вызова промежуточной подпрограммы) then
      добавить соответствующий формальный параметр - аргумент подпрограммы в список недостижимых атрибутов
    end_if
    if (ФС является доставляющей в рекурсивную процедуру) then
      удалить результат ФС из числа аргументов рекурсивного вызова
    end_if
  else_if (доставляющее ПВ является вызовом подпрограммы (п/п) and
    (вызов подпрограммы является безусловным or условная ветвь вызова подпрограммы является рекурсивной) and
    среди вызовов п/п, в которых недостижимые атрибуты участвуют в качестве аргумента содержится доставляющий вызов п/п текущего шага доказательства) then
      рекурсивно запустить процедуру очистки на доставляющем вызове подпрограммы текущего шага доказательства
  end_if
end_while
end
```

Поясним приведенный выше код. При первом вызове процедуры на вход подается вызов внешней подпрограммы, которая фиксируется как текущая. Выполняется последовательное перечисление всех шагов доказательства текущего вызова. Если доставляющая ФС очередного шага содержит в числе аргументов исключенный атрибут, то ее результат считается недостижимым и помещается в список исключенных атрибутов. Если результат ФС одновременно является входом в промежуточную подпрограмму или выходом из текущей подпрограммы, то в список исключенных атрибутов помещается, соответственно, формальный аргумент промежуточной подпрограммы или фактический аргумент вызова текущей. И, наконец, если результат ФС является входом во внутреннюю рекурсивную подпрограмму, то он удаляется из числа аргументов внутреннего вызова, что означает очередное сужение списка аргументов. Объект внутреннего вызова передается в процедуру очистки по ссылке, что обеспечивает отражение результатов обработки в процедуру построения замыкания и обнаружение сужения на очередной итерации.

Если очередной шаг доказательства содержит обращение к промежуточной подпрограмме, то выполняется рекурсивный запуск процедуры очистки на этой подпрограмме.

Оценка эффективности

Дополнительные затраты при работе в классе РДС-моделей приходятся на поиск потенциальных рекурсивных вхождений и построение замыкания. В остальном введение рекурсивных конструкций не ухудшает общей оценки скорости работы алгоритма, т. к. вывод на подзадачах ($t{:}T_1$, A_0, X_0/p) и ($t_1{:}T_1$, $t_1.X_0'$, $X_0/{\sim}p$) (2 и 3 этапы на рис. 1) проводится в рамках основного алгоритма поиска решения и полностью в него включается. Целью приведенных ниже расчетов не является вывод формулы расчета скорости, а получение вида зависимости скорости вычисления от объема исходных данных. В [1] приведена оценка времени работы алгоритма на схемах с условиями и подпрограммами, которая выражается полиномом первой степени $T(r, s, a, u, b)$, где r – количество схем в модели; s – количество не доставляющих ФС каждой подсхемы; a – количество аргументов каждой ФС; u – количество непервичных атрибутов в каждой схеме; b – количество доставляющих ФС каждой схемы.

Рассмотрим предельно сложный случай РДС-модели. Для начала введем переменную n, обозначающую количество атрибутов (как заголовка, так и вариантной части) каждой схемы. Предположим, что n для всех схем одинаково. Так как $s \le n$, примем $s=n$. Предположим, что каждая схема включает в себя вызовы всех подсхем модели. Тогда $u=r$ и $r \le n$. Примем $r=n$, следовательно $u=n$.

С учетом принятых допущений затраты на поиск рекурсивного вхождения определяются $T_{\text{поиск рекурсии}} \le O(|n|^3)$.

Предположим теперь, что рекурсивное вхождение встречается в каждой схеме, опосредовано всеми подсхемами модели и затрагивает все ФС этих схем (входные ФС нерекурсивной ветви условия и ФС выхода из рекурсивной подсхемы не исключаем). При допущении, что построение замыкания требует максимального количества итераций, равного количеству доставляющих ФС каждой подсхемы (равное b), суммарные затраты на построение всех замыканий определяются как $T_{\text{замыкание}} \le O(b \cdot |n|^3)$.

Из приведенных результатов, видно, что скорость вычислений на РДС-модели не выходит за рамки зависимости от объема исходных данных

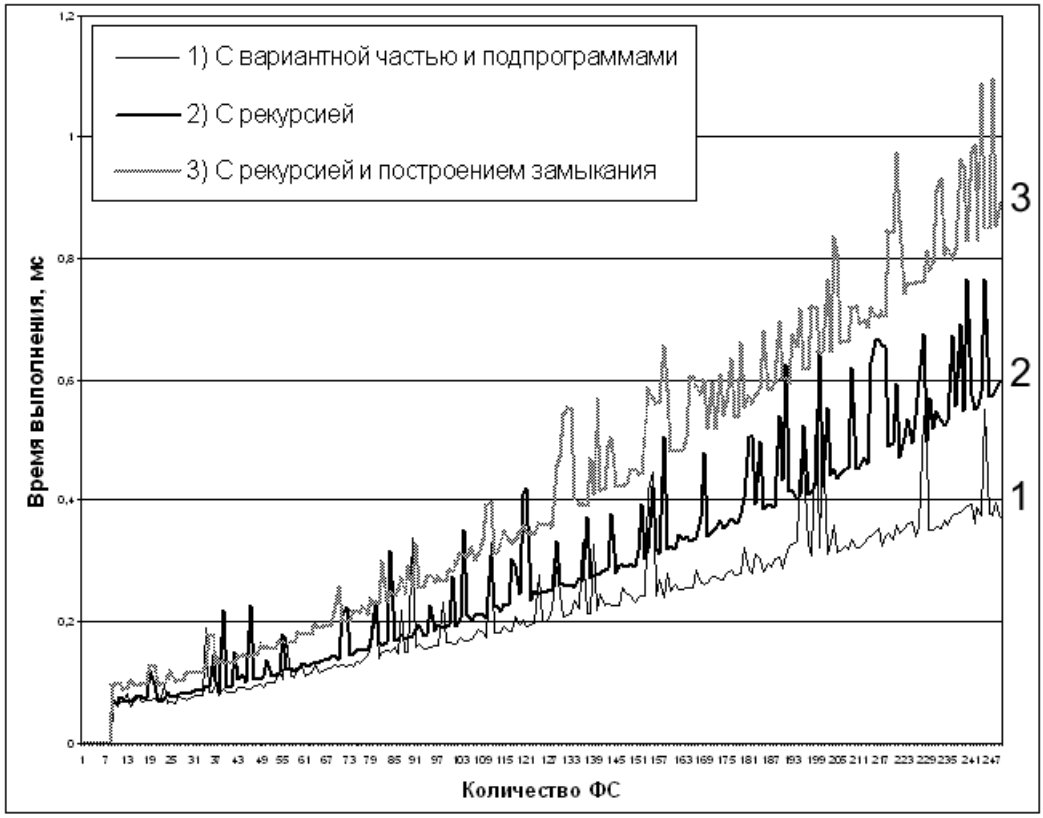

Рис. 3. *Результаты экспериментальной оценки эффективности алгоритма*

модели в виде полиномиальной функции третьей степени. Отметим, также, что предельный случай, используемый для расчетов, имеет мало общего с реальными практическими задачами, где грубые оценки времени поиска рекурсии и построения замыкания обычно не превышают $O(|n|^2)$ и $O(b\cdot|n|^2)$ соответственно.

Экспериментальная оценка, рис. 3, проводилась на моделях, содержащих один рекурсивный вызов, опосредованный всеми подсхемами модели, требующий двукратного прохождения этих подсхем для построения замыкания.

Заключение

Теоретические и экспериментальные результаты показывают эффективность РДС-моделей и реализации на ее базе системы логического вывода. Использование динамической развертки не ухудшает показателей скорости вычислений, а возможность обработки неявной рекурсии значительно увеличивает функциональные возможности системы синтеза программ. В перспективе последующих исследований – дальнейшее усовершенствование алгоритма и его использование для доказательства теорем логики высказываний и логик более высоких порядков.

СПИСОК ЛИТЕРАТУРЫ

1. Пинжин А.Е., Новосельцев В.Б. Эффективный алгоритм синтеза программ с условиями и подпрограммами // Известия Томского политехнического университета. – 2008. – Т. 313. – № 5. – С. 77–84.
2. Новосельцев В.Б., Пинжин А.Е. Реализация эффективного алгоритма синтеза линейных функциональных программ // Известия Томского политехнического университета. – 2008. – Т. 312. – № 5. – С. 32–35.
3. Новосельцев В.Б. Теория структурных функциональных моделей // Сибирский математический журнал. – 2006. – Т. 47. – № 5. – С. 1014–1030.
4. Новосельцев В.Б. Синтез рекурсивных программ в системах управления пакетами прикладных программ: Дис. ... канд. техн. наук. – Институт теоретической астрономии АН СССР. – Л., 1985. – 50 с.

Поступила 03.09.2008 г.