УДК 004.89

РЕАЛИЗАЦИЯ ЭФФЕКТИВНОГО АЛГОРИТМА СИНТЕЗА ЛИНЕЙНЫХ ФУНКЦИОНАЛЬНЫХ ПРОГРАММ

В.Б. Новосельцев, А.Е. Пинжин

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

Предлагается алгоритм синтеза линейных программ на основе заданной спецификации. Алгоритм позволяет добиться высокой производительности за счет предварительной подготовки специальных структур данных. Затраты на вывод и извлечение программы характеризуются линейной функцией от количества атрибутов и функциональных связей, объявленных в спецификации. Приведены результаты опытного сравнения с существующими алгоритмами.

Проблема синтеза линейных, ветвящихся и рекурсивных структур управления известна достаточно давно [1, 2], однако вопрос об эффективной реализации алгоритмов остается актуальным. Методы простого перебора или использование метода резолюций в его классической форме [1] порождают зависимость скорости вычислений от объема высказываний в виде экспоненциальной или, в лучшем случае, полиномиальной функции высокой степени. В ряде работ (например, [3, 4]) декларирована реализация полиномиальных (с первой либо второй степенью) алгоритмов вывода, однако последние либо обладают серьезными теоретическими ограничениями, либо содержат в оценке в качестве множителя очень большую константу. Таким образом, потребность в разработке эффективных алгоритмов по-прежнему существует. Важным также представляется экспериментальное сравнение сложностных характеристик известных алгоритмов с предлагаемым в настоящей работе.

В статье предлагается подход к решению задачи синтеза за счет подготовки специальных структур данных. Метод основан на идеях, изложенных в работах [5, 6].

Исходные данные и постановка задачи

Исходные данные для процедуры планирования (синтеза) линейной программы поставляются в виде множества имен атрибутов и функциональных связей.

Функциональная связь (ФС) определяется выражением вида f: $a_1,...,a_n \to a_0$, где f – имя, a_i – аргументы, a_0 – результат ФС. Обозначим совокупность атрибутов и ФС в виде следующего выражения, которое будем называть простой схемой: $T=(a_0,a_1,...,a_n|f\_set)$, где T – имя схемы, $a_0,a_1,...,a_n$ – список атрибутов схемы, f_set – множество ФС схемы.

ФС f: $a_1,...,a_n \to a_0$, входящая в схему T, называется допустимой, если и только если $a_0,a_1, ...,a_n$ – атрибуты схемы T. Схема называется синтаксически правильной, если f_set содержит только допустимые ФС. В дальнейшем будем рассматривать только синтаксически правильные схемы.

Постановка задачи планирования S определяется следующим образом: $S=(A_0,X_0,T)$, где A_0 и X_0 – наборы имён соответственно исходных и искомых атрибутов, а T – схема, в которой определены эти имена.

При планировании используется классическое правило вывода. Для данного выше определения ФС: $\frac{f: A \to x, A}{x}$.

Подготовка структур данных (компиляция)

Перед началом планирования выполняется специальная подготовка – компиляция описания схемы, на которой поставлена задача вывода. Заметим, что результат компиляции схемы не зависит от исходных и целевых атрибутов, а, следовательно, может быть использован многократно для выполнения разных задач планирования на этой схеме. Эффективная подготовка исходных данных является отдельной задачей и не учитывается при оценке эффективности алгоритма вывода. Структуры данных показаны на рис. 1.

Для схемы создается список объектов типа `Атрибут`, сформированный по всем атрибутам заголовка схемы. Для каждой ФС схемы создается объект типа `ФункциональнаяСвязь`. Каждый атрибут содержит список ссылок `фсАргумент` на функциональные связи, в которых он участвует в качестве аргумента. Функциональная связь в свою очередь содержит ссылку `атрибутРезультат` на атрибут, который является результатом этой ФС. Параметр `счетчикАргументов` перед началом планирования содержит число аргументов ФС.

Постановка задачи

Задача поставляется на вход машины вывода в виде списка исходных атрибутов и списка целевых атрибутов, рис. 2. Создаются два счетчика: счетчик необработанных атрибутов для хранения количества достижимых атрибутов, вхождения которых в ФС еще не были обработаны, и счетчик недостижимых целевых атрибутов для хранения количества целевых атрибутов, достижимость которых еще не установлена. Кроме того, создается список шагов доказательства для хранения последовательности достижимых атрибутов и доставляющих ФС.

Алгоритм доказательства

Процесс доказательства теоремы существования решения приведен на рис. 3.

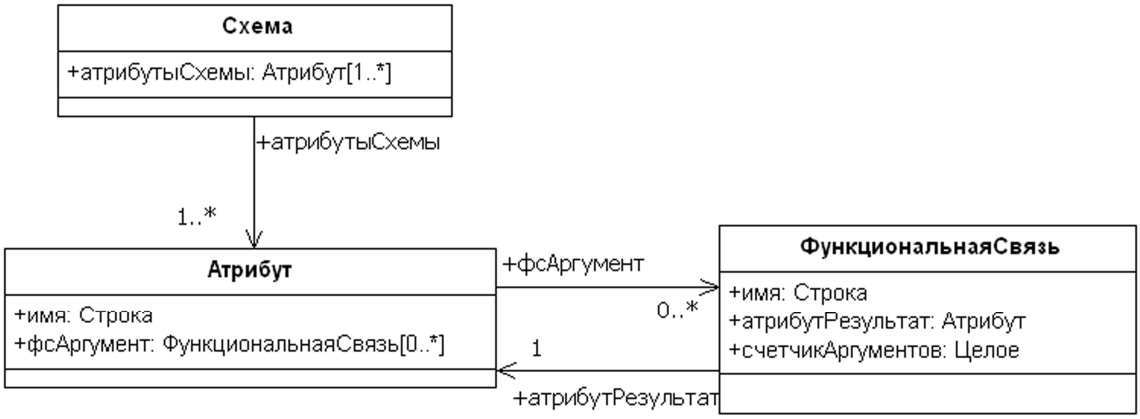

Рис. 1. *Структуры данных алгоритма вывода*

Рис. 2. *Структуры данных постановки задачи и результатов вывода*

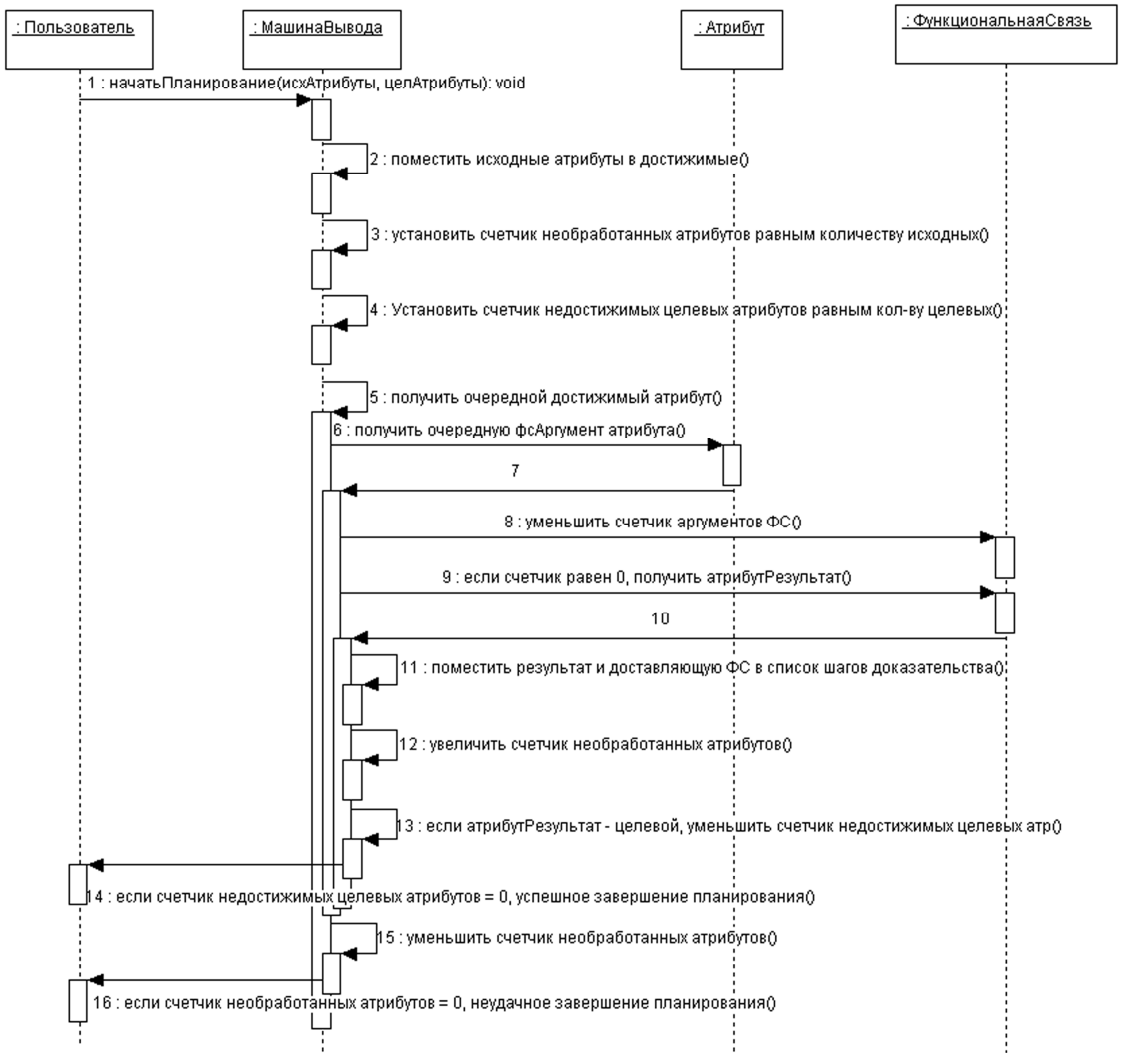

Рис. 3. *Процесс построения доказательства*

Алгоритм также можно представить с помощью некоторого псевдокода следующим образом:

```
begin
Поместить исходные атрибуты в список достижимых атрибутов шагов доказательства
Установить счетчик необработанных атрибутов равным количеству исходных атрибутов
Установить счетчик недостижимых целевых атрибутов равным количеству целевых атрибутов
While(счетчик необработанных атрибутов больше 0) begin
    Получить очередной достижимый атрибут из списка
    Получить список ФС, в которых достижимый атрибут участвует в качестве аргумента
  While(обработаны не все ФС) begin
      Получить очередную ФС
      Уменьшить счетчик аргументов ФС на 1
      If (счетчик аргументов равен 0) then
          Получить атрибут-результат ФС
          Поместить результат и доставляющую ФС в список шагов доказательства
          Увеличить счетчик необработанных атрибутов на 1
          If (результат входит в список целевых атрибутов) then
              Уменьшить счетчик недостижимых целевых атрибутов на 1
              If (счетчик недостижимых целевых атрибутов равен 0) then
                  Теорема доказана, выход из программы
              End If
          End If
      End If
  End While
  Уменьшить счетчик необработанных атрибутов на 1
End While
Теорема не доказана, выход из программы
End
```

Вполне очевидно, что затраты на планирование определяются некоторой линейной зависимостью от совокупного количества аргументов ФС схемы.

Синтез программы

В случае успешного планирования список шагов доказательства используется для синтеза программы. При осуществлении синтеза из доказательства устраняются ФС, не участвующие в получении целей.

Алгоритм синтеза заключается в обработке «снизу-вверх» списка шагов доказательства, полученных в результате планирования. На каждом шаге определяется вхождение результата доставляющей ФС в список целевых атрибутов. В случае утвердительного ответа аргументы ФС помещаются в список целевых атрибутов, иначе шаг доказательства отбрасывается. Полученный в результате список шагов доказательства является окончательной схемой линейной программы.

Эмпирическая оценка эффективности алгоритма

Очевидно, что задача планирования на схеме, является интерпретацией классической задачи вывода на логике высказываний, и может быть сведена к задаче доказательства выполнимости набора логических формул. Из постановки задачи $S=(A_0,X_0,T)$, каждая ФС вида f: $a_1,...,a_n \rightarrow a_0$, входящая в T, формирует дизъюнкт вида $\neg a_1 \vee ... \vee \neg a_n \vee a_0$. Исходные атрибуты, входящие в A_0, преобразуются во множество однолитерных дизъюнктов со знаком отрицания, аргументы-цели – во множество однолитерных дизъюнктов. Задача планирования состоит в доказательстве противоречивости полученного набора формул.

Проблема определения выполнимости набора высказываний в зарубежных источниках часто формулируется как проблема SAT (SATisfiability problem). Для сравнительной оценки был использован пакет sat4j [7], реализованный с использованием того же языка и средств компиляции, что и алгоритм, представленный в данной статье. Пакет sat4j содержит несколько реализаций машин вывода, участвовавших в конкурсе по решению SAT-проблемы в 2006 г. [8]. Среди них была выбрана реализация, показавшая по итогам конкурса максимальную производительность. В основе этой реализации лежит метод вывода DPLL [3, 4], который считается одним из наиболее эффективных.

Для целей тестирования выполнялась генерация схем с возрастающим числом ФС и атрибутов. ФС при генерации связывались таким образом, чтобы обеспечить самый трудоемкий случай – необходимость обработки всех связей при осуществлении планирования. Для сглаживания побочных эффектов генерация данных и планирование на выборке каждого объема производилась многократно, после чего подсчитывалось среднее значение периода времени, затраченного на вывод каждым алгоритмом. Запуск всех тестов выполнялся на одном и том же персональном компьютере.

Результаты сравнительной оценки (рис. 4) показали, что оба алгоритма обладают линейной зависимостью скорости выполнения от объема исходных данных, однако скорость выполнения доказательства у разработанного алгоритма почти в два раза выше.

Заключение

Несомненным преимуществом известных алгоритмов вывода является отсутствие необходимости

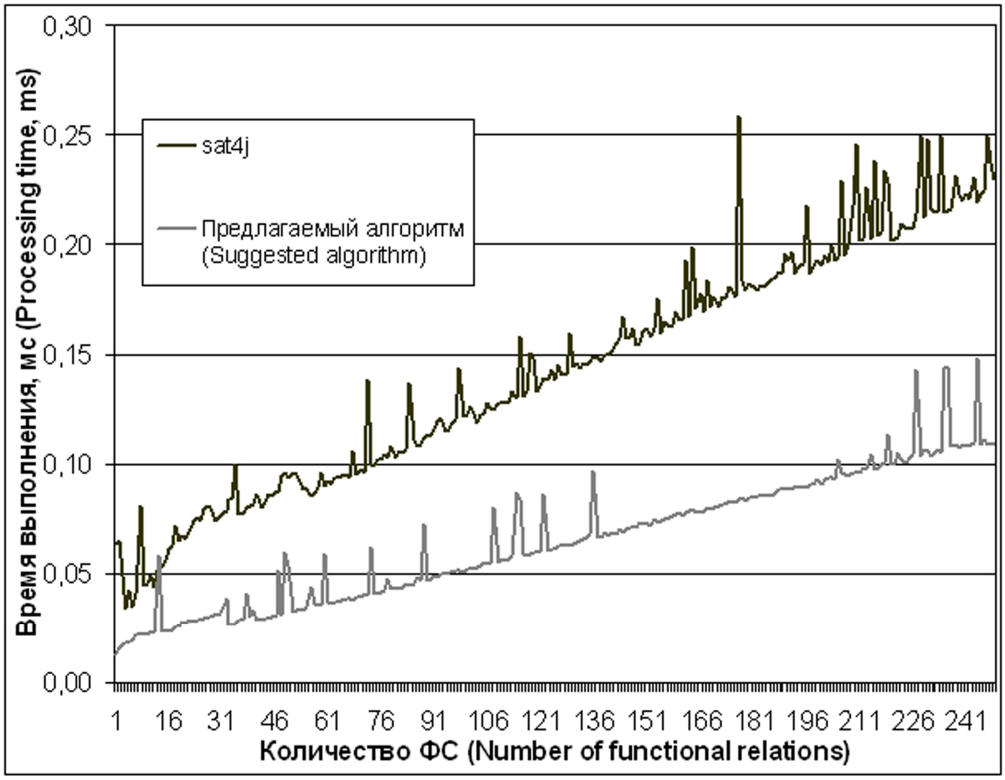

Рис. 4. *Сравнительная оценка производительности алгоритмов*

предварительной компиляции исходных данных. Однако в случае, когда модель исходных данных является достаточно устойчивой, подход, представленный в настоящей статье, может оказаться весьма эффективным. Также следует отметить, что существующие алгоритмы не предоставляют очевидных способов извлечения программы из доказательства.

В настоящее время завершается разработка алгоритма синтеза в классе ветвящихся программ с подпрограммами, в будущем планируется ввести возможность синтеза рекурсивных программ.

СПИСОК ЛИТЕРАТУРЫ

1. Чень Ч., Ли Р. Математическая логика и автоматическое доказательство теорем. – М: Наука, 1983. – 358 с.
2. Waldinger R.J., Lee R.C.T. PROW: A step toward automatic program writing // Proc. of the I Intern. Joint Conf. on Artificial Intelligence. – Bedford, UK, 1969. – P. 241–253.
3. Davis M., Logemann G., Loveland D. A machine program for theorem proving // Communications of the ACM. – 1962. – V. 5. – № 7. – P. 394–397.
4. Een N., Sorensson N. An extensible SAT solver // Proc. of the VI Intern. Conf. on Theory and Applications of Satisfiability Testing. – S. Margherita Ligure, Italy, 2003. – P. 502–518.
5. Диковский А.Я. Детерминированные вычислительные модели // Техническая кибернетика. – 1984. – Т. 84. – № 5. – С. 84–105.
6. Новосельцев В.Б. Теория структурных функциональных моделей // Сибирский математический журнал. – 2006. – Т. 47. – № 5. – С. 1014–1030.
7. Sat4j official site [Электронный ресурс]. – 2008. – Режим доступа: http://www.sat4j.org/
8. 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 15.04.2008 г.

Ключевые слова:

Функциональная связь, алгоритмы, логический вывод, синтез программ, функциональное программирование.