ОБНАРУЖЕНИЕ ЭМПИРИЧЕСКИХ ЗАКОНОМЕРНОСТЕЙ С ПОМОЩЬЮ ЭВМ
(Вычислительные системы)
1984 год Выпуск 102

УДК 681.3.06

ГРАММАТИЧЕСКИЙ МЕТОД СИНТЕЗА ПРОГРАММ

Н.А.Чуканова

Расширение сфер применения вычислительных машин и увеличение числа пользователей-непрограммистов определяют актуальность работ по созданию средств автоматизации программирования, ориентированных на массового пользователя.

Среди существующих подходов к решению проблемы автоматизации программирования вызывает интерес индуктивный синтез программ по примерам, так как примеры являются той информацией, которую в состоянии дать любой, даже самый неквалифицированный пользователь.

Очевидно, что задача извлечения программы из столь малой информации, как примеры, сложна. Трудно надеяться на создание универсальных алгоритмов, позволяющих в реальное время синтезировать сложные и большие программы на любом языке программирования.

Существующие методы индуктивного синтеза позволяют синтезировать программы для некоторых классов задач и определенных языков программирования. Большинство методов [1-3] предназначено для синтеза программ обработки плоских списков на языке LISP, который является языком обработки списков. В [4-5] предложен метод синтеза программ сортировки. Для записи программ используется специальный язык, основанный на формализации понятия многоточия.

В настоящей работе предлагается грамматический метод синтеза программ символьной обработки, основанный на моделировании текстов программ грамматиками некоторого специального вида. Задача синтеза программ сводится к задаче восстановления грамматик по примерам.

Обсуждаются основные результаты по восстановлению грамматик формальных языков и обосновывается выбор грамматик образов для моделирования текстов программ.

Описывается способ задания языков образов в Р-метаязыке [6]. Для записи программ предлагается графический язык, в котором структура обрабатываемых данных отделена от операций по собственно обработке данных.

Грамматический метод синтеза реализован для ЕС ЭВМ в среде ОС с использованием Р-технологического комплекса программиста [6], представляющего погружение Р-машины с Р-метаязыком программирования в операционную систему.

I. Неформально задачу индуктивного синтеза программ можно сформулировать следующим образом: в фиксированном языке (языке спецификаций) описывается конечное множество примеров. Задача состоит в преобразовании описаний примеров в текст на языке программирования. Можно выделить два типа примеров:

- пары "вход-выход", т.е. конечное множество входных и соответствующих им выходных данных;

- истории счета программы, т.е. примеры решения задачи с помощью какого-либо алгоритма.

Если $\sigma = \{(x_1, P(x_1)), \ldots, (x_k, P(x_k))\}$ - конечное множество пар "вход-выход", $S = \{x_1, \ldots, x_k\}$ - множество входных данных, $H = \{h_1, \ldots, h_k\}$ - множество историй счета, то в общем виде задачу индуктивного синтеза можно сформулировать следующим образом: по заданному конечному множеству S (или H) синтезировать программу P^T, которая на множестве S ведет себя так же как и P, т.е. $P^T(x_i) = P(x_i)$ при $x_i \in S$, но, вообще говоря, может иметь истории счета, отличные от H. Программа P^T должна быть в некотором смысле минимальной и удовлетворительно работать на "похожих" входных данных, не принадлежащих S.

В данном рассмотрении, когда x_i и $P(x_i)$ являются символьными последовательностями, а истории счета также можно трактовать как символьные последовательности в некотором алфавите, отвлекаясь от семантики входящих в них слов, постановка задачи синтеза имеет вид: по заданным конечным множествам S и H ($|S| \geq 2, |H| \geq 2$) восстановить (синтезировать) грамматики G_S и G_H такие, что $S \subseteq L(G_S)$ и $H \subseteq L(G_H)$, и для любых грамматик G_S^T и G_H^T, если $S \subseteq L(G_S^T)$ и $H \subseteq L(G_H^T)$, то $L(G_S^T)$ и $L(G_H^T)$ не являются собственными подмножествами $L(G_S)$ и $L(G_H)$ соответственно.

Следует отметить, что при такой постановке задачи синтеза мы предполагаем наличие как множества S, так и множества H. При спе-

цификации программы парами "вход-выход" истории счета явно не задаются. Для их получения будем фиксировать набор элементарных операций и правила их композиции.

Общая схема синтеза программ в предлагаемом методе включает следующие этапы:

- восстановление грамматики G_S по множеству входных данных S ;

- построение множества историй счета H (если они не заданы) по парам "вход-выход", фиксированному набору элементарных операций и правилам их композиции. Различные способы решения этой задачи были рассмотрены в [7];

- восстановление грамматики G_H по множеству историй счета H .

Таким образом, задачу синтеза программ можно свести к задаче восстановления грамматик по примерам. Понятно, что для практических приложений процедура восстановления должна работать в реальное время, трудоемкость определения принадлежности произвольной символьной последовательности языку должна лежать в практически приемлемых границах. Рассмотрим с точки зрения этих требований основные результаты по восстановлению грамматик.

2. Проблема восстановления грамматик для классов языков, входящих в иерархию Хомского, была исследована Голдом [8] и базировалась на концепции идентификации в пределе.

Модель идентификации Голда включает следующие компоненты:

- перечислимый класс грамматик U;

- множество примеров $\sigma = \{s_1, s_2, \dots\}$, в точности совпадающее с языком $L(G)$, порождаемых грамматикой, и, возможно, множество примеров $\bar{\sigma} = \{s_1^T, s_2^T, \dots\}$, принадлежащих дополнению языка. Множества σ и $\bar{\sigma}$ называются соответственно положительным и отрицательным образцами;

- алгоритм идентификации $\mathcal{E}_t$, выдающий по очередной порции информации $\{s_1, \dots, s_t\}$ гипотезу G_t, являющуюся именем грамматики из класса U .

Грамматика (язык) называется идентифицируемой в пределе с помощью алгоритма $\mathcal{E}_t$, если при достаточно большом количестве примеров, т.е. при $t \to \infty$, все гипотезы совпадают $G_t = G_{t+1} = \dots = G_\tau$.

Основные результаты по идентификации языков можно сформулировать в виде следующих теорем.

ТЕОРЕМА I. Класс конечных языков идентифицируем в пределе по положительному образцу.

ТЕОРЕМА 2. Классы регулярных, контекстно-свободных и контекстно-зависимых языков не идентифицируемы в пределе по положительному образцу.

ТЕОРЕМА 3. Классы регулярных, контекстно-свободных и контекстно-зависимых языков идентифицируемы в пределе по положительному и отрицательному образцам.

Таким образом, проблема восстановления грамматик по положительному образцу в общем случае разрешима лишь для класса конечных языков, дескриптивная мощность которых значительно слабее регулярных, контекстно-свободных или контекстно-зависимых языков. В то же время проблема принадлежности для конечных и регулярных языков эффективно разрешима с помощью конечных и детерминированных процедур.

В последнее время появился ряд грамматических моделей, не входящих в иерархию Хомского. Их появление отчасти объясняется поисками порождающего механизма, который бы лучше представлял синтаксис и/или семантику языков программирования, структуры данных и структурные отношения образов и проблема принадлежности была бы разрешима за полиномиальное время.

Англюин в [9] рассмотрела классы непустых рекурсивных языков и сформулировала необходимые и достаточные условия их выводимости по положительному образцу. Одно из условий, которое легко проверяется на практике, формулируется следующим образом:

ТЕОРЕМА 4. Класс непустых рекурсивных языков идентифицируем по положительному образцу при условии, что для любого непустого конечного множества $S \subset \Sigma^*$ (Σ - алфавит) множество $C(S) = \{L: S \subseteq L$ и $L = L_i$ для некоторого $i\}$ имеет конечную мощность [9].

Одним из классов языков, удовлетворяющих этому условию, является класс языков образов [10], различные типы которых мы и будем

использовать для моделирования примеров и историй счета, т.е. для записи обобщений.

3. Напомним основные определения и понятия, касающиеся языков образов.

Пусть Σ – конечный алфавит, содержащий хотя бы два символа, Σ^* – множество цепочек в алфавите Σ (элементы множества Σ называются константными символами, Σ^* – цепочками константных символов); $X = \{x_1, x_2, \ldots\}$ – счетное множество символов, называемых переменными ($X \cap \Sigma = \emptyset$).

Образ – это любая конечная цепочка в алфавите $\Sigma \cup X$. Множество всех образов обозначим через $\mathcal{P}^*$.

Количество различных символов из X, встретившихся в образе p, называется числом переменных.

Пусть f – гомоморфизм $\mathcal{P}^*$ в себя (относительно конкатенации). Если $f(a) = a$ для всех константных символов, то f называется подстановкой. Если $f(x) \in X$ и $f(x) = f(y)$ влечет равенство x и y для любых переменных x и y, то f называется переименованием переменных. Будем использовать запись $[a_1/v_1, \ldots, a_k/v_k]$ для обозначения подстановки, отображающей каждую переменную v_i в цепочку a_i и всех других символов в себя.

Определим на множестве $\mathcal{P}^*$ следующие отношения: а) $p \leq^T q$, если $p = f(q)$ для некоторой подстановки f; б) $p \equiv^T q$, если $p = f(q)$ для некоторой функции переименования переменных f.

Язык образа p, обозначаемый $L(p)$, – это множество таких цепочек $\omega \in \Sigma^*$, что $\omega \leq^T p$.

ТЕОРЕМА 5. К л а с с я з ы к о в о б р а з о в и д е н т и ф и ц и р у е м п о п о л о ж и т е л ь н о м у о б р а з ц у [10].

Мы не будем останавливаться здесь на свойствах языков образов, отметим лишь, что для произвольного языка образов проблема принадлежности NP-полна [10].

Для любого конечного множества $S \subseteq \Sigma^*$ существует в общем случае множество образов, генерирующих заданное множество S, и, в частности, тривиальный образ x, генерирующий все множество Σ^*. Нас будет интересовать минимальный образ p такой, что если $S \subseteq L(p)$, то для любого q, что $S \subseteq L(q)$, $L(q)$ не является собственным подмножеством $L(p)$. Англюин показала [10], что проблема построения такого образа (проблема MINL) NP-полна в общем случае. Был предложен некоторый вариант проблемы MINL – 1 -MINL – построение

образа максимальной длины. Для произвольных языков образов проблема 1-MINL также NP-полна.

В [7,10-11] рассмотрены различные типы языков образов, такие как

- регулярные, т.е. порождаемые образом с k переменными, каждая из которых встречается в нем только один раз, f - неувеличивающий гомоморфизм, запрещающий пустые подстановки;
- расширенные регулярные, т.е. регулярные, допускающие увеличивающие и пустые подстановки;
- языки образов с одной переменной;
- языки образов с двумя переменными инверсионного типа.

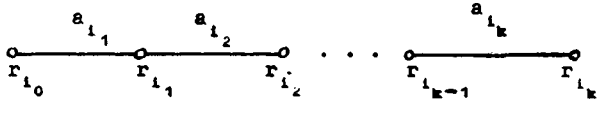
Для всех этих типов существуют алгоритмы полиномиальной сложности для решения проблемы 1-MINL. Проблема принадлежности также разрешима за полиномиальное время. Перечисленные типы грамматик мы и будем использовать для записи обобщений.

Многообразие используемых грамматик (которое, кстати, может быть расширено) объясняется тем, что рассмотрение того или иного конкретного приложения, как правило, дает дополнительную информацию о классе возможных закономерностей или желательном виде грамматик, что позволяет выбрать нужный класс грамматик либо исключить некоторые классы из рассмотрения.

4. Для задания языков образов будем пользоваться метаязыком Р-грамматик [6]. Регулярный язык образа можно представить в виде: $p = w_0 x_1 w_1 \dots x_n w_n$, где $w_i \in \Sigma^*$ $(i = 0,1,\dots,n)$, $x_j \in X^+$ $(j = 1,\dots,n)$. Безмагазинная Р-грамматика с набором правил $r_0 \sim \{w_0 \to r_1\}$, $r_1 \sim \{"\Sigma" \to r_2\}$, $r_2 \sim \{w_1 \to r_3, "\Sigma" \to r_2\}$ и т.д. задает регулярный язык образов. Здесь и далее "Σ" используется для обозначения синтерма, включающего все символы алфавита.

Расширенный регулярный образ имеет вид: $p = w_0 x_1 w_1 \dots x_n w_n$, где $w_0, w_n \in \Sigma^*$, $w_i \in \Sigma^+ (i=1,\dots,n-1)$, $x_j \in X^+ (j = 1,\dots,n)$. Безмагазинная Р-грамматика с набором правил $r_0 \sim \{w_0 \to r_1\}$, $r_1 \sim \{w_1 \to r_2, "\Sigma" \to r_1\}$ и т.д. задает расширенный регулярный язык образов.

Для снижения трудоемкости поиска вхождения $w_i = a_{i_1} \dots a_{i_k}$ в предъявляемую цепочку правила-преемники по несовпадению r_j в Р-грамматике



вычисляются следующим образом: j - наибольшее число $s < j \le i_k$, для которого $a_1 a_2 \dots a_s$ - суффикс цепочки $a_1 \dots a_j$, т.е. $a_1 \dots a_s = a_{j-s+1} a_{j-s+2} \dots a_j$. Если такого s нет, то $j = i_0$. Описанный способ вычисления правил-преемников является аналогом вычисления функции отказов [I2].

Классы языков образов с одной переменной и двумя переменными инверсионного типа можно задать различными способами в зависимости от используемых памятей. Приведем способ задания этих классов грамматик Р-грамматиками с одним счетчиком и одним вагоном.

Образ с одной переменной и двумя переменными инверсионного типа можно охарактеризовать тройкой (i, j, k), где i - количество константных символов, j - позиция самого левого вхождения переменной в образ, k - количество переменных. Если n - длина предъявляемой цепочки, то длина подстановки 1 равна (n-i)/k. Р-грамматика с правилами

$$r_0 \sim \{w_0 \to r_1\},\ r_1 \sim \left\{"\Sigma" \xrightarrow{W_1^C,\ W_1^{\Pi}(\emptyset)} r_2\right\},$$

$$r_2 \sim \left\{\emptyset \xrightarrow{W_2^C(1),\ W_1^{\Pi}(\#)} r_3,\ \emptyset \to r_1\right\},$$

$$r_3 \sim \{w_1 \to r_4\},\quad r_4 \sim \left\{"\Sigma" \xrightarrow{W_2^{\Pi}(\emptyset),\ W_1^{\Pi}(\emptyset)} r_5\right\},$$

$$r_5 \sim \left\{"\Sigma" \xrightarrow{W_2(\emptyset),\ W_1^{\Pi}(\emptyset)} r_5,\ \emptyset \xrightarrow{W_2^{\Lambda}(\#),\ W_1^{\Pi}(\#)} r_6\right\},$$

$$r_6 \sim \{w_2 \to r_7\} \text{ и т.д.}$$

задает язык образов с одной переменной и последовательностями длиной n. Здесь W_1^C - операция прибавления к счетчику; $W_1^{\Pi}(\emptyset)$, $W_1^{\Lambda}(\emptyset)$ - запись анализируемого символа в вагон с правого (левого) концов; $W_2^C(1)$ - операция чтения из счетчика, истинная при равенстве его содержимого 1; $W_2^{\Pi}(a)$, $W_2^{\Lambda}(a)$ - чтение из вагонной памяти с правого (левого) концов. Операция истинна, если символ в вершине вагона совпадает с a.

Для задания языка образов с двумя переменными инверсионного типа в зависимости от вида переменной в правилах r_4 и r_5 осуществляется чтение (запись) то с правого (левого), то с левого (правого) концов.

5. Для записи программ предлагается графический язык, в котором структура обрабатываемых данных отделена от операций по собственно обработке данных. Основной конструкцией языка является правило (команда), задаваемое парой:

УСЛОВИЕ: $\mathcal{A}$
ДЕЙСТВИЕ: $\mathcal{B}$

Выбор такой структуры обусловлен тем, что действия, которые выполняются в процессе работы любой программы, можно разделить на два класса. К первому относятся действия, связанные с распознаванием (анализом) входных данных (анализ структуры данных) и передачей управления в зависимости от текущей ситуации, ко второму - действия по собственно обработке (преобразованию) данных.

Графическая интерпретация правила - это дуга нагруженного ориентированного графа, где УСЛОВИЕ определяет условие обхода графа, ДЕЙСТВИЕ - линейную последовательность действий, выполняемых при совпадении условия.

Графическая программа - это нагруженный ориентированный граф с одним входом, дугами которого являются правила. В каждую вершину, кроме начальной, должно входить не более одной дуги. Дуги (правила), выходящие из одной вершины, будем называть комплексами правил или команд.

Правила обхода графической программы фиксированы слева направо и сверху вниз, т.е. при выполнении условия одной из команд комплекса выполняются соответствующие действия и осуществляется переход по дуге на следующий комплекс правил, в противном случае осуществляется переход на следующее правило комплекса и т.д.

Правила объединения команд в комплексы и комплексов в программы описывается также на графическом языке. Дисциплина программирования на графическом языке иерархическая. На верхних уровнях для нагрузки графов используются имена условий или действий, которые далее уточняются с помощью других имен, либо специфицируются примерами.

Для спецификации примеров будем пользоваться конструкциями вида:

ВХОД: $\mathcal{a}$ ВХОД: $\mathcal{a}$
ВЫХОД: $\mathcal{b}$ ИСТОРИЯ: $\mathcal{v}$,

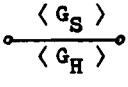
где $\mathcal{a},\mathcal{b},\mathcal{v}$ - цепочки символьных последовательностей (слов), отделенных друг от друга некоторым специальным символом, не встречающимся в самих последовательностях (например, "#"). Соответствие "вход-выход" будем задавать позициями слов в цепочках: первому слову из $\mathcal{a}$ соответствует первое слово из $\mathcal{b}$, второму - второе и т.д. Соответствие "вход-история счета" также будем задавать позициями слов $\mathcal{a}$ и $\mathcal{v}$.

Выделим следующие типы примеров: СЛОВАРЬ, СВОЙСТВО, ДЕЙСТВИЕ. Синтаксис описания примеров имеет вид:

1) СЛОВАРЬ ⟨имя⟩
   ВХОД: $\mathcal{a}$
2) СВОЙСТВО ⟨имя⟩
   ВХОД: $\mathcal{a}$
3) ДЕЙСТВИЕ ⟨имя⟩
   ВХОД: $\mathcal{a}$
   ВЫХОД: $\mathcal{b}$
4) ДЕЙСТВИЕ ⟨имя⟩
   ВХОД: $\mathcal{a}$
   ИСТОРИЯ: $\mathcal{v}$

Первые две конструкции используются для спецификации условий команд, последние - действий.

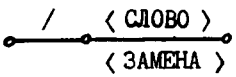
По описаниям примеров восстанавливается соответствующая грамматика и правило графического языка может быть представлено в виде:



Правило интерпретируется следующим образом: если входные данные допускаются грамматикой G_S, то выполняется действие, порождаемое грамматикой G_H.

6. Для иллюстрации предложенного метода рассмотрим небольшую задачу, возникающую при литературном редактировании текстов. Пусть необходимо для всех слов, помеченных специальным символом (например, "/"), выполнить некоторое преобразование с целью изменения модальности высказывания.

Программа на графическом языке будет иметь вид:



Условие СЛОВО описывается множеством примеров

СВОЙСТВО СЛОВО

ВХОД: наихудший#наилучший#...

Действие ЗАМЕНА описывается множеством пар "вход-выход"

ДЕЙСТВИЕ ЗАМЕНА

ВХОД: наилучший#наихудший#...

ВЫХОД: лучший#худший#...

Пусть набор операций включает операцию f(y), результатом применения которой к символьной последовательности является ее подпоследовательность, получаемая удалением первого символа. Полученное множество историй счета будет иметь вид: H = {f(f(f (наихудший))), f(f(f(наилучший)))...}. Грамматики, описывающие множества входных данных и историй счета, будут иметь вид: СЛОВО = наи x␣, ЗАМЕНА = f(f(f(y))).

Предложенный грамматический метод синтеза программ реализован в системе АЛИСА [7] для ОС ЕС ЭВМ.

Литература

I. SUMMERS P.D. A Methodology for LISP Program Constraction from Examples.-J.of the ACM,Jan.1977,v.24, N 1,p.161-175.

2. BIERMANN A.W. The Inference of Regular LISP Programs from Examples. - IEEE Trans. on SMC, Aug. 1978, v.SMC-8, N 8, p. 585-600.

3. BIERMANN A.W., SMITH D.R. A Production Rule Mechanism for Generating LISP code. - IEEE Trans. on SMC, May 1979,v.SMC-9, N 5, p.260-276.

4. BARZDIN J.M. On inductive syntesis of program. - LNCS , 1981, N 122,p.235-254.

5. БАРЗДИНЬ Я.М. Один подход к проблеме индуктивного вывода. -В кн.: Применение методов математической логики. Тез. докл. Таллин, 1983, с. 16-28.

6. ВЕЛЬБИЦКИЙ И.В., ХОДАКОВСКИЙ В.Н., ШОЛМОВ Л.И. Технологический комплекс производства программ на машинах ЕС ЭВМ и БЭСМ-6. -М.: Статистика, 1980. -263 с.

7. ЧУЖАНОВА Н.А. Об одном способе генерации языковых процессоров. -В кн.: Структурный анализ символьных последовательностей (Вычислительные системы, вып. 101), Новосибирск, 1983, с.44-55.

8. GOLD E.M. Language Identification in the Limit. - Inf.Control, 1967, N 10,p.447-474.

9. ANGLUIN D. Inductive Inference of Formal Languages from Positive Data.-Inf.Control,1980,v.45,p.117-135.

10. ANGLUIN D. Finding Patterns Common to Set of Strings.-J. of Computer and System Science,1980,v.21,N 1,p.46-62.

II. SHINOHARA T. Polinomial Time Inference of Extended Regular Pattern Languages.- LNCS,1983,N 147,p.115-127.

I2. АХО А., ХОПКРОФТ Дж., УЛЬМАН Дж. Построение и анализ вычислительных алгоритмов. -М.: Мир, 1979. - 535 с.

Поступила в ред.-изд.отд.
I2 сентября I984 года