АНАЛИЗ ТЕКСТОВ И СИГНАЛОВ
(Вычислительные системы)
1987 год Выпуск 123

УДК 681.3.06

О ГРАММАТИЧЕСКОМ МЕТОДЕ СИНТЕЗА ГРАФИЧЕСКИХ ПРОГРАММ

Н.А. Чуканова

Введение

Широкое внедрение ЭВМ как эффективного инструмента исследований в различных сферах научной и практической деятельности определяет актуальность работ по созданию средств автоматизации программирования, ориентированных на пользователей-непрограммистов (химиков, биологов, лингвистов и т.п.). Одним из подходов к решению данной проблемы является автоматический синтез программ по примерам.

В [1] предложен грамматический метод синтеза графических программ, основанный на восстановлении грамматик образов [2] по конечному множеству примеров. Метод ориентирован на пользователя-непрограммиста и предназначен для синтеза программ обработки символьных последовательностей, основной частью которых является идентификация цепочек в тексте (например, программы редактирования текстов, распознавания образов, поиска в базе данных и т.п.). При этом предполагается, что идентифицируемое множество цепочек может быть описано грамматиками образов.

Непосредственное использование метода для решения практических задач позволило выявить ряд его недостатков. Первый, присущий и другим известным методам индуктивного синтеза, связан с необходимостью тщательного подбора примеров [3]. В предыдущей реализации грамматического метода [1] от пользователя требовалось, чтобы во множество примеров он объединял последовательности, отражающие интересующую его ситуацию из предметной области и имеющие однотипную структуру. В случаях, когда множество примеров включало последовательности с разнотипной структурой, синтезированная программа описывала некоторую общую для всех последовательностей структуру.

Нередко такая программа оказывалась бесполезной, так как отражала либо общеизвестные, либо очень простые свойства исследуемых объектов (далее такие программы называются тривиальными). В то же время подбор последовательностей с однотипной структурой - задача далеко не всегда выполнимая, так как предполагает знание алгоритмов синтеза.

Второй недостаток связан с тем, что синтезированные программы могут оказаться менее эффективными по времени счета, чем создаваемые вручную. Существенный вклад в эффективность и правильность функционирования программы, как и в существующих алгоритмических языках, вносит порядок расположения альтернатив или условий в некоторых графических аналогах операторов case или if.

В данной работе предлагается расширение грамматического метода синтеза, направленное на преодоление указанных недостатков. В общую схему синтеза включается предобработка множества примеров и оптимизация полученной графической программы. Предобработка состоит в выявлении информативных зон в символьных последовательностях и кластеризации множества примеров. Задача оптимизации решается для случая, когда условия или альтернативы в графических аналогах операторов case или if, описанные грамматиками образов, вкладываются друг в друга. Задача оптимизации сводится к задаче упорядочения грамматик по вложенности. Формулируются условия вложенности для расширенных регулярных языков образов.

В качестве полигона для совершенствования метода синтеза использовалась задача обнаружения знаков пунктуации в генетических текстах. Знаки пунктуации представляют сложный и интересный класс символьных последовательностей, для которых, как отмечается в [4], характерно отсутствие четких границ, чередование неинформативных участков с информативными (последние и определяют принадлежность последовательности к некоторому классу знаков), сильное различие знаков даже в пределах одного класса.

§I. Грамматический метод синтеза программ

Напомним основные понятия грамматического метода синтеза графических программ.

Графический язык программирования представляет собой слабоформализованный язык, близкий к профессиональному языку пользователя. Язык позволяет использовать в качестве алфавита нетерминальных символов произвольные термины и обозначения предметной

области и описывать отношения (порядок, вложенность и т.п.) между ними. Все управляющие структуры (типа if, case, while) вынесены на графический уровень.

Основной конструкцией языка является правило (команда), задаваемое парой:

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

Графическая интерпретация правила - это дуга нагруженного, ориентированного графа, где УСЛОВИЕ - некоторый предикат $\mathcal{A}$, при истинности которого выполняется последовательность действий $\mathcal{B}$.

Графическая программа - это нагруженный ориентированный граф с одним входом, дугам которого соответствуют правила, а дугам, выходящим из одной вершины,- комплексы правил. Правила обхода графической программы фиксированы: слева направо и сверху вниз. В каждом комплексе правил выполняется первое по порядку просмотра правило, для которого истинно условие $\mathcal{A}_i$ ($1 \le i \le m$, m - число правил в комплексе). В этом случае выполняются соответствующие действия $\mathcal{B}_i$ и осуществляется переход по дуге на следующий комплекс правил. Все правила в комплексе должны быть упорядочены следующим образом:

- если условие $\mathcal{A}_i$ истинно на множестве S_i, а условие $\mathcal{A}_j$ - на множестве S_j и $S_i \subset S_j$, то $i < j$, т.е. правило с условием $\mathcal{A}_i$ должно быть раньше по порядку просмотра, чем правило с $\mathcal{A}_j$;
- если $S_i \cap S_j \neq \emptyset$ и $S_i \not\subseteq S_j$ или $S_i \cap S_j = \emptyset$, то порядок правил определяется пользователем, исходя из специфики решаемой задачи.

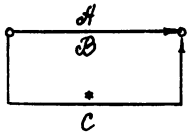
Аналогичным образом должны быть упорядочены правила, помеченные нетерминальными символами. В комплексе они должны располагаться после правил, помеченных терминальными символами.

Управляющие структуры в языке имеют графический эквивалент (см. рисунок).

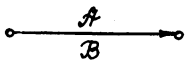
Наряду с такими стандартными для обработки символьных последовательностей предикатами, как "терм", "синтерм", "цепочка термов" и т.п. [5], в графическом языке введены два новых типа предикатов - "словарь" и "свойство".

Под словарем понимается некоторое множество цепочек термов. Значение такого предиката истинно, если существует некоторая подцепочка символов, начинающаяся с текущего анализируемого символа входной строки и совпадающая с одной из цепочек словаря.

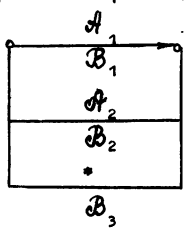
if $\mathcal{A}$ then $\mathcal{B}$ else C



if $\mathcal{A}$ then $\mathcal{B}$



if $\mathcal{A}_1$ then $\mathcal{B}_1$ else if $\mathcal{A}_2$ then $\mathcal{B}_2$ else $\mathcal{B}_3$



case x of

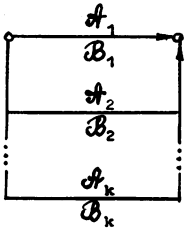
$\mathcal{A}_1 \rightarrow \mathcal{B}_1$

$\mathcal{A}_2 \rightarrow \mathcal{B}_2$

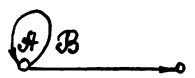
...

$\mathcal{A}_k \rightarrow \mathcal{B}_k$

end



while $\mathcal{A}$ do $\mathcal{B}$



Символом * обозначается безусловный переход.

Предикат "свойство" формулируется в терминах формальных грамматик. Значение предиката считается истинным, если существует некоторая подцепочка символов, начинающаяся с текущего анализируемого символа входной строки и принадлежащая языку $L(G)$, где G - формальная грамматика.

Все упомянутые выше предикаты можно рассматривать как частные случаи предиката "свойство". Так предикат "словарь" можно рассматривать как свойство принадлежности конечному языку, предикат "синтерм" - как свойство принадлежности конечному языку синтаксически равнозначных термов и т.д.

Условия и действия команд могут быть описаны как дедуктивно выражениями Р-метаязыка [5], так и индуктивно с помощью конечного множества примеров. В случае индуктивного описания собственно и возникает задача синтеза графической программы, которая формулируется следующим образом: по заданному конечному множеству примеров S синтезировать грамматику G такую, что $S \subseteq L(G)$ и для любой другой грамматики G' такой, что $S \subseteq L(G')$, $L(G')$ не является собственным подмножеством $L(G)$ (проблема MINL - Minimal Inductive Language).

Процедура синтеза включает синтез условий и действий команд. В данной работе рассматривается синтез условий, для описания которых используются классы формальных грамматик, называемых грамматиками образов [2].

§2. Грамматики образов

Образ - это любая конечная цепочка в алфавите $\Sigma \cup X$, где Σ - конечный алфавит $(\mathrm{card}(\Sigma) \geq 2)$, $X = \{x_1, x_2, \ldots\}$ - счетное множество символов, называемых переменными. Множество всех цепочек в алфавите Σ обозначим через Σ^*, множество всех образов - через $\mathcal{P}^*$. Количество различных символов из X, встретившихся в образе p, называется числом переменных.

Если $f(a) = a$ для любого символа a из Σ и гомоморфизма $f: \mathcal{P}^* \to \mathcal{P}^*$, то f называется подстановкой.

Если $f(x) \in X$ и $f(x) = f(y)$ влечет равенство x и y, то f называется переименованием переменных.

На множестве $\mathcal{P}^*$ определены следующие отношения:

а) $p \leq' q \Leftrightarrow p = f(q)$ для некоторой подстановки f;

б) $p \equiv' q \Leftrightarrow p = f(q)$ для некоторой функции переименования переменных f.

Будем использовать запись $p[a_1/x_1,\ldots,a_k/x_k]$ для обозначения подстановки, отображающей каждую переменную x_i образа p в цепочку a_i, а всех других символов в себя.

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

Неформально говоря, язык образа включает цепочки, полученные подстановкой некоторых цепочек $a_i \in \Sigma^*$ вместо каждого вхождения x_i в образ p. Образ описывает не только общую подпоследовательность для всех последовательностей языка, но и отражает структуру последовательностей (наличие повторов) путем многократного использования одних и тех же переменных.

Для описания предиката "свойство" используются следующие классы языков образов:

- регулярные, т.е. порождаемые образом с k переменными, каждая из которых встречается один раз, а f - не увеличивающий число переменных гомоморфизм, запрещающий пустые подстановки [6];
- расширенные регулярные, т.е. порождаемые образом с k переменными, каждая из которых встречается один раз, а f - произвольный гомоморфизм (в том числе и увеличивающий число переменных), допускающий пустые подстановки [6];
- языки образов с одной переменной [2] и сводимые к ним [7].

Для этих классов языков некоторый специальный случай проблемы MINL - 1-MINL (поиск образа максимальной длины) разрешим за полиномиальное время [2].

Говоря об образе, будем предполагать, что он представлен в канонической форме, т.е. самое левое вхождение x_i левее самого левого вхождения x_{i+1}. Для расширенного регулярного образа q каноническая форма $\hat{p}$ должна удовлетворять следующему свойству: $\forall q[\hat{p} \equiv' q \Rightarrow |\hat{p}| \leq |q|]$. Такой образ можно представить в виде: $\hat{p} = w_0 x_1 w_1 \ldots w_{n-1} x_n w_n$, где $w_0, w_n \in \Sigma^*$, $w_i \in \Sigma^+$ $(i = 1,\ldots,n-1)$, $x_i \in X^+$ $(i = 1,\ldots,n)$.

Здесь через Σ^+ обозначено множество Σ^* без пустой цепочки, через X^+ - множество X без пустого элемента.

На множестве Σ^+ определим отношение $\geq$ следующим образом: $\forall a = a_1,\ldots,a_k$ и $b \in \Sigma^*$, $a \geq b$ тогда и только тогда, когда $b = a_{i_1} \ldots a_{i_m}$ $(1 \leq i_1 < \ldots < i_m \leq k)$. Цепочка a называется надпоследовательностью (или суперпоследовательностью) b или b есть подпоследовательность a.

Если цепочка a представима в виде uvz, то u называется ее префиксом, v – подцепочкой, z – суффиксом. Будем считать, что пустая цепочка ε всегда является префиксом и суффиксом любой цепочки.

Приведем свойство языков образов, которое потребуется в дальнейшем.

ЛЕММА [2]. Для произвольных образов p и q справедливо $p \leq' q \Rightarrow L(p) \subseteq L(q)$.

Заметим, что проблема определения вложенности языков образов NP-полна в общем случае [2].

§3. Предварительная обработка множества примеров

Для любого конечного множества примеров $S \subset \Sigma^*$ существует множество образов, генерирующих заданное множество S. Очевидно, что такие образы, как x, $x_1 a x_2$ ($a \in \Sigma$) и другие "тривиальные" образы, дескриптивная мощность которых велика, не представляют интереса при синтезе реальных программ. Однако, как уже отмечалось, такие образы могут быть синтезированы в случаях, когда во множестве примеров закономерности типа повторов и общих подпоследовательностей проявляются слишком слабо, т.е. когда пользователь объединяет во множество разнотипные объекты.

Для более адекватного описания множества примеров в схему синтеза включается этап предварительной обработки, состоящий в выявлении информативных зон в последовательностях и кластеризации множества примеров. Очевидно, что выбор подходящих мер информативности и близости, отражающих наличие в последовательностях необходимых для формирования образа закономерностей, может оказать большое влияние на корректность синтезируемых программ. В данной работе в качестве мер информативности и близости используется коэффициент конкордации 1-го порядка с двойным ранжированием [4]. Варьирование параметра 1 и значения коэффициента конкордации внутри кластера позволяет выбрать некоторые оптимальные значения, дающие минимальное число ошибок идентификации [4].

В результате предобработки любая последовательность из множества примеров S представима в виде $W_0 z_1 W_1 \dots W_{m-1} z_m W_m$, где $W_0, W_m \in \Sigma^*$, $W_i \in \Sigma^+$ ($i = 1, \dots, m-1$), $z_i \in X^+$ ($i = 1, \dots, m$). Через z_i обозначены неинформативные участки. Множество S, специфицирующее условие $\mathcal{A}$, разбивается на k непересекающихся кластеров ($S = \bigcup_1^k S_j$).

§4. Стратегия синтеза

Общая схема синтеза графической программы в предлагаемом варианте реализации метода включает следующие этапы:

- предобработку множества примеров,
- синтез графической программы,
- оптимизацию графической программы.

Предобработку множества примеров можно рассматривать и как проверку множества примеров на однотипность. В случаях, когда в результате предобработки получено слишком много мелких (включающих одну или несколько последовательностей) кластеров, свидетельствующих о разнотипности множества примеров, переход к синтезу нецелесообразен. Можно потребовать от пользователя увеличить мощность множества примеров или задать новое множество. И только в случае, когда по каким-либо причинам пользователь не может этого сделать (например, при ограниченности числа примеров), можно попытаться синтезировать программу по имеющимся примерам.

Процедура синтеза графической программы применяется не ко всему множеству примеров S, а к примерам, попавшим в один кластер S_j, $j = 1, \ldots, k$. Условие $\mathcal{A}$, специфицированное множеством S, представляется в этом случае как дизъюнкция условий $\mathcal{A}_j$, истинных на множествах S_j, а правило с этим условием расщепляется на комплекс правил.

Может оказаться, что в комплексе правил условия $\mathcal{A}_i$ ($i = 1, \ldots, m$, где m – число правил в комплексе), сформулированные как свойства принадлежности языкам L_i, включают друг друга, т.е. существуют такие правила с номерами $r \le m$ и $l \le m$, что $L_r \subseteq L_l$. Для обеспечения эффективной и правильной работы комплекса встает задача оптимизации комплекса, состоящая в упорядочении языков L_i по включению.

§5. Оптимизация графической программы

Как уже отмечалось, проблема определения вложенности произвольных языков образов друг в друга NP-полна и связана с выявлением синтаксического отношения $\le'$ между образами.

Сформулируем простые условия вложенности для расширенных регулярных языков образов, проверка которых осуществима за полиномиальное время.

Будем предполагать, что образы представлены в канонической форме и имеют вид:

$p = v_0 x_1 v_1 \dots v_{m-1} x_m v_m$, $v_0, v_m \in \Sigma^*$, $v_i \in \Sigma^+$, $i = 1, \dots, m-1$,

$x_i \in X^+$, $i = 1, \dots, m$,

$q = w_0 y_1 w_1 \dots w_{n-1} y_n w_n$, $w_0, w_n \in \Sigma^*$, $w_j \in \Sigma^+$, $j = 1, \dots, n-1$,

$y_j \in X^+$, $j = 1, \dots, n$.

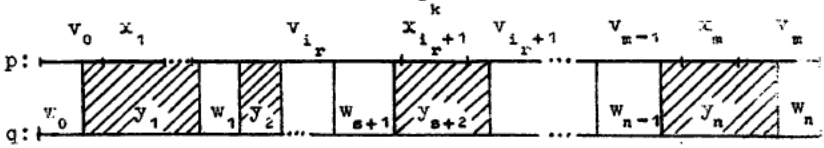
ТЕОРЕМА I. Пусть p и q – расширенные регулярные образы. Для того чтобы имело место включение $L(p) \subseteq L(q)$, достаточно выполнения следующих условий.

1) w_0 есть префикс v_0,

2) w_n есть суффикс v_m,

3) существует хотя бы один набор индексов $0 \le i_1 \le i_2 \le \dots \le i_{n-1} \le m$ такой, что для любого $k = 1, \dots, n-1$ w_k есть подцепочка v_{i_k} и в случае равных индексов $i_r = i_{r+1} = \dots = i_{r+s}$ $(1 \le r+s \le n-1)$ $v_{i_r} \ge w_r \dots w_{r+s}$.

ДОКАЗАТЕЛЬСТВО может быть проиллюстрировано графически. Выровняем образы p и q так, чтобы цепочки w_k располагались под соответствующими подцепочками в v_{i_k}:



Нетрудно видеть, что существует подстановка f (на графике она заштрихована), переводящая q в p. Для случая, представленного на графике, подстановку f можно определить следующим образом: $p = q[a_0 x_1 \dots a^1_{i_r}/y_1, a^2_{i_r}/y_2, \dots, a^{s+2}_{i_r} x_{i_r+1} a^1_{i_r+1}/y_{s+2}, \dots, a^i_{m-1} x_m a_m/y_n]$, где $v_0 = w_0 a_0$, $v_{i_r} = a^1_{i_r} w_1 a^2_{i_r} w_2 \dots w_{s+1} a^{s+2}_{i_r}$, $v_{i_r+1} = a^1_{i_r+1} \dots$ ..., $v_{m-1} = \dots w_{n-1} a^1_{m-1}$. Следовательно, $p \le q$, и по лемме $L(p) \subseteq L(q)$.

ТЕОРЕМА 2. Существует алгоритм сложности $O(|p| + |q|)$ для проверки условий теоремы I.

ДОКАЗАТЕЛЬСТВО. Можно построить детерминированный конечный автомат для поиска вхождений w_k в v_{i_k} на основе метода идентификации цепочек, предложенного в [8].

Сформулированные условия вложенности расширенных регулярных языков, а также линейный алгоритм их проверки дают возможность по-новому взглянуть на проблему MINL для расширенных регулярных языков.

Заключение

Предложен новый вариант реализации грамматического метода синтеза графических программ, включающий предобработку множества примеров, синтез и оптимизацию графической программы. Метод реализован для ОС ЕС ЭВМ и включен в состав второй версии ППП СИМВОЛ [9]. Предложенная схема синтеза апробировалась на задачах обнаружения знаков пунктуации (рибосомных сайтов связывания и терминаторов) в генетических текстах. Результаты работы синтезированной программы приведены в [4].

Литература

I. ЧУЖАНОВА Н.А. Грамматический метод синтеза программ //Обнаружение эмпирических закономерностей с помощью ЭВМ.-Новосибирск. - 1984. -Вып. 102: Вычислительные системы. - С. 32-42.

2. ANGLUIN D. Finding Patterns Common to Set of Strings// J. of Computer and System Sci.-1980.-Vol.11,N 1.-P.46-62.

3. ЧУЖАНОВА Н.А. Индуктивные методы синтеза программ //Логико-математические основы МОЗ. -Новосибирск. -1985. - Вып.107: Вычислительные системы. - С.137-150.

4. ГУСЕВ В.Д., КУЛИЧКОВ В.А., ЧУЖАНОВА Н.А. Алгоритмы обнаружения знаков пунктуации в генетических текстах //Настоящий сборник. - С. 1-24.

5. 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.

6. SHINOHARA T. Polinomial Time Inference of Extended Regular Pattern Languages// LNCS.-1983.-N 147.- P.115-127.

7. ЧУЖАНОВА Н.А. Об одном способе генерации языковых процессоров //Структурный анализ символьных последовательностей. - Новосибирск.- 1983. -Вып.101: Вычислительные системы. -С. 44-55.

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

9. Пакет прикладных программ для анализа произвольных символьных последовательностей значительной длины /Гусев В.Д., Косарев Ю.Г., Тимофеева М.К. и др. //Структурный анализ символьных последовательностей. -Новосибирск.-1984.-Вып. 101: Вычислительные системы. -С. 3-21.

Поступила в ред.-изд.отд.
7 сентября 1987 года