ЛОГИКО-МАТЕМАТИЧЕСКИЕ ОСНОВЫ ПРОБЛЕМЫ МОЗ
(Вычислительные системы)
1985 год Выпуск 107

УДК 681.3.06

ИНДУКТИВНЫЕ МЕТОДЫ СИНТЕЗА ПРОГРАММ

Н.А. Чуканова

В последнее время интенсивно развивается новый подход к решению проблемы автоматизации программирования - синтез программ. Попытки уточнить постановку задачи синтеза привели к появлению многочисленных формулировок и подходов к ее решению. Анализ существующих определений позволяет сформулировать задачу синтеза как преобразование спецификаций (описаний) некоторых свойств программы (каких именно - зависит от подхода) в текст на заданном языке программирования.

Настоящий обзор посвящен рассмотрению индуктивных методов синтеза программ.

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. Различают два типа примеров:

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

- истории счета программы, т.е. заданные для каждой пары "вход-выход" последовательности команд, выполняемых программой для преобразования входных данных в выходные.

Для решения задачи синтеза используют любые эффективные правила вывода, называемые алгоритмами или стратегиями синтеза.

В зависимости от языка, используемого для описания программ, рассмотрим задачу индуктивного синтеза программ для машин Тьюринга-Поста, LISP-программ, графических программ и порождающих грамматик.

2. Задача синтеза программ для машин Тьюринга-Поста по различным типам примеров была сформулирована и исследована Барздинем [1]. Различные аспекты задачи синтеза (под названием идентификации функций в пределе и прогнозирования функций) были изучены в работах Голда [2], Барздиня [3], Янтке [4] и др.

Задачу синтеза программ, вычисляющих одноместные общерекурсивные функции (класс таких функций обозначим через $\mathcal{R}$), в общем случае можно сформулировать следующим образом.

Пусть $N = \{0,1,2,\dots\}$ - множество натуральных чисел, $\mathcal{F}(N)$ - множество всех перестановок N. Каждой конечной n-ке $X = (x_0, x_1, \dots, x_n)$ поставим в соответствие номер из N, который будем обозначать $X[n]$. Аналогично обозначим через $P_X[n]$ номер $(P(x_0), P(x_1), \dots, P(x_n))$, через $P[n]$ - номер $(P(0), P(1), \dots, P(n))$ для всех $n \in N$ и $P \in \mathcal{P}$, где $\mathcal{P}$ - множество одноместных частично-рекурсивных функций и программа $P(m)$ определена $\forall m \le n$. Пусть φ - фиксированная геделевская нумерация всех одноместных частично-рекурсивных функций, $\mathcal{P}^n$ - множество частично-рекурсивных функций от n переменных, называемых алгоритмами или стратегиями синтеза.

Программа P называется синтезируемой в пределе с помощью алгоритма $g \in \mathcal{P}^n$, если для любого множества примеров алгоритм выдает в качестве гипотезы геделевский номер программы и при достаточно большом количестве примеров (в пределе) все гипотезы совпадают. Номер такой программы обозначается как $a = \lim g$.

Класс программ $U \subseteq \mathcal{R}$ называется синтезируемым в пределе по произвольному множеству пар "вход-выход" с помощью стратегии $g \in \mathcal{P}^2$ тогда и только тогда, когда для всех $P \in U$ и для всех $X \in \mathcal{F}(N)$ выполняются следующие условия:

1) $\forall n \in N$ $g(X[n], P_X[n])$ определена;

2) $a = \lim g(X[n], P_X[n])$ существует и $\varphi_a = P$.

Голд показал в [2], что класс всех общерекурсивных функций несинтезируем в пределе по множеству пар "вход-выход". Практически наиболее общими являются классы эффективно перечислимых программ. В [2] показана достаточность условия эффективной перечислимости для синтеза в пределе, однако в [3] доказано, что это условие не является необходимым. Представляет интерес задача поиска некоторых естественных характеристик и классов синтезируемых общерекурсивных функций, для которых существуют наилучшие стратегии синтеза, использующие минимальное число примеров.

Эффективно перечислимые классы программ исследованы в работах Барздиня и Фрейвалда [3]. Показано, что для любого эффективно перечислимого класса программ и любой вычислимой нумерации α этого класса существует стратегия $g \in \mathcal{P}^2$, что для любой входной последовательности $X \in \mathcal{F}(N)$ максимальное число гипотез не превосходит величины $\log_2 n + O(\log_2 n)$, где максимум берется по всем программам $P \in U$, α-номера которых не превосходят n.

Однако условие эффективной перечислимости является слишком слабым, чтобы можно было бы строить экономные алгоритмы синтеза, так как, например, существует эффективно перечислимый класс программ, такой что для любого алгоритма $g(P[n])$ число гипотез превосходит n.

В [I] рассмотрена ситуация, когда о программе дополнительно известно число ее команд — величина $|P|$. Показано, что это условие также является слишком слабым.

В [4] содержится обзор различных классов общерекурсивных функций, синтезируемых в пределе, их классификация и сравнение.

Синтез программ по множествам историй счета рассмотрен в [I]. Показано, что по такому типу примеров можно синтезировать класс всех общерекурсивных функций. Доказано, что существует такой алгоритм синтеза, который перебирает относительно небольшое число гипотез. Для случая, когда задана частичная история счета (не все логические команды известны), число гипотез не превосходит величины $3|P|\log_2|P| + 4$, для полной истории счета - $3\|P\|\log_2\|P\| + 4$, где $\|P\|$ - число логических команд. Показано, что в общем случае порядок оценки не может быть понижен.

Рассмотрим некоторую формализацию метода индуктивного синтеза программ, идея которой принадлежит К.Ф.Самохвалову и базируется на работе [5].

Пусть имеются программа P и отчет pr_0 о результатах испытания этой программы на конечном множестве входов. Кроме того, известна некоторая исходная информация H_0 о программе P. Задача индуктивного синтеза (идентификации) состоит в том, чтобы по паре $\langle pr_0, H_0 \rangle$ сформулировать новое предположение H_1, которое более определено высказывается о программе P.

Пусть f - функция из некоторого подходящего класса, однозначно ставящая в соответствие каждой паре $\langle pr_0, H_0 \rangle$ некоторую гипотезу H_1. Функцию f будем называть методом индуктивного синтеза (идентификации), если I) $H_1 = f(H_0, pr_0)$; 2) мы доверяем информации H_0, то мы доверяем и информации H_1.

В данном контексте проблема поиска методов индуктивного синтеза состоит в формулировке и обосновании ограничений на f, которые гарантировали бы наличие у f желаемых свойств, либо гарантировали отсутствие нежелательных.

Для изучения данной проблемы уточним понятия информации H_0 и H_1 о программе, понятие отчета pr_0 и т.п.

Пусть $\nu_2 : N \xrightarrow{1:1} N^2$ - взаимно-однозначная эффективная нумерация множества неотрицательных целых чисел на множество пар этих чисел (например, с помощью канторовских нумерующих функций). Каждому конечному эксперименту над программой P будем сопоставлять множество pr_0 натуральных чисел по правилу: $x \in pr_0$ тогда и только тогда, когда $\nu_2(x) = \langle n,m \rangle$, $P(n)=m$, где n - вход программы P, m - выход. Таким образом, pr_0 - это множество номеров пар чисел, принадлежащих той части графика функции, вычисляемой программой P, которая нам известна в результате эксперимента над P.

Информация H о программе P отождествляется с подходящей парой $H = \langle P,h \rangle$, где h - число, являющееся постовским номером некоторого рекурсивно-перечислимого множества $\pi(h) \subseteq N$. Принимая H, мы соглашаемся не ожидать среди результатов испытаний программы P тех пар чисел $\langle n,m \rangle$, номера которых в нумерации ν_2 принадлежат множеству с постовским номером h, т.е. испытание опровергает гипотезу H, если $P(n) = m$, $\nu_2^{-1}(\langle n, m \rangle) = x$ и $x \in \pi(h)$. Множество $\pi(h)$ назовем множеством потенциальных фальсификаторов информации (гипотезы) H. Требование рекурсивной перечислимости множества фальсификаторов является весьма существенным, поскольку в противном случае мы могли бы встретиться с ситуацией, когда некоторый эксперимент с программой P опровергает гипотезу H_0, но мы не в состоянии в этом убедиться.

Исходными данными для алгоритма синтеза f является тройка $\langle P, pr_0, h_0 \rangle$, где pr_0 - конечное множество натуральных чисел, h_0 - натуральное число, P - исследуемая программа. Будем говорить, что пара $\langle pr_0, h_0 \rangle$ является допустимой, и обозначать как $\langle pr_0, h_0 \rangle \in Ad$, если $pr_0 \cap \pi(h_0) = \emptyset$.

Пусть $\mathcal{P}$ - класс всех программ. Будем говорить, что $f: \mathcal{P} \times Ad \to \mathcal{P} \times N$ есть допустимый метод индуктивного синтеза, если для любых $\langle P, \langle pr_0, h_0 \rangle \rangle$ из $\mathcal{P} \times Ad$ f удовлетворяет следующим условиям:

1) $f(P, \langle pr_0, h_0 \rangle) = \langle P, ind_f(pr_0, h_0) \rangle$, где $ind_f : Ad \to N$ и ind_f не зависит от P, а $ind_f = Ad$ (условие универсальной применимости f);

2) $\forall \langle pr_0, h_0 \rangle \in Ad$, $h_1 \in N$, если $ind_f(pr_0, h_0) = h_1$, то $pr_0 \cap \pi(h_1) = \emptyset$ (условие непротиворечивости результатов f с исходными данными);

3) $\forall \langle pr_0\ h_0 \rangle \in Ad$, $h_1 \in N$, если $ind_f(pr_0, h_0) = h_1$, то $\pi(h_0) \subseteq \pi(h_1)$ (условие неослабления исходной информации);

4) $\exists \langle pr_0, h_0 \rangle \in Ad$, $\forall h_1 \in N$, если $ind_f(pr_0, h_0) = h_1$, то $\pi(h_1) \neq \pi(h_0)$ (условие существенности f).

Будем называть f регулярным допустимым методом индуктивного синтеза, если выполняется условие

5) $\forall \langle pr_0, h_0 \rangle \in Ad$, $h_1 \in N$, $g \in G$, если $f(pr_0, h_0) = h_1$, pr_0 g-устойчиво, $\pi(h_0)$ g-устойчиво, то и $\pi(h_1)$ g-устойчиво. Здесь G - группа всех рекурсивных перестановок из нумерации ν_2. g-устойчивое множество - это множество, устойчивое относительно перестановки g из G. Требование рекурсивности является весьма существенным, так как только оно гарантирует независимость применения метода f от субъективного произвола, связанного с выбором нумерации ν_2. Если это требование не выполняется, то возможны не согласующиеся друг с другом решения одной и той же задачи в зависимости от выбора нумерации. Нет никаких критериев предпочтения одной нумерации другой, если каждая из них может быть получена рекурсивной перестановкой другой.

ТЕОРЕМА. Для всякого регулярного метода индуктивного синтеза f^*, для всякой пары $\langle pr_0, h_0 \rangle \in Ad$ и $\forall h_1 \in N$, если $f^*(pr_0, h_0) = h_1$, то либо $\pi(h_1) = N - pr_0$, либо $\pi(h_1) = \pi(h_0)$.

Результат теоремы можно трактовать следующим образом: класс регулярных допустимых методов индуктивного синтеза является слишком узким, чтобы представлять практический интерес. В этом смысле результат является отрицательным. Очевидно, что попытки создания методов индуктивного синтеза программ должны вестись в нарушение сформулированных условий.

3. Объектом исследования большинства работ является синтез программ символьной обработки на языке LISP. Постановка задачи синтеза имеет следующий вид: по заданному конечному множеству пар "вход-выход" $\sigma = \{(x_1, P(x_1)), \ldots, (x_t, P(x_t))\}$, где x_1 и $P(x_1)$ -

списки, построить LISP-функцию φ, эквивалентную P для любого x_i из области определения P.

Харди [6] предложил прямой метод решения задачи синтеза для случая, когда задан единственный пример ($|\sigma| = 1$). Синтез сводится к эвристически выполняемому поиску подходящей программной конструкции в некотором фиксированном наборе схем LISP-функций.

Саммерс [7] доказал, что для подобранных некоторым специальным образом примеров возможен формальный вывод искомой LISP-функции φ. В зависимости от рекуррентных соотношений, связывающих множество примеров, функция φ может выражаться рекурсивно в терминах самой функции φ, в терминах другой рекурсивной функции от той же переменной x или в терминах другой рекурсивной функции с одной добавочной переменной.

Алгоритм синтеза, реализующий метод Саммерса, состоит в последовательном выполнении следующих шагов:

- преобразование каждой пары $(x_i, P(x_i))$ в историю счета вида $(p_i \rightarrow f_i(x))$, где p_i и $f_i(x)$ выражены через элементарные LISP-функции;

- поиск рекуррентных зависимостей между элементами во множествах $\{p_i\}$ и $\{f_i(x)\}$.

На основе найденных зависимостей осуществляется переход к искомой LISP-программе.

В качестве элементарных (примитивных) LISP-функций используются следующие функции для обработки списков: car - получение первого элемента (атома) списка, cdr - получение остатка списка после удаления первого элемента, cons- образование списка из составляющих (атома и списка).

Поскольку примитивная функция eq (эквивалентность) не используется, то невозможно определить эквивалентность двух выражений на основе анализа атомов, их составляющих, поэтому вся семантическая информация должна выражаться через структуру выражения.

Для задания списков, как правило, используются S-выражения языка LISP, содержащие полную информацию о том, каким образом было построено такое выражение. Любое неатомарное S-выражение единственным образом конструируется из двух более простых выражений с помощью операции cons. Функции car и cdr обеспечивают уникальную декомпозицию неатомарных S-выражений.

Использование точечной формы записи S-выражений и введение на множестве таких описаний отношения порядка позволяет упорядочить примеры и упростить поиск рекуррентных соотношений.

Рассмотрим пример синтеза программы представления списка списками его составляющих. Множество пар "вход-выход" имеет вид:

$$(A) \rightarrow ((A)),$$
$$(A,B) \rightarrow ((A),(B)),$$
$$(A,B,C) \rightarrow ((A),(B),(C)).$$

Используя примитивные операции языка LISP, получаем:

$f_1[x] = nil;$
$f_2[x] = cons[x,nil];$
$f_3[x] = cons[cons[car[x],nil],cons[cdr[x],nil]];$
$f_4[x] = cons[cons[car[x],nil],cons[car[cdr[x]],nil],$
$\quad cons[cdr[x], nil]];$
$p_1[x] = atom[x];\ p_3[x] = atom[cdr[cdr[x]]];$
$p_2[x] = atom[cdr[x]];\ p_4[x] = atom[cdr[cdr[cdr[x]]]].$

Во множествах $\{f_i\}$ и $\{p_i\}$ можно выделить следующие рекуррентные зависимости:

$f_1[x] = nil;$
$f_2[x] = cons[x,nil];$
$f_{k+1}[x] = cons[cons[car[x],nil], f_k[cdr[x]]]$ для $k=2,3;$
$p_1[x] = atom[x];$
$p_{k+1}[x] = p_k[cdr[x]]$ для $k=1,2,3.$

Используя схему преобразования из следствия I основной теоремы [7], определенную для рекуррентных зависимостей полученного вида, результирующая программа в терминах второй рекурсивной программы имеет вид:

$$\phi[x] \leftarrow [atom[x] \rightarrow nil;\ T \rightarrow \phi[x]];$$
$$\phi[x] \leftarrow [atom[cdr[x]] \rightarrow cons[x,nil];\ T \rightarrow cons[cons[car[x],nil], \phi[cdr[x]]]].$$

В [8] определен класс LISP-программ, аналогичных автоматам с конечным числом состояний и названных регулярными LISP-программами. Класс допустимых примеров описывается регулярным выражением вида $p_i = atom[cw_ir[x]]$, где $w_i \in (a+d)^*$ для $i=1,\ldots,n-1$, w_i

есть собственный суффикс w_{i+1} для i=1,...,n-2 и cr - тождественная функция. Для обеспечения уникальности декомпозиции выходного списка в терминах входного фиксируется схема вывода и порядок выполнения примитивных функций. Доказывается, что задача синтеза регулярных LISP-программ разрешима, и предлагается алгоритм синтеза таких программ по парам "вход-выход".

Отметим ряд трудностей, возникающих при практическом использовании рассмотренных методов синтеза. Во-первых, трудоемкость синтеза экспоненциально зависит от длины программы. Предложенные способы снижения трудоемкости синтеза [9] за счет декомпозиции исходной задачи в иерархию задач и сканирования примеров требует довольно сложной сканирующей системы, поскольку трудоемкость синтеза будет сильно зависеть от сканирования. Во-вторых, методы требуют тщательного подбора примеров, особенно для случаев синтеза по единственному примеру [6]. В-третьих, степень автоматизации процесса синтеза невелика. Большая часть работ (например, переход от множества пар "вход-выход" к историям счета) осуществляется вручную.

4. В работах Барздиня [10-11] предлагается система правил индуктивного вывода, основанная на усмотрении закономерностей типа повторов и арифметических прогрессий в историях счета программы. Для записи обобщений и программ предложено два языка - графический язык и язык, основанный на формализации понятия многоточия, которые легко транслируются друг в друга. Применительно к каждому из языков формализованы соответствующие правила вывода и доказана их асимптотическая полнота, означающая, что для любой графической программы P существует такая константа C_P, что результатами применения правил вывода к любой истории счета h_i для $x_i \geq C_P$ являются графические программы, эквивалентные (асимптотически эквивалентные) программе P (проблема эквивалентности двух графических программ эффективно разрешима).

Язык спецификации примеров состоит из конструкций вида:

ВХОД: $\mathcal{A}$

ОПИСАНИЕ: $\mathcal{B}$

где $\mathcal{A}$, $\mathcal{B}$ - символьные последовательности в некотором фиксированном алфавите. Некоторые числовые элементы в $\mathcal{A}$ могут быть подчеркнуты и означают, что вместо них в общем случае могут стоять произвольные натуральные числа. Последовательность $\mathcal{B}$ есть история работы программы на входных данных $\mathcal{A}$.

Основное правило вывода базируется на выявлении регулярных сегментов в последовательности $\mathcal{L}$. Регулярный сегмент последовательности $\mathcal{L}$ – это связная последовательность вида $x_1^1 \dots x_{\lambda_1}^1 \dots$ $\dots x_1^s \dots x_{\lambda_s}^s$ такая, что $\lambda_1=\lambda_2=\dots=\lambda_s=\lambda$ и для всякого $i(1 \le i \le \lambda)$

– если $x_i^1,\dots,x_i^s$ – буквенные элементы, то они совпадают,

– если $x_i^1,\dots x_i^s$ – числовые элементы, то они либо совпадают, либо удовлетворяют следующему условию:

(*) $x_i^2-x_i^1 = x_i^3-x_i^2 = \dots = x_i^s-x_i^{s-1} = \delta$, $\delta \in \{+1, -1\}$ и $s \ge 2$.

Пусть $i_1,\dots,i_q$ – набор всех индексов таких, что для них выполняется условие (*). Пусть

$$a' = \min(x_{i_1}^1,\dots,x_{i_q}^1), \quad a'' = \max(x_{i_1}^1,\dots,x_{i_q}^1),$$

$$b' = \min(x_{i_1}^s,\dots,x_{i_q}^s), \quad b'' = \max(x_{i_1}^s,\dots,x_{i_q}^s),$$

$$a = [(a'+a'')/2], \qquad b = [(b'+b'')/2].$$

Тогда регулярный сегмент можно записать с помощью графического DO-оператора следующим образом:

$$[I: a \text{ to } b; x_1 \dots x_{i_1-1} \underline{I+c_1}\, x_{i_1+1} \dots x_{i_q-1} \underline{I+c_q} x_{i_q+1} \dots x_\lambda],$$

где $c_k = x_{i_k} - a$.

Рассмотрим пример "пузырьковой" сортировки массива длиной 4, состоящего из элементов

```
ВХОД: A(1:4);
ОПИСАНИЕ: IF A(1) > A(2) THEN A(1)↔ A(2);
          IF A(2) > A(3) THEN A(2)↔ A(3);
          IF A(3) > A(4) THEN A(3)↔ A(4);
          IF A(1) > A(2) THEN A(1)↔ A(2);
          IF A(2) > A(3) THEN A(2)↔ A(3);
          IF A(1) > A(2) THEN A(1)↔ A(2);

          RES<-A;
```

В программе можно выделить два регулярных сегмента с $\delta = +1$. Заменив регулярные сегменты графическим DO-оператором, получим программу вида:

```
ВХОД: A(1:4);
ОПИСАНИЕ: [I:1 to 3; IF A(I) > A(I+1) THEN A(I)⟷ A(I+1);]
          [I:1 to 2; IF A(I) > A(I+1) THEN A(I)⟷ A(I+1);]
           IF A(1) > A(2) THEN A(1)⟷ A(2);
           RES<-A;
```

Повторное применение правила к полученной программе приводит ее к следующему виду:

```
ВХОД: A(1:4);
ОПИСАНИЕ:[J:3 от 2:[I:1 to J;IF A(I)>A(I+1)THEN A(I)⟷A(I+1)]]
         IF A(1) > A(2) THEN A(1)⟷ A(2);
         RES<-A;
```

Следующее правило - правило обобщения - применяется в случае, когда правило сегментации уже неприменимо. Пусть $\delta_1, \delta_2, \ldots, \delta_v$ - подчеркнутые числовые элементы в $\mathcal{A}$ и $M_0 = \min\{\delta_1, \ldots, \delta_v\}$, $c_1 = \delta_1 - M_0, \ldots, c_v = \delta_v - M_0$. Берем некоторую букву M, не входящую в программу, и подставляем в $\mathcal{A}$ вместо $\delta_1, \delta_2, \ldots, \delta_v$ соответственно выражения $M+c_1, \ldots, M+c_v$. Буква M в данном случае обозначает внешнюю переменную. Анализируя программу, для каждого числового элемента выясняем, к чему он ближе - к I ("абсолютная константа") или к M_0. Если ближе к M_0, то вместо соответствующего числового элемента x_i подставляем величину $M+w$, где $w = x_i - M_0$.

Для описанного ранее примера применение правила обобщения приводит программу к следующему виду:

```
ВХОД: A(1:M);
ОПИСАНИЕ: [J:M-1 от 2;[I:1 to J;A(I)>A(I+1) THEN A(I)⟷A(I+1);]]
          IF A(I) > A(2) THEN A(1)⟷ A(2);
          RES<-A
```

Заметим, что при синтезе программ семантика входящих в нее слов не рассматривается. Однако когда программа синтезирована, можно придать входящим в нее словам семантику, которую они имели в примере. В результате графическая программа превращается в описание соответствующего алгоритма для общего случая. Таким образом, предложенные методы позволяют в процессе синтеза отделить синтаксис от семантики и изучать процесс синтеза с точки зрения синтаксиса.

Предложенные в [10] правила индуктивного вывода позволяют строить доказательства утверждений о программах.

В отличие от описанных ранее методов трудоемкость метода Барздиня пропорциональна квадрату длины программы, длины же примеров, необходимых для синтеза программ, лежат в практически приемлемых границах. (Заметим, что трудоемкость синтеза определяется трудоемкостью поиска регулярных сегментов. Существуют более экономные алгоритмы, например, в [I2] для решения этой задачи.)

5. Предложенный в [I3] грамматический метод предназначен для синтеза программ символьной обработки по множеству пар "вход-выход" $\sigma=\{(s_1,P(s_1)),\ldots,(s_k,P(s_k))\}$, где s_i и $P(s_i)$ - символьные последовательности в некотором алфавите и $|\sigma| \geq 2$, либо по множеству историй счета $H = \{h_1,\ldots,h_k\}$, $|H| \geq 2$. Метод основан на моделировании текстов программ грамматиками некоторого специального вида.

Задача синтеза формулируется следующим образом: по конечному множеству $S = \{s_1,\ldots,s_k\}$ и H восстановить (синтезировать) грамматики G_S и G_H, такие, что $S \subseteq L(G_S)$ и $H \subseteq L(G_H)$ и для любых других грамматик G_S' и G_H', если $S \subseteq L(G_S')$ и $H \subseteq L(G_H')$, то $L(G_S')$ и $L(G_H')$ не являются собственными подмножествами $L(G_S)$ и $L(G_H)$ соответственно.

Таким образом, задача синтеза программ сводится к задаче восстановления грамматик по примерам и общая схема синтеза имеет вид:

- восстановление грамматики G_S по множеству входных данных S;

- построение множества историй счета H (если они не заданы) по парам "вход-выход", набору элементарных операций и правилам их композиции;

- восстановление грамматики G_H по множеству историй счета H.

Проблема восстановления грамматик как проблема идентификации в пределе для классов языков, входящих в иерархию Хомского, была исследована Голдом [I4]. Основные результаты приведены в таблице.

Таким образом, проблема синтеза грамматик по примерам последовательностей, принадлежащих языку, разрешима лишь для классов конечных языков, представляющих малый интерес в задачах синтеза программ.

В последнее время были предложены грамматические модели, не входящие в иерархию Хомского. Их появление отчасти объясняется поисками порождающего механизма, который бы лучше представлял синтаксис и/или семантику языков программирования и других объектов, представляемых символьными последовательностями.

Таблица

Результаты по идентификации языков в пределе

Тип примеров	Класс синтезируемых языков
Аномальный текст	рекурсивно-перечислимые рекурсивные
Последовательности, принадлежащие языку и его дополнению	примитивно-рекурсивные контекстно-зависимые контекстно-свободные регулярные суперконечные
Последовательности, принадлежащие языку	конечные

Англюин [I5] рассмотрела классы непустых рекурсивных языков и сформулировала условия, характеризующие их выводимость по положительной информации. Одно из условий, которое легко проверяется, формулируется следующим образом: индексированное семейство непустых рекурсивных языков $L_1, L_2, \ldots$ выводимо по положительной информации при условии, что для каждого непустого конечного множества $S \subseteq \Sigma^*$ (Σ -алфавит) множество $C(S) = \{L : S \subseteq L$ и $L = L_i$ для некоторого $i\}$ конечно.

Одним из классов языков, удовлетворяющих этому требованию, является класс языков образов [I6], который и используется в [I3] для моделирования примеров и программ.

Введем некоторые определения. Пусть Σ - конечный алфавит символов, называемых константными, Σ^* - множество всех цепочек в алфавите Σ, $X = \{x_1, x_2, \ldots\}$ - счетное множество символов, называемых переменными ($X \cap \Sigma = \emptyset$).

Образ - это конкатенация константных символов и переменных (например, $x_1 01\, x_2$, $11 x_{30} x_{32}$). Образ p порождает множество цепочек или язык $L(p)$, получаемый подстановкой ненулевых цепочек $a_i \in \Sigma^*$ вместо каждого вхождения x_i в p. Число переменных образа p - это количество различных символов из X, встретившихся в p (например, $p = 01 x_1 1 x_2 x_1$ - образ с двумя переменными x_1 и x_2).

Для любого конечного множества S символьных последовательностей существует в общем случае множество образов, генерирующих данное множество, и, в частности, тривиальный образ x. Нас будет интересовать минимальный образ p (называемый описателем) такой, что

если $S \subseteq L(p)$, то для любого такого q, что $S \subseteq L(q)$, $L(q)$ не является собственным подмножеством $L(p)$ (имеется каноническое представление, когда самое левое вхождение x_i левее самого левого вхождения x_{i+1}). Англюин доказала, что существует эффективная процедура, которая по заданному образцу S строит описатель p.

Различные варианты данной проблемы и алгоритмы построения описателей рассмотрены в [I6-I8]. В систему синтеза, реализующую грамматический метод, включены алгоритмы построения описателей с одной переменной [I6], двумя переменными инверсионного типа [I7], регулярного и расширенного регулярного языков образов [I8]. Многообразие используемых грамматик (которое, кстати, может быть расширено) объясняется тем, что рассмотрение того или иного конкретного приложения, как правило, дает дополнительную информацию о желательном виде грамматики или классе возможных закономерностей, что позволяет выбрать нужный класс грамматик или исключить некоторые классы из рассмотрения.

Различные варианты построения преобразований "вход-выход" (историй счета программы) по заданному набору элементарных операций и правилам их композиции рассмотрены в [I7].

Для записи программ предложен язык, в котором структура обрабатываемых данных отделена от операций по собственно обработке данных.

6. Мы рассмотрели наиболее проработанные теоретически методы индуктивного синтеза программ. Обзор прикладных методов можно найти в [I9].

Литература

I. БАРЗДИНЬ Я.М. О синтезе программ по отдельным примерам.-В кн.: Теория программирования. Труды симпозиума. Ч.I. Новосибирск, I972, с. 8I-94.

2. GOLD E.M. Limiting Recursion.- J.Symb.Log.,1965,v.30,p.28-48.

3. БАРЗДИНЬ Я.М., ФРЕЙВАЛД Р.В. О прогнозировании общерекурсивных функций. - ДАН СССР, I972, т.206, № 3, с. 52I-524.

4. JANTKE K.P., BEICK H.-R. Combining Postulates of Naturalness in Inductive Inference.-EIK,1981,v.17,N 8/9,p.465-484.

5. ЗАГОРУЙКО Н.Г., САМОХВАЛОВ К.Ф., СВИРИДЕНКО Д.И. Логика эмпирических исследований. - Новосибирск, I978. - 65 с.

6. HARDY S. Synthesis of LISP programs from examples.-Proc. 4th Int.Joint Conf.Artifical Intelligence, Tbilisi, Georgia,USSR, 1975,Sept., p.240-245.

7. SUMMERS R.D. A Methodology for LISP Program Constraction from Examples.- Journal of ACM,1977, v.24, N 1, p.161-175.

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

9. BIERMANN A.W., SMITH D.R. The Hierarchical Synthesis of LISP Scanning Programs. - In: Inf.Processing 77. Proc.of IFIP Congres,1977,v.7,p.41-45.

I0. BARZDIN J.M. On inductive synthesis of programs. - LNCS, 1981, N 122,p.235-254.

II. БАРЗДИНЬ Я.М. Один подход к проблеме индуктивного вывода. -В кн.: Применение методов математической логики. Тезисы докладов. Таллин, I983, с. I6-28.

I2. ГУСЕВ В.Д., КОСАРЕВ Ю.Г., ТИТКОВА Т.Н. О задаче поиска повторяющихся отрезков текста. -В кн.: Ассоциативное кодирование (Вычислительные системы, вып. 62). Новосибирск, I973, с. 49-7I.

I3. ЧУЖАНОВА Н.А. Грамматический метод синтеза программ. - В кн.: Обнаружение эмпирических закономерностей с помощью ЭВМ (Вычислительные системы, вып. I02). Новосибирск, I984, с. 32-42.

I4. GOLD E.M. Language Identification in the Limit.- Inf.Contr. 1967,v.10,p.447-474.

I5. ANGLUIN D. Inductive Inference of Formal Languages from Positive Data. - Inf.Contr.,1980, v.45,p.117-135.

I6. Angluim D. Finding Patterns Common to Set of Strings. - J. of Computer and System Science,1980,v.21, N 1,p.46-62.

I7. ЧУЖАНОВА Н.А. Об одном способе генерации языковых процессоров. -В кн.: Структурный анализ символьных последовательностей (Вычислительные системы, вып. I0I). Новосибирск, I984, с.44-55.

I8. SHINOHARA T. Polynomial Time Inference of Extended Regular Pattern Languages.- LNCS,1983, N 147,p.115-127.

19. BIERMANN A.W. Approaches to Automatic Programming.- Advances in Computers,1976,v.15,p.1-63.

Поступила в ред.-изд.отд.
I4 сентября I984 года