СТРУКТУРНЫЙ АНАЛИЗ СИМВОЛЬНЫХ ПОСЛЕДОВАТЕЛЬНОСТЕЙ
(Вычислительные системы)
1984 год Выпуск 101

УДК 681.3.06

ОБ ОДНОМ СПОСОБЕ ГЕНЕРАЦИИ ЯЗЫКОВЫХ ПРОЦЕССОРОВ

Н.А.Чуханова

В связи с расширением сфер применения вычислительных машин проблема создания языковых и программных средств, обеспечивающих взаимодействие пользователя-непрограммиста с ЭВМ, становится все более актуальной.

Одним из путей решения этой проблемы является разработка методов и средств, позволяющих пользователю самому конструировать свои собственные языки и средства их поддержки - языковые процессоры, выполняющие функции компиляции, ассемблирования или интерпретации программы, написанной на некотором входном языке.

Сложность создания языковых процессоров во многом зависит от метаязыка для описания синтаксиса и семантики конструируемого языка. Интуитивно ясно, что метаязык, ориентированный на массового пользователя, должен быть наглядным, близким к профессиональному языку пользователя, простым в изучении.

Среди существующих метаязыков, используемых для задания синтаксиса языков, в большей степени удовлетворяет сформулированным требованиям Р-язык [1]. Р-язык допускает задание практически любых синтаксических конструкций, имеет наглядную графическую интерпретацию и позволяет использовать для нагрузки графов нетерминалы, соответствующие терминам предметной области. Как показывают результаты эксперимента по обучению различных категорий пользователей Р-языку, время на его освоение практически не зависит от квалификации пользователей и составляет 3-4 часа [2].

Для задания семантики в существующих метаязыках, как правило, используется механизм семантических определений, задающих соответствие элементов языка понятиям предметной области, либо элемен-

тов некоторой исходной области элементам результирующей. Если набор семантических определений реализован и набор параметров невелик, то пользователь может оперировать их именами, не вдаваясь в тонкости реализации. Однако при создании новых языков, как правило, необходимо конструирование новых семантических определений и их описание на одном из допустимых языков программирования. Трудности использования для этих целей языков программирования связаны со сложностью их освоения. Как показывают результаты эксперимента, время на освоение, например, такого языка как РТРАН [I] (уровень которого в метрической теории программ Холстеда [3] выше таких языков как ПЛ/I, Фортран, Алгол-68 [4]) в 20-50 раз больше, чем для программистов-профессионалов [2]. Кроме того, для работы с существующими языками программирования недостаточно знать только сами языки. Необходимо знание организации вычислительного процесса в машине, организацию хранения данных, организацию библиотек, средства отладки, системное обеспечение и т.п.

Таким образом, Р-язык удобен для задания синтаксиса языков, но требует поиска способов задания семантических определений, доступных массовому пользователю.

Для задания семантики (семантических определений) элементов языка предлагается использовать:

- множество оформленных некоторым специальным образом модулей, предназначенных для решения того класса задач, для которого создается язык;

- множество примеров, соответствующих элементу языка в предметной области.

Описываются методы синтеза семантических определений по произвольному множеству модулей и множеству примеров, являющихся символьными последовательностями.

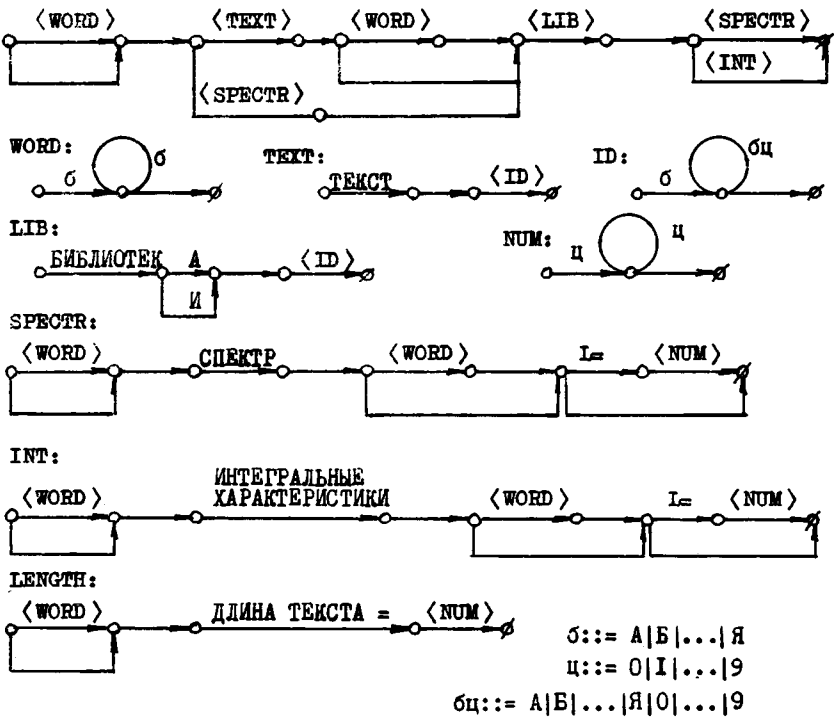
Предложенные методы реализованы в системе АЛИСА, осуществляющей автоматическую генерацию языковых процессоров по описанию синтаксиса в Р-языке и семантике, заданной описанными выше способами.

Применение предложенных средств демонстрируется на примере построения пакета прикладных программ СИМВОЛ [5].

I. Для задания синтаксиса языка используется Р-язык. Элементы и конструкции языка описываются в виде нагруженного ориентированного графа. На дугах графа записываются символы, цепочки символов, имена синтермов (синтаксически равнозначных в данном при-

ложении символов), понятия языка, синтаксическая структура которых описывается отдельно.

ПРИМЕР I. Описание синтаксиса фрагмента языка ППП СИМВОЛ [5] имеет вид:



2. В настоящее время практически в каждой прикладной области накоплены значительные программные фонды, что позволяет перейти на качественно новый уровень описания семантики конструируемых для данной предметной области проблемно-ориентированных языков.

Основным элементом описания семантики в данном случае является модуль - поименованная функционально независимая часть программы, реализующая некоторую функцию. Модуль должен обладать следующими свойствами: независимостью, повторной используемостью (изменения в одном модуле не должны вызывать изменения в другом), раздельной компиляцией и параметризацией. Последнее свойство свя-

зано с определением основных характеристик модуля,задающих способ обращения к нему.

Характеристики модуля описываются в специальной информационной части - паспорте (рис.I).

ПАСПОРТ: ⟨ имя модуля ⟩ ⟨(список формальных параметров) ⟩

ЯЗЫК : ⟨ язык программирования ⟩

ВХОД : ⟨ список входных параметров модуля ⟩

ВЫХОД : ⟨ список выходных параметров модуля ⟩

ОПИСАНИЯ: ⟨ описание входных и выходных параметров средствами того языка программирования, на котором написан модуль ⟩

СПЕЦИФИКАЦИЯ: ⟨ описание модуля через элементы языка ⟩

Рис. I

В разделе СПЕЦИФИКАЦИЯ модуль описывается через элементы языка. При этом различают элементы, соответствующие входным и выходным параметрам модуля. Структура раздела фиксирована и имеет вид:

ВХОД: ⟨ имя элемента языка ⟩ ... ⟨ имя элемента языка ⟩
ВЫХОД: ⟨ имя элемента языка ⟩... ⟨ имя элемента языка ⟩

Спецификация ВХОД-ВЫХОД должна однозначно описывать модуль.

Для задания соответствия фактических параметров формальным имя формального параметра записывается под соответствующей дугой графа,описывающего элемент языка. Поскольку модули могут быть написаны различными программистами, то имена параметров, соответствующих одному и тому же элементу языка,могут не совпадать.Поэтому при описании элементов языка необходимо задать соответствие некоторому произвольному имени,а затем задать список эквивалентностей этих имен формальным параметрам (EQ-список).

ПРИМЕР 2. Пусть имеется некоторая программа VVOD чтения последовательного текстового файла с диска в некоторый буфер. Параметры программы следующие:

NAMED - имя библиотечного набора данных - входной параметр;

NAME - имя раздела библиотеки или последовательного набора данных - входной параметр;

NAME1- имя буфера ввода - выходной параметр;

N - длина текста - выходной параметр.

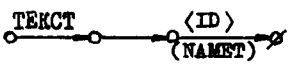
Спецификация модуля в терминах элементов языка из примера I имеет вид:

ВХОД: TEXT , LIB

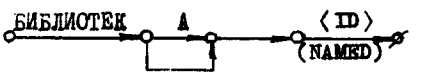
ВЫХОД: TEXT ,LENGTH

Для задания соответствия фактических параметров формальным необходимо доопределить графы соответствующих элементов языка следующим образом:

TEXT:



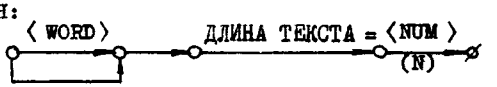
LIB:



EQ-список:

NAMET=NAME=NAME1

LENGTH:



Для задания допустимых последовательностей действий используется M-список: если $F = \{f_1, f_2, \ldots, f_k\}$ - набор модулей, то $M \subseteq F^*$. По M-списку строится RD-таблица [6], имеющая следующую структуру:

Адрес	Вход	Выход	Действие	Адрес по несовпадению
	α_i		f_i	
	. . .	. . .	. . . . .	. . . . .
	α'_{i+n}		f_{i+n}	A_j
	. . .	. . .	. . . . .	. . . . .
	α'_m	β_m	f_m	
A_j	α'_j	β_j	f_j	

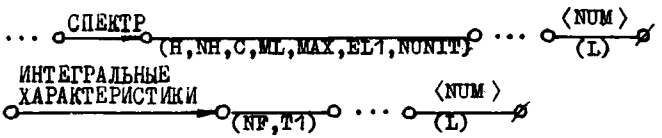
В ней $f_i \ldots f_{i+n} \ldots f_m, f_i \ldots f_{i+n-1} f_j$ - допустимые последовательности; A_j - адрес - продолжение второй последовательности ($f_j \neq f_{i+n}$); α_i - спецификация входа модуля f_i; β_m, β_j - спецификации выходов модулей f_m и f_j соответственно; $\alpha'_{i+n} = \alpha_{i+n} \beta_{i+n-1}$ - дополнительные элементы спецификации входов, необходимые для вы -

полнения очередного модуля. Запись в поле выхода означает,что существует допустимая последовательность модулей, заканчивающаяся этим модулем.

ПРИМЕР 3. Дополним набор модулей из примера 2 модулями ST и INTCH, вычисляющими спектральные и интегральные характеристики текста [5]. Спецификация модулей имеет вид:

```
ST:     ВХОД  : TEXT, LENGTH
        ВЫХОД : SPECTR
INTCH:  ВХОД  : SPECTR
        ВЫХОД : INT
```

Соответствие фактических и формальных параметров задано на графах элементов языка из примера I:



M-список задан последовательностями: VVOD, ST, INTCH; ST, INTCH; VVOD, ST.

RD-таблица будет иметь вид:

Адрес	Вход	Выход	Действие	Адрес по несовпадению
	TEXT			
	LIB	-	VVOD	A1
		SPECTR	ST	
		INT	INTCH	
A1	SPECTR	INT	INTCH	

3. Второй способ описания семантики состоит в задании множества примеров, соответствующих элементу языка в предметной области. Очевидно, что для произвольного множества примеров задача синтеза семантических определений является весьма трудной. В данной работе рассматривается метод синтеза по примерам, являющимся символьными последовательностями некоторой определенной структуры.

Примеры будем задавать в несколько формализованном виде и представлять парой:

ВХОД : $\mathcal{A}$

ВЫХОД : $\mathcal{B}$

где $\mathcal{A}, \mathcal{B}$ - цепочки символьных последовательностей (слов), отделенных друг от друга некоторым символом, не встречающимся в самих последовательностях (например, "#"). Соответствие "вход-выход" задается позициями слов в цепочках: первому слову из $\mathcal{A}$ соответствует первое слово из $\mathcal{B}$, второму - второе и т.д.

Введем следующие типы формальных примеров: СЛОВАРЬ, СВОЙСТВО, ДЕЙСТВИЕ. Синтаксис задания этих типов следующий:

1) СЛОВАРЬ ⟨ имя элемента языка ⟩
   ВХОД: $\mathcal{A}$
2) СВОЙСТВО ⟨ имя элемента языка ⟩
   ВХОД: $\mathcal{A}$
3) ДЕЙСТВИЕ ⟨ имя элемента языка ⟩
   ВХОД: $\mathcal{A}$
   ВЫХОД: $\mathcal{B}$

Возможны комбинированные типы, такие, как СЛОВАРЬ-ДЕЙСТВИЕ или СВОЙСТВО-ДЕЙСТВИЕ.

В зависимости от вида формального примера применяется одна из трех процедур синтеза семантического определения, либо их последовательность для комбинированных типов.

Процедура I. Построение последовательности действий. Задан формальный пример вида:

ВХОД : $\mathcal{A}$

ВЫХОД : $\mathcal{B}$

где $\mathcal{A} = A_1\#\ldots\#A_n$, $\mathcal{B} = B_1\#\ldots\#B_n$ и набор операций $F = \{f_1, f_2, \ldots, f_k\}$.

На набор операций F наложены следующие ограничения:

- области определения различных $f_i \in F$ не пересекаются;
- никакая последовательность из области определения операции f_i не является собственным префиксом никакой другой последовательности.

Каждая операция из F также должна быть описана в виде формального примера. Для каждой пары (A_i, B_i) строится последова-

тельность операций F_i и формальный пример преобразуется к виду:

ВХОД : $A_1 \# \ldots \# A_n$

ДЕЙСТВИЕ: $F_1 \# \ldots \# F_n$

где F_i - слово в алфавите F.

В случае произвольного набора операций задача является нетривиальной в вычислительном отношении и требует привлечения дополнительных критериев выбора последовательности операций, таких, например, как минимум памяти, требуемой для реализации последовательности, времени счета, сложности и т.п.

Процедура 2. Разбиение. Задан формальный пример, преобразованный к виду:

ВХОД : $A_1 \# \ldots \# A_n$

ДЕЙСТВИЕ: $F_1 \# \ldots \# F_n$

На множестве слов $A = \{A_1, \ldots, A_n\}$ определим отношение эквивалентности σ следующим образом: $A_i \sigma A_j \Leftrightarrow F_i = F_j$, т.е. последовательности операций совпадают. Отношение σ индуцирует на множестве A разбиение на классы эквивалентности, которые будем обозначать $\alpha_i (\alpha_i \in A/\sigma)$.

В результате применения процедуры 2 формальный пример преобразуется к виду:

ВХОД : $\alpha_1 \# \ldots \# \alpha_m$

ДЕЙСТВИЕ : $F'_1 \# \ldots \# F'_m$

где $F'_i \in \{F_1, F_2, \ldots, F_n\}$, $m \leq n$.

Процедура 3. Обобщение. Задан формальный пример, преобразованый к виду

ВХОД : $\alpha_1 \# \ldots \# \alpha_m$

ДЕЙСТВИЕ : $F'_1 \# \ldots \# F'_m$

Задача состоит в обобщении свойств последовательностей, попавших в один класс эквивалентности. Метод обобщений основан на выявлении закономерностей типа повторов и инверсий в символьных последовательностях. Для представления закономерности используется конструкция, называемая образом [7]. Образ - конечная цепочка над алфавитом $\Sigma \cup X$, где Σ - алфавит символьных последовательностей, $X = \{x_1, x_2, \ldots\}$ - счетное множество символов, называемых переменными ($X \cap \Sigma = \emptyset$). Число различных x_i, входящих в образ p, назы-

вается числом переменных (например, $p = abx_1x_2cx_1$ - образ с двумя переменными x_1 и x_2). Если p - образ, то язык образа L(p) есть множество элементов из Σ^*, получаемых из p подстановкой ненулевых цепочек s_i вместо каждого вхождения x_i в p для $i \geq 1$.

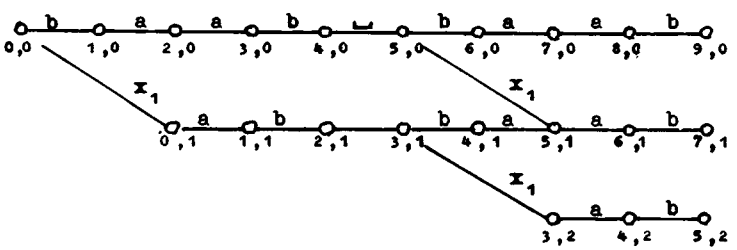
В [7] предложен алгоритм полиномиальной сложности, строящий по положительному множеству примеров, принадлежащих языку, образ с одной переменной. Доказано, что проблема построения образа с k переменными NP-полна.

В данной работе рассматривается частный случай построения образа с двумя переменными, когда они являются инверсиями друг друга (например, $x_1 = abc$, $x_2 = cba$).

Идея алгоритма состоит в сведении случая двух переменных к случаю одной переменной. Для этого для каждой последовательности, принадлежащей одному классу эквивалентности, строится последовательность $A_i' = A_i \sqcup A_i^r$, где A_i^r - инверсия A_i, а символ "$\sqcup$" не принадлежит Σ. Для каждой такой последовательности построим автомат $M(A_i', t)$ образа с одной переменной [7]. Состояния автомата будем обозначать парой (i,j), начальное состояние - (0,0). Функции переходов автомата определим следующим образом:

$\delta((i,j),b) = (i+1,j)$, если символ в позиции $i+j|t|+1$ есть b;
$\delta((i,j),x_1) = (i,j+1)$, если цепочка t начинается в позиции $i+j|t|+1$;
$\delta((i,j),\sqcup) = (i+1,j)$, если символ в позиции $i+j|t|+1$ есть $\sqcup$.

ПРИМЕР 4. Пусть задана последовательность A_1 = baab. Автомат образа с одной переменной $M(A_1', t)$ для t = ba будет иметь вид:



Далее аналогично случаю одной переменной строим автомат $B_r(i,j,k)$, где i - количество константных символов в p; j - общее количество переменных x_1 и x_2; k - позиция самого левого вхождения переменной x_1 в p.

Автомат $B_r(i,j,k)$ строим из автомата $M(A_1,t)$ следующим образом:

- удаляем все x_1-переходы из состояния $(u,0)$, где $u < k-1$;
- удаляем все константные переходы из состояния $(k-1,0)$;
- заменяем все константные переходы из состояния (i,j) длиной $|t|$ на x_2, если существует x_1-переход из состояния (i',j'), где $j' \geq j$, $i' = 2|A_1| - i - j|t| + 1$;
- удаляем все состояния (i,j), если $i+j|t| > |A_1|$.

Таким образом, случай двух переменных сводится к случаю одной переменной. Трудоемкость выполнения дополнительных операций построения автомата B_r пропорциональна числу состояний автомата, т.е. $O(n^2)$, где $n = |A_1|$.

ПРИМЕР 5. Множество примеров состоит из последовательностей {0II00, 0III0II0, I0III00I}. Образ, описывающий это множество, имеет вид: $p = x_1 11 x_1 x_2$.

4. Предложенные методы синтеза семантических определений реализованы в системе АЛИСА, осуществляющей автоматическую генерацию языковых процессоров по описанию синтаксиса языка в Р-языке и семантики множеством модулей или примеров.

Система АЛИСА имеет два уровня (рис.2). На первом уровне по описаниям синтаксиса и семантики языка генерируется языковый процессор транслирующего или интерпретирующего типа (в зависимости от заданной семантики). На втором уровне обрабатывается собственно программа пользователя, написанная на проблемно-ориентированном языке.

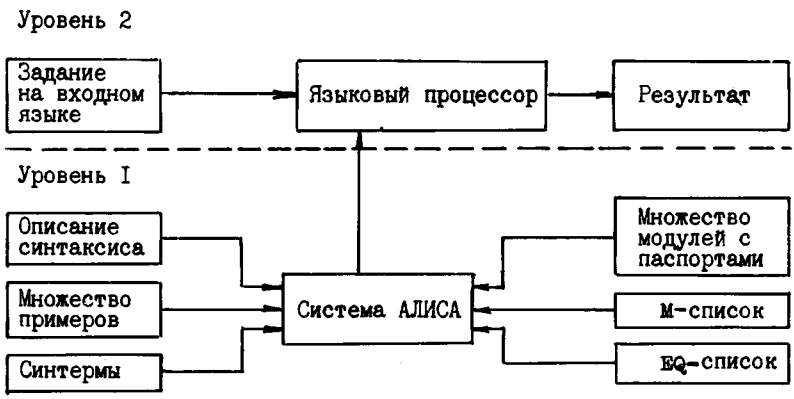


Рис. 2

Система АЛИСА реализована погружением в технологический комплекс программиста РТК ОС ЕС [I], что обеспечивает следующие функциональные характеристики системы:

- проектирование и использование языковых процессоров может осуществляться по безбумажной схеме в диалоговой среде;

- пояснение семантики множеством примеров или модулей может применяться наряду с семантическими определениями, написанными на одном из допустимых в РТК языков программирования;

- для описания синтаксиса и семантики языков могут использоваться средства РТК.

Допустимыми языками для написания модулей являются языки ПЛ/I, Фортран, РТРАН, Ассемблер. Преобразование разнотипных данных и создание соответствующей среды выполняется автоматически.

Языковые процессоры, сгенерированные в системе АЛИСА, представляются в языке РСТРАН [I].

5. Предложенный способ описания языковых процессоров позволяет упростить проектирование пакетов прикладных программ из уже имеющихся модулей, унифицировать структуру пакетов, легко пополнять пакеты новыми модулями и языковыми конструкциями, конструировать пользователю собственные проблемно-ориентированные языки.

Литература

I. ВЕЛЬБИЦКИЙ И.В., ХОДАКОВСКИЙ В.Н., ШОЛМОВ Л.И. Технологический комплекс производства программ на машинах ЕС ЭВМ и БЭСМ-6. -М.: Статистика, I980. - 263 с.

2. ЧУЖАНОВА Н.А. Индуктивный синтез Р-программ по образцам пар "вход-выход". -В кн.: Р-технология программирования и средства ее инструментальной поддержки. Применение Р-технологии программирования для решения прикладных задач (техническое описание).- Киев, ИК АН УССР, I982, с. 92-94.

3. ХОЛСТЕД М.Х. Начала науки о программах.-М.: Финансы и статистика, I980. -I28 с.

4. МОЩИЦКИЙ А.В. Сравнительная оценка языков программирования РТК ОС ЕС. -В кн.: Р-технология программирования; Тез. докл. I Всесоюз. конф. Ч.I. Средства автоматизации. -Киев, ИК АН УССР, I983, с. 79-82.

5. ГУСЕВ В.Д. и др. Пакет прикладных программ для анализа произвольных символьных последовательностей значительной длины (СИМОЛ). - Настоящий сборник, с. 3-2I.

6. КОСАРЕВ Ю.Г., ЧУЖАНОВА Н.А. Автоматический синтез алгоритмов по динамическим входным данным. -В кн.: Методы обработки информации (Вычислительные системы, вып. 74). Новосибирск, I978, с. 96-I07.

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

Поступила в ред.-изд.отд.
I4 ноября I983 года