Мартыненко Борис Константинович

УДК 519.682.1 +681.142.2

СИНТАКСИЧЕСКИЕ ДИАГРАММЫ Н. ВИРТА И ГРАФ-СХЕМЫ В SYNTAX-ТЕХНОЛОГИИ

Аннотация

Цель данной статьи – методологическая. Описывается опыт применения модифицированных синтаксических диаграмм Н. Вирта в SYNTAX-технологии, разработанной в связи с реализацией языка программирования Алгол 68 в начале 1970-х.

Ключевые слова: грамматики, синтаксические диаграммы Н. Вирта, граф-схемы, трансляции.

ВВЕДЕНИЕ

Среди календарных дат 2014 года есть одна, связанная с 80-летием Никлауса Вирта, автора языка программирования Паскаль [1], широко используемого в мире и в России для обучения программированию в старших классах школы и на первых курсах вузов. Вирт ввёл понятие синтаксических диаграмм, которое он использовал для наглядного представления структуры конструкций языка Паскаль. Впоследствии это понятие, родственное понятию графа, получило дальнейшие развитие и применение в качестве средства реализации систем обработки данных, управляемых синтаксисом языков.

Использование графов[1] в математике началось ещё задолго до появления компьютеров. А когда они появились в 1950-х годах и программирование велось в кодах вычислительной машины в условных адресах, то программисты часто применяли блок-схемы для документирования управляющей структуры машинных программ. Вскоре это привело к появлению теории схем программ[2] [2].

Плодотворно ставятся на графах и другие задачи программирования: оптимальное распределение памяти, исследование потока данных в программах, эквивалентные преобразования КС-грамматик, автоматическая генерация тестов, семантические сети Г.С. Цейтина [3] в задачах искусственного интеллекта и т. д.

В начале 1970-х группа сотрудников Вычислительного центра и лаборатории математической лингвистики Ленинградского государственного университета под научным руководством Г.С. Цейтина приступила к изучению языка программирования Алгол 68 [4], и вскоре

[1] Отцом теории графов (так же как и топологии) является Эйлер (1707–1782), решивший в 1736 г. широко известную в то время задачу, называвшуюся проблемой кёнигсбергских мостов.

[2] Схемы Янова — *схемы программ*, которые были введены в литературу А.А. Ляпуновым и Ю.И. Яновым в 1958 г. и направлены на разработку общей теории условий, управляющих последовательностью выполнений операторов в программе.

к его реализации на ЕС ЭВМ[1] по заказу Научно-исследовательского центра электронной вычислительной техники (НИЦЭВТ) Министерства радиоэлектронной промышленности СССР (отчёт об этом проекте *см*. в [5]). Язык Алгол 68 в то время ещё не был «заморожен»: поправки поступали в каждом номере Алгол-Бюллетеня, некоторые выпуски которого содержали вложения – микрофиши с описанием синтаксиса этого языка в виде синтаксических диаграмм Н. Вирта. Это привело к идее использовать синтаксические диаграммы Вирта как основы для внутреннего представления структуры предложений входного языка в системе его реализации.

Синтаксис Алгола 68 определяется двухуровневыми грамматиками А. ван Вейнгаардена, порождающими рекурсивно перечислимые множества, задача распознавания которых в общем случае алгоритмически не разрешима. Исходную грамматику можно было рассматривать лишь как первоначальную форму спецификации синтаксиса языка, то есть только как документ. Требовалось найти эффективный метод анализа языка Алгол 68 на основе других, более простых, классов грамматик и построить адекватный инструмент, его реализующий. Позднее он был воплощён в технологическом комплексе (ТК) SYNTAX [6].

1. ТРАНСЛЯЦИОННЫЕ RBNF-ГРАММАТИКИ

В SYNTAX-технологии трансляционные RBNF-грамматики являются двухуровневым формализмом для спецификации трансляций, который состоит из управляющей RBNF-грамматики, представляющей синтаксический уровень спецификации языка, и описания операционной среды, представляющего его семантический уровень.

Трансляционная грамматика есть формальная система $G_t = (G_c, \mathscr{E})$, где G_c – управляющая грамматика; $\mathscr{E}$ – описание операционной среды.

1.1. Управляющая RBNF-грамматика определяет синтаксис входного языка, вернее, способ его обработки (трансляции) через синтаксическую структуру его предложений отличается от обычной BNF-грамматики[2] тем, что в ней помимо алфавитов нетерминалов и терминалов имеются два дополнительных алфавита *контекстных символов*: *семантических* и *резольверных*, а правые части правил представляют собой регулярные выражения над символами всех алфавитов грамматики. Эти выражения трактуются как формулы с регулярными операциями над множествами цепочек, составленных из терминальных и контекстных символов. При этом считается, что терминальный или контекстный символ, используемый в качестве операнда регулярного выражения, представляет множество из одной односимвольной цепочки, а нетерминальный символ – множество[3] всех цепочек, представляющее его значение.

Таким образом, множество правил RBNF-грамматики рассматривается как своего рода «алгебраическая» система, неявно определяющая значения всех нетерминалов в качестве её неизвестных, в то время как все прочие символы, точнее одноэлементные множества, ими представляемые, играют роль её коэффициентов.

Множество цепочек, являющееся значением начального нетерминала, называется *синтаксическим управлением,* порождаемым данной управляющей RBNF-грамматикой, а каждая цепочка – элемент этого множества – *управляющей цепочкой.*

Если в управляющих цепочках игнорировать контекстные символы, мы получим бесконтекстную составляющую входного языка, то есть некоторый КС-надъязык над входным языком.

С другой стороны, интерпретация контекстных символов с каждым семантическим символом ассоциирует некоторое преобразование состояния операционной среды, а с резоль-

[1] Советский аналог серии System/360 и System/370 фирмы IBM, выпускавшихся в США с 1964 г.
[2] КС-грамматика, представленная в форме Бэкуса–Наура.
[3] Возможно, бесконечное.

верным символом – некоторый предикат, заданный на множестве состояний операционной среды.

Интерпретация управляющей цепочки состоит в исполнении преобразований, ассоциированных с составляющими её семантическими символами при условии, что предикаты, ассоциированные с её резольверными символами, выполняются[1]. Таким образом, предикаты «следят» за выполнением контекстных условий во входном предложении, а семантические преобразования операционной среды воплощают его «смысл».

Управляющая RBNF-грамматика есть формальная система $G_c = (N, T, \Re, \Sigma, P, S)$, где N – словарь нетерминальных символов или нетерминалов; T – словарь терминальных символов или терминалов; $\Re$ – словарь резольверных символов или резольверов; Σ – словарь семантических символов или семантик; $P = \{A: \mathscr{R}_A. \mid A \in N, \mathscr{R}_A$ – регулярное выражение относительно символов из множества $N \cup T \cup \Re \cup \Sigma$ – множество правил, S – начальный нетерминал.

Управляющая грамматика специфицирует *синтаксическое управление* – множество цепочек $\mathscr{C}(G_c) = \lambda(\mathscr{R}_S)$ над терминальными и контекстными символами (то есть символами из множества $T \cup \Re \cup \Sigma$, где $\lambda(\mathscr{R})$ определяется в зависимости от вида $\mathscr{R}$ следующим образом:

$$\lambda(\mathscr{R}) = \begin{cases} \{a\}, & \text{если } \mathscr{R} = a,\ a \in (T \cup \Re \cup \sum) \text{ или } a = \varepsilon; \\ \lambda(\mathscr{R}_A), & \text{если } \mathscr{R} = A,\ A \in N,\ \exists\, p \in P,\ p = A: \mathscr{R}_A.; \\ \bigcup\limits_{k=0}^{\infty} (\lambda(\mathscr{R}_1))^k, & \text{если } \mathscr{R} = \mathscr{R}_1{}^*; \\ \bigcup\limits_{k=1}^{\infty} (\lambda(\mathscr{R}_1))^k, & \text{если } \mathscr{R} = \mathscr{R}_1{}^+; \\ \lambda(\mathscr{R}_1) \bigcup\limits_{k=1}^{\infty} (\lambda(\mathscr{R}_2)\lambda(\mathscr{R}_1))^k, & \text{если } \mathscr{R} = \mathscr{R}_1 \# \mathscr{R}_2; \\ \lambda(\mathscr{R}_1)\, \lambda(\mathscr{R}_2), & \text{если } \mathscr{R} = \mathscr{R}_1,\ \mathscr{R}_2; \\ \lambda(\mathscr{R}_1) \cup \lambda(\mathscr{R}_2), & \text{если } \mathscr{R} = \mathscr{R}_1;\ \mathscr{R}_2; \\ \lambda(\mathscr{R}_1) \cup \{\varepsilon\}, & \text{если } \mathscr{R} = [\mathscr{R}_1]; \end{cases}$$

Здесь $\mathscr{R}_1$ и $\mathscr{R}_2$ – некоторые регулярные выражения. Определение вида регулярного выражения производится с учётом старшинства операций и расстановки скобок. Предполагается, что наивысшее старшинство имеют унарные операции *-Клини и $^+$-Клини, затем бинарные операции – итерация с разделителем (#), далее конкатенация (,) и, наконец, объединение (;).

1.2. Описание операционной среды. *Описанием операционной среды* называется формальная система $\mathscr{E} = (E, H, \mathscr{I}_\Re, \mathscr{I}_\Sigma, e_0)$, где E – *пространство состояний операционной среды* – область определения *предикатов*, ассоциированных с резольверными символами: $\mathscr{I}_\Re = \{\iota_\rho : E \to \{false, true\} \mid \rho \in \Re\}$, и *преобразований состояний операционной среды*, ассоциированных с семантическими символами: $\mathscr{I}_\Sigma = \{\iota_\sigma : E \to E \mid \sigma \in \Sigma\}$; H – *объектное подпространство*, то есть та часть операционной среды, состояние которой представляет особый интерес[2]; $e_0 \in E$ – *начальное состояние операционной среды*.

2. ТРАНСЛЯЦИЯ, ОПРЕДЕЛЯЕМАЯ ТРАНСЛЯЦИОННОЙ ГРАММАТИКОЙ

Пусть $e \in E$ – некоторое состояние операционной среды, $\rho \in \Re^*$ – некоторая резольверная цепочка, а $\sigma \in \Sigma^*$ – некоторая семантическая цепочка.

[1] Порядок синхронизации преобразований с вычислением предикатов определяется в разд. 3.

[2] Например, выходной файл.

Положим по определению

$$\iota_\rho(e) = \begin{cases} \iota_{\rho_1}(e)\ \&\ \iota_{\rho'}(e), \text{ если } \rho = \rho_1\rho',\ \rho_1 \in \Re,\ \rho' \in \Re^*, \\ true, \text{ если } \rho = \varepsilon; \end{cases}$$

$$\iota_\sigma(e) = \begin{cases} \iota_{\sigma'}(\iota_{\sigma_1}(e)), \text{ если } \sigma = \sigma_1\sigma',\ \sigma_1 \in \Sigma,\ \sigma' \in \Sigma^*, \\ e, \text{ если } \sigma = \varepsilon; \end{cases}$$

Здесь ι_{ρ_1} и ι_{σ_1} определяются описанием операционной среды, а $\iota_{\rho'}$ и $\iota_{\sigma'}$ – рекурсивные ссылки на соответствующие определения, приведённые выше.

Трансляционная грамматика определяет *трансляцию*:
$\tau(G_t) = \{(x, [e]_H) \mid \exists c_x(c_x \in \mathscr{C}(G_c),\ c_x = \kappa_0 a_1 \kappa_1 a_2 ... \kappa_{m-1} a_m \kappa_m$ – управляющая цепочка; $\kappa_i \in (\Re \cup \Sigma)^*$ $(i = 0, 1, 2, ..., m)$ – цепочки контекстных символов, в которых $\rho_i \in \Re^*$ – резольверные подцепочки, $\sigma_i \in \Sigma^*$ – семантические подцепочки; $a_j \in T$ $(j = 1, 2, ..., m)$ – терминальные символы; $x = a_1 a_2 ... a_m$ – входная цепочка (предложение входного языка); $e = \iota_{\sigma_m}(...\iota_{\sigma_1}(e_0))$ – финальное состояние операционной среды при условии, что $\iota_{\rho_0}(e_0)\ \&\ \iota_{\rho_1}(e_1)\ \&\ ...\ \&\ \iota_{\rho_m}(e_m)$ выполняется, где $e_{n+1} = \iota_{\sigma_n}(e_n)$ $(n = 0, 1, ..., m)$; очевидно, что $e = e_{m+1}$)}. Здесь $[e]_H$ обозначает проекцию точки $e \in E$ на объектное подпространство H – *результат трансляции* входной цепочки x.

Другими словами, трансляция есть множество пар, в которых первая компонента есть предложение входного языка, а вторая – проекция состояния операционной среды после её преобразований посредством семантик, входящих в соответствующую управляющую цепочку. Заметим, что управляющая цепочка и вместе с ней предложение входного языка определяются с учётом контекстных условий, задаваемых резольверами. При этом предполагается, что резольверы вычисляются *синхронно* с семантическими преобразованиями по порядку вхождений контекстных символов в управляющую цепочку.

Синхронизация означает, что сначала вычисляются предикаты, ассоциированные с резольверными символами, предшествующими терминалу a_1, в порядке их следования. Если все они выполняются, то исполняются преобразования состояния операционной среды, ассоциированные с семантическими символами, предшествующими терминалу a_1 (также в текстуальном порядке). Затем аналогичным порядком выполняются резольверы и семантики, предшествующие терминалу a_2, и т. д. Процесс заканчивается, когда успешно завершается вычисление конъюнкции предикатов, следующих за символом a_m, и исполнение соответствующих семантик.

Синтаксически правильными считаются лишь те входные цепочки, которые входят в состав управляющих цепочек, на которых выполняются все предикаты, ассоциированные с их резольверными символами. «Смысл» входного предложения x представляется семантической цепочкой $\sigma_1\sigma_2 ... \sigma_n$.

Если некоторому входному предложению соответствует несколько управляющих цепочек с разными семантическими составляющими, то такое предложение называется *семантически неоднозначным.* В SYNTAX-технологии семантическая неоднозначность не допускается[1]. Однако использование резольверов во многих случаях помогает избегать этой опасности.

В ТК SYNTAX управляющие грамматики записываются на языке TSL[2]. Именно по ним генерируются граф-схемы. Операционная среда описывается на языке программирования Borland Pascal 7.0 и после компиляции представляется в виде DLL-библиотеки.

Приведём пример трансляционной грамматики, иллюстрирующий спецификацию интерпретатора арифметических выражений (калькулятора).

[1] Фактически семантическая неоднозначность контролируется при построении управляющих таблиц процессоров (*см.* [6]).

[2] Описание языка TSL дано в [6, гл. 8].

CALC – трансляционная грамматика калькулятора

```
MICROLEXICS

Lexical classes: d, '.', 'e', '+', '-', '*', '/', '(', ')', EOF, Escaped Symbols.

d: '0' ..'9'.     Escaped Symbols: #32, #13, #10.     EOF: ''.

SYNTAX

Nonterminals: PROGRAM (ПРОГРАММА), EXPRESSION (ВЫРАЖЕНИЕ).

Terminals: 'd', '.', 'e', '+', '-', '*', '/', '(' , ')', 'EOF'.

Auxiliary notions: NUMBER (ЧИСЛО), INTEGER NUMBER (ЦЕЛОЕ),
FRACTIONAL PART (ДРОБНАЯ_ЧАСТЬ), EXPONENT PART (ПОРЯДОК), DIGIT (ЦИФРА),
MONADIC OPERATION (УНАРНАЯ_ОПЕРАЦИЯ), DYADIC OPERATION (БИНАРНАЯ_ОПЕРАЦИЯ),
OPERAND (ОПЕРАНД).

Forward pass semantics: Reset, Complete, Push Mon{adic} Op{eration}, Push Dyadic
Op{eration}, Push Op{eran}d, Push Open Par{enthisis}, Unload And Discard Open
Par{enthisis}, Init Int{eger Number}, App{end} Dig{it}, Set Int{eger} Part, App{end}
Exp{onent} Part, Set Fr{actional} Part, App{end} Fr{actional} Part, Set Exp{onent},
Set Dig{it}, Set Sign.

PROGRAM: Reset, EXPRESSION, Complete, 'EOF'.

EXPRESSION: ((MONADIC OPERATION)*, OPERAND) # DYADIC OPERATION.

OPERAND: NUMBER, Push Op{eran}d;
       Push Open Par{enthisis}, '(', EXPRESSION, Unload And Discard Open Par{enthisis}, ')'.
NUMBER: INTEGER NUMBER, Set Int{eger} Part, [FRACTIONAL PART, App{end} Fr{actional} Part],
                                          [EXPONENT PART, App{end} Exp{onent} Part].
INTEGER NUMBER: Init Int{eger Number}, (Set Dig{it}, DIGIT, App{end} Dig{it})+.
FRACTIONAL PART: '.', INTEGER NUMBER, Set Fr{actional} Part.
EXPONENT PART: 'e', Set Sign, (['+']; '-'), INTEGER NUMBER, Set Exp{onent}.
DIGIT: 'd' .
MONADIC OPERATION: Push Mon{adic} Op{eration}, ('+'; '-').
DYADIC OPERATION: Push Dyadic Op{eration}, ('+' ; '-' ; '*' ; '/').
```

Поясним, что спецификация трансляционной грамматики состоит из трёх разделов.

Раздел **MICROLEXICS** содержит информацию для построения транслитератора. Задача последнего – распределить символы входного алфавита по лексическим классам. Именно лексические классы, а не сами символы, управляют процессом обработки входного текста на стадии синтаксиса, описываемого в разделе **SYNTAX**.

Из словаря нетерминалов выделяются вспомогательные понятия (Auxiliary notions), если они определяются прямо или косвенно регулярными выражениями. Они вставляются в граф-схему механизмом макроподстановок.

Разделы **ENVIRONMENT** и **IMPLEMENTATION**, описывающие окружение и интерпретацию контекстных символов в этом окружении, здесь не показаны. Они, как сказано выше, преобразуются в DLL-библиотеку посредством транслятора на той платформе, на которой реализуется инструментальный комплекс SYNTAX-технологии.

3. СПЕЦИФИКАЦИЯ ТРАНСЛЯЦИЙ ПРИ ПОМОЩИ ТРАНСЛЯЦИОННЫХ ГРАФ-СХЕМ

В SYNTAX-технологии используется три различных формы спецификации трансляций, одна из которых (RBNF-грамматика) удобна для первоначального задания трансляций, другая (процессор) является формой реализации трансляций, а третья (трансляционная граф-

схема) есть промежуточная форма между двумя первыми. Эта форма, аналогичная синтаксическим диаграммам Вирта, может использоваться для описания синтаксиса языков и спецификации трансляций непосредственно, тем более что граф-схемы являются более гибким аппаратом для задания трансляций, чем грамматики[1].

Трансляционная граф-схема $\mathcal{G} = (\mathcal{G}_c, \mathscr{E})$, как и трансляционная грамматика, состоит из двух компонент: *управляющей граф-схемы* ($\mathcal{G}_c$) и *описания операционной среды* ($\mathscr{E}$), причём описание операционной среды ($\mathscr{E}$) наследуется непосредственно от трансляционной грамматики (*см*. разд. 1).

Управляющая граф-схема является графовым аналогом управляющей грамматики. Каждое правило RBNF-грамматики представляется в виде отдельного связного ориентированного графа, вершины которого помечены терминалами или нетерминалами грамматики, а дуги – цепочками, составленными из резольверных и семантических символов, с возможными петлями, циклами и параллельными дугами (так что уместнее было бы говорить о псевдомультиграфе).

Такой граф имеет две особые вершины: *начальную* и *конечную*. Особенность этих вершин в том, что в начальную вершину ни одна дуга не входит, а из конечной вершины ни одна дуга не выходит. Все другие, *внутренние вершины,* помечены терминальными или нетерминальными символами. В частности, дуги могут помечаться пустыми цепочками, и тогда они называются *непомеченными*. Если все дуги управляющей граф-схемы не помечены, то она называется *синтаксической граф-схемой.*

Пример граф-схемы, построенной по управляющей грамматике калькулятора до раскрытия вспомогательных понятий[2], показан на рис. 1. Пунктирными рамками изображены вспомогательные понятия (`Auxiliary notions`), которые раскрываются с помощью макроподстановок, перед тем как по управляющей граф-схеме строится процессор. Дуги граф-схем помечены контекстными символами, в то время как дуги синтаксических диаграмм Вирта никак не помечаются. Только в этом их внешнее различие.

В общем случае управляющая граф-схема – это несвязный помеченный граф, в котором каждая компонента связности представляет одно правило управляющей грамматики. Очевидно, что управляющая граф-схема как формальная система вполне аналогична управляющей грамматике с той лишь разницей, что правила имеют графовую форму, то есть $\mathcal{G}_c=(N, T, \mathfrak{R}, \Sigma, \mathrm{K}, S)$, где N – нетерминалы, T – терминалы, $\mathfrak{R}$ – резольверы, Σ – семантики, $\mathrm{K}=\{\mathrm{K}_A \mid A \in N, \mathrm{K}_A$ – компонента графа, определяющая нетерминал $A\}$, $S \in N$ – начальный нетерминал.

Компоненту K_S, определяющую начальный нетерминал S, назовем *заглавной компонентой* управляющей граф-схемы.

Как и управляющая грамматика, управляющая граф-схема порождает синтаксическое управление. Порождение управляющих цепочек реализуется с помощью *порождающих маршрутов* следующим образом.

Назовем маршрут в некоторой компоненте управляющей граф-схемы *полным*, если он начинается в начальной и заканчивается в конечной вершине этой компоненты. *Следом маршрута* назовем последовательность меток вершин и дуг, составляющих этот маршрут.

Рассмотрим некоторый полный маршрут в заглавной компоненте управляющей граф-схемы и определим его след. Каждое вхождение нетерминала в следе заменим на след некоторого полного маршрута в компоненте граф-схемы, определяющей этот нетерминал. Цепочку, получаемую в результате многократного повторения таких подстановок и не содержащую ни одного вхождения нетерминала, назовем *структурированной управляющей це-*

[1] Действительно, существуют граф-схемы, для которых невозможно построить прообраз управляющей грамматики, преобразование которого в граф-схему давало бы оригинальную граф-схему.

[2] На рис. 1 они показаны в пунктирных рамках.

почкой. Начальные и конечные метки придают ей структуру, соотносящую отдельные её фрагменты с компонентами, их породившими.

Отметим, что компоненты используются лишь, после того, как в них раскрыты все вспомогательные понятия с помощью макроподстановок.

Фактически используется линейное представление «раскрытых» диаграмм в виде массива записей из двух полей: тип записи и информационное поле, соответствующее этому типу.

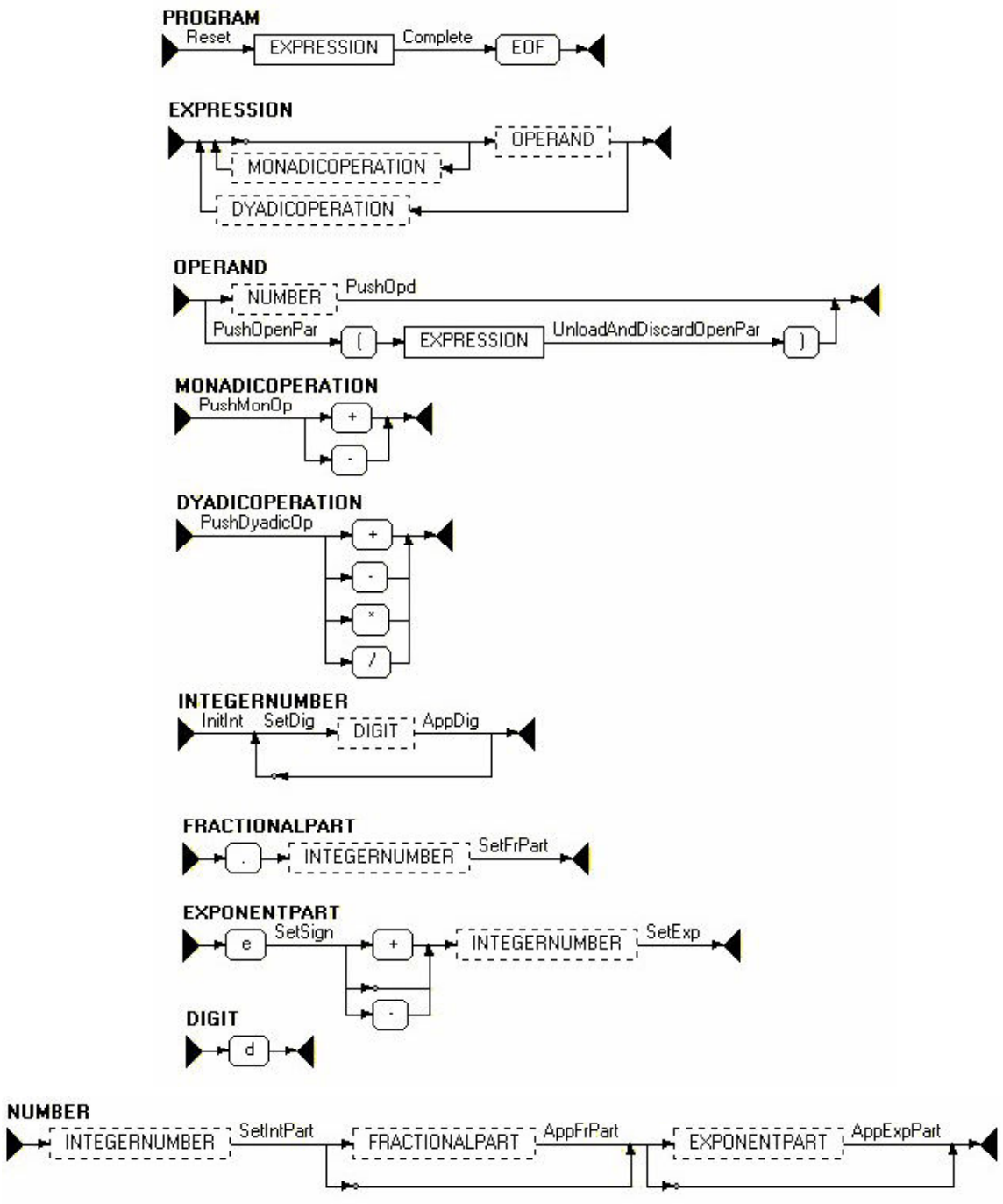

Рис. 1. Представление управляющей граф-схемы, построенной по грамматике CALC, до раскрытия вспомогательных понятий

Машинное (линейное) представление графа есть последовательность занумерованных записей. Такая запись состоит из двух полей, одно из которых определяет её тип, а другое – информационное.

Записи типа «begin» и «end» представляют начальную и конечную вершину соответственно. Их информационные поля содержат определяемый данной компонентой нетерминальный символ.

Записи типа T и N представляют терминальные и нетерминальные вершины, а их информационные поля содержат символы соответствующих словарей.

Типы записей FR, BR, FS и BS — представляют соответственно резольверы и семантики прямого и обратного просмотров.

Записи типа «разветвление» (-<) и «переход» (→), в которых информационные поля представлены номерами других записей, определяют возможный порядок обхода компонент управляющей граф-схемы.

Записи типа { и }, ограничивающие макровставки, используются для генерации диагностических сообщений о бесконтекстных ошибках во входных цепочках.

Для рассматриваемого примера калькулятора линейное представление граф-схемы имеет вид:

```
0    begin   PROGRAM
1    FS      Reset
2    N       EXPRESSION
3    FS      Complete
4    T       'EOF'
5    end     PROGRAM
6    begin   EXPRESSION
7    -<      16
8    {       MONADIC OPERATION
9    FS      PushMonOp
10   -<      13
11   T       '+'
12   →       14
13   T       '-'
14   }       MONADIC OPERATION
15   →       7
16   {       OPERAND
17   -<      68
18   {       NUMBER
19   {       INTEGER NUMBER
20   FS      InitInt
21   FS      SetDig
22   {       DIGIT
23   T       'd'
24   }       DIGIT
25   FS      AppDig
26   -<      21
27   }       INTEGER NUMBER
28   FS      SetIntPart
29   -<      44
30   {       FRACTIONAL PART
31   T       '.'
32   {       INTEGER NUMBER
33   FS      InitInt
34   FS      SetDig
35   {       DIGIT
36   T       'd'
37   }       DIGIT
38   FS      AppDig
39   -<      34
40   }       INTEGER NUMBER
41   FS      SetFrPart
42   }       FRACTIONAL PART
43   FS      AppFrPart
44   -<      65
45   {       EXPONENT PART
46   T       'E'
47   FS      SetSign
48   -<      52
49   -<      51
50   T       '+'
51   →       53
52   T       '-'
53   {       INTEGER NUMBER
54   FS      InitInt
55   FS      SetDig
56   {       DIGIT
57   T       'd'
58   }       DIGIT
59   FS      AppDig
60   -<      55
61   }       INTEGER NUMBER
62   FS      SetExp
63   }       EXPONENT PART
64   FS      AppExpPart
65   }       NUMBER
66   FS      PushOpd
67   →       73
68   FS      PushOpenPar
69   T       '('
70   N       EXPRESSION
71   FS      UnloadDiscardOpenPar
```

```
72    T       ')'                    81    T       '-'
73    }       OPERAND                82    →       87
74    -<      89                     83    -<      86
75    {       DIADIC OPERATION       84    T       '*'
76    FS      PushDyadicOp           85    →       87
77    -<      80                     86    T       '/'
78    T       '+'                    87    }       DIADIC OPERATION
79    →       87                     88    →       7
80    -<      83                     89    end     EXPRESSION
```

4. ЧЕЛНОЧНЫЙ СПЛАЙНОВЫЙ ПРОЦЕССОР

Челночным сплайновым процессором назовем формальную систему $\mathscr{P}_s = (\mathscr{P}_c^f, \mathscr{P}_c^b, E)$, состоящую из управляющего сплайнового процессора прямого просмотра $\mathscr{P}_c^f$, управляющего сплайнового процессора обратного просмотра $\mathscr{P}_c^b$ и операционной среды E, компилируемой по её описанию $\mathscr{E}$.

4.1. Прямой просмотр. *Управляющим сплайновым процессором прямого просмотра* назовем формальную систему $\mathscr{P}_c^f = (Q^f, \Sigma^f, \Gamma^f, \Delta^f, \mathfrak{R}^f, \delta^f, q^f_0, F^f)$, в которой Q^f – *множество состояний управления прямого просмотра*; Σ^f – *входной алфавит прямого просмотра*; Γ^f – *алфавит магазинных символов прямого просмотра*; Δ^f – *алфавит семантических символов прямого просмотра*; $\mathfrak{R}^f$ – *алфавит резольверных символов прямого просмотра*; $\delta^f = (\delta^f_1, \delta^f_2, \delta^f_3)$ – *управляющая таблица прямого просмотра*, состоящая из *таблицы резольверов прямого просмотра*: δ^f_1: $Q^f \times (\Sigma^f \cup \{\varepsilon\}) \to 2^{\mathfrak{R}^{f*}}$, *таблицы управляющих элементов прямого просмотра*: δ^f_2: $Q^f \times (\Sigma^f \cup \{\varepsilon\}) \times \mathfrak{R}^{f*} \to (Q^f \cup \{\mathbf{Sup}\}) \times \Gamma^{f*} \times \Delta^{f*}$, и *таблицы возвратных состояний*: δ^f_3: $Q^f \times \Gamma^f \times \mathfrak{R}^{f*} \to Q^f$; q^f_0 – *начальное состояние прямого просмотра*; $F^f \subseteq Q^f$ – *множество конечных состояний прямого просмотра*.

4.2. Обратный просмотр. *Управляющим сплайновым процессором обратного просмотра* назовем формальную систему $\mathscr{P}_c^b = (Q^b, \Sigma^b, \Gamma^b, \Delta^b, \mathfrak{R}^b, \delta^b, q^b_0, F^b)$, в которой Q^b – *множество состояний управления обратного просмотра*; Σ^b – *входной алфавит обратного просмотра*; Γ^b – *алфавит магазинных символов обратного просмотра*; Δ^b – *алфавит семантических символов обратного просмотра*; $\mathfrak{R}^b$ – *алфавит резольверных символов обратного просмотра*; $\delta^b = (\delta^b_1, \delta^b_2)$ – *управляющая таблица обратного просмотра*, в которой δ^b_1: $Q^b \times (\Sigma^b \cup \{\varepsilon\}) \to 2^{\mathfrak{R}^{b*}}$ – *таблица резольверов обратного просмотра*, а δ^b_2: $Q^b \times (\Sigma^b \cup \{\varepsilon\}) \times \mathfrak{R}^{b*} \to (Q^b \cup \{\mathbf{Pop}\}) \times (\Gamma^b \cup \{\varepsilon\}) \times \Delta^{b*}$ – *таблица управляющих элементов обратного просмотра*; q^b_0 – *начальное состояние обратного просмотра*; $F^b \subseteq Q^b$ – *множество конечных состояний обратного просмотра*.

Заметим, что управляющая таблица обратного просмотра состоит только из таблицы резольверов и таблицы управляющих элементов, тогда как таблица возвратных состояний не используется. Это связано с тем, что в случае ε-движения возвратное состояние снимается непосредственно с вершины магазина. Особенностью обратного просмотра является и то, что в магазин помещается не более одного магазинного символа одновременно.

4.3. Операционная среда. Понятие операционной среды, включающей интерпретацию контекстных (то есть резольверных и семантических резольверных) символов, определённое для трансляционных грамматик, используется в сплайновых процессорах с учётом того, что $\mathfrak{R} = \mathfrak{R}^f \cup \mathfrak{R}^b$, $\mathfrak{R}^f \cap \mathfrak{R}^b = \varnothing$, $\Delta = \Delta^f \cup \Delta^b$, $\Delta^f \cap \Delta^b = \varnothing$[1], и что контекстные символы из $\mathfrak{R}^f$ и

[1] В трансляционных грамматиках для обозначения семантических символов вместо символа Δ используется символ Σ, причём предполагается, что $\Sigma = \Sigma^f \cup \Sigma^b$ и $\Sigma^f \cap \Sigma^b = \varnothing$.

Δ^f интерпретируются на прямом просмотре, а резольверы и семантики из $\Re^b$ и Σ^b – на обратном.

5. ФУНКЦИОНИРОВАНИЕ ЧЕЛНОЧНОГО СПЛАЙНОВОГО ПРОЦЕССОРА

Работу челночного сплайнового процессора опишем в терминах его конфигураций.

5.1. Прямой просмотр действует с начальной конфигурации $(q^f_0, x, \varepsilon, e^f_0)$, где $q^f_0 \in Q^f$ – начальное состояние его конечного управления; $x \in \Sigma^{f*}$ – входная цепочка; ε означает, что первоначально магазин пуст; e^f_0 – начальное состояние операционной среды.

Пусть $(q^f_1, ax, \alpha, e^f_1)$ – текущая конфигурация прямого просмотра, в которой $q^f_1 \in Q^f$ – текущее состояние его управления; $ax \in \Sigma^{f*}$ – необработанная часть входной цепочки, где $a \in \Sigma^f$ – текущий входной символ, $x \in \Sigma^{f*}$ – остаток входной цепочки; $\alpha \in \Gamma^{f*}$ – текущая магазинная цепочка, причём считается, что на вершине магазина находится крайний левый символ цепочки α; e^f_1 – текущее состояние операционной среды. Очередное движение определяется управляющей таблицей прямого δ^f.

Случай 1. $\delta^f_1(q^f_1, a) \neq \varnothing$ – существуют контексты приёма входного символа a.

Случай 1.1. $\forall(\rho \in \delta^f_1(q^f_1, a)): (\sim\iota_\rho(e^f_1))$.

– Диагностируется контекстная ошибка по входному символу a.

Случай 1.2. $\exists(\rho, \rho' \in \delta^f_1(q^f_1, a)): ((\rho \neq \rho') \,\&\, (\rho \neq \varepsilon) \,\&\, (\rho' \neq \varepsilon) \,\&\, \iota_\rho(e^f_1) \,\&\, \iota_{\rho'}(e^f_1))$.

– Диагностируется контекстная неоднозначность по входному символу a.

Случай 1.3. $(\delta^f_1(q^f_1, a) = \{\rho\}) \,\&\, (\rho \in \Re^{f*}) \,\&\, \iota_\rho(e^f_1)$ – приём a.

Пусть $\delta^f_2(q^f_1, a, \rho) = (q^f_2, \gamma, \sigma)$, где $q^f_2 \in Q^f$ – *переходное* состояние, $\gamma \in \Gamma^{f*}$ – магазинная цепочка, $\sigma \in \Delta^{f*}$ – семантическая цепочка. Тогда

$$(q^f_1, ax, X\alpha, e^f_1) \underset{\mathscr{P}^f_c}{\vdash} (q^f_2, x, \gamma X\alpha, e^f_2).$$

После этого новая конфигурация анализируется сначала.

Случай 2. $\delta^f_1(q^f_1, a) = \varnothing$ – не существуют контексты приёма a.

Случай 2.1. $\delta^f_1(q^f_1, \varepsilon) \neq \varnothing$ – существуют контексты приёма ε.

Случай 2.2.1. $\forall(\rho \in \delta^f_1(q^f_1, \varepsilon)) : (\sim\iota_\rho(e^f_1))$.

– Диагностируется контекстная ошибка по ε-движениям.

Случай 2.2.2. $\exists(\rho, \rho' \in \delta^f_1(q^f_1, \varepsilon)) : ((\rho \neq \rho') \,\&\, (\rho \neq \varepsilon) \,\&\, (\rho' \neq \varepsilon) \,\&\, \iota_\rho(e^f_1) \,\&\, \iota_{\rho'}(e^f_1))$.

– Диагностируется контекстная неоднозначность выбора ε-движения.

Случай 2.2.3. $(\delta^f_1(q^f_1, \varepsilon) = \{\rho\}) \,\&\, (\rho \in \Re^{f*}) \,\&\, \iota_\rho(e^f_1)$.

Движение процессора однозначно определяется управляющим элементом $\delta^f_2(q^f_1, \varepsilon, \rho)$.

Пусть $\delta^f_2(q^f_1, \varepsilon, \rho) = (\mathbf{Sup}^1, \gamma, \sigma)$, где $\gamma \in \Gamma^{f*}$, $\sigma \in \Delta^{f*}$. Процессор выполняет переход вида

$$(q^f_1, ax, X\alpha, e^f_1) \underset{\mathscr{P}^f_c}{\vdash} (q^f_1, ax, \gamma X\alpha, e^f_2).$$ Здесь $e^f_2 = \iota_\sigma(e^f_1)$.

Пусть $\gamma X\alpha = Z\beta$, где $Z \in \Gamma^f$, $\beta \in \Gamma^{f*}$. Тогда фактически $(q^f_1, ax, \gamma X\alpha, e^f_2) = (q^f_1, ax, Z\beta, e^f_2)$. Если при этом $\delta^f_3(q^f_1, Z, \rho) = q^f_2$, то процессор совершает ещё один шаг перехода вида

$$(q^f_1, ax, Z\beta, e^f_2) \underset{\mathscr{P}^f_c}{\vdash} (q^f_2, ax, \beta, e^f_2).$$

Новое состояние q^f_2 называется *возвратным*, а исходное – *подавляемым*. Далее анализ новой конфигурации начинается сначала при том же текущем входном символе.

---

[1] Для ε-движения вместо переходного состояния всегда даётся специальное значение **Sup** (**Suppress**), означающее, что следующее (возвратное) состояние надо определять по таблице δ^f_3.

Прямой просмотр заканчивает свою работу тогда, когда он достигает одного из своих конечных состояний при пустом магазине. Вся последовательность движений прямого просмотра может быть представлена следующим образом:

$$(q^{\mathrm{f}}_0, x, \varepsilon, e^{\mathrm{f}}_0) \overset{*}{\underset{\mathscr{P}^{\mathrm{f}}_{\mathrm{c}}}{\vdash}} (p^{\mathrm{f}}, \varepsilon, \varepsilon, e^{\mathrm{f}}), \text{ где } p^{\mathrm{f}} \in F^{\mathrm{f}}.$$

Достигнутое состояние операционной среды e^{f} является начальным для обратного просмотра. Подразумевается, что вся последовательность состояний управления прямого просмотра регистрируется на его выходе и передаётся на вход обратного просмотра.

5.2. Обратный просмотр, сканируя в обратном порядке последовательность состояний управления прямого просмотра, продолжает изменение состояний операционной среды, начиная со своей начальной конфигурации, которая имеет вид

$$(q^{\mathrm{b}}_0, p^{\mathrm{f}} \ldots q^{\mathrm{f}}_0, \varepsilon, e^{\mathrm{f}}).$$

Пусть $(q^{\mathrm{b}}_1, q^{\mathrm{f}}_k, q^{\mathrm{f}}_{k-1} \ldots q^{\mathrm{f}}_0, \alpha, e^{\mathrm{b}}_1)$ – текущая конфигурация обратного просмотра. Очередное движение обратного просмотра определяется его управляющей таблицей δ^{b} в зависимости от ситуации.

Случай 1. $\delta^{\mathrm{b}}_1(q^{\mathrm{b}}_1, q^{\mathrm{f}}_k) = \varnothing$ – существуют контексты приёма q^{f}_k.

Случай 1.1. $\forall\, (\rho \in \delta^{\mathrm{f}}_1(q^{\mathrm{b}}_1, q^{\mathrm{f}}_k))\colon (\sim\iota_\rho(e^{\mathrm{b}}_1))$.

– Диагностируется контекстная ошибка по состоянию прямого просмотра q^{f}_k.

Случай 1.2. $\exists(\rho, \rho' \in \delta^{\mathrm{b}}_1(q^{\mathrm{b}}_1, q^{\mathrm{f}}_k))\colon ((\rho \neq \rho') \,\&\, (\rho \neq \varepsilon) \,\&\, (\rho' \neq \varepsilon) \,\&\, \iota_\rho(e_1{}^{\mathrm{b}}) \,\&\, \iota_{\rho'}(e_1{}^{\mathrm{b}}))$.

– Диагностируется контекстная неоднозначность по состоянию прямого просмотра q^{f}_k.

Случай 1.3. $(\delta^{\mathrm{b}}_1(q^{\mathrm{b}}_1, q^{\mathrm{f}}_k) = \{\rho\}) \,\&\, (\rho \in \Re^{\mathrm{b}*}) \,\&\, \iota_\rho(e^{\mathrm{b}}_1)$.

Пусть $\delta^{\mathrm{b}}_2(q^{\mathrm{b}}_1, q^{\mathrm{f}}_k, \rho) = (q^{\mathrm{b}}_2, \gamma, \sigma)$, где $q^{\mathrm{b}}_2 \in Q^{\mathrm{b}}$ – *переходное состояние*, $\gamma \in (\Gamma^{\mathrm{b}} \cup \{\varepsilon\})$ – магазинная цепочка[1], $\sigma \in \Delta^{\mathrm{b}*}$ — семантическая цепочка. В терминах конфигураций имеем

$$(q^{\mathrm{b}}_1, q^{\mathrm{f}}_k q^{\mathrm{f}}_{k-1} \ldots q^{\mathrm{f}}_0, \alpha, e^{\mathrm{b}}_1) \underset{\mathscr{P}^{\mathrm{f}}_{\mathrm{c}}}{\vdash} (q^{\mathrm{b}}_2, q^{\mathrm{f}}_{k-1} \ldots q^{\mathrm{f}}_0, \gamma\alpha, e^{\mathrm{b}}_2).$$

После этого новая конфигурация анализируется сначала.

Случай 2. $\delta^{\mathrm{b}}_1(q^{\mathrm{b}}_1, q^{\mathrm{f}}_k) = \varnothing$ – не существуют контексты приёма q^{f}_k.

Случай 2.1. $\delta^{\mathrm{b}}_1(q^{\mathrm{b}}_1, \varepsilon) \neq \varnothing$ – существуют контексты приёма ε.

Случай 2.2.1. $\forall\, (\rho \in \delta^{\mathrm{b}}_1(q^{\mathrm{b}}_1, \varepsilon))\colon (\sim\iota_\rho(e^{\mathrm{b}}_1))$.

– Диагностируется контекстная ошибка по ε.

Случай 2.2.2. $\exists(\rho, \rho') \in \delta^{\mathrm{b}}_1(q^{\mathrm{b}}_1, \varepsilon)\colon ((\rho \neq \rho') \,\&\, (\rho \neq \varepsilon) \,\&\, (\rho' \neq \varepsilon) \,\&\, \iota_\rho(q^{\mathrm{b}}_1) \,\&\, \iota_{\rho'}(e^{\mathrm{b}}_1))$.

– Диагностируется контекстная неоднозначность выбора ε-движения.

Случай 2.2.3. $(\delta^{\mathrm{b}}_1(q^{\mathrm{b}}_1, \varepsilon) = \{\rho\}) \,\&\, (\rho \in \Re^{\mathrm{b}*}) \,\&\, \iota_\rho(e^{\mathrm{b}}_1)$.

Пусть $\delta^{\mathrm{b}}_2(q^{\mathrm{b}}_1, \varepsilon, \rho) = (\mathbf{Pop}^2, \gamma, \sigma)$, где $\gamma \in (\Gamma^{\mathrm{b}} \cup \{\varepsilon\})$, $\sigma \in \Delta^{\mathrm{b}*}$. В этом случае выполняется переход вида

$$(q^{\mathrm{b}}_1, q^{\mathrm{f}}_k q^{\mathrm{f}}_{k-1} \ldots q^{\mathrm{f}}_0, \alpha, e^{\mathrm{b}}_1) \underset{\mathscr{P}^{\mathrm{f}}_{\mathrm{c}}}{\vdash} (q^{\mathrm{b}}_1, q^{\mathrm{f}}_k q^{\mathrm{f}}_{k-1} \ldots q^{\mathrm{f}}_0, \gamma\alpha, e^{\mathrm{b}}_2), \text{ где } e^{\mathrm{b}}_2 = \iota_\sigma(e^{\mathrm{b}}_1).$$

Пусть магазинная цепочка, образовавшаяся после записи γ над верхним символом магазина, есть $\gamma\alpha = X\beta$. Тогда имеем

$$(q^{\mathrm{b}}_1, q^{\mathrm{f}}_k q^{\mathrm{f}}_{k-1} \ldots q^{\mathrm{f}}_0, \gamma\alpha, e^{\mathrm{b}}_2) = (q^{\mathrm{b}}_1, q^{\mathrm{f}}_k q^{\mathrm{f}}_{k-1} \ldots q^{\mathrm{f}}_0, X\beta, e^{\mathrm{b}}_2).$$

Далее процессор совершает ещё один шаг перехода, а именно:

---

[1] Односимвольная или пустая.

[2] Для ε-движения вместо переходного состояния всегда даётся специальное значение **Pop**. Следующее (возвратное) состояние следует брать с вершины магазина.

$$(q^{\mathrm{b}}{}_1, q^{\mathrm{f}}{}_k q^{\mathrm{f}}{}_{k-1} \ldots q^{\mathrm{f}}{}_0, X\beta, e^{\mathrm{b}}{}_2) \underset{\mathscr{P}^{\mathrm{f}}_{\mathrm{c}}}{\vdash} (q^{\mathrm{b}}{}_2, q^{\mathrm{f}}{}_k q^{\mathrm{f}}{}_{k-1} \ldots q^{\mathrm{f}}{}_0, \beta, e^{\mathrm{b}}{}_2), \text{ где } q^{\mathrm{b}}{}_2 = X.$$

Это последнее движение затрачивает верхний символ магазина, но текущий входной символ всё ещё остаётся не принятым. Новое состояние $q^{\mathrm{b}}{}_2$ называется *возвратным*, а исходное $q^{\mathrm{b}}{}_1$ – *подавляемым*.

Далее разбор новой конфигурации начинается по той же схеме и при том же текущем входном символе.

Вся последовательность движений обратного просмотра может быть представлена следующим образом:

$$(q^{\mathrm{b}}{}_0, p^{\mathrm{f}} \ldots q^{\mathrm{f}}{}_0, \varepsilon, e^{\mathrm{f}}) \underset{\mathscr{P}^{\mathrm{f}}_{\mathrm{c}}}{\overset{*}{\vdash}} (p^{\mathrm{b}}, \varepsilon, \varepsilon, e^{\mathrm{b}}).$$

Здесь $p^{\mathrm{b}} \in F^{\mathrm{b}}$ – конечное состояние обратного просмотра, прочитана вся последовательность состояний прямого просмотра и магазин пуст.

Итак, *трансляцией, реализуемой челночным сплайновым процессором,* которую мы также будем называть *челночной трансляцией,* называется множество пар:

$$\tau(\mathscr{P}_{\mathrm{s}}) = \{(x, [e^{\mathrm{b}}]_{\mathrm{H}}) \mid (q^{\mathrm{f}}{}_0, x, \varepsilon, e^{\mathrm{f}}{}_0) \underset{\mathscr{P}^{\mathrm{f}}_{\mathrm{c}}}{\overset{*}{\vdash}} (p^{\mathrm{f}}, \varepsilon, \varepsilon, e^{\mathrm{f}}), p^{\mathrm{f}} \in F^{\mathrm{f}},$$

$$(q^{\mathrm{b}}{}_0, p^{\mathrm{f}} \ldots q^{\mathrm{f}}{}_0, \varepsilon, e^{\mathrm{f}}) \underset{\mathscr{P}^{\mathrm{f}}_{\mathrm{c}}}{\overset{*}{\vdash}} (p^{\mathrm{b}}, \varepsilon, \varepsilon, e^{\mathrm{b}}), p^{\mathrm{b}} \in F^{\mathrm{b}}\}.$$

ЗАКЛЮЧЕНИЕ

Описанный здесь метод трансляции использует RBNF-грамматики, подобные КС-грамматикам, правила которых определяют терминальные порождения через регулярные выражения относительно символов алфавитов грамматики и вновь введённых контекстных символов – семантик и резольверов (предикатов). С каждым семантическим символом ассоциируется некоторое преобразование операционной среды – пространства данных, составляющих контекст, а с каждым из резольверных символов – предикат, заданный в том же пространстве. Такая грамматика порождает язык, аппроксимируемый регулярными сплайнами.

Соответствующий языковой процессор подобен множеству конечных автоматов, каждый из которых распознает свой фрагмент входной цепочки – регулярный сплайн и обрабатывает его с учётом состояния контекста.

Известно [7], что структура любого предложения КС-языка может быть представлена в виде дерева вывода, отображающего его на правила грамматики. В описываемом методе используется другая постановка задачи синтаксического анализа: по данному предложению КС-языка выделить из леса деревьев, порождающих язык, то дерево, результат которого равен этому предложению.

Внутренней задачей транслятора является реконструкция управляющей цепочки по предложению входного языка и её интерпретация. Эта реконструкция равносильна нахождению в управляющей граф-схеме маршрутов, порождающих данное предложение входного языка. Эти маршруты начинаются в начальной вершине заглавной компоненты, заканчиваются в конечной её вершине и проходят через некоторые другие компоненты управляющей граф-схемы. Заход в другие компоненты (или повторное прохождение той же самой компоненты) связано с прохождением нетерминальных вершин. Дуги, составляющие порождающий маршрут, несут на себе контекстные условия, при выполнении которых только и возможен проход по ним, а также задают преобразования текущего состояния операционной среды, выполняемые при их проходе.

Эта задача нахождения порождающего маршрута решается в два просмотра, один из которых – прямой – сканирует входную цепочку в прямом направлении, а другой – обратный – сканирует последовательность состояний конечного управления прямого просмотра

в качестве своей входной цепочки. Каждый из этих просмотров использует магазин[1]. Естественно, контекстные символы относятся к соответствующим просмотрам. Собственно трансляция осуществляется семантическими процедурами при учёте контекста предикатами, интерпретирующими резольверные символы.

Построение и оптимизация челночного сплайнового процессора, описанного в разд. 4, производится по управляющей грамматике, представленной в виде граф-схемы с ограничениями, гарантирующими его детерминизм. Эти ограничения формулируются в терминах граф-схемы и учитываются во время его построения [6].

Класс языков, определяемых RBNF-граф-схемами через порождающие маршруты, есть LL(1) и, следовательно, имеет линейную оценку сложности по времени относительно длины входного предложения [7].

Технология реализована в виде пакета инструментальных программ, названного технологическим комплексом (ТК) SYNTAX [6]. Он достаточно полно инструментирует все основные этапы разработки средств синтаксически управляемой обработки данных (СУОД), таких как анализаторы языков программирования, интерпретаторы и компиляторы, конверторы, синтаксические редакторы и т. д.

SYNTAX-технология была успешно использована при реализации ряда языков программирования, таких как Алгол 68, Ада, Паскаль, при написании конвертора Reduce 2 → Mathematica, а также при реализации самого языка спецификации трансляций TSL – входного языка ТК SYNTAX.

В течение ряда лет ТК SYNTAX использовался в спецсеминаре по технологии трансляции на математико-механическом факультете СПбГУ.

Литература

1. *Wirth N.* The Programming Language Pascal. Acta Informatica, 1971. Vol. 1. P. 35–63.

2. *Янов Ю.И.* О логических схемах алгоритмов // Проблемы кибернетики, 1958. Вып. 1. С. 75–127.

3. *Цейтин Г.С.* Программирование посредством ассоциативных сетей. В: ЭВМ в проектировании и производстве. Вып. 2 / Под ред. Г.В. Орловский. Л.: «Машиностроение», 1985, С. 16–48.

4. Пересмотренное сообщение об Алголе 68 / Под ред. А. Ван Вейнгаарден, Б. Майу, Дж. Пек, К. Костер, М. Синцов, Ч. Линдси, Л. Меертенс, Р. Фискер. М.: «Мир», 1979.

5. Алгол 68. Методы реализации / Под ред. Г.С. Цейтина. Л.: Изд-во Ленингр. ун-та, 1976.

6. *Мартыненко Б.К.* Синтаксически управляемая обработка данных. СПб.: Изд-во СПбГУ, 2004.

7. *Ахо А., Ульман Дж.* Теория синтаксического анализа, перевода и компиляции. Т. 1. Синтаксический анализ; Т. 2. Компиляция. М.: Мир, 1978.

Приложение

УПРАВЛЯЮЩАЯ ТАБЛИЦА КАЛЬКУЛЯТОРА

Табл. 1. δ^f_2

Вход	Семантика	Магазин	Состояние
1	**2**	**3**	**4**
State 1={0}			
d	Reset;InitInt;SetDig	1	2
+	Reset;PushMonOp	1	3
–	Reset;PushMonOp	1	4
(	Reset;PushOpenPar	1	5

[1] Если граф-схема не однокомпонентная.

Вход	Семантика	Магазин	Состояние
1	**2**	**3**	**4**
State 2={23}			
ε	AppDig; SetIntPart; PushOpd		**Sup**
d	AppDig;SetDig		2
.	AppDig;SetIntPart		6
E	AppDig;SetIntPart		7
+	AppDig; SetIntPart; PushOpd; PushDyadicOp		8
–	AppDig; SetIntPart; PushOpd; PushDyadicOp		9
*	AppDig; SetIntPart; PushOpd; PushDyadicOp		10
/	AppDig; SetIntPart; PushOpd; PushDyadicOp		11
State 3={11}; 4={13}; 8={78}; 9={81}; 10={84}; 11={86}			
d	InitInt;SetDig		2
+	PushMonOp		3
–	PushMonOp		4
(	PushOpenPar		5
State 5={69}			
d	InitInt;SetDig	2	2
+	PushMonOp	2	3
–	PushMonOp	2	4
(	PushOpenPar	2	5
State 6={31}			
d	InitInt;SetDig		12
State 7={46}			
d	SetSign;InitInt;SetDig		13
+	SetSign		14
–	SetSign		15
State 12 = {36}			
ε	AppDig;SetFrPart;AppFrPart;PushOpd		**Sup**
d	AppDig;SetDig		12
E	AppDig;SetFrPart;AppFrPart		7
+	AppDig;SetFrPart;AppFrPart;PushOpd;PushDyadicOp		8
–	AppDig;SetFrPart;AppFrPart;PushOpd;PushDyadicOp		9
*	AppDig;SetFrPart;AppFrPart;PushOpd;PushDyadicOp		10
/	AppDig;SetFrPart;AppFrPart;PushOpd;PushDyadicOp		11
State 13 = {57}			
ε	AppDig;SetExp;AppExpPart;PushOpd		**Sup**
d	AppDig;SetDig		13
+	AppDig;SetExp;AppExpPart;PushOpd;PushDyadicOp		8
–	AppDig;SetExp;AppExpPart;PushOpd;PushDyadicOp		9
*	AppDig;SetExp;AppExpPart;PushOpd;PushDyadicOp		10
/	AppDig;SetExp;AppExpPart;PushOpd;PushDyadicOp		11
State 14 ={50}; 15={52}			
d	InitInt;SetDig		13
1	**2**	**3**	**4**
State 16 ={2}			
EOF	Complete		18
State 17 ={70}			
)	UnloadAndDiscardOpenPar		19

Вход	Семантика	Магазин	Состояние
1	**2**	**3**	**4**
State 18 = {4} (final)			
ε			**Stop**
State 19 = {72}			
ε			**Sup**
+	PushDyadicOp		8
–	PushDyadycOp		9
*	PushDyadycOp		10
/	PushDyadycOp		11

Табл. 2. δ^f_3. Подавляемые состояния: 2, 12, 13, 19

Магазинный символ	**Возвратное состояние**
1	16
2	17

Некоторые пояснения

1. Подтаблица δ^f_1 оказывается не у дел, так как резольверные символы не используются в трансляционной грамматике калькулятора.

2. Подобные строки таблицы δ^f_2, соответствующей состояниям 3, 4, 8, 9, 10, 11, представляются одной копией, также так же как и состояниям 14 и 15.

3. Комментарии, подобные (2), касаются также таблицы δ^f_3 по подавленным состояниям 2, 12, 13, 19.

4. Процессор обратного прохода не используется.

Рассмотрим пример вычисления арифметического выражения 2 * (3 * (1 + 4)) + 5 (см. табл. 3). Процессор-калькулятор воспроизводит модифицированный алгоритм Дейкстры, совмещающий преобразования выражений в обратную польскую форму с одновременным их вычислением. Арифметические операции над значениями, находящимися на вершине магазина операндов, выполняются процессором в момент, когда оригинальный алгоритм пересылает их из магазина операций на выход. Результат образуется в магазине операндов (на шаге 17). После его печати магазин операндов опустошается. Вычисление заканчивается в состоянии 18, которое является конечным. При благополучном завершении вычислений магазин процессора также оказывается пустым.

Протокол работы процессора-калькулятора (см. табл. 3) состоит из двух частей, одна из которых, озаглавленная «Управляющий процессор», показывает его работу, а другая, под заголовком «Операционная среда» описывает её состояние в соответствующие моменты времени. Эти моменты (шаги) пронумерованы в первой графе протокола. Графа «Сост.» показывает текущее состояние управления процессора. В графе «Вход» представлены текущие входные символы, составляющие интерпретируемое арифметическое выражение. Пустая клетка в этой графе означает, что в соответствующий момент повторно анализируется входной символ, который не был принят на предыдущем шаге процессирования[1] (см., например, шаги 11, 13 и 17). В графе «Магазин» показано текущее состояние магазина управляющего процессора. Его вершина на каждом шаге процессирования определяется положением последней записи (для удобства читателя графа «Магазин» оцифрована). В графе «Семантики» указывается, какие семантики исполняются в соответствующий момент обработки.

[1] Это соответствует ε-движениям, использующим символы с вершины магазина.

В разделе «Операционная среда» графа «Операнд» представляет магазин значений операндов, а графа «Операция» – магазин операций, с помощью которого обеспечивается надлежащий порядок вычисления выражения в соответствии со старшинством операций и расстановкой скобок. Обе графы этих магазинов также оцифрованы, чтобы показать, сколько записей в них имеется на каждый текущий момент процессирования. Положения вершин этих магазинов определяются номерами последних записей. Остальные графы представляют текущие значения переменных, составляющих операционную среду процессора, которые используются при интерпретации данного арифметического выражения.

Табл. 3

УПРАВЛЯЮЩИЙ ПРОЦЕССОР							ОПЕРАЦИОННАЯ СРЕДА											
Шаг	Сост.	Вход	Магазин			Семантики	Операнд				Операция					Integer	DigNmb	Digit	Number
			1	2	3		1	2	3	4	1	2	3	4	5				
1	1	2	1			Reset;InitInt; SetDig										0	0	2	
2	2	*	1			AppDig;SetIntPart; PushOpd; PushDyadicOp	2				*					2	1	2	2
3	10	(	1			PushOpenPar	2				*	(				2	1	2	2
4	5	3	1	2		InitInt;SetDig	2				*	(				0	0	3	
5	2	*	1	2		AppDig;SetIntPart; PushOpd; PushDyadicOp	2	3			*	(	*			3	1	3	3
6	10	(	1	2		PushOpenPar	2	3			*	(	*	(		3	1	3	3
7	5	1	1	2	2	InitInt;SetDig	2	3			*	(	*	(		0	0	1	
8	2	+	1	2	2	AppDig;SetIntPart; PushOpd; PushDyadicOp	2	3	1		*	(	*	(	+	1	1	1	1
9	8	4	1	2	2	InitInt;SetDig	2	3	1		*	(	*	(	+	0	0	4	
10	2	)	1	2	2	AppDig;SetIntPart; PushOpd	2	3	1	4	*	(	*	(	+	4	1	4	4
11	17		1	2		UnloadAndDiscard	2	3	5		*	(	*			4	1	4	4
12	19	)	1	2			2	3	5		*	(	*			4	1	4	4
13	17		1			UnloadAndDiscard	2	15			*					4	1	4	4
14	19	+	1			PushDyadicOp	30				+					4	1	4	4
15	8	5	1			InitInt;SetDig	30				+					0	0	5	
16	2	EOF				AppDig;SetIntPart; PushOpd	30	5			+					5	1	5	5
17	16					Complete	35												
18	18	**Stop**																	

WIRTH SYNTACTIC CHARTS AND GRAPH-SCHEMES IN THE SYNTAX-TECHNOLOGY

Abstract

The purpose of the article is a methodological one. An application of modified the Wirth's syntactic charts is described in connection with the implementation of the programming language Algol 68 in the early 1970s.

Keywords: grammars, Wirth syntactic flowcharts, graph-schemes, regular splines, translations.

Мартыненко Борис Константинович,
доктор физико-математических наук, профессор кафедры информатики математико-механического факультета СПбГУ,

mbk@ctinet.ru