Автоматная модель визуального описания синтаксического разбора*

А. П. Стасенко
Институт систем информатики им. А.П. Ершова СО РАН, Новосибирск, Россия
e-mail: astasenko@gmail.com

The paper introduces and investigates an automaton model suitable for a clear visual description of effective descending syntax analysis of programming languages. It is shown that in the deterministic case the presented automaton model allows class of LL_1 languages. The presented automaton model indirectly allows more powerful languages via its context states and supports hierarchical processing of indeterminacy for implementing syntax error handling without overhead of completely specified automaton. The paper shows the ways to improve efficiency of automaton of the presented model such as state minimization, removal of ε-transitions and unreachable states. The advantages of translator implementation that use automatons of the presented model are shown.

Введение

В настоящее время синтаксический разбор языков программирования [1] обычно осуществляется с помощью автоматов, сгенерированных по грамматике языков системами построения трансляторов (СПТ). При этом в СПТ, ориентированных на восходящий разбор, таких как YACC [2] или Bison [3], возникают сложности с написанием семантических правил и читабельностью сгенерированного распознавателя [4], тогда как СПТ, ориентированные на нисходящий разбор, такие как ANTRL [5], допускают часто неприемлемо узкий класс LL_k-языков [6]. Существуют подходы, комбинирующие сильные стороны восходящих и нисходящих разборов [7, 8], но все они так или иначе требуют, чтобы входная грамматика принадлежала к определенному классу, и затем генерируют распознающий автомат в виде таблицы и/или программного кода.

Требование принадлежности грамматики к определенному классу связано с необходимостью преобразования (часто ручного) исходной грамматики для получения допустимой грамматики, что, во-первых, не всегда возможно, во-вторых, приводит к отрыву от обычно более наглядного пользовательского описания языка и, в-третьих, может быть источником ошибок [9]. Генерация автомата также приводит к отрыву от исходного описания языка, так как обычно неизбежно наступает этап, когда необходимо внести изменения в уже сгененированный код. Положение особенно осложняется при необходимости частого внесения изменений в исходный язык, требующих повторных ручных преобразований грамматики и перегенерации автомата.

*Работа частично поддержана Российским фондом фундаментальных исследований (грант № 07-07-12050).

Указанные проблемы особенно обостряются в трансляторах промышленного уровня и обычно решаются с помощью вручную написанных распознавателей. Существуют работы по ручному заданию восходящих распознавателей [10], но большее распространение получили распознаватели[1], основывающиеся на намного более естественном нисходящем разборе, использующем контекстную информацию там, где это необходимо для преодоления ограниченности класса LL_k-языков. Недостатками вручную написанных трансляторов обычно являются:

— недостаточно сильное разделение разбора синтаксиса и семантики;

— плохое сочетание кода разбора синтаксиса, представляющего собой большие конструкции выбора, с кодом семантического разбора;

— использование больших линейно проверяемых конструкций выбора вместо логарифмического поиска по упорядоченному множеству условий;

— сложность ручного устранения снижающих эффективность переходов по умолчанию в конструкциях выбора;

— трудность ручной минимизации числа конструкций выбора.

В данной статье предлагается автоматная модель визуального описания синтаксического разбора, которая, во-первых, позволяет избежать недостатков существующих СПТ путем наглядного описания синтаксиса языка в непосредственно исполняемом виде. Во-вторых, данная модель делает возможным упростить трансляторы, разрабатываемые с ее помощью, и повысить их эффективность путем устранения указанных выше недостатков написанных вручную трансляторов.

Наглядность предлагаемой автоматной модели проистекает из того факта, что она задается не табличным способом, как классические магазинные автоматы, осуществляющие синтаксический анализ [6], а с помощью графа, сравнимого по сложности с графом конечного автомата, тогда как существующие способы графического задания магазинных автоматов [11, 12] требуют громоздких пометок на дугах, соответствующих переходам. Графические способы задания синтаксиса языка достаточно удобны, что подтверждается синтаксическими диаграммами Вирта [13], с помощью которых был описан язык Паскаль. Предлагаемая автоматная модель по сути является развитием и уточнением идеи, заложенной в синтаксических диаграммах Вирта.

Статья состоит из нескольких разделов. В разд. 1 вводится определение модели γ-автомата и его частных случаев: γ_s-, γ_1- и γ_{s1}-автоматов, графическое изображение которых дается в разд. 2. Наибольший интерес представляет класс γ_1-автоматов, допускающих детерминированное исполнение и класс контекстно-зависимых языков, однако для полноты картины в разд. 3 исследуется класс языков, представимых γ_s-автоматами, в которых отсутствуют контекстные переходы, снижающие наглядность графического представления γ-автоматов. В двух следующих разделах рассмотрение сосредоточивается на классе γ_{s1}-автоматов, сочетающем наглядность представления γ_s-автоматов и детерминированность исполнения γ_1-автоматов. В разд. 4 доказывается существование класса грамматик, эквивалентного классу γ_{s1}-автоматов. В разд. 5 характеризуется

[1]Многие промышленные компиляторы языка Си++ и Си используют компилятор переднего плана Edison Design Group (http://www.edg.com), основанный на вручную написанном нисходящем распознавателе. Компиляторы Open Watcom также используют вручную написанные нисходящие распознаватели (http://www.openwatcom.org/index.php/Compiler_Architecture). Компилятор переднего плана G++ начиная с версии 3.4 перестал использовать СПТ YACC и был переписан вручную для реализации нисходящего разбора, для повышения эффективности трансляции и устранения конфликтов распознавателя (http://gcc.gnu.org/gcc-3.4/changes.html).

класс языков, представимых γ_{s1}-автоматами, и исследуется влияние на него механизма иерархической обработки исключений. В разд. 6 рассматривается совпадение класса контекстно-зависимых языков и класса языков γ_1-автоматов, а в разд. 7 речь идет о способах оптимизации γ_1-автоматов. Статья завершается разделом 8, где описываются преимущества реализации транслятора, синтаксический разбор которого дан с помощью графического представления γ_1-автоматов.

1. Определение модели

Вводимая модель автомата для визуального описания синтаксического разбора (далее — просто γ-автомата) основывается на классической модели магазинного автомата [14]. Магазинный автомат подобен конечному автомату, но в отличие от последнего имеет рабочую память — магазин, в который записываются символы некоторого алфавита. Каждое движение магазинного автомата определяется функцией перехода в зависимости от текущего состояния, входного символа или не зависимо от него (так называемые ϵ-движения) и от верхнего символа магазина. Одно движение магазинного автомата состоит в замещении верхнего символа магазина некоторой магазинной цепочкой, в частности пустой (стирание верхнего символа магазина), и переходе в новое состояние управления. При этом текущим входным символом становится текущий символ на входной ленте, если выполняется движение, зависящее от входного символа, либо текущий входной символ остается тем же самым, если выполняется ϵ-движение.

С одной стороны, модель γ-автомата — это сужение модели магазинного автомата, так как на нее наложены ограничения на вид функции перехода, призванные повысить наглядность ее графического представления. С другой стороны, модель γ-автомата дополнительно имеет контекстные переходы для расширения класса языков, представимых γ-автоматами, и магазин исключений с целью дальнейшего повышения читабельности графического представления модели.

Модель γ-автомата определяется кортежем $A = (T, T_k, S, s_0, S_{end}, S_k, \tau, \tau_k, K, k_0, \Phi, \mu, f_e)$, где T — конечный входной алфавит, T_k — множество контекстных пометок, S — конечное множество состояний, $s_0 \in S$ — начальное состояние, $S_{end} \subseteq S$ — множество конечных состояний, S_k — множество контекстных состояний ($S_k \cap S = \emptyset$), $\tau : S \times (T \cup \{\epsilon^2\}) \times (M \cup \{\epsilon\}) \to 2^{(S \cup S_k) \times M^*}$ — функция переходов, изменяющая содержимое магазина M, содержащего состояния из S, $\tau_k : S_k \times T_k \to S \cup S_k$ — функция контекстных переходов, K — множество контекстов, $k_0 \in K$ — начальный контекст, Φ — множество контекстных функций вида $K \to K \times T_k$, $\mu : S_k \to \Phi$ — функция разметки состояний контекстными функциями, $f_e : S \times (M_e \cup \{\epsilon\}) \to M_e^*$ — функция изменения магазина исключений M_e (ИМИ-функция), содержащего состояния из S. Модель γ-автомата имеет входную ленту с алфавитом T, в котором выделен специальный символ t_{end}, находящийся в конце ленты. Также модель γ-автомата имеет магазин M и магазин исключений M_e.

Функция переходов γ-автомата τ для каждого состояния $s_1 \in S$ и символа перехода $\delta \in T \cup \{\epsilon\}$ имеет один из следующих трех видов (упоминаемые далее как переходы первого, второго и третьего видов соответственно):

1) семейство переходов $\tau(s_1, \delta, \epsilon) = (s_2, \epsilon)$, не зависящих от состояния и не меняющих содержимое магазина M;

[2] Символ ϵ обозначает пустую цепочку.

2) семейство переходов $\tau(s_1, \delta, \epsilon) = (s_n, s_2, \ldots, s_{n-1})$, не зависящих от содержимого магазина M, но добавляющих в него цепочку $s_2, \ldots, s_{n-1}$ (s_2 — в первую очередь);

3) семейство переходов $\tau(s_1, \delta, s_2) = (s_2, \epsilon)$ для всех $s_2 \in M$ в состояние, вытолкнутое из магазина M.

Функция f_e для каждого состояния $s \in S$ имеет один из следующих четырех видов:

— функция вида $f_e(s, \epsilon) = (\epsilon)$, не зависящая от состояния и не меняющая содержимое магазина M_e;

— функция вида $f_e(s, \epsilon) = (s_e)$, добавляющая состояние s_e в магазин M_e;

— функция вида $f_e(s, m_e) = (s_e)$ для всех $m_e \in M_e$, заменяющая состояние m_e на s_e на вершине магазина M_e;

— функция вида $f_e(s, m_e) = (\epsilon)$ для всех $m_e \in M_e$, выталкивающая состояние из магазина M_e.

Конфигурацией автомата называется элемент множества $T^* \times (S \cup S_k) \times M^* \times M_e^* \times K$. Работа автомата определяется сменой конфигураций от *начальной конфигурации* $(\omega_1, s_0, \epsilon, \epsilon, k_0)$, где ω_1 обозначает *начальную цепочку символов* на входной ленте, до *конечной конфигурации*, принадлежащей множеству $T^* \times S_{end} \times M^* \times M_e^* \times K$. Введем определение вспомогательных понятий. В γ-автомате *переходом* функции переходов τ для конфигурации $(\omega,\ s,\ m,\ m_e,\ k)$ называется множество $\tau(s, t, m) \cup \tau(s, \epsilon, m)$, где $t \in T$ определяется из равенства $\omega = t\omega_1$. Переход τ для конфигурации $(\omega,\ s,\ m,\ m_e,\ k)$ называется *детерминированным*, если $|\tau(s, t, m)| = 1$ и $|\tau(s, \epsilon, m)| \leq 1$. Переход τ для конфигурации $(\omega,\ s,\ m,\ m_e,\ k)$ называется *неопределенным*, если $|\tau(s, t, m) \cup \tau(s, \epsilon, m)| = 0$.

Если для состояния $s \in S$ функция переходов τ предназначена для конфигурации $(\omega_1,\ s_1,\ m_1,\ m_{e1},\ k_1)$, то следующая конфигурация $(\omega_2,\ s_2,\ m_2,\ m_{e2},\ k_2)$ определяется таким образом. Состояние s_2 и цепочка m', использующаяся для определения цепочки $m_2 = m''m'$, где цепочка m'' получается из равенства $m_1 = m''m$, связаны с функцией переходов τ как $(s_2, m') = \tau(s_1, \delta = \{t, \epsilon\}, m)$. Если $\delta = t$, то цепочка ω_2 находится из равенства $\omega_1 = t\omega_2$, иначе — $\omega_2 = \omega_1$. Цепочка m'_e, использующаяся для определения цепочки $m_{e2} = m''_e m'_e$, где цепочка m''_e получается из равенства $m_{e1} = m''_e m_e$, связана с ИМИ-функцией f_e как $m'_e = f_e(s_1, m_e)$. Контекст не изменяется: $k_2 = k_1$.

Если для состояния $s \in S$ функция переходов τ не определена для конфигурации $(\omega_1,\ s_1,\ m_1,\ m_{e1},\ k_1)$, то следующая конфигурация $(\omega_2,\ s_2,\ m_2,\ m_{e2},\ k_2)$ находится таким образом: $\omega_2 = \omega_1$, состояние s_2 и цепочка m_{e2} определяются из равенства $m_{e1} = m_{e2}s_2$[3], $m_2 = m_1$, $k_2 = k_1$. Тем самым, ИМИ-функция f_e и магазин исключений M_e задают *механизм иерархической обработки неопределенностей* (*ИОН-механизм*). Под *γ-автоматом без ИОН-механизма* подразумевается γ-автомат, в котором все функции f_e не зависят от содержимого магазина M_e и не меняют его.

Для состояния $s \in S_k$ смена конфигурации $(\omega_1,\ s_1,\ m_1,\ m_{e1},\ k_1)$ на конфигурацию $(\omega_2,\ s_2,\ m_2,\ m_{e2},\ k_2)$ рассчитывается иначе. Новое состояние s_2 определяется функцией переходов τ_k как $s_2 = \tau_k(s_1, t_k)$, где t_k и новое состояние контекста k_2 характеризуются функцией ϕ на основании текущего контекста $(t_k, k_2) = \phi(k_1)$, при этом функция $\phi \in \Phi$ зависит от функции разметки μ для текущего состояния: $\phi = \mu(s_1)$. Содержимое магазинов и входной ленты не изменяется: $m_2 = m_1$, $m_{e2} = m_{e1}$ и $\omega_2 = \omega_1$.

Определение 1. *Под языком, задаваемым γ-автоматом, подразумевается множество начальных цепочек символов в алфавите T на входной ленте γ-автомата, для которых автомат достигает конечного состояния.*

[3]Если $m_{e1} = \epsilon$, то поведение γ-автомата не определено.

Будем говорить, что *автоматы эквивалентны*, если задаваемые ими языки совпадают. *Два класса* (множества) *автоматов эквивалентны*, если для каждого автомата в первом классе найдется эквивалентный автомат во втором классе и наоборот.

Введем подкласс класса γ-автоматов без контекстных состояний и ИОН-механизма, который интересен тем, что он является подклассом обычных недетерминированных магазинных автоматов с более ограниченным видом функции переходов τ, что, однако, как показано в разд. 3, не повлекло за собой сужения класса задаваемых ими языков.

Определение 2. *Под γ_s-автоматом подразумевается γ-автомат без ИОН-механизма, в котором $S_k = \emptyset$.*

Заметим, что у γ_s-автомата можно не задавать компоненты K, k_0, Φ, μ, τ_k и T_k, а в конфигурации автомата указывать только первые три элемента $T^* \times S \times M^*$.

Введем понятие класса детерминированных γ-автоматов, который важен тем, что автоматы из этого класса допускают детерминированное исполнение и позволяют задавать широкий класс контекстно-зависимых языков, что доказывается в разд. 6.

Определение 3. *Под γ_1-автоматом подразумевается γ-автомат, в котором все переходы функции переходов τ детерминированы, а неоднозначность выбора между переходами $\tau(s, \epsilon, m)$ и $\tau(s, t, m)$ разрешается предпочтением перехода по t.*

Введем подкласс класса γ-автоматов, который интересен тем, что выступает как подкласс обычных детерминированных магазинных автоматов с более ограниченным видом функции переходов τ, что, как показано в разд. 5, повлекло за собой сужение класса задаваемых ими языков.

Определение 4. *Пересечение класса γ_s-автоматов и класса γ_1-автоматов обозначается как класс γ_{s1}-автоматов.*

Рассмотрим примеры γ-автоматов. Начнем с примера γ_{s1}-автомата A_{s1}, определив его компоненты следующим образом. Пусть $T = \{\text{‘(’}, \text{‘)’}, t_{end}\}$, $S = \{s_0, s_1, s_2, s_{end}\}$, $S_{end} = \{s_{end}\}$. Функция переходов τ γ_{s1}-автомата A_{s1} характеризуется следующим образом: $\tau(s_0, \text{‘(’}, m) = (s_1, ms_{end})$, $\tau(s_1, \text{‘(’}, m) = (s_1, ms_2)$, $\tau(s_1, \text{‘)’}, s) = (s, \epsilon)$, $\tau(s_2, \epsilon, m) = (s_1, m)$[4].

Если на входной ленте находится цепочка ‘(()())’, то работа задающего ее γ_{s1}-автомата A_{s1} будет определяться следующей последовательностью конфигураций (конфигурации упорядочены слева направо и сверху вниз):

$$
\begin{array}{llllll}
(\text{‘(()())’}, & s_0, \epsilon), & (\text{‘()())’}, & s_1, s_{end}), & (\text{‘)())’}, & s_1, s_{end}\ s_2), \\
(\text{‘())’}, & s_2, s_{end}), & (\text{‘())’}, & s_1, s_{end}), & (\text{‘))’}, & s_1, s_{end}\ s_2), \\
(\text{‘)’}, & s_2, s_{end}), & (\text{‘)’}, & s_1, s_{end}), & (\epsilon, & s_{end}, \epsilon).
\end{array}
$$

Заметим, что автомат A_{s1} задает скобочный язык $L = \{\text{’}(\alpha)\text{’}\}$, где $\alpha \in A = \{\epsilon, \text{’}(\alpha_1)\alpha_2\text{’}\}$ с $\alpha_1, \alpha_2 \in A$.

Рассмотрим пример γ_1-автомата A_1, определив его компоненты следующим образом. Пусть $T = \{\text{‘(’}, \text{‘)’}, a, \ldots, z, t_{end}\}$, $S = \{s_0, s_1, s_3, s_4, s_5, s_6, s_7, s_8, s_{end}\}$, $S_k = \{s_2\}$, $S_{end} = \{s_{end}\}$, $K = \{\text{цепочка чисел, задающая содержимое магазина } X\}$, $k_0 = 1$, $\Phi = \{\phi\}$, $T_k = \{true, false\}$. Функция μ автомата A_1 определяется как $\mu(s_2) = \phi$. Функция переходов τ_k определяется как $\tau_k(s_2, true) = s_3$, $\tau_k(s_2, false) = s_5$. Функция f_e определяется как $f_e(s_3, m_e) = (m_e s_7)$, $f_e(s_4, m_e) = (\epsilon)$ и $f_e(s, m_e) = (m_e)$ для $s \in \{s_1, s_5, s_6, s_7, s_8\}$.

[4]Правила $\tau(s_1, \text{‘(’}, m) = (s_1, ms_2)$ и $\tau(s_2, \epsilon, m) = (s_1, m)$ можно было бы заменить правилом $\tau(s_1, \text{‘(’}, m) = (s_1, ms_1)$, но это не было сделано для поддержания сходства с рассматриваемыми далее γ-схемами, которые не могут непосредственно представить указанное правило.

Функция ϕ автомата A_1 возвращает значение $true$, если число на вершине магазина X равно глубине магазина X, и возвращает значение $false$ — если иначе. Также (после проверки предыдущего равенства) функция ϕ увеличивает на единицу число на вершине магазина X. Для целей реальной трансляции состояниям и переходам модели γ-автомата приписываются пометки, обозначающие различные транслирующие процедуры над контекстом K, что более подробно рассматривается в разд. 8. Тем самым вводятся функция ϕ_{push}, которая добавляет к магазину X единицу, и функция ϕ_{pop}, которая выталкивает число из магазина X. Состояние s_5 помечено функцией ϕ_{push}.

Функция переходов τ автомата A_1 определяется следующим образом: $\tau(s_0, \text{`('}, m) = (s_1, ms_{end})$, $\tau(s_1, \text{`('}, m) = (s_2, m)$, $\tau(s_1, \text{`)'}, s) = (s, \epsilon)$ (переход помечен функцией $\phi_{pop}$), $\tau(s_3, \epsilon, m) = (s_8, m)$, $\tau(s_4, \epsilon, m) = (s_1, m)$, $\tau(s_5, \epsilon, m) = (s_1, ms_6)$, $\tau(s_6, \epsilon, m) = (s_1, m)$, $\tau(s_7, t, m) = (s_7, m)$ для всех $t \in T \setminus \{\text{`)'}\}$, $\tau(s_7, \text{`)'}, m) = (s_1, m)$, $\tau(s_8, \text{`)'}, m) = (s_4, m)$, $\tau(s_8, \text{`}a\text{'}, m) = (s_8, m)$.

Если на входной ленте находится цепочка '$((abba)())$', то работа задающего ее γ_1-автомата A_1 будет определяться следующей последовательностью конфигураций:

('$((abba)())$',	$s_0, \epsilon, \epsilon, 1)$,	('$(abba)())$',	$s_1, s_{end}, \epsilon, 1)$,	('$abba)())$',	$s_2, s_{end}, \epsilon, 1)$,
('$abba)())$',	$s_3, s_{end}, \epsilon, 2)$,	('$abba)())$',	$s_8, s_{end}, s_7, 2)$,	('$bba)())$',	$s_8, s_{end}, s_7, 2)$,
('$bba)())$',	$s_7, s_{end}, \epsilon, 2)$,	('$ba)())$',	$s_7, s_{end}, \epsilon, 2)$,	('$a)())$',	$s_7, s_{end}, \epsilon, 2)$,
('$)())$',	$s_7, s_{end}, \epsilon, 2)$,	('$())$',	$s_1, s_{end}, \epsilon, 2)$,	('$))$',	$s_2, s_{end}, \epsilon, 2)$,
('$))$',	$s_5, s_{end}, \epsilon, 3)$,	('$))$',	$s_1, s_{end}\ s_6, \epsilon, 3\ 1)$,	('$)$',	$s_6, s_{end}, \epsilon, 3)$,
('$)$',	$s_1, s_{end}, \epsilon, 3)$,	$(\epsilon,$	$s_{end}, \epsilon, \epsilon, \epsilon)$.		

Заметим, что автомат A_1 задает скобочный язык $L = \{\text{'}(\alpha)\text{'}\}$, где $\alpha \in A = \{\beta, \text{'}(\alpha_1)\alpha_2\text{'}\}$ с $\alpha_1, \alpha_2 \in A$, $\beta \in B$. Причем $B = \{a, \ldots, z\}^*$ для скобок с номером n в скобках с уровнем вложенности, равным n, и $B = \epsilon$ для всех остальных случаев. Также заметим, что в автомате A_1 специальным образом задается разбор языка $L_2 = a^*$ (начиная с состояния s_8), являющегося подмножеством множества B. В качестве языка L_2 можно взять произвольный язык, задаваемый γ_1-автоматом. Автомат A_1 пытается разобрать язык L_2 и, в случае неудачи, пропускает часть цепочки до первого символа закрывающей скобки (являющегося признаком окончания цепочки языка L_2). Данная ситуация близка к реальной задаче разбора языка программирования с восстановлением разбора после синтаксических ошибок.

2. Изображение модели

Для визуального представления модели γ-автомата было разработано графическое представление, называемое γ-*схемой* [15], которое отображает модель γ-автомата в виде ориентированного размеченного графа. Вершины γ-схемы делятся на следующие классы:

1) прямоугольные вершины, соответствующие состояниям $s \in S \cup S_k$:

   а) вершины, соответствующие состояниям $s \in S$:

      i) вершины со сплошной границей, для соответствующих состояний которых ИМИ-функция f_e имеет вид $f_e(s, m_e) = (m_e)$ и $f_e(s, m_e) = (m_e s_e)$ (рис. $1, a$);

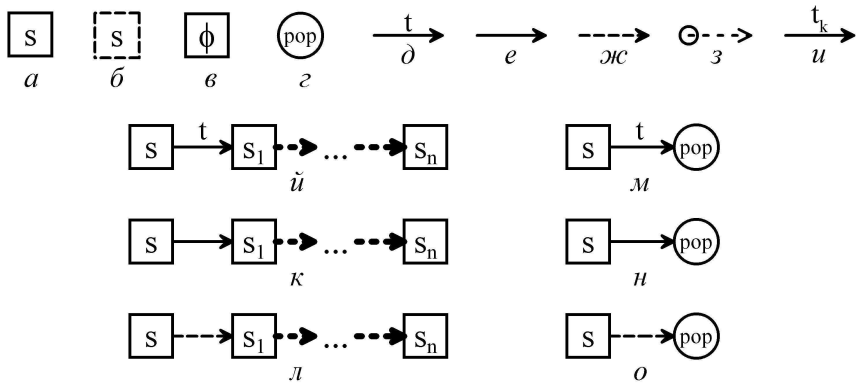


Рис. 1. Базовые элементы γ-схемы

ii) вершины со штриховой границей, для соответствующих состояний которых ИМИ-функция f_e имеет вид $f_e(s, m_e) = (s_e)$ и $f_e(s, m_e) = (\epsilon)$ (рис. 1, *б*);

b) вершины, соответствующие состояниям из S_k и имеющие сплошную границу и пометку элементом множества $\phi \in \Phi$ (рис. 1, *в*);

2) вспомогательные круглые вершины с пометкой "pop" (рис. 1, *г*).

Дуги γ-схем также подразделяются на несколько классов:

1) дуги из вершин, соответствующих состояниям $s \in S$:

a) дуги, задающие переходы первого вида $\tau(s, \delta, m) = (s_1, m)$ различными способами в зависимости от символа δ:

i) сплошная дуга с меткой $t \in T$ для любого $\delta \in T$ (рис. 1, *д*);

ii) сплошная дуга без метки для всех $\delta \in T \setminus T_s$, где T_s — это метки всех других помеченных дуг, исходящих из вершины, соответствующей состоянию s (рис. 1, *е*);

iii) штриховая дуга без метки для $\delta = \epsilon$ (рис. 1, *ж*);

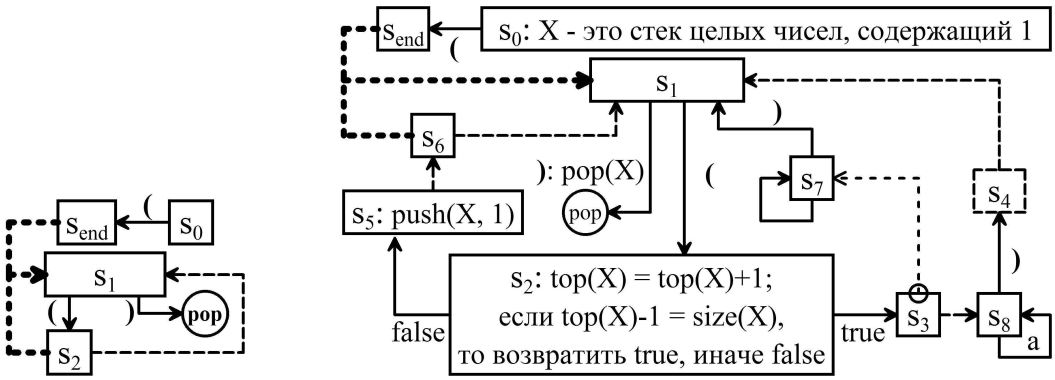
b) непомеченные жирные пунктирные дуги, задающие переходы второго вида $\tau(s, \delta, m) = (s_n, ms_1 \dots s_{n-1})$ из состояния s[5] в состояние s_n и образующие путь от вершины состояния s_1 до вершины состояния s_n (рис. 1, *й,к,л*);

c) входящие в круглые вершины с пометкой "pop" дуги, задающие переходы третьего вида $\tau(s, \delta, s_1) = (s_1, \epsilon)$ из состояний s и имеющие различный вид в зависимости от символа δ (рис. 1, *м,н,о*);

d) непомеченные пунктирные дуги с символом круга в начале дуги, задающие ИМИ-функции f_e вида $f_e(s, m_e) = (m_e s_e)$ и $f_e(s, m_e) = (s_e)$, с началом в вершине состояния s и концом в вершине состояния s_e (рис. 1, *з*);

2) сплошные дуги с меткой $t_k \in T_k$, исходящие из вершин, соответствующих состояниям из S_k (рис. 1, *и*).

[5]Вершина состояния s имеет исходящую дугу, которая может быть разного вида (в зависимости от символа δ), — она входит в вершину состояния s_1.



а *б*

Рис. 2. Примеры: *а* — γ_{s1}-схема; *б* — γ_1-схема

Если γ-схема построена по $\gamma_{\{1,s1\}}$-автомату, как показано выше, то она называется $\gamma_{\{1,s1\}}$-схемой. Исходя из определения γ_1-автомата, γ_1-схема обладает следующими свойствами: из любой вершины не может исходить несколько одинаково помеченных дуг, из вершины, соответствующей состоянию из S, не может исходить более одной непомеченной сплошной или штриховой дуги, более одной жирной пунктирной дуги и более одной пунктирной дуги с символом круга в начале дуги.

Рассмотрим примеры γ-схем. На рис. 2, *а* приведена γ_{s1}-схема, задающая γ_{s1}-автомат A_{s1} из разд. 1. На рис. 2, *б* приведена γ_1-схема, задающая γ_1-автомат A_1 из разд. 1. Нетрудно заметить, что рис. 2, *б* более компактно и наглядно дает описание автомата A_1, занимающее половину страницы в текстовом виде.

3. Свойства γ_s-автомата

Напомним понятие грамматики. Под *грамматикой* подразумевается четверка $(T,\ N,\ s_0,\ R)$, где T — это терминальный алфавит, N — нетерминальный алфавит, $s_0 \in N$ — начальный символ грамматики, R — множество правил вывода. Под *языком, задаваемым грамматикой*, подразумевается множество цепочек терминальных символов, выводимых согласно правилам грамматики из начального символа грамматики. *Контекстно-свободной грамматикой* называется грамматика, правила вывода которой имеют вид $B \to \alpha$, где $B \in N$, $\alpha \in (N \cup T)^*$. *Контекстно-свободным языком* называется язык, задаваемый некоторой контекстно-свободной грамматикой. *Грамматика и автомат эквивалентны*, если задаваемые ими языки совпадают. *Классы грамматик и автоматов эквивалентны*, если для каждой грамматики в классе грамматик найдется эквивалентный ей автомат в классе автоматов и наоборот.

Известно, что класс недетерминированных магазинных автоматов эквивалентен классу контекстно-свободных грамматик [14]. В данном разделе показывается, что класс γ_s-автоматов эквивалентен классу контекстно-свободных грамматик. Сначала покажем, что класс контекстно-свободных языков — это подкласс класса языков γ_s-автоматов.

Утверждение 1. *Для любого контекстно-свободного языка существует γ_s-автомат, его задающий.*

Доказательство. Возьмем контекстно-свободную грамматику $G = \{T', N', s'_0, R\}$. Построим γ_s-автомат A, такой что $L_A = L_G$. Определим компоненты γ_s-автомата A

следующим образом. Входной алфавит определим как $T = T' \cup \{t_{end}\}$. Множество состояний $S = N' \cup N'' \cup \{s_{end}\} \cup \{s_0\}$, где $N'' = \{r_{i,j}\}$, $i = 1 \dots |R|$, $j = 1 \dots J$, где J — это длина правой части i-го правила вывода, $N' \cap N'' = \emptyset$, $s_{end} \notin N' \cup N''$, $s_0 \notin N' \cup N''$, $s_{end} \neq s_0$. В качестве конечного состояния возьмем $S_{end} = \{s_{end}\}$. Функция переходов τ содержит переход из начального состояния $\tau(s_0, \epsilon, \epsilon) = (s'_0, s_{end})$, добавляющий конечное состояние s_{end} в магазин M.

В качестве начального состояния входной ленты возьмем произвольную цепочку языка L_G. Для каждого i-го правила грамматики $B_i \to \alpha_{i,1} \dots \alpha_{i,J}$, где $\alpha_{i,1\dots J} \in N' \cup T'$, добавим следующие переходы к функции переходов τ:

1) для каждого $j = 1 \dots J$ полагаем следующее:

   a) если $\alpha_{i,j} \in T'$, то добавляем переход $\tau(r_{i,j-1}, \alpha_{i,j}, m) = (r_{i,j}, m)$, полагая, что $r_{i,0} = B_i$;

   b) если $\alpha_{i,j} \in N'$, то добавляем переход $\tau(r_{i,j-1}, \epsilon, m) = (\alpha_{i,j}, mr_{i,j})$;

2) добавляем переход в состояние на вершине магазина M: $\tau(r_{i,J}, \epsilon, s) = (s, \epsilon)$.

Тем самым, одно правило магазинного автомата, разбирающего данную грамматику G, заменено несколькими правилами γ_s-автомата A с помощью дополнительных магазинных символов. □

Следующее утверждение говорит, что справедливо и обратное включение.

Утверждение 2. *Язык, задаваемый любым γ_s-автоматом, является контекстно-свободным.*

Доказательство. Доказательство напрямую следует из того, что любой γ_s-автомат — это недетерминированный магазинный автомат (с ограничением на функцию переходов τ). □

Так как класс языков, задаваемых γ_s-автоматами, совпадает с классом контекстно-свободных языков, то введенные ограничения на функцию переходов τ γ_s-автомата не повлекли за собой сужения класса языков, задаваемых недетерминированными магазинными автоматами. К сожалению, ввиду недетерминированности γ_s-автоматы не допускают эффективного[6] исполнения и поэтому не интересны для применений в реальных трансляторах [14].

4. Связь γ_{s1}-автомата с классами грамматик

В данном разделе показывается, что существует класс грамматик, эквивалентный классу γ_{s1}-автоматов и не совпадающий с известным классом LL_1-грамматик [6].

Определим класс грамматик $G_\gamma = (T, N, s_0, R)$ с множеством правил R вида: (1) "$N_1 \to t_1\alpha$", (2) "$N_1 \to N_2 t_1 \alpha$" и (3) "$s_0 \to \epsilon$", где $N_{1,2} \in N$, $t_1 \in T$ и $\alpha \in (N \cup T)^*$, со следующими ограничениями:

1) все правила множества R с одинаковым нетерминалом N_1 имеют один вид;

2) каждое правило множества R с одинаковым нетерминалом N_1 уникально по терминалу t_1;

3) каждое правило второго вида множества R с одинаковым нетерминалом N_1 имеет одинаковый нетерминал N_2.

[6] Исполнение за время, пропорциональное длине входной цепочки.

Покажем, что класс грамматик G_γ эквивалентен классу γ_{s1}-автоматов.

Утверждение 3. *Любой грамматике из класса G_γ соответствует γ_{s1}-автомат, задающий язык этой грамматики.*

Доказательство. Рассмотрим доказательство утверждения 1 для грамматики из G_γ в месте построения функции переходов γ_s-автомата A. Докажем, что γ_s-автомат A является γ_1-автоматом, так как все переходы функции переходов τ автомата A для грамматики из G_γ детерминированы. Недетерминированность переходов может возникнуть в пунктах 1a и 1b утверждения 1 при $j = 1$, если $r_{i,0} = B_i$. В пункте 1a переход $\tau(r_{i,0}, \alpha_{i,1}, m) = (r_{i,1}, m)$ будет детерминированным ввиду того, что в грамматике из G_γ каждое правило с одинаковым нетерминалом $r_{i,0}$ в левой части имеет уникальный терминал $\alpha_{i,1}$. В пункте 1b правило перехода $\tau(r_{i,0}, \epsilon, m) = (\alpha_{i,1}, mr_{i,1})$ будет детерминировано, так как:

— в грамматике из G_γ все правила с одинаковым нетерминалом $r_{i,0}$ в левой части начинаются с одинакового нетерминала $\alpha_{i,1}$ в правой части, что напрямую следует из третьего ограничения в определении класса грамматик G_γ;

— состояния $r_{i,1}$ для совпадающих $r_{i,0}$ нетерминалов можно объединить в одно, потому что переход из них однозначно определяется терминалом $\alpha_{i,2}$ и это напрямую следует из второго ограничения в определении класса грамматик G_γ.

Очевидно, что автомат A является γ_{s1}-автоматом, так как оставшийся переход в родительское состояние пункта 2 детерминирован ввиду уникальности состояния $r_{i,J}$. □

Утверждение 4. *Любому γ_{s1}-автомату соответствует грамматика из G_γ, задающая язык этого автомата.*

Доказательство. Возьмем γ_{s1}-автомат $A = \{T, S, s_0, S_{end}, \tau\}$. Согласно утверждению 13 (см. дальше), устраним в автомате A мнимые переходы первого и второго видов. Построим грамматику $G_\gamma = (T, S, s_0, R)$. Для каждого перехода первого вида $\tau(s_1, t, m) = (s_2, m)$ к множеству R добавим правило "$s_1 \to ts_2$". Для каждого перехода второго вида $\tau(s_1, t, m) = (s_n, ms_2 \dots s_{n-1})$ к множеству R добавим правило "$s_1 \to ts_n s_{n-1}, \dots, s_2$". Для каждого перехода третьего вида $\tau(s_1, t, s_2) = (s_2, \epsilon)$ к множеству R добавим правило "$s_1 \to t$". Для каждого перехода третьего вида $\tau(s_1, \epsilon, s_2) = (s_2, \epsilon)$ к множеству R добавим правило "$s_1 \to \epsilon$". Удалим мнимые переходы из грамматики, используя известный алгоритм для контекстно-свободных грамматик [14]. Очевидно, что каждое правило множества R с одинаковым нетерминалом s_1 в левой части имеет уникальный терминал t, по свойству γ_{s1}-автомата. □

Покажем, что класс грамматик G_γ не совпадает с классом LL_1-грамматик. Для определения класса LL_1-грамматик введем понятие множества ВЫБОР. *Множеством ВЫБОР* для правила $B \to \alpha$ контекстно-свободной грамматики называется множество терминальных символов, с которых начинаются всевозможные терминальные цепочки, выводимые из цепочки α. Будем говорить, что грамматика является *LL_1-грамматикой*, если она является контекстно-свободна и множества ВЫБОР всех ее правил с одинаковой левой частью не пересекаются.

Утверждение 5. *Существуют LL_1-грамматики, не принадлежащие классу G_γ.*

Доказательство. Рассмотрим грамматику $G = (T = \{a, b, c\}, N = \{S, A, B\}, S, R)$, где $R = \{$"$S \to Ac$", "$S \to Bc$", "$A \to a$", "$B \to b$"$\}$. Грамматика G, очевидно, принадлежит классу LL_1, но не принадлежит классу G_γ, так как в ней существуют правила с одинаковой левой частью, но различными нетерминалами в начале правой части. □

Утверждение 6. *В классе G_γ существуют грамматики, не принадлежащие классу LL_1.*

Доказательство. Рассмотрим грамматику $G = (T = \{a, b, c, d\}, N = \{S, A\}, S, R)$, где $R = \{$"$S \to Ac$", "$S \to Ad$", "$A \to a$", "$A \to b$"$\}$. Грамматика G удовлетворяет всем требованиям класса грамматик G_γ, но не принадлежит классу LL_1, так как содержит неоднозначность по первому терминалу выбора правила с нетерминалом S в левой части. □

5. Связь γ_{s1}-автомата с классами языков

Будем говорить, что язык является LL_1-*языком*, если существует LL_1-грамматика, его задающая [6]. В этом разделе показано, что класс языков, задаваемых γ_{s1}-автоматами, совпадает с классом LL_1-языков. Также исследуется класс языков, задаваемых γ_{s1}-автоматами с механизмом иерархической обработки неопределенностей.

Утверждение 7. *Любой LL_1-язык можно задать γ_{s1}-автоматом.*

Доказательство. Рассмотрим LL_1-грамматику G произвольного LL_1-языка. Изменим грамматику G так, чтобы она продолжала задавать язык исходной грамматики, но принадлежала классу G_γ. Этого достаточно, так как уже доказано существование γ_{s1}-автомата, задающего язык L_{G_γ}. В грамматике G множества ВЫБОР, построенные для правил с одинаковой левой частью, не содержат одинаковых элементов, поэтому очевидно, что каждое правило грамматики G можно заменить множеством правил первого вида $N_1 \to t_1\alpha$ для $t_1 \in$ ВЫБОР правил грамматики класса G_γ путем замены первого нетерминала правой части правила. Очевидно, что после таких замен грамматика останется в классе LL_1 и непересечение множеств ВЫБОР для правил с одинаковой левой частью будет означать соблюдение второго ограничения в определении класса грамматик G_γ. Первое и второе ограничения в определении класса грамматик G_γ для измененной грамматики будут очевидно удовлетворены. □

Утверждение 8. *Любой γ_{s1}-автомат задает LL_1-язык.*

Доказательство. Рассмотрим грамматику из G_γ, соответствующую взятому γ_{s1}-автомату. Правила первого вида грамматики из G_γ с одинаковой левой частью имеют непересекающиеся множества ВЫБОР ввиду того, что они различны по первому терминалу.

Рассмотрим правила второго вида грамматики из G_γ с одинаковой левой частью "$N_1 \to N_2 t_1 \alpha_1$". Введем новый нетерминал N_1' для каждого нетерминала N_1, стоящего в левой части правила второго вида. К грамматике добавим правила первого вида "$N_1' \to t_1\alpha_1$".

Если правила первого нетерминала N_2 имеют второй вид "$N_2 \to N_3 t_2 \alpha_2$", то после подстановки снова получаются правила второго вида "$N_1 \to N_3 t_2 \alpha_2 t_1 \alpha_1$". Сделав еще одну подстановку, получим правила вида "$N_1 \to N_3 t_2 \alpha_2 N_1'$", которые будут удовлетворять второму ограничению из определения класса грамматик G_γ, так как для каждого нетерминала N_1 они различны по терминалу t_2, ввиду уникальности терминала t_2 для одинаковых нетерминалов N_2. Там самым для полученных правил можно повторять процедуру замены первого нетерминала на правила второго вида до тех пор, пока первый нетерминал не будет стоять в левой части правил первого вида.

Если правила первого нетерминала N_2 имеют первый вид "$N_2 \to t_2\alpha_2$", то после подстановки получатся правила первого вида "$N_1 \to t_2 \alpha_2 N_1'$", которые также будут удовлетворять второму ограничению из определения класса грамматик G_γ, так как для каждого нетерминала N_1 они различны по первому терминалу. Тем самым правила

второго вида грамматики из G_γ с одинаковой левой частью также имеют непересекающиеся множества ВЫБОР.

Все правила грамматики с одинаковой левой частью имеют непересекающиеся множества ВЫБОР, поэтому грамматика из G_γ принадлежит классу LL_1. □

Очевидно, что γ_{s1}-автомат является детерминированным магазинным автоматом с ограничением на функцию переходов τ. Известно из [6], что класс LL_1-языков — это собственный подкласс класса детерминированных контекстно-свободных языков[7]. Таким образом, введенные ограничения на функцию переходов τ γ_{s1}-автомата повлекли за собой сужение класса детерминированных контекстно-свободных языков до класса LL_1-языков.

Рассмотрим влияние ИОН-механизма на класс языков, представимых γ_{s1}-автоматами. ИОН-механизм γ-автоматов, задаваемый ИМИ-функцией f_e и магазином исключений M_e, был введен для удобства построения и сохранения наглядности реальных γ-схем, описывающих синтаксический анализ языков программирования. ИОН-механизм позволяет избежать необходимости полного определения функции переходов τ для описания γ-автоматов с возможностью обработки синтаксических ошибок во входной программе. По своей сути ИОН-механизм сходен с механизмом обработки исключений известных языков программирования.

Определение 5. *Под γ_{se1}-автоматом подразумевается γ_{s1}-автомат с ИОН-механизмом.*

Хотя ИОН-механизм не предназначается для расширения класса языков, задаваемых γ_{s1}-автоматами, можно показать, что класс языков γ_{se1}-автоматов включает неконтекстно-свободный язык и не включает контекстно-свободный язык. Тем самым язык γ_{se1}-автомата не совпадает ни с одним из известных классов языков, но, очевидно, включает в себя класс LL_1-языков.

Утверждение 9. *Существует неконтекстно-свободный язык, задаваемый γ_{se1}-автоматом.*

Доказательство. Используем известный неконтекстно-свободный язык, состоящий из цепочек вида $a^n b^n c^n$ для всех $n \geq 0$. Рассмотрим γ_{se1}-автомат, в котором $T = \{a, b, c, t_{end}\}$, $S = \{s_0, s_1, s_2, s_3, s_{yes}, s_{no}\}$, $S_{end} = \{s_{yes}\}$. Функция переходов τ определяется следующим образом: $\tau(s_0, \epsilon, m) = (s_1, ms_3)$, $\tau(s_1, a, m) = (s_1, ms_2)$, $\tau(s_1, \epsilon, m) = (m, \epsilon)$, $\tau(s_2, b, m) = (m, \epsilon)$, $\tau(s_3, c, m) = (s_3, m)$, $\tau(s_{no}, t, m, m_e) = (s_{no}, m, m_e)$ для всех $t \in T$. Функция f_e определяется так: $f_e(s_0, m_e) = (m_e s_{yes})$, $f_e(s_1, m_e) = (m_e s_{no})$, $f_e(s_3, m_e) = (\epsilon)$, $f_e(s_{no}, m_e) = (\epsilon)$.

При распознавании букв a по сути их количество n запоминается в магазинах M и M_e. Магазин M используется для распознавания ровно n букв b, а магазин M_e — для распознавания ровно n букв c. Автомат может достигнуть конечного состояния s_{yes} только в случае возникновения неопределенной ситуации в состоянии s_3, когда сработает переход в состояние на вершине магазина M_e, причем только в том случае, если состоянием s_3 из магазина M_e было вытолкнуто n раз состояние s_{no}. Попав в состояние s_{no}, автомат с неизбежностью попадет в неопределенное положение при исчерпании магазина M_e. □

Утверждение 10. *Существует контекстно-свободный язык, не задаваемый γ_{se1}-автоматом.*

[7] Язык детерминированного магазинного автомата называется *детерминированным контекстно-свободным языком.*

Доказательство. Рассмотрим язык L, задаваемый контекстно-свободной грамматикой $G = (T = \{a, b, c, d\}, N = \{S, A, B\}, S, R)$, где $R = \{$"$S \to A$", "$S \to B$", "$A \to aAbAc$", "$B \to aBbBd$"$\}$. Язык L не принадлежит классу LL_k-языков, так как для любого наперед заданного числа k существует достаточно длинная цепочка, принадлежащая языку L, по первым k символам которой нельзя определить, нетерминал A или нетерминал B ее задает.

Ввиду того что нетерминалы A и B задаются с помощью ветвящейся рекурсии, их невозможно сымитировать итеративными способами, наподобие трюка, описанного при доказательстве утверждения 5. Значит, γ_{se1}-автомат, задающий язык L, должен использовать магазин M для обеспечения корректной вложенности нетерминалов A и B. Различить, какой из этих нетерминалов задает распознаваемую цепочку языка, можно только в момент считывания терминала c или d. Даже если каким-то образом изменить содержимое магазина M_e, чтобы отразить, какая из букв (c или d) была встречена, при рекурсивном возврате по магазину M автомат сможет воспользоваться этой информацией только при возникновении неопределенной ситуации (функция переходов τ не определена для текущей конфигурации), которая невозможна ввиду ее отсутствия при движении автомата по той же части пути до момента добавления этой информации в магазин M_e. □

6. Свойства γ_1-автомата

Как показано ранее, γ_{s1}-автоматы ограничены довольно узким классом LL_1-языков и поэтому малоприменимы для синтаксического анализа реальных языков программирования, спроектированных без учета принадлежности к этому классу формальных языков. Поэтому с точки зрения мощности множества задаваемых языков интерес представляет более широкий класс γ_1-автоматов, класс языков которого, как будет показано в этом разделе, совпадает с классом контекстно-зависимых языков[8]. Рассмотрение ограничивается контекстно-зависимыми языками, в предположении, что контекст исполнения ограничен конечной памятью реальных вычислительных систем. Следующее утверждение показывает, что класс контекстно-зависимых языков — это подкласс класса языков γ_1-автомата.

Утверждение 11. *Для любого контекстно-зависимого языка существует γ_1-автомат, его задающий.*

Доказательство. Возьмем произвольный контекстно-зависимый язык L с грамматикой (T', N', s'_0, R'). Как известно из [14], класс контекстно-зависимых языков совпадает с классом языков линейно ограниченных автоматов. Таким образом, существует линейно ограниченный автомат A_L, задающий язык L.

Рассмотрим γ_1-автомат, в котором $T = T' \cup \{t_{end}\}$, $S = \{s_0, s_1, s_3\}$, $S_{end} = \{s_3\}$, $S_k = \{s_2\}$, $K = \{X \mid$ строка символов из алфавита $T'\}$, $T_k = \{true, false\}$, $\mu(s_2) = \phi_{decide}$, $\Phi = \{\phi_{decide}\}$. Состояние s_0 помечено действием "сделать строку символов X пустой". Состояние s_1 помечено действием "добавить последний прочитанный символ входной ленты к строке символов X". Функция переходов τ, определяется так: $\tau(s_0, t, m) = (s_1, m)$, $\tau(s_0, t_{end}, m) = (s_2, m)$, $\tau(s_1, t, m) = (s_2, m)$, $\tau(s_1, t_{end}, m) = (s_2, m)$ для любого $t \in T$ и любого $m \in M$. Функция ϕ_{decide} реализует автомат A_L, используя в качестве

[8] *Контекстно-зависимый язык* задается контекстно-зависимой грамматикой (T, N, s_0, R), правила R которой имеют вид $\alpha \to \beta$, где $\alpha, \beta \in (N \cup T)^*$ и $|\alpha| \le |\beta|$.

его ленты строку символов X и возвращая $true$, если строка допустима, и $false$ — если иначе. Функция переходов τ_k содержит переход $\tau_k(s_2, true) = (s_3)$, помеченный действием "сообщить, что входная цепочка принадлежит языку L". Функция переходов τ_k также содержит переход $\tau_k(s_2, false) = (s_3)$, помеченный действием "сообщить, что входная цепочка не принадлежит языку L".

Построенный γ_1-автомат с очевидностью задает язык L, так как по сути он только накапливает символы входной ленты для использования ее в качестве ленты линейно ограниченного автомата[9]. □

Следующее утверждение говорит, что справедливо и обратное включение.

Утверждение 12. *Язык, задаваемый любым γ_1-автоматом, является контекстно-зависимым.*

Доказательство. Доказательство напрямую следует из того, что ввиду конечности памяти реальных вычислительных систем γ_1-автомат является подклассом линейно ограниченного автомата. □

Таким образом, класс языков, задаваемых γ_1-автоматами, совпадает с классом контекстно-зависимых языков.

7. Оптимизация γ_1-автомата

В данном разделе показаны конструктивные способы оптимизации, применимые к классу γ_1-автоматов, такие как минимизация состояний, устранение мнимых переходов[10] и недостижимых состояний, позволяющие повысить эффективность работы автомата для практических нужд. Сначала рассмотрим устранение мнимых переходов в γ_{s1}-автоматах.

Утверждение 13. *В рамках модели γ_{s1}-автомата, не сужая класс представимых им языков, можно устранить все мнимые переходы первого и второго видов.*

Доказательство. Для устранения мнимого перехода первого вида $\tau(s_1, \epsilon, m) = (s_2, m)$ достаточно заменить его множеством переходов $A \cup B \cup C \cup \{d\}$ для всех t, не участвующих в переходах из состояния s_1, где:

— $A = \{\tau(s_1, t, m) = (s_3, m) \mid$ для каждого перехода $\tau(s_2, t, m) = (s_3, m)\}$;

— $B = \{\tau(s_1, t, m) = (s_{n+1}, ms_3 \dots s_n) \mid$ для каждого перехода $\tau(s_2, t, m) = (s_{n+1}, ms_3 \dots s_n)\}$;

— $C = \{\tau(s_1, t, s_3) = (s_3, \epsilon) \mid$ для каждого перехода $\tau(s_2, t, s_3) = (s_3, \epsilon)\}$;

— d — мнимый переход из состояния s_2, если он существует, в котором исходное состояние s_2 заменено на состояние s_1.

Мнимый переход второго вида $\tau(s_1, \epsilon, m) = (s_n, ms_2 \dots s_{n-1})$ устраняется его заменой на множество переходов $A \cup B \cup C \cup \{d\}$ для всех t, не участвующих в переходах из состояния s_1, где:

— $A = \{\tau(s_1, t, m) = (s_{n+1}, ms_2 \dots s_{n-1}) \mid$ для каждого перехода $\tau(s_n, t, m) = (s_{n+1}, m)\}$;

— $B = \{\tau(s_1, t, m) = (s_{n+m}, ms_2 \dots s_n \dots s_{n+m-1}) \mid$ для каждого перехода $\tau(s_n, t, m) = (s_{n+m}, ms_{n+1} \dots s_{n+m-1})\}$;

[9] Используемый в доказательстве утверждения 6 автомат не нагляден, так как скрывает все детали разбора в контекстной функции ϕ_{decide}. Однако для большинства языков программирования, синтаксис которых в основном представим детерминированным контекстно-свободным языком, можно сохранить разумный баланс наглядности автомата и его контекстных функций, как, например, на рис. 2, *б*.

[10] Переход функции переходов τ называется *мнимым*, если его второй аргумент равен ϵ.

— $C = \{\tau(s_1, t, m) = (s_{n-1}, ms_2 \dots s_{n-2}) \mid$ для каждого перехода $\tau(s_n, t, s_{n-1}) = (s_{n-1}, \epsilon)\}$;

— d — мнимый переход из состояния s_n, если он существует, в котором исходное состояние s_n заменено на состояние s_1.

Очевидно, что все вышеперечисленные замены не меняют язык, задаваемый γ_{s1}-автоматом. Также очевидно, что достаточно выполнить конечное число таких замен для полного устранения мнимых дуг первого и второго видов, так как каждая замена сокращает их конечное число на единицу. □

Для устранения мнимых переходов третьего вида в модели γ_{s1}-автомата необходимо расширить переход третьего вида до перехода $\tau(s_1, t, m_1) = (s_2, \epsilon)$, выталкивающего состояние из магазина M и использующего его для определения состояния s_2. Такое расширение ввиду роста числа переходов, зависящих от содержимого магазина, было сочтено нецелесообразным для их наглядного изображения в рамках γ-схем. В рамках класса γ_1-автомата невозможно избавиться от мнимых переходов в состояния, которым сопоставлены функции f_e, изменяющие содержимое магазина M_e, так как модель γ-автомата не позволяет приписывать изменение стека M_e переходам функции переходов τ.

Некоторые из состояний γ_1-автомата могут оказаться недостижимыми[11]. Недостижимые состояния могут быть удалены для экономии размера автомата.

По γ_1-автомату часто можно построить γ_1-автомат с меньшим числом состояний, эквивалентный исходному. Соответствующий процесс называется *минимизацией* γ_1-автомата.

Утверждение 14. *Для любого γ_1-автомата можно построить эквивалентный ему γ_1-автомат с числом состояний меньшим или равным числу состояний исходного автомата.*

Доказательство. Минимизация γ_1-автомата схожа с минимизацией конечного автомата [9]. Сначала все состояния автомата разбиваются на класс конечных состояний (S_{end}) и классы состояний, разбиваемые отношением эквивалентности $\equiv_0$. Пара состояний (s_1, s_2) принадлежит отношению $\equiv_0$, если верен предикат $(A_1 \vee A_2) \wedge B_1 \wedge (C_1 \vee C_2)$, где:

— терм A_1 истинен тогда и только тогда, когда из состояний s_1 и s_2 не существует мнимых переходов;

— терм A_2 истинен тогда и только тогда, когда из состояний s_1 и s_2 существуют мнимые переходы;

— терм B_1 истинен тогда и только тогда, когда состояния s_1 и s_2 имеют один вид функции f_e;

— терм C_1 истинен тогда и только тогда, когда $s_1, s_2 \in S$;

— терм C_2 истинен тогда и только тогда, когда $s_1, s_2 \in S_k$ и $\mu(s_1) = \mu(s_2)$.

Отношение $\equiv_0$ рефлексивно и симметрично, так как все отношения термов предыдущего предиката рефлексивны и симметричны. Отношение $\equiv_0$ транзитивно, так как все отношения термов предыдущего предиката транзитивны, а отношения термов с одинаковой буквой не пересекаются. Тем самым отношение $\equiv_0$ является отношением эквивалентности.

Так как γ_1-схемы имеют дополнительные (повышающие их наглядность) обозначения, то для минимизации γ_1-автомата в терминах γ_1-схем к предикату отношения $\equiv_0$

[11] *Недостижимыми* называются состояния, не отмеченные обходом (при обходе рассматриваются переходы, заданные синтаксисом автомата) по переходам τ, τ_k и f_e (рассматриваемой здесь тоже как функция перехода) из состояния s_0.

необходимо добавить терм A_3, проверяющий, что из вершин, соответствующих состояниям s_1 и s_2, исходят сплошные непомеченные дуги. Также необходимо добавить терм D_1, проверяющий, что вершины, соответствующие состояниям s_1 и s_2, и все переходы из них, вызванные одинаковыми пометками, имеют одинаковые пометки транслирующих процедур.

Далее каждый полученный класс эквивалентности разбивается отношением эквивалентности $\equiv_1$. Пара состояний (s_1, s_2) принадлежит отношению $\equiv_1$, если любой одинаково помеченный переход τ, τ_k и f_e (рассматриваемой здесь как функция перехода) из этих состояний ведет к состояниям из одного класса эквивалентности предыдущего разбиения. Разбиение продолжается до тех пор, пока общее число классов эквивалентности будет увеличиваться, после чего совокупность представителей полученных классов эквивалентности будет состояниями минимизированного автомата. □

8. Описание транслятора

С помощью γ_1-схем можно реализовать транслятор, осуществляющий лексический, синтаксический и семантический разборы языков программирования. Для этого к вершинам и дугам γ_1-схем добавляются пометки, состоящие из идентификаторов транслирующих процедур. Реализация процедур в формализме γ_1-схем не уточняется и может иметь общий модифицируемый контекст исполнения, позволяющий задавать трансляцию произвольной сложности.

Транслятор разделяется на графическую, текстовую и интерпретирующую части. Графическая часть описывает γ_1-схему, задающую γ_1-автомат, который задает синтаксис языка. Текстовая часть содержит описание контекстно-зависимых переходов ("семантики отношений") и транслирующих процедур ("операционной семантики"), использующихся в γ_1-автомате. Интерпретирующая часть транслятора исполняет его графическую и текстовую части.

Разделение транслятора на графическую и текстовую части позволяет сочетать сильные качества обеих форм представления, что косвенно подтверждается распространенными средствами визуального моделирования [16]. Графические спецификации подходят для проектирования и документирования из-за богатства способов представления, легче усваиваемых кратковременной памятью человека. Текстовые спецификации лучше подходят для реализации программы по причине сочетания строгости, гибкости, компактности записи и переносимости. К другим преимуществам разделения транслятора на графическую и текстовую части относятся:

— простота модификаций входного языка путем наглядных изменений γ_1-схемы;

— высокая переносимость по причине интерпретируемости γ_1-автомата;

— упрощение реализации и тестирования процедур (проверки входных и выходных условий) за счет их логической обособленности;

— более экономное расходование памяти за счет общего контекста процедур без накладных расходов, присущих иерархии локальных контекстов вложенных функциональных вызовов;

— более эффективное и легкое восстановление общего контекста процедур, что обычно нереализуемо в языках высокого уровня без дополнительных расходов, связанных с обработкой исключений;

— автоматизация тестирования транслятора за счет наличия исполняемого описания синтаксиса в виде γ_1-автомата;

— возможность использования динамических оптимизаций на уровне интерпретатора γ_1-автомата, настраивающих его на транслируемый язык или даже на конкретную программу языка в процессе ее разбора;

— возможность легкой инструментации транслятора на уровне интерпретатора γ_1-автомата и сбора статистики программ транслируемого языка.

Компилятор переднего плана потокового языка Sisal 3.1 системы функционального программирования [17] спроектирован с использованием рассмотренного разделения на графическую, текстовую и интерпретирующую части. Компилятор переднего плана осуществляет лексический и синтаксический разбор (совмещенный с семантическим) текста программ языка Sisal 3.1 во внутреннее представление, основанное на потоке данных. Разработанный транслятор обеспечивает простоту учета модификаций языка, удовлетворительную скорость трансляции[12], качественные сообщения об ошибках и о предупреждениях и развитые механизмы восстановления после ошибок разбора. Использование γ_1-схем позволило сократить объем текста транслятора переднего плана языка Sisal 3.1 до 10 тыс. строк по сравнению с 30 тыс. строк кода аналогичной части транслятора OSC 12.0 для более простой версии языка Sisal 1.2 [18].

Заключение

В статье вводится и исследуется новая автоматная модель, предназначенная для описания синтаксического разбора языков программирования. Модель допускает наглядное изображение в виде графа и упрощает ручное построение высокоэффективных распознавателей, минуя недостатки, свойственные их автоматическому построению. В детерминированном случае автоматы введенной модели задают класс LL_1-языков. Задание более широких классов языков описывается неявно с помощью контекстных состояний, используемых для разбора трудных фрагментов языка и, как правило, не влияющих на наглядность графа всего автомата, что подтверждается опытом реализации компилятора переднего плана потокового языка Sisal 3.1. Показаны способы повышения эффективности автомата введенной модели, такие как минимизация состояний, устранение мнимых переходов и недостижимых состояний. Описываются преимущества реализации транслятора с использованием автоматов введенной модели.

Список литературы

[1] Ахо А.В., Сети Р., Ульман Д.Д. Компиляторы: принципы, технологии и инструменты. СПб.: Вильямс, 2001. 768 с.

[2] Johnson S.C. YACC — yet another compiler compiler. N.Y.: Murray Hill, 1975. 33 p. (Tech. Rep. / AT&T Bell Labs; N 32).

[3] Donnelly C., Stallman R.M. Bison Manual: Using the YACC-Compatible Parser Generator. Boston, MA: Free Software Foundation, 2003. 136 p.

[12]Невысокая скорость разбора, до десяти раз медленнее обычных показателей трансляторов промышленного уровня, объясняется тем, что задача построения высокоскоростного транслятора изначально не ставилась. В частности, высокая скорость разбора ограничивается интерпретацией γ_1-автомата и накладными расходами вызова методов COM-интерфейсов, с помощью которых транслятор переднего плана был реализован.

[4] Grune D., Jacobs C. Parsing techniques: a practical guide. New Jersey: Ellis Horwood, 1990. 322 p.

[5] Parr T. The Complete Antlr Reference Guide. Pragmatic Bookshelf, 2007. 361 p.

[6] Льюис Ф., Розенкранц Д., Стринз Р. Теоретические основы проектирования компиляторов. М.: Мир, 1979. 656 с.

[7] Leermakers R. The Functional Treatment of Parsing. Norwell, MA: Kluwer Academic Publishers, 1993. 158 p.

[8] Horspool R.N. Recursive ascent-descent parsing // Computer Languages. N.Y.: Pergamon Press, 1993. Vol. 18, N 1. P. 1–15.

[9] Полетаева И.А. Методы трансляции: конспект лекций. Новосибирск: НГТУ, 1997. 59 с.

[10] Nunes-Harwitt A. CPS Recursive Ascent Parsing // Proceedings of International LISP Conference, 2003. 6 p.

[11] McDonald J. Interactive Pushdown Automata Animation // ACM SIGCSE Bulletin. N.Y.: ACM Press, 2002. Vol. 34, N 1. P. 376–380.

[12] Cavalcante R., Finley T., Rodger S.H. A visual and interactive automata theory course with JFLAP 4.0 // ACM SIGCSE Bulletin. N.Y.: ACM Press, 2004. Vol. 36, N 1. P. 140–144.

[13] Вирт Н., Йенсен К. Паскаль. Руководство для пользователя и описание языка. М.: Финансы и статистика, 1989. 255 с.

[14] Касьянов В.Н., Поттосин И.В. Методы построения трансляторов. Новосибирск: Наука, 1986. 330 с.

[15] Стасенко А.П. Графический метаязык для описания транслятора // Сб. тр. аспирантов и молодых ученых "Молодая информатика". Новосибирск: ИСИ СО РАН, 2005. С. 105–113.

[16] Кузнецов Б.П. Психология автоматного программирования // Журнал BYTE, 2000. № 11. С. 22–29.

[17] SFP — An interactive visual environment for supporting of functional programming and supercomputing / Kasyanov V.N., Stasenko A.P., Gluhankov M.P. et al. // WSEAS Transactions on Computers. Athens: WSEAS Press, 2006. Vol. 5, N 9. P. 2063–2070.

[18] Стасенко А.П. Использование автоматного подхода для построения компилятора переднего плана // Тез. конф.-конкурса "Технологии Microsoft в теориии и практике программирования". Новосибирск, 2006. С. 37–39.

Поступила в редакцию 26 июля 2006 г.,
в переработанном виде — 25 мая 2008 г.