А.П. Стасенко

АВТОМАТНАЯ МОДЕЛЬ ВИЗУАЛЬНОГО ОПИСАНИЯ СИНТАКСИЧЕСКОГО РАЗБОРА*

1. 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 определённому классу, и затем генерируют распознающий автомат в виде таблицы и/или программного кода.

Требование принадлежности грамматики к определённому 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.

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 распозна-

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

ватели[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 путём наглядного описания синтаксиса языка в непосредственно исполняемом виде, и во-вторых позволяет упростить и повысить эффективность трансляторов, разрабатываемых с её помощью, путём устранения указанных выше недостатков написанных вручную трансляторов.

Предлагаемая автоматная модель наглядна, поскольку она задаётся 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,

[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).

по сути, является развитием и уточнением идеи, заложенной в синтаксических диаграммах Вирта.

Статья состоит из десяти разделов. В разд. 2 вводится определение модели γ-автомата и его частных случаев: γ_s-автоматов, γ_1-автоматов и γ_{s1}-автоматов, графическое изображение которых описывается далее в разделе 3. Наибольший интерес представляет класс γ_1-автоматов, допускающих детерминированное исполнение и класс контекстно-зависимых языков, однако для полноты картины в разд. 4 исследуется класс языков, представимых γ_s-автоматами. В двух следующих разделах рассмотрение сосредотачивается на γ_{s1}-автоматах, как важного частного случая γ_1-автоматов. В разд. 5 доказывается существование класса грамматик, эквивалентного классу γ_{s1}-автоматов. В разд. 6 исследуется класс языков, представимых γ_{s1}-автоматами, и показывается его ограниченность. В разд. 7 рассматривается соответствие между классами языков, классом γ_1-автоматов и некоторых его подклассов. В разд. 8 рассматриваются способы оптимизации γ_1-автоматов. Статья завершается разделом 9, в котором описываются преимущества реализации транслятора, синтаксический разбор которого описывается с помощью графического представления γ_1-автоматов.

2. ОПРЕДЕЛЕНИЕ МОДЕЛИ

Вводимая далее модель автомата для визуального описания синтаксического разбора (далее просто γ-автомата) основывается на классической модели магазинного автомата [14], на которую наложены ограничения на вид функции перехода, призванные повысить наглядность её графического представления. С другой стороны, модель γ-автомата содержит дополнения, направленные на расширение класса языков, представимых автоматами модели и дальнейшее повышение читабельности её графического представления.

Модель γ-автомата определяется кортежем $A = (T,\ S,\ S_k,\ s_0,\ S_{end},\ K,\ k_0,\ \Phi,\ \mu,\ \tau,\ \tau_k,\ T_k,\ f_e)$, где T — конечный входной алфавит, S — конечный алфавит состояний, $S_k \subseteq S$ — множество состояний с контекстным переходами, $s_0 \in S$ – начальное состояние, $S_{end} \subseteq S$ — множество конечных состояний, 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 функци-

ями, $\tau : (S \setminus S_k) \times (T \cup \{\epsilon^2\}) \times M \to 2^{S \times M^*}{}^3$ – функция переходов, $\tau_k : S_k \times T_k \to S$ — функция контекстных переходов, T_k – множество пометок контекстных переходов, $f_e : (S \setminus S_k) \times M_e \to M_e^*$ — функция изменения магазина исключений M_e[4].

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

1) семейство переходов $\tau(s_1, \delta, m) = (s_2, m)$ для всех $m \in M$, не зависящих от состояния и не меняющих содержимое магазина M;
2) семейство переходов $\tau(s_1, \delta, m) = (s_n, ms_2 \dots s_{n-1})$ для всех $m \in M$, не зависящих от содержимого магазина M, но добавляющих в него цепочку $s_2 \dots s_{n-1}$ (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 $s \in S \setminus S_k$ имеет один из следующих четырёх видов:

- функция вида $f_e(s, m_e) = (m_e)$ для всех $m_e \in M_e$, не зависящая от состояния и не меняющая содержимое магазина M_e;
- функция вида $f_e(s, m_e) = (m_e s_e)$ для всех $m_e \in M_e$, 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.

Автомат читает символы множества входной ленты с алфавитом T, в котором выделен специальный символ t_0, находящийся в конце ленты. Автомат имеет магазин M и магазин исключений M_e, содержащие символы из множества состояний $S \setminus S_k$. Конфигурацией автомата называется элемент множества $T^* \times S \times M^* \times M_e^* \times K$. Работа автомата определяется сменой конфигураций от начальной конфигурации $(\omega_1, s_0, \epsilon, \epsilon, k_0)$, где ω_1 обозначает начальную цепочку символов, до ко-

[2]Символ ϵ обозначает пустую цепочку.

[3]Звёздочка после множества X обозначает множество цепочек в алфавите X, включающее пустую цепочку.

[4]Помимо основного магазина M, автомат содержит дополнительный магазин исключений M_e.

нечной конфигурации, принадлежащей множеству $T^* \times S_{end} \times M^* \times M_e^* \times K$.

Для состояния $s \in S \setminus S_k$ смена конфигурации (ω, s, m, m_e, k) на конфигурацию $(\omega', s', m', m'_e, k')$ определяется следующим образом: $m'_e = f_e(s, m_e)$, $k' = k$, $(s', m') = \tau(s, \delta = \{t, \epsilon\}, m)$, где если $\delta = t$, то $\omega = t\omega'$, иначе $\omega' = \omega$. Если функция переходов τ неопределена для текущей конфигурации, то новая конфигурация определяется следующим образом: $\omega' = \omega$, состояние s' равно состоянию на вершине магазина $m''_e = f_e(s, m_e)$[5], $m' = m$, m'_e равно содержимому магазина m''_e с вытолкнутым состоянием.

Для состояния $s \in S_k$ смена конфигурации (ω, s, m, m_e, k) на конфигурацию $(\omega', s', m', m'_e, k')$ определяется иначе. Новое состояние s' определяется функцией переходов τ_k как $s' = \tau_k(s, t_k)$, где t_k и новое состояние контекста k' определяются функцией ϕ на основании текущего контекста $(t_k, k') = \phi(k)$, где функция $\phi \in \Phi$, определяется функцией разметки μ для текущего состояния $\phi = \mu(s)$. Содержимое магазинов и входной ленты не изменяется: $m' = m$, $m'_e = m_e$ и $\omega' = \omega$.

Определение 1. *Языком γ-автомата называется множество цепочек, начиная работу с которых, автомат достигает конечного состояния.*

Введём подклассы класса γ_s-автоматов.

Определение 2. *Под γ_s-автоматом подразумевается γ-автомат, в котором $S_k = \emptyset$ и все функции f_e не зависят и не меняют содержимое магазина M_e.*

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^*, S, M^*). Класс γ_s-автоматов интересен тем, что он является подклассом обычных недетерминированных магазинных автоматов с более ограниченным видом функции переходов τ, что, однако, как показано в разделе 4, не повлекло за собой сужения класса представимых ими языков.

Определение 3. *В γ-автомате переходом функции переходов τ называется множество $\tau(s, \delta, m)$.*

[5]Если $m_e = \epsilon$, то поведение γ-автомата не определено.

Определение 4. *В γ-автомате переход $\tau(s, \delta, m)$ функции переходов τ называется детерминированным, если $|\tau(s, \delta, m)| \leq 1$.*

Определение 5. *Под γ_1-автоматом (детерминированным γ-автоматом) подразумевается γ-автомат, в котором все переходы функции переходов τ детерминированы, а неоднозначность выбора между переходами $\tau(s_1, \epsilon, m)$ и $\tau(s_1, t, m)$ разрешается предпочтением перехода по t.*

Класс γ_1-автоматов важен тем, что такие автоматы допускают детерминированное исполнение и позволяют разбирать широкий класс контекстно-зависимых языков, что доказывается в разделе 7.

Определение 6. *Пересечение класса γ_s-автоматов и класса γ_1-автоматов обозначается как класс γ_{s1}-автоматов.*

Класс γ_{s1}-автоматов интересен тем, что он является подклассом обычных детерминированных магазинных автоматов с более ограниченным видом функции переходов τ, что, как показано в разделе 6, повлекло за собой сужение класса представимых ими языков.

Рассмотрим примеры γ-автоматов, первый из которых задаёт язык контекстно-свободной грамматики. Под грамматикой G подразумевается четвёрка $\{T, N, s_0, R\}$, где T — это терминальный алфавит, N — нетерминальный алфавит, $s_0 \in N$ — начальный символ грамматики, R — множество правил вывода, для контекстно-свободных грамматик имеющих вид $B \to \alpha$, где $B \in N$, $\alpha \in (N \cup T)^*$ [14]. Под языком грамматики подразумевается множество цепочек, выводимых из начального символа грамматики.

Рассмотрим примеры γ-автоматов. Начнём с примера γ_{s1}-автомата, задающего язык контекстно-свободной грамматики. Под грамматикой G подразумевается четвёрка $\{T, N, s_0, R\}$, где T — это терминальный алфавит, N — нетерминальный алфавит, $s_0 \in N$ — начальный символ грамматики, R — множество правил вывода, для контекстно-свободных грамматик имеющих вид $B \to \alpha$, где $B \in N$, $\alpha \in (N \cup T)^*$ [14]. Под языком грамматики подразумевается множество цепочек, выводимых из начального символа грамматики.

В качестве контекстно-свободного языка возьмём скобочной язык L_{s1}, задаваемый грамматикой $G_{s1} = (\{\text{‘(’}, \text{‘)’}\}, \{S, A\}, S, R)$, где $R = \{\text{“}S \to (A)\text{”}, \text{“}A \to S\text{”}, \text{“}A \to SA\text{”}\}$. Компоненты γ_{s1}-автомата A_{s1}, задающего язык L_{s1}, определяются так: $T = \{\text{‘(’}, \text{‘)’}, t_0\}$, $S = \{s_0, s_1, s_2, s_{end}\}$, $S_{end} = \{s_{end}\}$. Функция переходов τ определяется следующим образом:

$\tau(s_0, \text{‘(’}, m) = (s_1, ms_{end})$, $\tau(s_1, \text{‘(’}, m) = (s_1, ms_2)$, $\tau(s_1, \text{‘)’}, s) = (s, \epsilon)$, $\tau(s_2, \epsilon, m) = (s_1, m)$[6].

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

‘(()())’,	s_0, ϵ),	‘()())’,	s_1, s_{end}),	‘))’,	$s_1, s_{end}\ s_2$),
‘())’,	s_2, s_{end}),	‘())’,	s_1, s_{end}),	‘))’,	$s_1, s_{end}\ s_2$),
‘)’,	s_2, s_{end}),	‘)’,	s_1, s_{end}),	$(\epsilon$,	s_{end}, ϵ).

Рассмотрим пример γ_1-автомата, разбирающего контекстно-зависимый язык L_1. Зададим язык L_1, изменив скобочный язык L_{s1} дополнительным требованием того, что содержимое скобок с номером n в скобках с уровнем вложенности равным n может содержать цепочки произвольного алфавита не содержащего скобок. В то же время данные цепочки предполагаются принадлежащими некоторому контекстно-зависимому языку L_e, который необходимо попытаться разобрать и, в случае неудачи, пропустить оставшуюся часть цепочки. Данная задача очень близка к реальной задаче разбора языка программирования с восстановлением разбора после синтаксических ошибок.

Пусть γ_1-автомат A_e, распознающий язык L_e, имеет следующие компоненты: (T', S', S'_k, s'_0, S'_{end}, K', k'_0, Φ', μ', τ', τ'_k, T'_k, f'_e). Компоненты γ_1-автомата A_1, распознающего язык L_1, определяются так (в предположении, что все множества, участвующие в последующих объединениях, не пересекаются): $T = \{\text{‘(’}, \text{‘)’}, t_0\} \cup T'$, $S = \{s_0, \ldots, s_7, s_{end}\} \cup S'$, $S_k = \{s_2\} \cup S'_k$, $S_{end} = \{s_{end}\}$, $K = \{\text{содержимое магазина чисел } L\} \times K'$, $k_0 = (\text{магазин чисел } L, \text{содержащий число } 1, k'_0)$, $\Phi = \{\phi\} \cup \Phi'$, $T_k = \{true, false\} \cup T'_k$.

Функция μ автомата A_1 для состояния s_2 определяется как $\mu(s_2) = \phi$, а для состояний из множества S'_k — тождественна функции μ'. Функция переходов τ_k для состояния s_2 определяется как $\tau_k(s_2, true) = s_3$, $\tau_k(s_2, false) = s_5$, а для состояний из множества S'_k – тождественна функции τ'_k. Функция f_e для состояний из множества $S \setminus S_k$ определяется как $f_e(s_3, m_e) = (m_e s_7)$, $f_e(s_4, m_e) = (\epsilon)$, а для состояний из множества $S' \setminus S'_k$ — тождественна функции f'_e.

Функция ϕ автомата A_1 возвращает значение $true$, если число на

[6]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.

вершине магазина L равно глубине магазина L, и возвращает значение $false$ иначе. Также (после проверки предыдущего равенства) функция ϕ увеличивает на единицу число на вершине магазина L. Для целей реальной трансляции состояниям и переходам модели γ-автомата приписываются пометки, обозначающие различные транслирующие процедуры над контекстом K, что более подробно рассматривается в разделе 9. Тем самым вводятся функция ϕ_{push}, которая добавляет к магазину L единицу, и функция ϕ_{pop}, которая выталкивает число из магазина L. Состояние s_5 помечено функцией ϕ_{push}.

Функция переходов τ автомата A_1 для состояний из множества $S' \backslash S'_k$ тождественна функции перехода τ', а для состояний из $S \setminus S_k$ 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 $\phi_{pop}$), $\tau(s_3, \epsilon, m) = (s'_0, m)$, $\tau(s_4, \epsilon, m) = (s_1, m)$, $\tau(s_5, \epsilon, m) = (s_1, ms_6)$, $\tau(s_6, \epsilon, m) = (s_1, m)$, $\tau(s_7, t, m) = (s_7, m)$ для всех $t \in T \setminus \{\text{`)'}\}$, $\tau(s_7, \text{`)'}, m) = (s_1, m)$, $\tau(s'_{end}, \text{`)'}, m) = (s_4, m)$ для всех $s'_{end} \in S'_{end}$.

Если цепочка '*abc*' не принадлежит языку L_e, и на входной ленте находится допустимая цепочка '((*abc*))', то работа распознающего её γ_1-автомата A_1 будет определяться следующей последовательностью конфигураций:

('((*abc*))',	$s_0, \epsilon, \epsilon, 1$[7]),	('(*abc*))',	$s_1, s_{end}, \epsilon, 1$),	('*abc*))',	$s_2, s_{end}, \epsilon, 1$),
('*abc*))',	$s_3, s_{end}, \epsilon, 2$),	('*abc*))',	$s'_0, s_{end}, s_7, 2$),	... [8],	
('ω^9))',	$s_7, s_{end}, \epsilon, 2$),	... [10],		('))',	$s_1, s_{end}, \epsilon, 2$),
('))',	$s_2, s_{end}, \epsilon, 2$),	('))',	$s_5, s_{end}, \epsilon, 3$),	('))',	$s_1, s_{end}\ s_6, \epsilon, 3\ 1$),
(')',	$s_6, s_{end}, \epsilon, 3$),	(')',	$s_1, s_{end}, \epsilon, 3$),	(ϵ,	$s_{end}, \epsilon, \epsilon, \epsilon$).

3. ИЗОБРАЖЕНИЕ МОДЕЛИ

Для визуального представления модели γ-автомата было разработано графическое представление [15], называемое γ-схемой, которое отображает модель γ-автомата в виде ориентированного размеченного графа. Ниже перечислены классы вершин графа γ-схемы.

[7] Единицей обозначается содержимое магазина L.

[8] Многоточием обозначается работа автомата A_e, который не сможет разобрать цепочку '*abc*', то есть попадёт в неопределённое состояние.

[9] Символом ω обозначается цепочка из языка $\{\epsilon, \text{`}c\text{'}, \text{`}bc\text{'}, \text{`}abc\text{'}\}$.

[10] Многоточием обозначается повторение предыдущей конфигурации с уменьшающейся по одному символу входной строкой до тех пор, пока не исчерпается цепочка ω.

1. Прямоугольные вершины, соответствующие состояниям $s \in S$:
   (a) вершины, соответствующие состояниям $s \in S \setminus S_k$:
      i. вершины со сплошной границей, для состояний которых функция f_e имеет вид $f_e(s, m_e) = (m_e)$ и $f_e(s, m_e) = (m_e s_e)$ (рис. 1(а));
      ii. вершины со штриховой границей, для состояний которых 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 дополнительную пометку элементом множества $\phi \in \Phi$ (рис. 1(в)).
2. Круглые вершины v с пометкой "pop", задающие переходы третьего типа $\tau(s, \delta, s_1) = (s_1, \epsilon)$ из состояний s, вершины которых имеют исходящую дугу, которая входит в вершину v и имеет различный вид в зависимости от символа δ (рис. 1(м,н,о)).

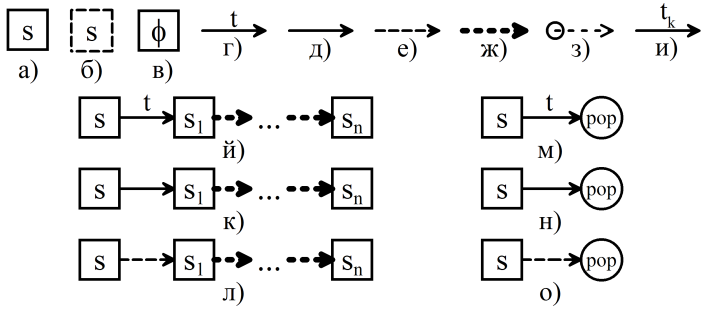

Рис. 1. Базовые элементы γ-схемы

Дуги γ-схем также подразделяются на несколько классов.

1. Дуги из вершин, соответствующих состояниям $s \in S \setminus S_k$:
   (a) дуги, задающие переходы первого типа различным способом в зависимости от второго аргумента δ функции переходов τ:
      i. сплошная дуга с меткой $t \in T$ для $\delta \in T$ (рис. 1(г));
      ii. сплошная дуга без метки для $\delta \in T \setminus T_s$, где где T_s — это метки всех других помеченных дуг, исходящих из вершины состояния s (рис. 1(д))
      iii. штриховая дуга без метки для $\delta = \epsilon$ (рис. 1(е));
   (b) непомеченные пунктирные дуги без символа круга в начале

дуги, образующие путь от вершины состояния s_1 до вершины состояния s_n и задающие переходы второго типа $\tau(s, \delta, m) = (s_n, ms_1 \dots s_{n-1})$ из состояния s, из вершины которого исходит дуга, которая входит в вершину состояния s_1 и имеет различный вид в зависимости от символа δ (рис. 1(й,к,л));

(c) непомеченные пунктирные дуги с символом круга в начале дуги, ведущие из вершины состояния s в вершину состояния s_e, которые участвуют в функции f_e вида $f_e(s, m_e) = (m_e s_e)$ и $f_e(s, m_e) = (s_e)$ (рис. 1(з)).

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

Если γ-схема задаёт γ_1-автомат, то она называется γ_1-схемой, и на неё накладывается несколько дополнительных условий. Ни из одной вершины не может исходить несколько одинаково помеченных дуг. Из вершины, соответствующей состоянию из $S \setminus S_k$, не может исходить более одной непомеченной сплошной или штриховой дуги, более одной пунктирной дуги без символа круга в начале дуги и более одной пунктирной дуги с символом круга в начале дуги.

Рассмотрим примеры γ-схем. На рис. 2 приведена γ_{s1}-схема, задающая γ_{s1}-автомат A_{s1} из раздела 2. На рис. 3 приведена γ_1-схема, задающая γ_1-автомат A_1 из разд. 2. Нетрудно заметить, что рис. 3 более компактно и наглядно задаёт описание автомата A_1, занимающее половину страницы в текстовом виде.

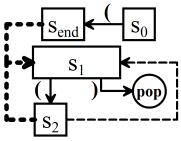

Рис. 2. Пример γ_{s1}-схемы

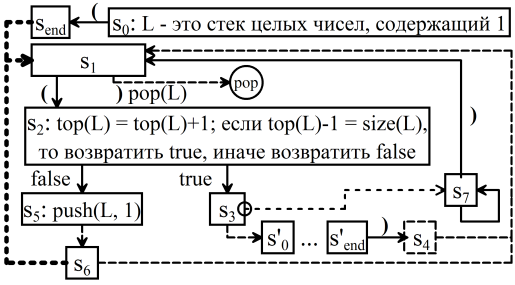

Рис. 3. Пример γ_1-схемы

4. СВОЙСТВА γ_S-АВТОМАТА

В данном разделе показывается, что γ_s-автоматы по мощности представимых языков равны классу контекстно-свободных языков. Таким образом, введённые ограничения на функцию переходов τ не повлекли за собой сужения класса языков, представимых недетерминированными магазинными автоматами.

Теорема 1. *Любой контекстно-свободный язык [14] можно задать γ_s-автоматом.*

Доказательство. Возьмём 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 $T = T' \cup \{t_0\}$. 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 возьмём $S_{end} = \{s_{end}\}$. 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 возьмём 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 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

$$\tau(r_{i,j-1}, \epsilon, m) = (\alpha_{i,j}, m r_{i,j}).$$

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. ■

Множества языков, задаваемых γ_s-автоматами и контекстно-свободными грамматиками, равны, так как любой γ_s-автомат задаёт контекстно-свободный язык, ввиду очевидного факта, что любой γ_s-автомат является магазинным автоматом. К сожалению, ввиду недетерминированности γ_s-автоматы не допускают эффективное[11] исполнение и поэтому не интересны для применений в реальных трансляторах [14].

5. 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. По определению, классы автоматов или грамматик эквивалентны между собой, если для каждого объекта A одного класса, найдётся объект в другом классе, задающий язык объекта A.

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.

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

Теорема 2. *Любой грамматике из класса G_γ соответствует γ_{s1}-автомат, задающий язык этой грамматики.*

Доказательство. Рассмотрим доказательство теоремы 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 1.a и 1.b теоремы 1 при $j = 1$, когда $r_{i,0} = B_i$. В пункте 1.a 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 1.b правило перехода $\tau(r_{i,0}, \epsilon, m) = (\alpha_{i,1}, m r_{i,1})$ будет детерминировано, так как

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

- 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, так как переход из них однозначно определяется терминалом $\alpha_{i,2}$, что 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 $r_{i,J}$. ■

Теорема 3. *Любому γ_{s1}-автомату соответствует грамматика из G_γ, задающая язык этого автомата.*

Доказательство. Возьмём γ_{s1}-автомат $A = \{T, S, s_0, S_{end}, \tau\}$. Согласно теореме 11, 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.

Определение 7. *Грамматика принадлежит классу LL_1 [6], если множества ВЫБОР для правил грамматики с одинаковой левой частью не пересекаются.*

Определение 8. *Множество ВЫБОР правила грамматики по определению содержит терминальные символы, при появлении которых под читающей головкой распознаватель должен применять это правило.*

Теорема 4. *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. ■

Теорема 5. *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. ■

6. СВЯЗЬ γ_{S1}-АВТОМАТА С КЛАССАМИ ЯЗЫКОВ

В этом разделе показывается, что множества языков, задаваемых γ_{s1}-автоматами и LL_1-грамматиками, равны. Тем самым, введённые ограничения на функцию переходов τ повлекли за собой сужение класса детерминированных контекстно-свободных языков (представимых детерминированными магазинными автоматами) до класса LL_1-языков.

Определение 9. *Язык принадлежит классу LL_1, если существует LL_1-грамматика, его задающая.*

Теорема 6. *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, путём 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 изменённой грамматики будут очевидно удовлетворены. ■

Теорема 7. *Любой γ_{s1}-автомат задаёт 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 "$N_1 \to N_2 t_1 \alpha_1$". Введём 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 "$N_1 \to N_3 t_2 \alpha_2 t_1 \alpha_1$". Делая ещё 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. Тем 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. 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. ■

7. 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 учёта

принадлежности к этому классу языков. Поэтому с точки зрения мощности множества поддерживаемых языков, интерес представляет более широкий класс γ_1-автоматов.

Теорема 8. *Любой контекстно-зависимый язык можно задать γ_1-автоматом.*

Доказательство.[12] Возьмём грамматику $G = (T', N', s'_0, R')$ произвольного контекстно-зависимого языка. Известно, что существует линейно-ограниченный автомат A_G, задающий язык L грамматики G [14].

Рассмотрим γ_1-автомат, в котором $T = T' \cup \{t_0\}$, $S = \{s_0, s_1, s_2, s_3\}$, $S_{end} = \{s_3\}$, $S_k = \{s_2\}$, $K = \{\text{строка символов } L\}$, $T_k = \{true, false\}$, $\mu(s_2) = \phi_{decide}$, $\Phi = \{\phi_{decide}\}$. Состояние s_0 помечено действием "сделать строку символов L пустой". Состояние s_1 помечено действием "добавить последний прочитанный символ входной ленты к строке символов L". Функция переходов τ, определяется так: $\tau(s_0, t, m) = (s_1, m)$, $\tau(s_0, t_0, m) = (s_2, m)$, $\tau(s_1, t, m) = (s_2, m)$, $\tau(s_1, t_0, m) = (s_2, m)$, для для любого $t \in T$ и любого $m \in M$.

Функция ϕ_{decide} реализует автомат A_G, используя в качестве его ленты строку символов L и возвращая $true$, если строка допустима, и $false$ 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 автомат с очевидностью задаёт язык L, так как, по сути, он только накапливает символы входной ленты для использования её в качестве ленты линейно-ограниченного автомата. ■

Используемый в доказательстве теоремы 8 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. 3.

Механизм иерархической обработки неопределённостей γ-автоматов, задаваемый функцией изменения магазина исключений f_e, был введён для удобства построения и сохранения наглядности реальных γ-схем,

[12] Доказательство ограничивается рассмотрением контекстно-зависимых языков в предположении, что контекст исполнения ограничен конечной памятью реальных вычислительных систем.

описывающих синтаксический анализ. Данный механизм позволяет избежать необходимости полного определения функции переходов τ, особенно в случае, когда реакция по умолчанию, которая может рассматриваться как ошибка разбора, легко унифицируема. По своей сути механизм иерархической обработки неопределённостей сходен с механизмом обработки исключений известных языков программирования.

Определение 10. *Под γ_{se1}-автоматом подразумевается γ_1-автомат, в котором $S_k = \emptyset$.*

Хотя механизм иерархической обработки неопределённостей не предназначается для расширения класса языков, представимых γ_{s1}-автоматами, можно показать, что язык γ_{se1}-автомата включает не контекстно-свободный язык и не включает контекстно-свободный язык. Таким образом, язык γ_{se1}-автомата не совпадает ни с одним из известных классов языков но, очевидно, включает в себя класс LL_1-языков.

Теорема 9. *Существует не контекстно-свободный язык, распознаваемый γ_{se1}-автоматом.*

Доказательство. Рассмотрим известный не контекстно-свободный язык, состоящий из цепочек вида $a^n b^n c^n$ для всех $n \geq 0$. Рассмотрим γ_{se1}-автомат, в котором $T = \{a, b, c, t_0\}$, $S = \{s_0, s_1, s_2, s_3, s_{yes}, s_{no}\}$, $S_{end} = \{s_{yes}\}$. 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 неопределённой ситуации в состоянии s_3, когда сработает переход в состояние на вершине магазина M_e, причём только если состоянием s_3 из магазина M_e было вытолкнуто n раз состояние s_{no}. Попав в состояние s_{no} автомат с неизбежность попадёт в неопределённое положение при исчерпании магазина M_e. ■

Теорема 10. *Существует контекстно-свободный язык, не распознаваемый γ_{se1}-автоматом.*

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 наперёд заданного числа k существует достаточно длинная цепочка, принадлежащая языку L, по первым k символам которой нельзя определить, нетерминал A или нетерминал B её задаёт.

Ввиду того, что нетерминалы A и B задаются с помощью ветвящейся рекурсии, их невозможно сымитировать итеративными способами, наподобие трюка, описанного при доказательстве теоремы 9. Значит, γ_{se1}-автомат, распознающий язык L, должен использовать магазин M для обеспечения корректной вложенности нетерминалов A и B. Различить, какой из этих нетерминалов задаёт распознаваемую цепочку языка, можно только в момент считывания терминала c или d. Даже если каким-то образом изменить содержимое магазина M_e, для того чтобы отразить, какая из букв c или d была встречена, при рекурсивном возврате по магазину M, автомат сможет воспользоваться этой информацией только при возникновении неопределённой ситуации (функция переходов τ не определена для текущей конфигурации), которая невозможна ввиду её отсутствия при движении автомата по той же части пути до момента добавления этой информации в магазин M_e. ■

8. ОПТИМИЗАЦИЯ γ_1-АВТОМАТА

В данном разделе показываются конструктивные способы оптимизации, применимые к классу γ_1-автоматов, такие как минимизация состояний, устранение мнимых переходов (перехода функции переходов τ со вторым аргументом равным ϵ) и недостижимых состояний, позволяющие повысить эффективность работы автомата для практических нужд. Сначала рассмотрим устранение мнимых переходов в γ_{s1}-автоматах.

Теорема 11. *В рамках модели γ_{s1}-автомата, не сужая класс представимых им языков, можно устранить все мнимые переходы первого и второго вида.*

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 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 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 $\tau(s_n, t, m) = (s_{n+m}, ms_{n+1} \dots s_{n+m-1})\}$;
- $C = \{\tau(s_1, t, m) = (s_{n-1}, ms_2 \dots s_{n-2}) \mid$ 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 вида, так как каждая замена сокращает их конечное число на единицу. ■

Для устранения мнимых переходов третьего вида в модели γ_{s1}-автомата необходимо расширение перехода 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 избавится 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 из состояния s_0. Недостижимыми называются состояния, не отмеченные обходом[13] по переходам τ, τ_k и f_e (тоже рассматриваемой здесь как функция перехода) из состояния s_0. Недостижимые состояния могут быть удалены для экономии размера автомата.

[13]При обходе рассматриваются переходы, заданные синтаксисом автомата.

По γ_1-автомату часто можно построить γ_1-автомат с меньшим числом состояний, эквивалентный исходному. Соответствующий процесс называется минимизацией γ_1-автомата. 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 $s_1 \in S \setminus S_k$ и $s_2 \in S \setminus S_k$;
- терм C_2 истинен тогда и только тогда, когда $s_1 \in S_k$, $s_2 \in S_k$ 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. Таким образом, 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 $\equiv_0$ 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 ведёт к состояниям из одного класса эквивалентности предыдущего разбиения. Разбиение продолжается до тех пор, пока общее число классов

эквивалентности увеличивается, после чего совокупность представителей полученных классов эквивалентности будет состояниями минимизированного автомата.

9. 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 распознаёт 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 распространёнными 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 следующие:

- простота модификаций входного языка путём наглядных изменений γ_1-схемы;
- высокая переносимость по причине интерпретируемости γ_1-автомата;
- использование γ_1-автомата для других целей, таких как тестирование;
- обособленность процедур, упрощающая их реализацию, тестирование и проверку входных и выходных условий каждой процедуры;

- эффективная раздельная трансляция γ_1-автомата и её процедур;
- общий контекст процедур более экономно расходует память за счёт отсутствия накладных расходов, присущих иерархии локальных контекстов, получаемых при вложенных функциональных вызовах;
- общий контекст процедур легче контролировать и быстрее восстанавливать в случае ошибочных ситуаций, что обычно нереализуемо в языках высокого уровня без дополнительных расходов, связанных с обработкой исключений;
- интерпретация γ_1-автомата допускает динамические оптимизации, настраивающие его на транслируемый язык или даже на конкретную программу языка в процессе её разбора;
- лёгкое изменение интерпретатора для инструментации транслятора 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 (совмещённый с семантическим) текста программ языка Sisal 3.1 во внутреннее представление, основанное на потоке данных. Разработанный транслятор обеспечивает простоту учёта модификаций языка, удовлетворительную скорость трансляции[14], качественные сообщения об ошибках и предупреждениях и развитые механизмы восстановления после ошибок разбора. Использование γ_1-схем позволило сократить объём текста транслятора переднего плана языка Sisal 3.1 до десяти тысяч строк по сравнению с тридцатью тысячами строк кода аналогичной части транслятора OSC 12.0 для более простой версии языка Sisal 1.2 [18].

10. ЗАКЛЮЧЕНИЕ

В статье вводится и исследуется новая автоматная модель, предназначенная для описания синтаксического разбора языков программирования. Модель допускает наглядное изображение в виде графа и упро-

[14]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.

щает ручное построение высокоэффективных распознавателей, минуя недостатки, свойственные их автоматическому построению. В детерминированном случае введённая автоматная модель допускает класс LL_1-языков. Разбор 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 введённой модели, такие как минимизация состояний, устранение мнимых переходов и недостижимых состояний.

СПИСОК ЛИТЕРАТУРЫ

1. Ахо А.В., Сети Р., Ульман Д.Д. Компиляторы: принципы, технологии и инструменты. — С.-Пб.: Вильямс, 2001. — 768 с.
2. Johnson S.C. YACC — yet another compiler compiler. — NY: 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.
4. Grune D., Jacobs C. Parsing techniques: a practical guide. — NJ: 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. — NY: 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 // Proc. of Internat. LISP Conf., 2003. — 6 p.
11. McDonald J. Interactive Pushdown Automata Animation // ACM SIGCSE Bull. NY: 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 Bull. — NY: 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 // Сборник трудов аспирантов и молодых учёных "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., Dortman P.A., Pyjov K.A., Sinyakov A.I. // 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.