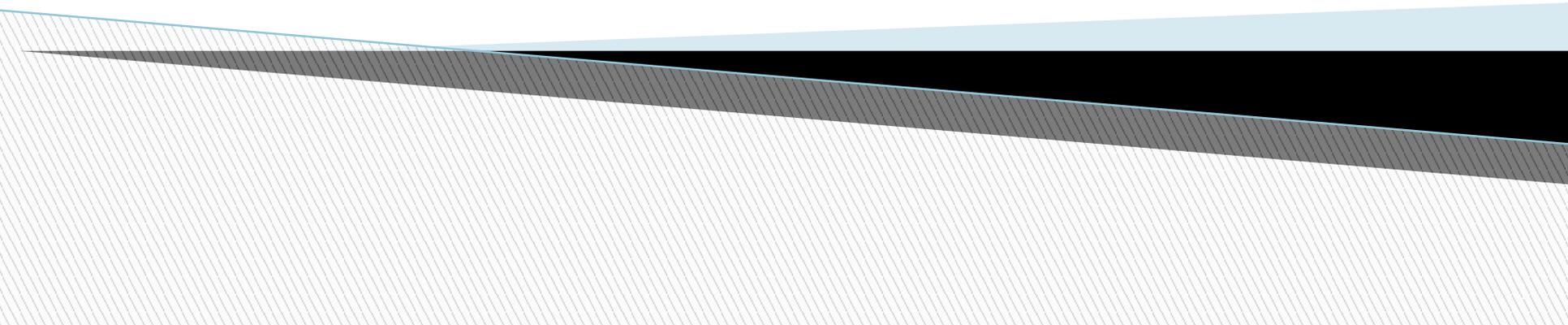


Лексер + Парсер. Часть 2.

Илья Филиппов
05.10.2015



Этапы компиляции

- Фронтэнд
- Машинно независимые оптимизации
- Код генерация + машинно зависимые оптимизации

Этапы фронтэнда

- Лексический анализатор (лексер)
- Синтаксический анализатор (парсер)
- Семантический анализатор
- Генерация промежуточного представления

Схема работы

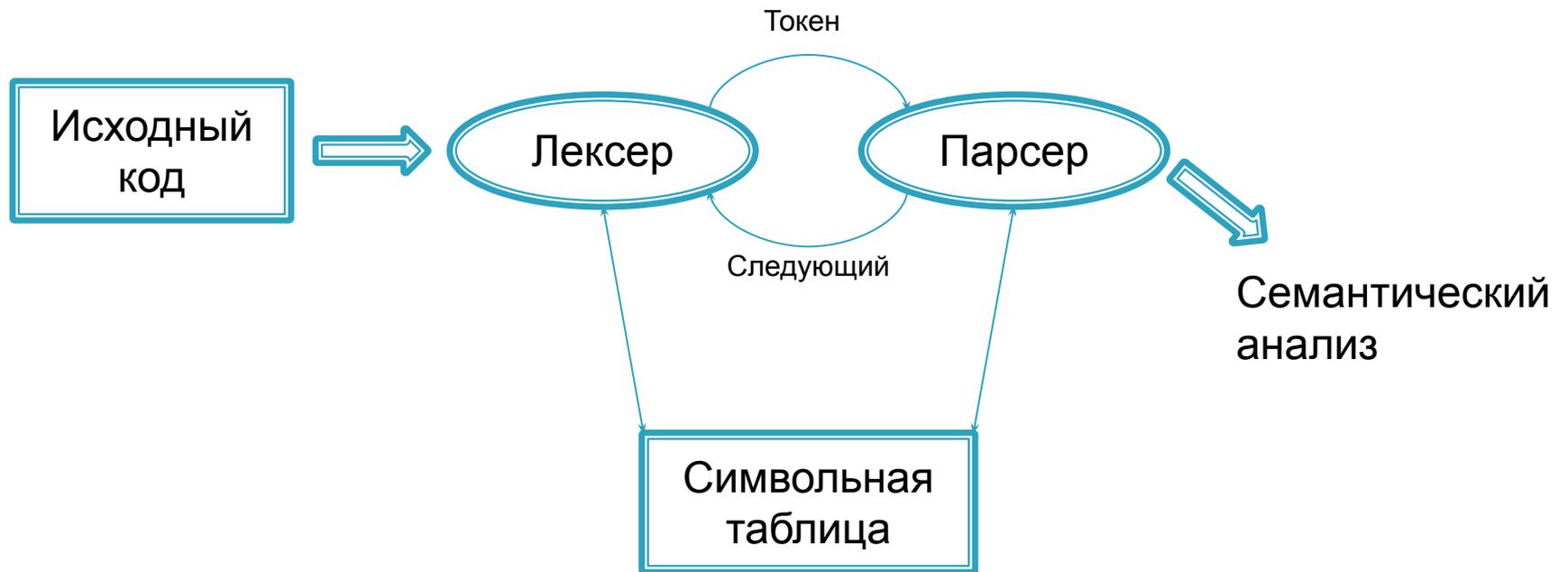


Схема работы

- ▣ Лексер формирует последовательности входных символов в лексемы и отправляет токены парсеру.
 - Лексема – минимальная единица языка, имеющая самостоятельный смысл.
 - Токен – тип лексемы + атрибут
- ▣ Парсер формирует исходное выражение языка, запрашивая токены.

Схема работы

- ▣ Обрабатываемые лексером и парсером последовательности символов и лексем напрямую зависят от спецификации языка.
- ▣ Необходим способ описания
 - «что в языке может быть»
 - «что в языке не может быть»

Строгие определения. Грамматики.

- Алфавит – множество символов, используемых в языке
- Терминальный символ - символ из алфавита
- Нетерминальный символ – символ не из алфавита
- Цепочка — последовательность символов
- Терминальная цепочка – цепочка, состоящая из терминальных символов
- Язык – множество терминальных цепочек

Грамматика — $G = (N, T, P, S)$

- N — множество нетерминальных символов (напр. A, B, C, \dots)
- T (иногда E) — алфавит терминальных символов ($N \cap T = \emptyset$)
- P — конечное множество правил вывода
 - $P = \{\alpha \rightarrow \beta \mid \alpha \in (N \cup T)^+; \beta \in (N \cup T)^*\}$
- $S \in N$ — аксиома (или начальный символ) грамматики

Пример грамматики:

$S \rightarrow aQbZ$

$Q \rightarrow ab \mid cc \mid Qd$

$Z \rightarrow aQa \mid c \mid \varepsilon$

Классификация грамматик по Хомскому

- Тип 0: неограниченные
- Тип 1: контекстно-зависимые / неукорачивающие
- Тип 2: контекстно-свободные
- Тип 3: регулярные:
праволинейные/леволинейные

Строгие определения. Типы грамматик.

- Регулярные грамматики:
 - праволинейные ($A \rightarrow w, A \rightarrow wB, A, B \in N, w \in T^*$)
 - леволинейные ($A \rightarrow w, A \rightarrow Bw, A, B \in N, w \in T^*$)
- Контекстно-свободные грамматики:
 - ($A \rightarrow w, A \in N, w \in (T \cup N)^*$)

Соответствие языков и грамматик

- ▣ Тип 0 (неограниченные): естественные языки:
 - Русский
 - Английский
- ▣ Тип 2 (контекстно-свободные): большинство языков программирования:
 - Java
 - C++
- ▣ Тип 3: (регулярные): описание отдельных лексем в языках программирования:
 - Идентификатор
 - Числовая константа

Способы разбора грамматик

- ▣ Тип 2 контекстно – свободная грамматика:
 - может быть описана с помощью конечного автомата с магазинной памятью
 - Используется для анализа последовательности токенов синтаксическим анализатором
- ▣ Тип 3 регулярная грамматика:
 - Может быть описана с помощью конечного автомата
 - Используется для формирования лексем лексическим анализатором

Строгие определения. Конечные автоматы с магазинной памятью.

Недетерминированный

Недетерминированный автомат с магазинной памятью (МП-автомат) — семёрка $M = (Q, T, \Gamma, D, q_0, Z_0, F)$, где

1. Q — конечное множество состояний, представляющее всевозможные состояния управляющего устройства
2. T — конечный входной алфавит
3. Γ — конечный алфавит магазинных символов
4. D — отображение множества $Q \times (T \cup \{\varepsilon\}) \times \Gamma$ в множество всех конечных подмножеств $Q \times \Gamma^*$, называемое функцией переходов
5. $q_0 \in Q$ — начальное состояние управляющего устройства
6. $Z_0 \in \Gamma$ — символ, находящийся в магазине в начальный момент (начальный символ магазина)
7. $F \subseteq Q$ — множество заключительных состояний

Детерминированный

Кроме того, должны выполняться следующие условия:

1. Множество $D(q, a, Z)$ содержит не более одного элемента для любых $q \in Q, a \in T \cup \{\varepsilon\}, Z \in \Gamma$
2. Если $D(q, \varepsilon, Z) \neq \emptyset$, то $D(q, a, Z) = \emptyset$ для всех $a \in T$

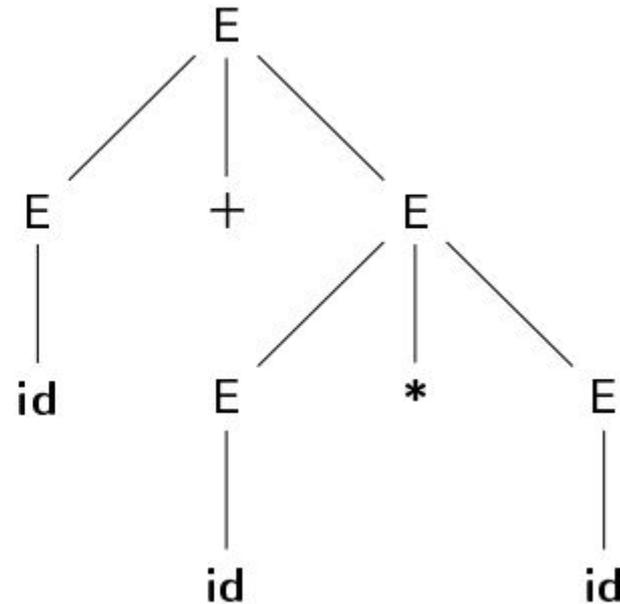
Варианты синтаксического анализа

- ▣ Нисходящий анализ
 - Метод рекурсивного спуска (парсер с возвратом)
 - Предиктивный анализатор (предсказывающий анализатор) (LL анализатор)
- ▣ Восходящий анализ (LR анализатор)

Дерево разбора

□ $id + id * id$

$E \rightarrow E A E \mid (E) \mid -E \mid id$
 $A \rightarrow + \mid - \mid * \mid /$



Метод рекурсивного спуска

- Дерево разбора строится сверху вниз, слева направо
- Токены от лексера рассматриваются в порядке поступления

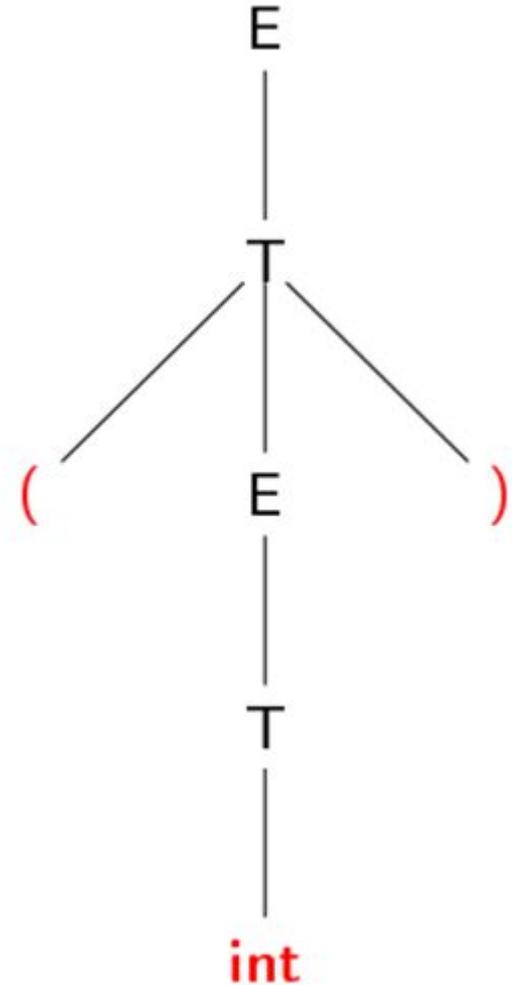
Метод рекурсивного спуска

- Каждому нетерминалу соответствует процедура, в которую жёстко зашиты продукции данного нетерминала
- Процедура последовательно сравнивает пришедшие токены с терминалами продукции
- Если ни одна продукция не подходит – ошибка
- Если встречается нетерминал управление переходит в процедуру нетерминала

Пример применения метода рекурсивного спуска

$$E \rightarrow T \mid T + E$$
$$T \rightarrow \mathbf{int} \mid \mathbf{int} * T \mid (E)$$

(int₅)



Рекурсивный спуск, минусы

- Грамматика жёстко зашита в алгоритм. В последующих алгоритмах на основании грамматики строятся таблицы
- Алгоритм выполняется за экспоненциальное время
- Алгоритм может зацикливаться
- Использование неявного стека

Предиктивный анализатор

- Идея – убрать откатывание. По каждому символу должно быть изначально ясно куда переходим.
- Преобразование грамматики
 - Удаление левой рекурсии
 - Левая факторизация
- Вычисление множеств
 - FIRST
 - FOLLOW
- Составление таблицы

Предиктивный анализатор

Удаление левой рекурсии

■ $A \rightarrow Aa \mid Ab \mid \dots \mid Ak \mid m \mid n \mid \dots \mid z$



■ $A \rightarrow mB \mid nB \mid \dots \mid zB$

■ $B \rightarrow aB \mid bB \mid \dots \mid kB \mid \epsilon$

Левая факторизация

■ $A \rightarrow ac \mid adf \mid adg \mid b$



■ $A \rightarrow aB \mid b$

■ $B \rightarrow c \mid df \mid dg$



■ $A \rightarrow aB \mid b$

■ $B \rightarrow c \mid dC$

■ $C \rightarrow f \mid g$

Предиктивный анализатор

Пример преобразования грамматики:

$G = \{S, A, B\}, \{a, b, c\}, P, S\}$

P:

- $S \rightarrow SAbB \mid a$
- $A \rightarrow ab \mid aa \mid \epsilon$
- $B \rightarrow c \mid \epsilon$

Удаление левой рекурсии для S:

- $S \rightarrow aS_1$
- $S_1 \rightarrow AbBS_1 \mid \epsilon$

Левая факторизация для A:

- $A \rightarrow aA_1 \mid \epsilon$
- $A_1 \rightarrow b \mid a$

Итоговая грамматика:

- $S \rightarrow aS_1$
- $S_1 \rightarrow AbBS_1 \mid \epsilon$
- $A \rightarrow aA_1 \mid \epsilon$
- $A_1 \rightarrow b \mid a$
- $B \rightarrow c \mid \epsilon$

Предиктивный анализатор

- Определение множества FIRST
 - $\text{FIRST}(a)$, $a \in (\text{NUT})^*$ - множество терминалов или ϵ , с которых может начинаться a .
- Построение множества FIRST

Для терминалов

- Для любого терминала x , $x \in T$, $\text{FIRST}(x) = \{x\}$

Для нетерминалов

- Если X — нетерминал, то положим $\text{FIRST}(X) = \{\emptyset\}$
- Если в грамматике есть правило $X \rightarrow \epsilon$, то добавим ϵ к $\text{FIRST}(X)$
- Для каждого нетерминала X и для каждого правила вывода $X \rightarrow Y_1 \dots Y_k$

добавим в $\text{FIRST}(X)$ множества FIRST всех символов в правой части правила до первого, из которого не выводится ϵ , включая его

Для цепочек

- Для цепочки символов $X_1 \dots X_k$ FIRST есть объединение FIRST входящих в цепочку символов до первого, у которого $\epsilon \notin \text{FIRST}$, включая его.

Предиктивный анализатор

- Определение множества FOLLOW
 - FOLLOW(A), $A \in N$ – множества терминалов, которые могут появиться сразу за A, включая концевой маркер \$
- Построение множества FOLLOW

- Пусть FOLLOW(X) = { \emptyset }
- Если X — аксиома грамматики, то добавить в FOLLOW маркер \$
- Для всех правил вида $A \rightarrow \alpha X \beta$ добавить FIRST(β) \setminus \{\epsilon\} к FOLLOW(X) (за X могут следовать те символы, с которых начинается β)
- Для всех правил вида $A \rightarrow \alpha X$ и $A \rightarrow \alpha X \beta$, $\epsilon \in \text{FIRST}(\beta)$ добавить FOLLOW(A) к FOLLOW(X) (то есть, за X могут следовать все символы, которые могут следовать за A, в случае, если в правиле вывода символ X может оказаться крайним правым)
- Повторять предыдущие два пункта, пока возможно добавление символов в множество

Предиктивный анализ

Пример построения FIRST, FOLLOW

- $S \rightarrow aS_1$
- $S_1 \rightarrow AbBS_1 \mid \epsilon$
- $A \rightarrow aA_1 \mid \epsilon$
- $A_1 \rightarrow b \mid a$
- $B \rightarrow c \mid \epsilon$

- $FIRST(S) = \{a\}$
- $FIRST(A) = \{a, \epsilon\}$
- $FIRST(A_1) = \{b, a\}$
- $FIRST(B) = \{c, \epsilon\}$
- $FIRST(S_1) = \{a, b, \epsilon\}$

- $FIRST(aS_1) = \{a\}$
- $FIRST(AbBS_1) = \{a, b\}$
- $FIRST(\epsilon) = \{\epsilon\}$
- $FIRST(aA_1) = \{a\}$

- $FIRST(a) = \{a\}$
- $FIRST(b) = \{b\}$
- $FIRST(c) = \{c\}$

- $FOLLOW(S) = \{\$ \}$
- $FOLLOW(S_1) = \{\$ \}$ (S_1 — крайний правый символ в правиле $S \rightarrow aS_1$)
- $FOLLOW(A) = \{b\}$ (после A в правиле $S_1 \rightarrow AbBS_1$ следует b)
- $FOLLOW(A_1) = \{b\}$ (A_1 — крайний правый символ в правиле $A \rightarrow aA_1$, следовательно, добавляем $FOLLOW(A)$ к $FOLLOW(A_1)$)
- $FOLLOW(B) = \{a, b, \$ \}$ (добавляем $FIRST(S_1) \setminus \{\epsilon\}$ (следует из правила $S_1 \rightarrow AbBS_1$), $FOLLOW(S_1)$ (так как есть $S_1 \rightarrow \epsilon$))

Предиктивный анализ

□ Построение таблицы

В таблице M для пары нетерминал-терминал (в ячейке $M[A, a]$) указывается правило, по которому необходимо выполнять свёртку входного слова. Заполняется таблица следующим образом: для каждого правила вывода заданной грамматики $A \rightarrow \alpha$ (где под α понимается цепочка в правой части правила) выполняются следующие действия:

1. Для каждого терминала $a \in \text{FIRST}(\alpha)$ добавить правило $A \rightarrow \alpha$ к $M[A, a]$
2. Если $\epsilon \in \text{FIRST}(\alpha)$, то для каждого $b \in \text{FOLLOW}(A)$ добавить $A \rightarrow \alpha$ к $M[A, b]$
3. $\epsilon \in \text{FIRST}(\alpha)$ и $\$ \in \text{FOLLOW}(A)$, добавить $A \rightarrow \alpha$ к $M[A, \$]$
4. Все пустые ячейки — ошибка во входном слове

Предиктивный анализ

Пример построения таблицы

	a	b
S	$S \rightarrow aS_1$ (Первое правило, вывод $S \rightarrow aS_1$, $a \in \text{FIRST}(aS_1)$)	Error (Четвёртое правило)
S₁	$S_1 \rightarrow AbBS_1$ (Первое правило, вывод $S_1 \rightarrow AbBS_1$, $a \in \text{FIRST}(AbBS_1)$)	$S_1 \rightarrow AbBS_1$ (Первое правило, вывод $S_1 \rightarrow AbBS_1$, $b \in \text{FIRST}(AbBS_1)$)
A	$A \rightarrow aA_1$ (Первое правило, вывод $A \rightarrow aA_1$, $a \in \text{FIRST}(aA_1)$)	$A \rightarrow \epsilon$ (Второе правило, вывод $A_1 \rightarrow \epsilon$, $b \in \text{FOLLOW}(A_1)$)
A₁	$A_1 \rightarrow a$ (Первое правило, вывод $A_1 \rightarrow a$, $a \in \text{FIRST}(a)$)	$A_1 \rightarrow b$ (Первое правило, вывод $A_1 \rightarrow b$, $b \in \text{FIRST}(b)$)
B	$B \rightarrow \epsilon$ (Второе правило, вывод $B \rightarrow \epsilon$, $a \in \text{FOLLOW}(B)$)	$B \rightarrow \epsilon$ (Второе правило, вывод $B \rightarrow \epsilon$, $a \in \text{FOLLOW}(B)$)
	c	\$
S	Error (Четвёртое правило)	Error (Четвёртое правило)
S₁	Error (Четвёртое правило)	$S_1 \rightarrow \epsilon$ (Третье правило, вывод $S_1 \rightarrow \epsilon$, $\epsilon \in \text{FIRST}(\epsilon)$, $\$ \in \text{FOLLOW}(S_1)$)
A	Error (Четвёртое правило)	Error (Четвёртое правило)
A₁	Error (Четвёртое правило)	Error (Четвёртое правило)
B	$B \rightarrow c$ (Первое правило, вывод $B \rightarrow c$, $c \in \text{FIRST}(c)$)	$B \rightarrow \epsilon$ (Третье правило, вывод $B \rightarrow \epsilon$, $\$ \in \text{FOLLOW}(B)$)

Предиктивный анализ, разбор

Процесс разбора строки довольно прост. Его суть в следующем:
на каждом шаге считывается верхний символ v из стека анализатора и берется крайний символ c входной цепочки.

- Если v - терминальный символ
 - Если v совпадает с c , то они оба уничтожаются, происходит сдвиг
 - Если v не совпадает с c , то сигнализируется ошибка разбора
 - Если v - нетерминальный символ, c возвращается в начало строки, вместо v в стек возвращается правая часть правила, которое берется из ячейки таблицы $M[v, c]$
- Процесс заканчивается, когда и строка и стек дошли до концевого маркера (#).

Предиктивный анализ, пример

стек	строка	действие
S#	aabbaabcb\$	$S \rightarrow aS_1$
aS₁#	aabbaabcb\$	сдвиг
S₁#	abbaabcb\$	$S_1 \rightarrow AbBS_1$
AbBS₁#	abbaabcb\$	$A \rightarrow aA_1$
aA₁bBS₁#	abbaabcb\$	сдвиг
A₁bBS₁#	bbaabcb\$	$A_1 \rightarrow b$
bbBS₁#	bbaabcb\$	сдвиг
bBS₁#	baabcb\$	сдвиг
BS₁#	aabcb\$	$B \rightarrow \epsilon$
S₁#	aabcb\$	$S_1 \rightarrow AbBS_1$
AbBS₁#	aabcb\$	$A \rightarrow aA_1$
AbBS₁#	aabcb\$	$A \rightarrow aA_1$

aA₁bBS₁#	aabcb\$	сдвиг
A₁bBS₁#	abcb\$	$A_1 \rightarrow a$
abBS₁#	abcb\$	сдвиг
bBS₁#	bcб\$	сдвиг
BS₁#	cb\$	$B \rightarrow c$
cS₁#	cb\$	сдвиг
S₁#	b\$	$S_1 \rightarrow AbBS_1$
AbBS₁#	b\$	$A \rightarrow \epsilon$
bBS₁#	b\$	сдвиг
BS₁#	\$	$B \rightarrow \epsilon$
S₁#	\$	$S_1 \rightarrow \epsilon$
#	\$	ГОТОВО

Восходящий анализатор

- ▣ Строим дерево разбора начиная с листьев
- ▣ Для большинства языков программирования можно построить LR грамматику
- ▣ Наиболее общий метод анализа без отката
- ▣ Существуют генераторы LR анализаторов

Построение LR(1) анализатора

- 1. Пополнение грамматики
- 2. Построение канонической системы множеств допустимых LR(1)-ситуаций
- 3. Построение таблицы анализатора
 - 3.1 Построение таблицы Goto
 - 3.2 Построение таблицы Actions
- 4. Разбор цепочки

Пополнение грамматики

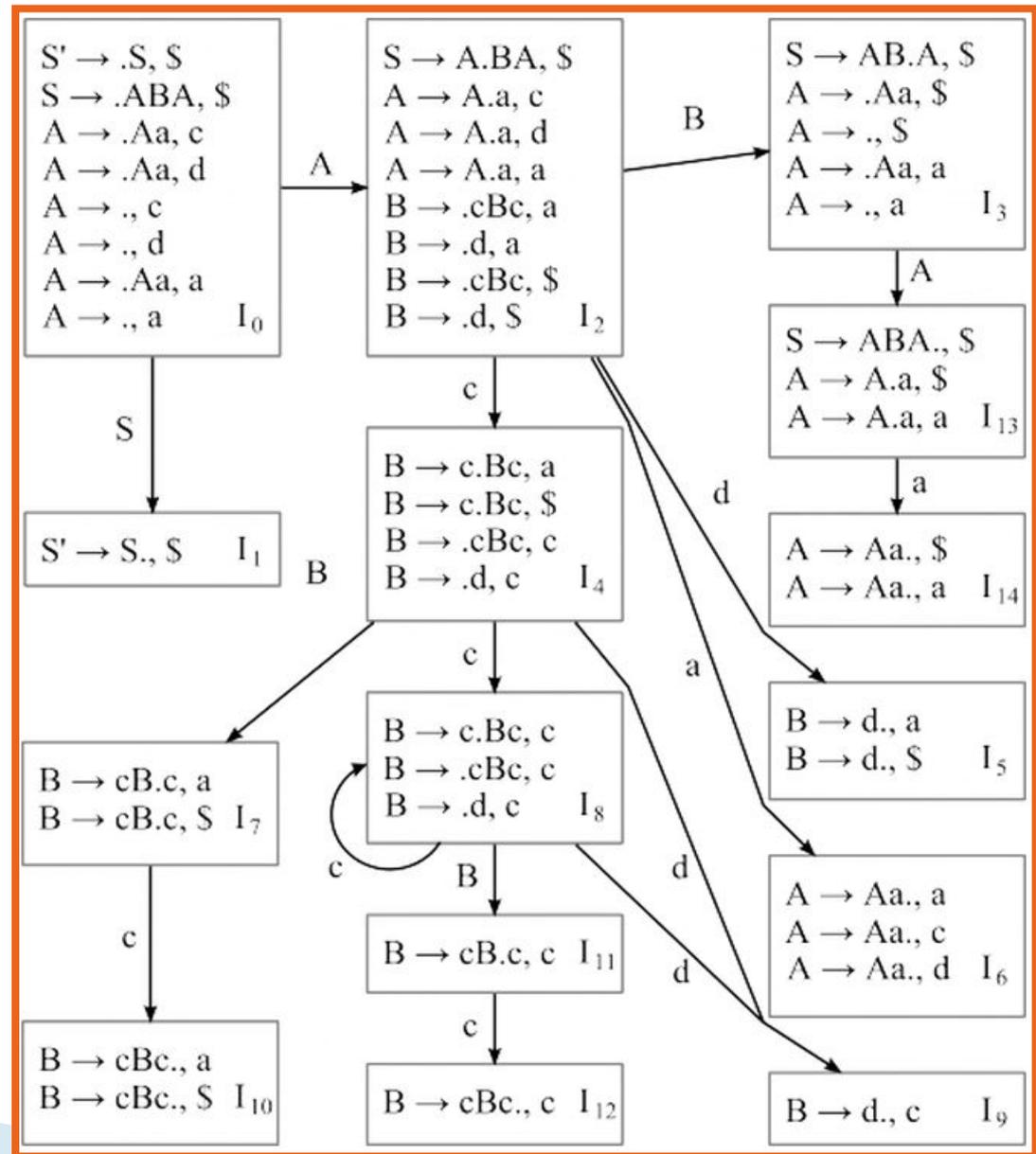
- Добавим новое правило $S' \rightarrow S$, и сделаем S' аксиомой грамматики.
- Допуск имеет место \Leftrightarrow когда возможно осуществить свёртку по правилу $S \rightarrow S'$.

Каноническая система множеств

- Вначале имеется множество I_0 с конфигурацией анализатора $S' \rightarrow .S, \$$
- К этой конфигурации применяется операция закрытия до тех пор, пока в результате её применения не перестанут добавляться новые конфигурации.
- Строятся переходы в новые множества путём сдвига точки на один символ вправо

Пример

- $S' \rightarrow S$
- $S \rightarrow ABA$
- $A \rightarrow Aa \mid \epsilon$
- $B \rightarrow cBc \mid d$



Построение Goto

- Если в канонической системе множеств есть переход из I_i в I_j по символу A , то в $\text{Goto}(I_i, A)$ мы ставим состояние I_j . После заполнения таблицы полагаем, что во всех пустых ячейках $\text{Goto}(I_i, A) = \text{Error}$
- После заполнения таблицы полагаем, что во всех пустых ячейках $\text{Goto}(I_i, A) = \text{Error}$

Построение action

- Если есть переход по терминалу a из состояния I_i в состояние I_j , то $\text{Action}(I_i, a) = \text{Shift}(I_j)$
- Если $A \neq S'$ и есть конфигурация $A \rightarrow \alpha.$, a , то $\text{Action}(I_i, a) = \text{Reduce}(A \rightarrow \alpha)$
- Для состояния I_i , в котором есть конфигурация $S' \rightarrow S., \$$, $\text{Action}(I_i, \$) = \text{Accept}$
- Для всех пустых ячеек $\text{Action}(I_i, a) = \text{Error}$

Пример

- $S' \rightarrow S$
- $S \rightarrow ABA$
- $A \rightarrow Aa \mid \epsilon$
- $B \rightarrow cBc \mid d$

	Action				Goto						
	a	c	d	\$	S	S'	A	B	a	c	d
l_0	Reduce($A \rightarrow \epsilon$)	Reduce($A \rightarrow \epsilon$)	Reduce($A \rightarrow \epsilon$)		l_1		l_2				
l_1				Accept							
l_2	Shift(l_6)	Shift(l_4)	Shift(l_5)					l_3	l_6	l_4	l_5
l_3	Reduce($A \rightarrow \epsilon$)			Reduce($A \rightarrow \epsilon$)			l_{13}				
l_4		Shift(l_8)	Shift(l_9)					l_7		l_8	l_9
l_5	Reduce($B \rightarrow d$)			Reduce($B \rightarrow d$)							
l_6	Reduce($A \rightarrow Aa$)	Reduce($A \rightarrow Aa$)	Reduce($A \rightarrow Aa$)								
l_7		Shift(l_{10})								l_{10}	
l_8		Shift(l_8)	Shift(l_9)					l_{11}		l_8	l_9
l_9		Reduce($B \rightarrow d$)									
l_{10}	Reduce($B \rightarrow cBc$)			Reduce($B \rightarrow cBc$)							
l_{11}		Shift(l_{12})								l_{12}	
l_{12}		Reduce($B \rightarrow cBc$)									
l_{13}	Shift(l_{14})			Reduce($S \rightarrow ABA$)						l_{14}	
l_{14}	Reduce($A \rightarrow Aa$)			Reduce($A \rightarrow Aa$)							

Восходящий анализ, разбор

На каждом шаге считывается верхний символ v из стека анализатора и берется крайний символ c входной цепочки.

Если в таблице действий на пересечении v и c находится:

- Shift(I_k), то в стек кладется c и затем I_k . При этом c удаляется из строки.
- Reduce($A \rightarrow u$), то из верхушки стека вычищаются все терминальные и нетерминальные символы, составляющие цепочку u , после чего смотрится состояние I_m , оставшееся на верхушке. По таблице переходов на пересечении I_m и A находится следующее состояние I_s . После чего в стек кладется A , а затем I_s . Строка остается без изменений.
- Асепт, то разбор закончен
- пустота - ошибка

Восходящий анализ, пример

Стек	Строка	Действие
l_0	aaaccdcc\$	Reduce($A \rightarrow \epsilon$), goto l_2
$l_0 A l_2$	aaaccdcc\$	Shift(l_6)
$l_0 A l_2 a l_6$	aaccdcc\$	Reduce($A \rightarrow Aa$), goto l_2
$l_0 A l_2$	aaccdcc\$	Shift(l_6)
$l_0 A l_2 a l_6$	accdcc\$	Reduce($A \rightarrow Aa$), goto l_2
$l_0 A l_2$	accdcc\$	Shift(l_6)
$l_0 A l_2 a l_6$	ccdcc\$	Reduce($A \rightarrow Aa$), goto l_2
$l_0 A l_2$	ccdcc\$	Shift(l_4)
$l_0 A l_2 c l_4$	cdcc\$	Shift(l_8)
$l_0 A l_2 c l_4 c l_8$	dcc\$	Shift(l_9)
$l_0 A l_2 c l_4 c l_8 d l_9$	cc\$	Reduce($B \rightarrow d$), goto l_{11}
$l_0 A l_2 c l_4 c l_8 B l_{11}$	cc\$	Shift(l_{12})
$l_0 A l_2 c l_4 c l_8 B l_{11} c l_{12}$	c\$	Reduce($B \rightarrow cBc$), goto l_7
$l_0 A l_2 c l_4 B l_7$	c\$	Shift(l_{10})
$l_0 A l_2 c l_4 B l_7 c l_{10}$	\$	Reduce($B \rightarrow cBc$), goto l_3
$l_0 A l_2 B l_3$	\$	Reduce($A \rightarrow \epsilon$), goto l_{13}
$l_0 A l_2 B l_3 A l_{13}$	\$	Reduce($S \rightarrow ABA$), goto l_1
$l_0 S l_1$	\$	Accept

Обрабатываемые ошибки

- Баланс скобок {...{...(...)...(((...))...)}...(...)}...
- Верность выражений “+” между expr

Должны быть ясны вопросы:

- ▣ Место выполнения синтаксического анализатора
- ▣ Нисходящий анализ
 - Метод рекурсивного спуска
 - Предиктивный анализатор
- ▣ Восходящий анализ