# Лексер + Парсер. Часть 2.

Илья Филиппов
05.10.2015

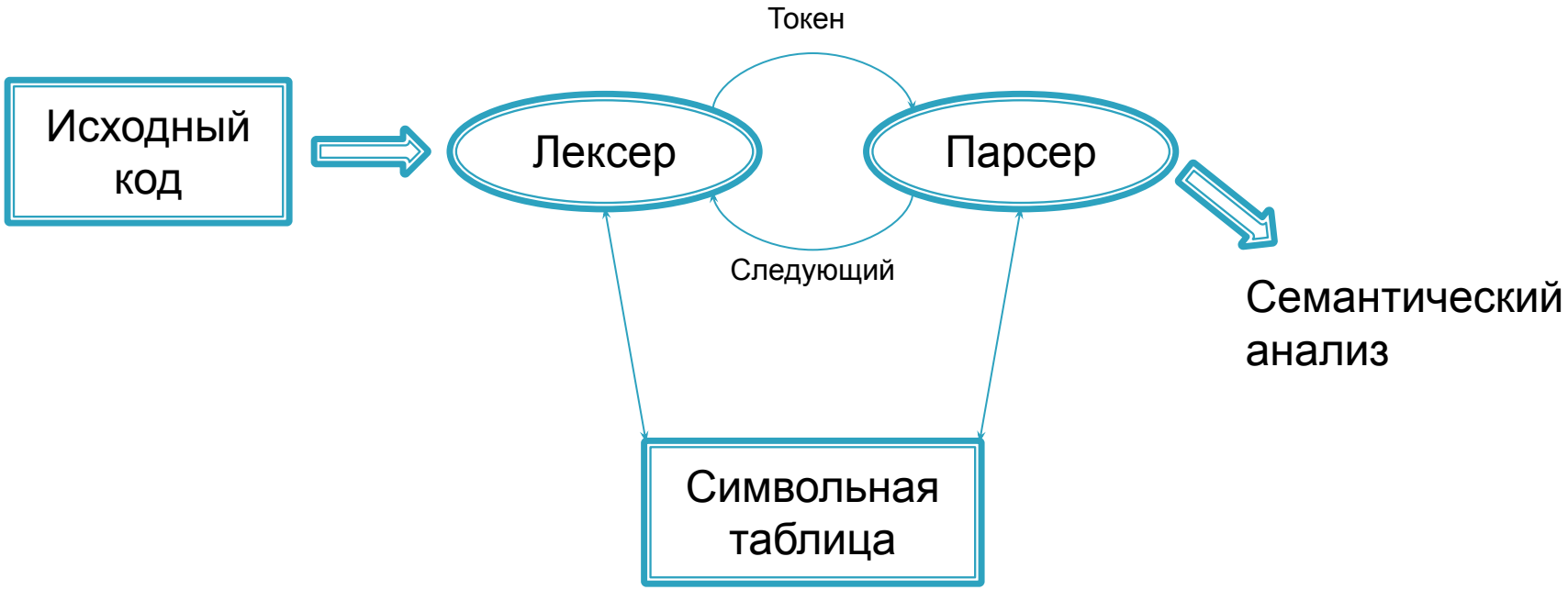
# Этапы компиляции

- 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 + аттрибут
- 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...)
- T (иногда E) — алфавит терминальных символов ($N \cap T = \varnothing$)
- P — конечное множество правил вывода
  - $P = \{\alpha \to \beta \mid \alpha \in (N \cup T)^+; \beta \in (N \cup T)^*\}$
- $S \in N$ — аксиома (или начальный символ) грамматики

Пример грамматики:

S->aQbZ

Q->ab | cc | Qd

Z -> aQa | c | ε

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

# 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 лексемы 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. $\Gamma$ — 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 $D(q, \varepsilon, Z) \neq \varnothing$, то $D(q, a, Z) = \varnothing$ для всех $a \in T$

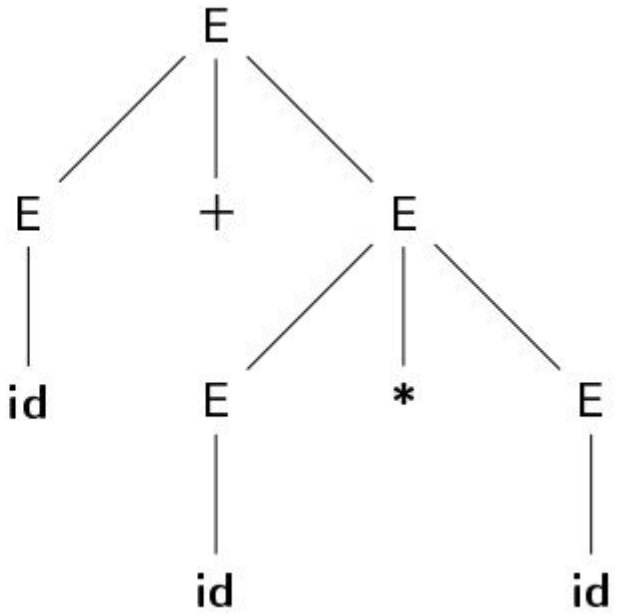
# Варианты синтаксического анализа

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

# Дерево разбора

- id + id * id

$$E \to E\ A\ E \mid (E) \mid -E \mid \mathbf{id}$$
$$A \to + \mid - \mid * \mid /$$



# Метод рекурсивного спуска

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

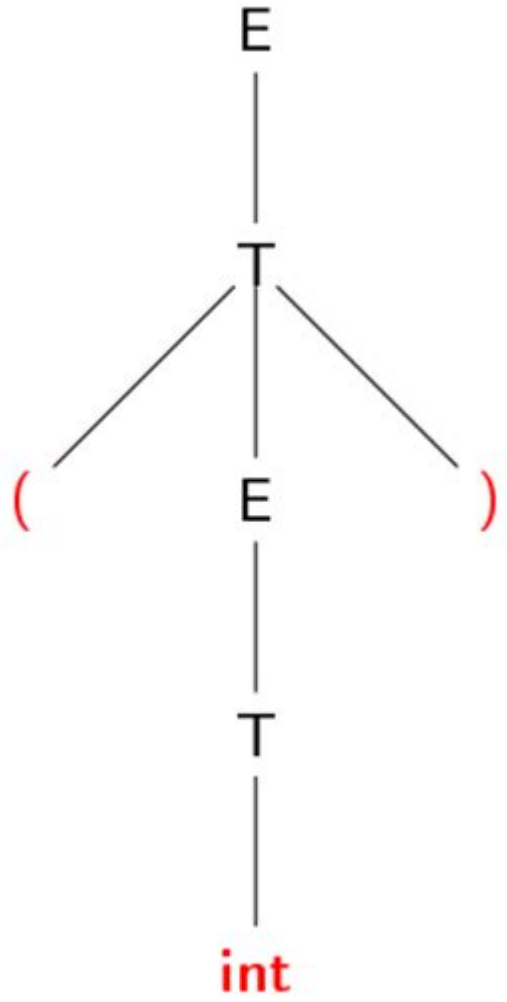
# Метод рекурсивного спуска

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

# Пример применения метода рекурсивного спуска

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

$(\mathbf{int}_5)$



# 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

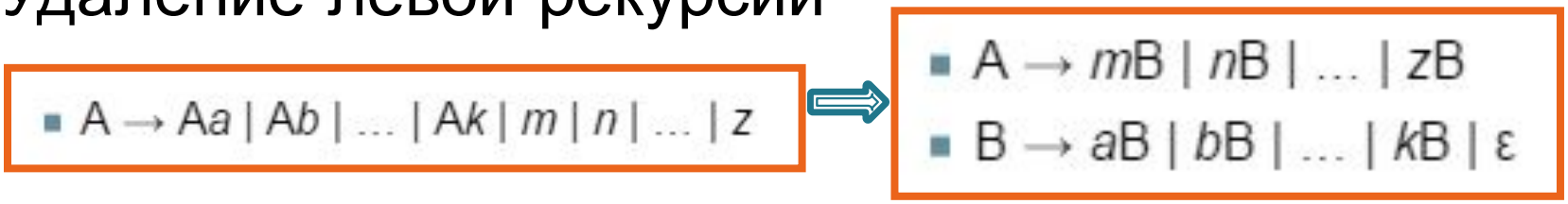
# Предиктивный анализатор

- Идея – убрать откатывание. По каждому символу должно быть изначально ясно куда переходим.

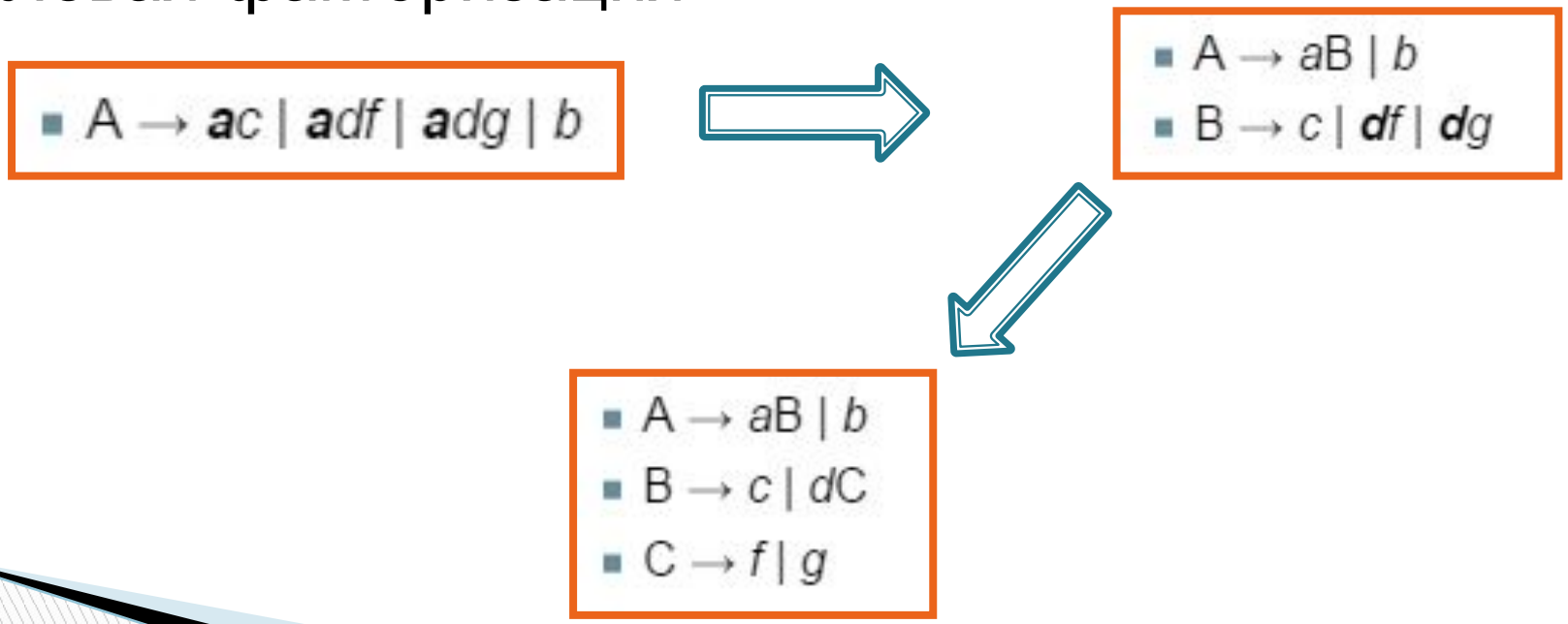
- Преобразование грамматики
  - Удаление левой рекурсии
  - Левая факторизация
- Вычисление множеств
  - FIRST
  - FOLLOW
- Составление таблицы

# Предиктивный анализатор

- Удаление левой рекурсии




- Левая факторизация



# Предиктивный анализатор

Пример преобразования грамматики:

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

P:

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

**Удаление левой рекурсии для S:**

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

**Левая факторизация для A:**

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

**Итоговая грамматика:**

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

# Предиктивный анализатор

- Определение множества FIRST
  - FIRST(a), $a \in (N \cup T)^*$ - множество терминалов или ε, с которых может начинаться a.
- Построение множества FIRST

**Для терминалов**

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

**Для нетерминалов**

- Если $X$ — нетерминал, то положим $FIRST(X) = \{\varnothing\}$
- Если в грамматике есть правило $X \to \varepsilon$, то добавим ε к $FIRST(X)$
- Для каждого нетерминала $X$ и для каждого правила вывода $X \to Y_1 \dots Y_k$ добавим в $FIRST(X)$ 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 $\varepsilon \notin FIRST$, 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) = {∅}
- Если X — аксиома грамматики, то добавить в FOLLOW маркер $
- Для всех правил вида $A \to \alpha X \beta$ добавить FIRST(β)\{ε} к FOLLOW(X) (за X могут следовать те символы, с которых начинается β)
- Для всех правил вида $A \to \alpha X$ и $A \to \alpha X \beta$, $\varepsilon \in$ FIRST(β) 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

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

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

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

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

- $FOLLOW(S) = \{\$\}$
- $FOLLOW(S_1) = \{\$\}$ ($S_1$ — 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 $FIRST(S_1)\backslash\{\varepsilon\}$ (следует из правила $S_1 \rightarrow AbBS_1$), $FOLLOW(S_1)$ (так как есть $S_1 \rightarrow \varepsilon$))

# 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 → α (где под α понимается цепочка в правой части правила) выполняются следующие действия:

1. Для каждого терминала $a \in$ FIRST(α) добавить правило $A \to \alpha$ к *M*[*A*, *a*]
2. Если $\varepsilon \in$ FIRST(α), то для каждого $b \in$ FOLLOW(*A*) добавить $A \to \alpha$ к *M*[*A*, *b*]
3. $\varepsilon \in$ FIRST(α) и \$ ∈ FOLLOW(*A*), добавить $A \to \alpha$ к *M*[*A*, \$]
4. Все пустые ячейки — ошибка во входном слове

# Предиктивный анализ

## Пример построения таблицы

| | a | b |
|---|---|---|
| S | $S \to aS_1$ (Первое правило, вывод $S \to aS_1$, $a \in FIRST(aS_1)$) | **Error** (Четвёртое правило) |
| $S_1$ | $S_1 \to AbBS_1$ (Первое правило, вывод $S_1 \to AbBS_1$, $a \in FIRST(AbBS_1)$) | $S_1 \to AbBS_1$ (Первое правило, вывод $S_1 \to AbBS_1$, $b \in FIRST(AbBS_1)$) |
| A | $A \to aA_1$ (Первое правило, вывод $A \to aA_1$, $a \in FIRST(aA_1)$) | $A \to \varepsilon$ (Второе правило, вывод $A_1 \to \varepsilon$, $b \in FOLLOW(A_1)$) |
| $A_1$ | $A_1 \to a$ (Первое правило, вывод $A_1 \to a$, $a \in FIRST(a)$) | $A_1 \to b$ (Первое правило, вывод $A_1 \to b$, $b \in FIRST(b)$) |
| B | $B \to \varepsilon$ (Второе правило, вывод $B \to \varepsilon$, $a \in FOLLOW(B)$) | $B \to \varepsilon$ (Второе правило, вывод $B \to \varepsilon$, $a \in FOLLOW(B)$) |

| | c | $ |
|---|---|---|
| S | **Error** (Четвёртое правило) | **Error** (Четвёртое правило) |
| $S_1$ | **Error** (Четвёртое правило) | $S_1 \to \varepsilon$ (Третье правило, вывод $S_1 \to \varepsilon$, $\varepsilon \in FIRST(\varepsilon)$, $\$ \in FOLLOW(S_1)$) |
| A | **Error** (Четвёртое правило) | **Error** (Четвёртое правило) |
| $A_1$ | **Error** (Четвёртое правило) | **Error** (Четвёртое правило) |
| B | $B \to c$ (Первое правило, вывод $B \to c$, $c \in FIRST(c)$) | $B \to \varepsilon$ (Третье правило, вывод $B \to \varepsilon$, $\$ \in FOLLOW(B)$) |

# 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**# | **a**abbaabcb$ | $S \rightarrow aS_1$ |
| **a**$S_1$# | **a**abbaabcb$ | сдвиг |
| **$S_1$**# | **a**bbaabcb$ | $S_1 \rightarrow AbBS_1$ |
| **A**b$BS_1$# | **a**bbaabcb$ | $A \rightarrow aA_1$ |
| **a**$A_1$b$BS_1$# | **a**bbaabcb$ | сдвиг |
| **$A_1$**b$BS_1$# | **b**baabcb$ | $A_1 \rightarrow b$ |
| **b**b$BS_1$# | **b**baabcb$ | сдвиг |
| **b**$BS_1$# | **b**aabcb$ | сдвиг |
| **B**$S_1$# | **a**abcb$ | $B \rightarrow \varepsilon$ |
| **$S_1$**# | **a**abcb$ | $S_1 \rightarrow AbBS_1$ |
| **A**b$BS_1$# | **a**abcb$ | $A \rightarrow aA_1$ |
| **A**b$BS_1$# | **a**abcb$ | $A \rightarrow aA_1$ |

| | | |
|---|---|---|
| **a**$A_1$b$BS_1$# | **a**abcb$ | сдвиг |
| **$A_1$**b$BS_1$# | **a**bcb$ | $A_1 \rightarrow a$ |
| **a**b$BS_1$# | **a**bcb$ | сдвиг |
| **b**$BS_1$# | **b**cb$ | сдвиг |
| **B**$S_1$# | **c**b$ | $B \rightarrow c$ |
| **c**$S_1$# | **c**b$ | сдвиг |
| **$S_1$**# | **b**$ | $S_1 \rightarrow AbBS_1$ |
| **A**b$BS_1$# | **b**$ | $A \rightarrow \varepsilon$ |
| **b**$BS_1$# | **b**$ | сдвиг |
| **B**$S_1$# | **$** | $B \rightarrow \varepsilon$ |
| **$S_1$**# | **$** | $S_1 \rightarrow \varepsilon$ |
| **#** | **$** | 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

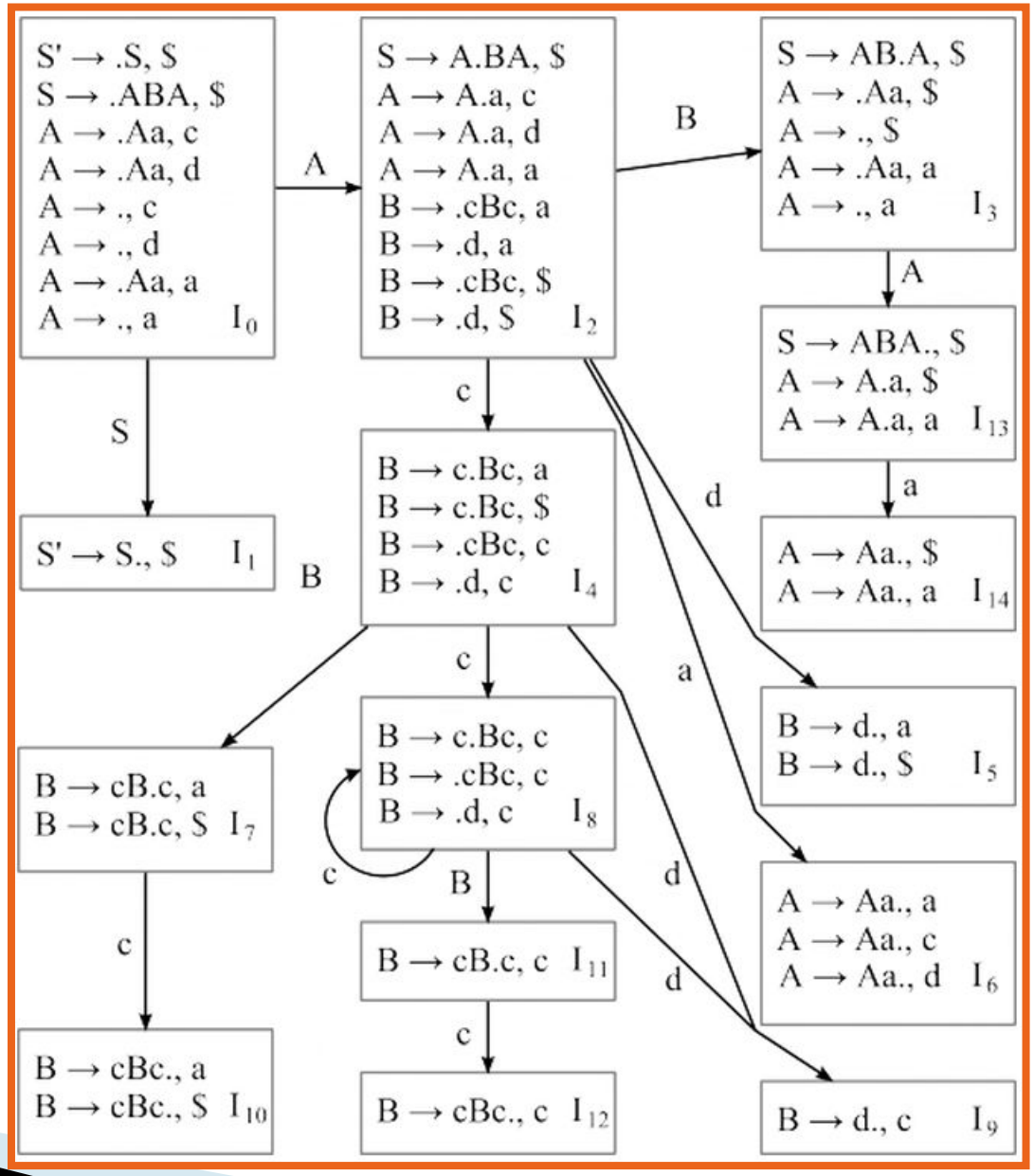
- Добавим новое правило $S' \rightarrow S$, и сделаем $S'$ аксиомой грамматики.
- Допуск имеет место $\Leftrightarrow$ когда возможно осуществить свёртку по правилу $S \rightarrow S'$.

# 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' → S
- S → ABA
- A → Aa | ε
- B → cBc | d



# Построение Goto

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

# Построение action

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

# Пример

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

| | Action | | | | Goto | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | a | c | d | $ | S | S' | A | B | a | c | d |
| $I_0$ | Reduce($A \rightarrow \varepsilon$) | Reduce($A \rightarrow \varepsilon$) | Reduce($A \rightarrow \varepsilon$) | | $I_1$ | | $I_2$ | | | | |
| $I_1$ | | | | Accept | | | | | | | |
| $I_2$ | Shift($I_6$) | Shift($I_4$) | Shift($I_5$) | | | | | $I_3$ | $I_6$ | $I_4$ | $I_5$ |
| $I_3$ | Reduce($A \rightarrow \varepsilon$) | | | Reduce($A \rightarrow \varepsilon$) | | | $I_{13}$ | | | | |
| $I_4$ | | Shift($I_8$) | Shift($I_9$) | | | | | $I_7$ | | $I_8$ | $I_9$ |
| $I_5$ | Reduce($B \rightarrow d$) | | | Reduce($B \rightarrow d$) | | | | | | | |
| $I_6$ | Reduce($A \rightarrow Aa$) | Reduce($A \rightarrow Aa$) | Reduce($A \rightarrow Aa$) | | | | | | | | |
| $I_7$ | | Shift($I_{10}$) | | | | | | | | $I_{10}$ | |
| $I_8$ | | Shift($I_8$) | Shift($I_9$) | | | | | $I_{11}$ | | $I_8$ | $I_9$ |
| $I_9$ | | Reduce($B \rightarrow d$) | | | | | | | | | |
| $I_{10}$ | Reduce($B \rightarrow cBc$) | | | Reduce($B \rightarrow cBc$) | | | | | | | |
| $I_{11}$ | | Shift($I_{12}$) | | | | | | | | $I_{12}$ | |
| $I_{12}$ | | Reduce($B \rightarrow cBc$) | | | | | | | | | |
| $I_{13}$ | Shift($I_{14}$) | | | Reduce($S \rightarrow ABA$) | | | | | $I_{14}$ | | |
| $I_{14}$ | 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($\mathbf{l_k}$), то в стек кладется **c** и затем $\mathbf{l_k}$. При этом **c** удаляется из строки.
- Reduce($\mathbf{A} \rightarrow \mathbf{u}$), то из верхушки стека вычищаются все терминальные и нетерминальные символы, составляющие цепочку **u**, после чего смотрится состояние $\mathbf{l_m}$, оставшееся на верхушке. По таблице переходов на пересечении $\mathbf{l_m}$ и **A** находится следующее состояние $\mathbf{l_s}$. После чего в стек кладется A, а затем $\mathbf{l_s}$. Строка остается без изменений
- Accept, то разбор закончен
- пустота - ошибка

# Восходящий анализ, пример

| Стек | Строка | Действие |
|---|---|---|
| $\mathbf{I_0}$ | **a**aaccdcc$ | Reduce(A → ε), goto $I_2$ |
| $I_0$ A $\mathbf{I_2}$ | **a**aaccdcc$ | Shift($I_6$) |
| $I_0$ A $I_2$ a $\mathbf{I_6}$ | **a**accdcc$ | Reduce(A → Aa), goto $I_2$ |
| $I_0$ A $\mathbf{I_2}$ | **a**accdcc$ | Shift($I_6$) |
| $I_0$ A $I_2$ a $\mathbf{I_6}$ | **a**ccdcc$ | Reduce(A → Aa), goto $I_2$ |
| $I_0$ A $\mathbf{I_2}$ | **a**ccdcc$ | Shift($I_6$) |
| $I_0$ A $I_2$ a $\mathbf{I_6}$ | **c**cdcc$ | Reduce(A → Aa), goto $I_2$ |
| $I_0$ A $\mathbf{I_2}$ | **c**cdcc$ | Shift($I_4$) |
| $I_0$ A $I_2$ c $\mathbf{I_4}$ | **c**dcc$ | Shift($I_8$) |
| $I_0$ A $I_2$ c $I_4$ c $\mathbf{I_8}$ | **d**cc$ | Shift($I_9$) |
| $I_0$ A $I_2$ c $I_4$ c $I_8$ d $\mathbf{I_9}$ | **c**c$ | Reduce(B → d), goto $I_{11}$ |
| $I_0$ A $I_2$ c $I_4$ c $I_8$ B $\mathbf{I_{11}}$ | **c**c$ | Shift($I_{12}$) |
| $I_0$ A $I_2$ c $I_4$ c $I_8$ B $I_{11}$ c $\mathbf{I_{12}}$ | **c**$ | Reduce(B → cBc), goto $I_7$ |
| $I_0$ A $I_2$ c $I_4$ B $\mathbf{I_7}$ | **c**$ | Shift($I_{10}$) |
| $I_0$ A $I_2$ c $I_4$ B $I_7$ c $\mathbf{I_{10}}$ | **$** | Reduce(B → cBc), goto $I_3$ |
| $I_0$ A $I_2$ B $\mathbf{I_3}$ | **$** | Reduce(A → ε), goto $I_{13}$ |
| $I_0$ A $I_2$ B $I_3$ A $\mathbf{I_{13}}$ | **$** | Reduce(S → ABA), goto $I_1$ |
| $I_0$ S $\mathbf{I_1}$ | **$** | 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