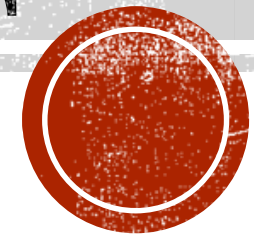
ЯЗЫКИ ПРОГРАММИРОВАНИЯ И МЕТОДЫ ТРАНСЛЯЦИИ



Лекции 1-2

9 февраля 2018 г.

ПОСТРОЕНИЕ АМП

Непростая грамматика

1. $S \rightarrow ASB$
2. $S \rightarrow \varepsilon$
3. $A \rightarrow a$
4. $B \rightarrow b$

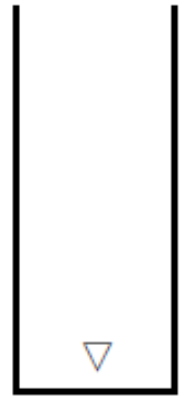
- Как построить АМП для непростых грамматик?
- Можно ли это сделать автоматически?

Автомат с магазинной памятью

S1	a	b	↩
X	Push (X) Advance S1	Pop Advance S2	Reject
▽	Push (X) Advance	Reject	Accept

S2	a	b	↩
X	Reject	Pop Advance S2	Reject
▽	Reject	Reject	Accept

Магазин



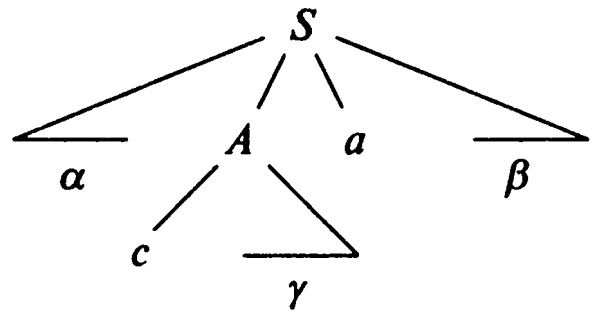
Initial Stack

FIRST & FOLLOW

Пусть **A** – один нетерминал, **a** – один терминал, α, β, γ – строка терминалов и нетерминалов

Опр. 1: FIRST(α) – множество терминалов, с которых начинаются строки, выводимые из α. Если из α выводимо ε, то ε тоже входит в FIRST(α).

Опр. 2: **FOLLOW(A)** – множество терминалов **a**, которые могут располагаться непосредственно после (справа от) **A** в некоторой сентенциальной форме, то есть множество терминалов **a** таких, что существует порождение вида **S** ->* **αAaβ** для некоторых α и β.



FIRST(A) содержит **c**

FOLLOW(A) содержит **a**

	FIRST	FOLLOW
Входное значение	Любая последовательность терминалов и нетерминалов	Один нетерминал
Выходное значение	Множество терминалов и ε	Множество терминалов и **$**

АЛГОРИТМ FIRST

Опр. 3: FIRSTForG – таблица, содержащая значения функции FIRST, вычисленные для каждого символа грамматики (терминала и нетерминала)

Алгоритм 1: функция **FIRST(α)**

1. если $\alpha = \varepsilon$, то возвращаем ε
2. если α – одиночный терминал/нетерминал a, то возвращаем FIRSTForG[α]
3. пусть $\alpha = X_1X_2X_3 \dots X_n$, где X_i – произвольный терминал или нетерминал G:
   a) добавляем к результату все терминалы из $FIRST(X_1)$; если ε входит в $FIRST(X_1)$, то переходим к шагу (b), если нет, то выходим из функции FIRST с накопленным результатом
   b) добавляем к результату все терминалы из $FIRST(X_2)$; если ε входит в $FIRST(X_2)$, то переходим к шагу (c), если нет, то выходим из функции FIRST с накопленным результатом
   c) …
   d) добавляем к результату все терминалы из $FIRST(X_n)$; **если ε входит в $FIRST(X_n)$, то добавляем ε в результат**, иначе просто выходим из функции FIRST с накопленным результатом

АЛГОРИТМ FIRST

Алгоритм 2: вычисление значений функции **FIRST** для всех символов грамматики (алгоритм заполнения таблицы FIRSTForG)

1. для каждого терминала x в грамматике G добавляем в FIRSTForG[x] множество {x}, состоящее из одного этого терминала
2. для каждого правила вида A -> ε добавляем ε в FIRSTForG[A]
3. для каждого правила грамматики G вида A -> α, **добавляем** FIRST(α) в FIRSTForG[A]
4. если во время выполнения шага 3 в FIRSTForG было добавлено хотя бы одно новое значение, то переходим снова к шагу 3, иначе выходим из функции

ПРИМЕР РАБОТЫ АЛГОРИТМА FIRST

1. S → ASB
2. S → ε
3. A → a
4. B → b

Символ грамматики **x**	FIRSTForG[x]
a	
b	
A	
B	
S	

АЛГОРИТМ FOLLOW

Опр. 4: FOLLOWForG – таблица, содержащая значения функции FOLLOW, вычисленные для каждого нетерминала грамматики.

Алгоритм 3: вычисление значений функции **FOLLOW** для всех нетерминалов грамматики

1. добавляем $ в FOLLOWForG[S], где S – стартовый нетерминал грамматики G
2. для каждого правила грамматики G выполняем одно из двух действий:
    1. если правило вида A -> αBβ, то добавляем в FOLLOWForG[B] все элементы FIRST(β) кроме ε;
    2. если правило вида A -> αB или A -> αBβ, где FIRST(β) содержит ε, то добавляем в FOLLOWForG[B] все элементы FOLLOWForG[A];
3. если во время выполнения шага 2 в FOLLOWForG было добавлено хотя бы одно новое значение, то переходим снова к шагу 2, иначе выходим из функции

ПРИМЕР РАБОТЫ АЛГОРИТМА FOLLOW

1. S → ASB
2. S → ε
3. A → a
4. B → b

Символ грамматики **x**	**FOLLOWForG[x]**
A	
B	
S	

УПРАЖНЕНИЕ

Упражнение 1
Построить FIRST и FOLLOW для следующей грамматики: **S -> A c | B b | S C a b** **A -> a | b B | C** **B -> A C | d** **C -> a S | C d B d | AB | ε**

LL(1)-ГРАММАТИКА

Опр. 5: **LL**(1) – грамматика – это грамматика нисходящего синтаксического анализатора, которому на каждом шаге работы для выбора очередной продукции (правила) грамматики достаточно просматривать один символ входной ленты.

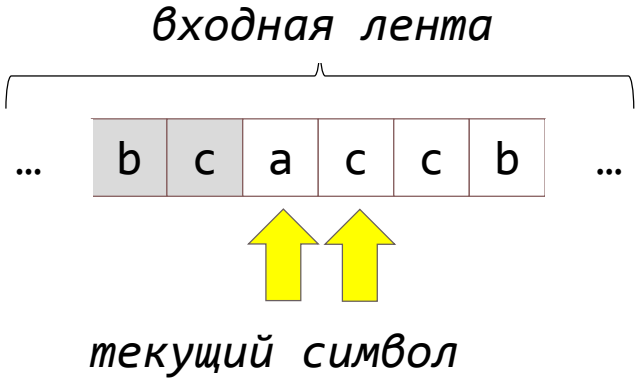
Грамматика принадлежит классу **LL**(1) тогда и только тогда, когда для любых двух различных правил **A** -> α | β выполняются следующие три условия:

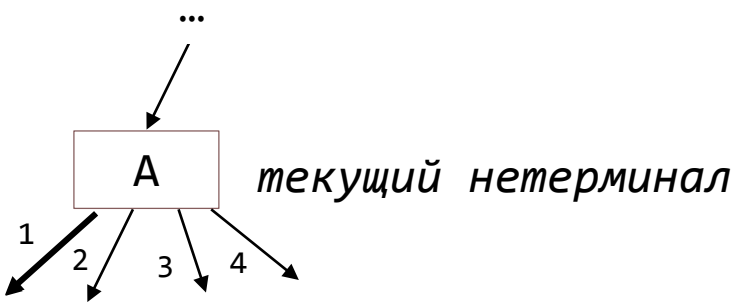
1. Не существует такого терминала **a**, для которого и α, и β порождают строку, начинающуюся с **a**.
2. Пустую строку может порождать не более чем одна из продукций α или β.
3. Если $\beta \overset{*}{\Rightarrow} \varepsilon$, то α не порождает ни одну строку, начинающуюся с терминала из **FOLLOW**(**A**). Аналогично, если $\alpha \overset{*}{\Rightarrow} \varepsilon$, то β не порождает ни одну строку, начинающуюся с терминала из **FOLLOW**(**A**).

ПРЕДИКТИВНЫЙ АНАЛИЗАТОР

Опр. 6: Предиктивный синтаксический анализатор – это такой нисходящий синтаксически анализатор, который работает без возвратов.

Пример:
…
1) **A -> aXYc**
2) **A -> aDc**
3) **A -> aXAbc**
4) **A -> ε**
…





- синтаксический анализатор, работающий с возвратами, будет пробовать каждую альтернативу, пока либо они не закончатся, либо не будет найдена правильная
- если среди предложенных правильной альтернативы нет, то анализатор поднимется по дереву вверх, пока не найдет узел, в котором есть еще неиспробованные альтернативы

Предиктивный анализатор на каждом шаге знает, какую единственно правильную альтернативу нужно выбрать. Если она не подходит, то слово не принадлежит распознаваемому языку.

АЛГОРИТМ ПОСТРОЕНИЕ АМП СА

Алгоритм 4: построение таблицы АМП предиктивного синтаксического анализатора

1. для каждого правила **A -> α** грамматики G выполняем:
   1. для каждого терминала a из FIRST(α) добавляем **Rep(α^T),R** в ячейку **M[A][a]**
   2. если ε входит в FIRST(α), то для каждого символа b из FOLLOW(A) добавляем **Rep(α^T),R** в **M[A][b]**
2. для каждого терминала x грамматики G добавляем **Pop,A** в ячейку **M[x][x]**
3. добавляем **Accept** в ячейку **M[▼][$]**
4. добавляем **Error** во все оставшиеся пустые ячейки

УПРАЖНЕНИЯ

УПРАЖНЕНИЯ

Упражнение 2
Построить АМП СА для следующей грамматики 1. E -> E + T | E – T | T 2. T -> T * F | T / F | F 3. F -> id | (E)
Упражнение 3
Построить АМП СА для следующей грамматики: 1. S -> if E then S S' | a 2. S' -> else S | ε 3. E -> b
Упражнение 4
Построить АМП СА для следующей грамматики: 1. IfStmt -> Matched 2. IfStmt -> Unmatched 3. Matched -> if (Expr) Matched else Matched 4. Matched -> OtherStmt 5. Unmatched -> if (Expr) Stmt 6. Unmatched -> if (Expr) Matched else Unmatched