



Автоматы и формальные языки

Карпов Юрий Глебович
профессор, д.т.н., зав.кафедрой
“Распределенные вычисления и компьютерные сети”
Санкт-Петербургского Политехнического университета

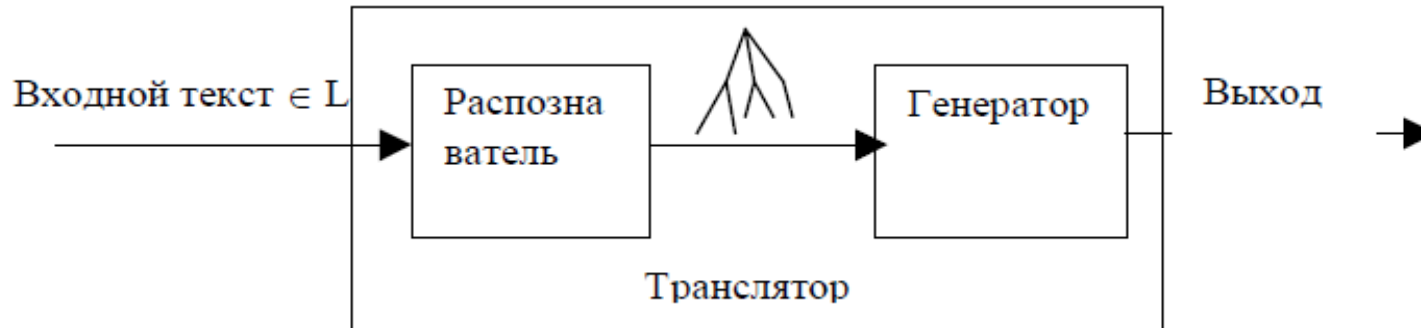
karpov@dcn.infos.ru



Структура курса

- Конечные автоматы-распознаватели – 4 л
- Порождающие грамматики Хомского – 3 л
- Атрибутные трансляции и двусмысленные КС-грамматики – 2 л
- Распознаватели КС-языков и трансляция – 6 л
 - Лекция 10. s-грамматики, LL(k)-грамматики, грамматики рекурсивного спуска
 - Лекция 11. Построение транслятора языка Милан методом рекурсивного спуска
 - Лекция 12. Грамматики предшествования, LR(k)-грамматики
 - Лекция 13. SLR(k) и LALR(k)-грамматики.
 - Лекция 14. Компиляторы компиляторов. Yacc и Bison.
 - Лекция 15. Грамматики Кока-Янгера-Касами и Эрли
- Дополнительные лекции - 2 л

Проблема синтаксического анализа



- В соответствии с идеей синтаксически-ориентированной трансляции, процесс трансляции в информатике связывается с двумя основными этапами:
 - На первом этапе блок, который можно назвать распознавателем, строит структуру входной цепочки НА ОСНОВЕ ПОРОЖДАЮЩЕЙ ГРАММАТИКИ ВХОДНОГО ЯЗЫКА
 - На втором этапе построенная структура используется для генерации выхода, выражающего смысл входной цепочки
- Во многих трансляторах языков программирования процессы распознавания и генерации разделены не так явно. Но во всех случаях метод синтаксически-ориентированной трансляции основан на том, что строится целиком или по частям структура входной цепочки

Восходящие алгоритмы синтаксического анализа

$S \Rightarrow_1 abScB \Rightarrow_6 abSc c \Rightarrow_2 abbAcc \Rightarrow_4 abbcBAcc \Rightarrow_3 abbcBabcc \Rightarrow_6 abbcCabcc$

1. $S \rightarrow abScB$

2. $S \rightarrow bA$

3. $A \rightarrow ab$

4. $A \rightarrow cBA$

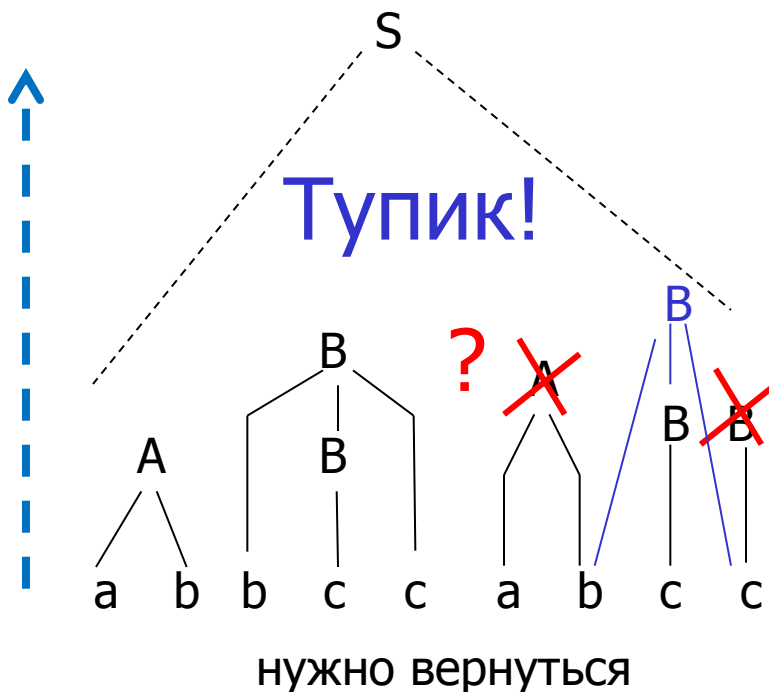
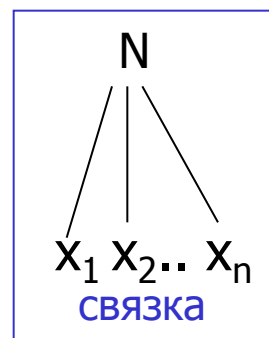
5. $B \rightarrow bBc$

6. $B \rightarrow c$

сентенциальные формы
(промежуточные цепочки вывода)

правый вывод

$S \Leftarrow ?? \Leftarrow abbcCabcc$



Восходящий синтаксический анализ работает так:

Начиная от терминальной строки листьев дерева пытаемся найти "связку" – правую часть продукции грамматики, которую нужно свернуть – заменить нетерминалом (левой частью продукции), чтобы получить новый узел синтаксического дерева.

Поскольку ищем самую левую связку, восстанавливается правый вывод цепочки.

$S \Leftarrow \dots \Leftarrow_4 abbcBAcc \Leftarrow_3 abbcBabcc \Leftarrow_6 abbcCabcc$



Восходящие методы СА

- Восходящие алгоритмы синтаксического анализа просматривая слева направо произвольную sentенциальную форму порождаемого грамматикой языка пытаются найти в ней подцепочку, являющуюся "**СВЯЗКОЙ**" или "**ОСНОВОЙ**" – правой частью продукции грамматики, которую нужно заменить нетерминалом - левой частью этой продукции - с получением предыдущей sentенциальной формы.
- Последовательное выполнение этой операции (возможно, с возвратами) позволяет восстановить правый канонический вывод данной цепочки из начального символа грамматики
- Простейший алгоритм "грубой силы", пытающийся решить эту задачу простым перебором, ищет очередную связку простой проверкой на совпадение правых частей всех продукций грамматики и подстрок sentенциальной формы.

Как найти связку в очередной сентенциальной форме?

$S \xleftarrow{1} abScB \xleftarrow{6} abScc \xleftarrow{2} abbAcc \xleftarrow{4} abbcBAcc \xleftarrow{3} abbcBabcc \xleftarrow{6} abbccabcc$

1. $S \rightarrow abScB$

2. $S \rightarrow bA$

3. $A \rightarrow ab$

4. $A \rightarrow cBA$

5. $B \rightarrow bVc$

6. $B \rightarrow c$

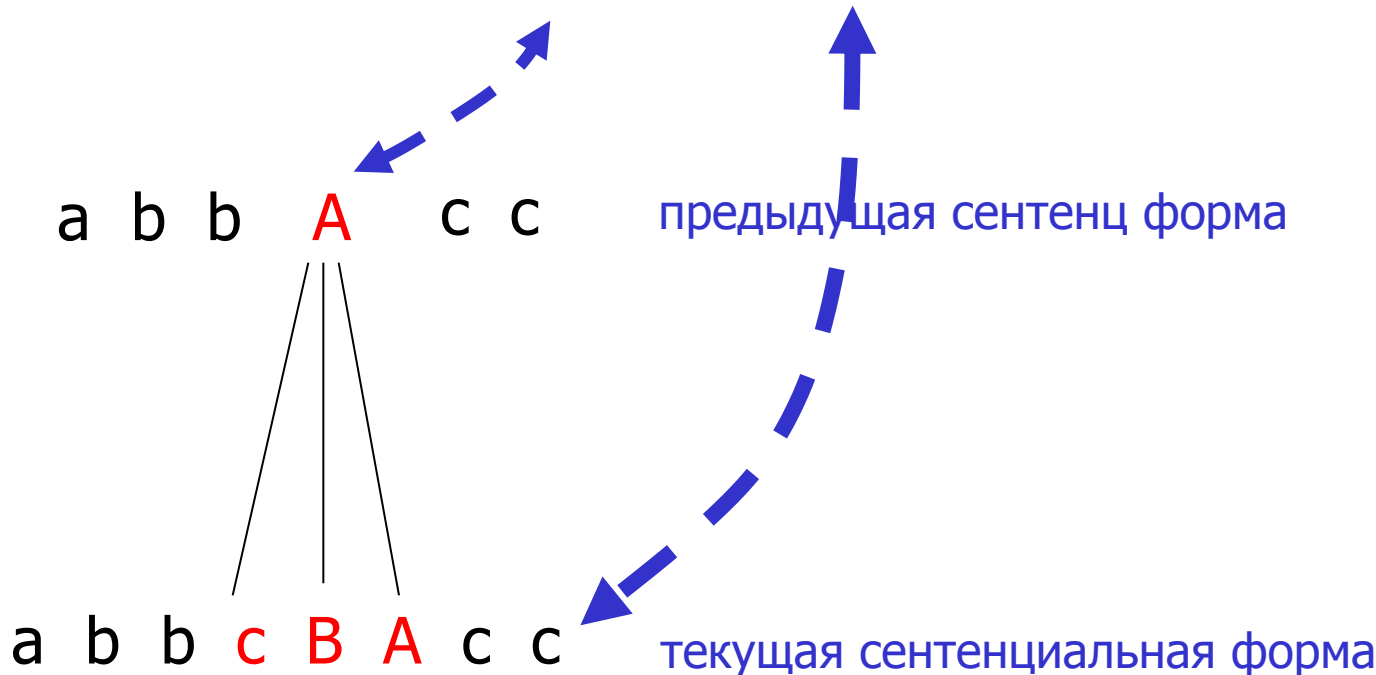
Почему при второй свертке нужно выбрать именно эту пару ab и свернуть к A ?

Почему при первой свертке нужно выбрать именно это c и свернуть к B ?

- Простой алгоритм “грубой силы” не использует никакой информации, кроме информации о простом вхождении символов в сентенциальную форму, и поэтому чрезвычайно неэффективен.
- Были разработаны методы, позволяющие выполнить восходящий синтаксический анализ с более полным учетом информации, содержащейся во входной цепочке и грамматике
- Мы рассмотрим два метода выделения самой левой связки в сентенциальной форме:
 - метод отношений предшествования
 - метод, основанный на использовании всей информации, содержащейся во входной цепочке до связки, и k символов после искомой связки, приводящий к выделению так называемого класса $LR(k)$ -грамматик

Повторяющаяся задача: как перейти от очередной сентенциальной формы к предыдущей?

$S \xleftarrow{1} abScB \xleftarrow{2} abScc \xleftarrow{4} abbAcc \xleftarrow{6} abbcBAcc \xleftarrow{3} abbcBabcc \xleftarrow{6} abbccabcc$



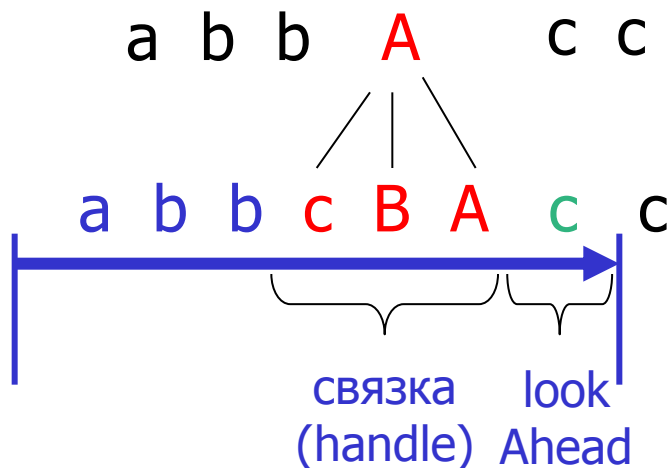
1. $S \rightarrow abScB$
2. $S \rightarrow bA$
3. $A \rightarrow ab$
4. $A \rightarrow cBA$
5. $B \rightarrow bBc$
6. $B \rightarrow c$

Почему не $A \leftarrow ab$, не $B \leftarrow c$ а именно $A \leftarrow cBA$???

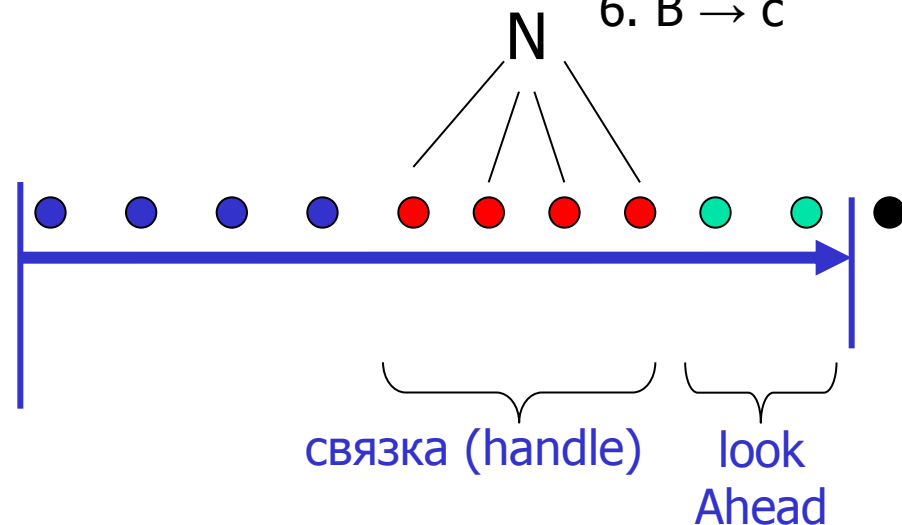
Идя по сентенциальной форме слева направо, хотим точно знать, где находится связка, заглянув не более, чем на k символов за нее. Свернув связку к нужному нетерминалу, получим предыдущую сентенц. форму

Как перейти от очередной сентенциальной формы к предыдущей?

$S \xleftarrow{1} abScB \xleftarrow{2} abScc \xleftarrow{4} abbAcc \xleftarrow{6} abbcBAcc \xleftarrow{3} abbcBabcc \xleftarrow{6} abbccabcc$



1. $S \rightarrow abScB$
2. $S \rightarrow bA$
3. $A \rightarrow ab$
4. $A \rightarrow cBA$
5. $B \rightarrow bBc$
6. $B \rightarrow c$



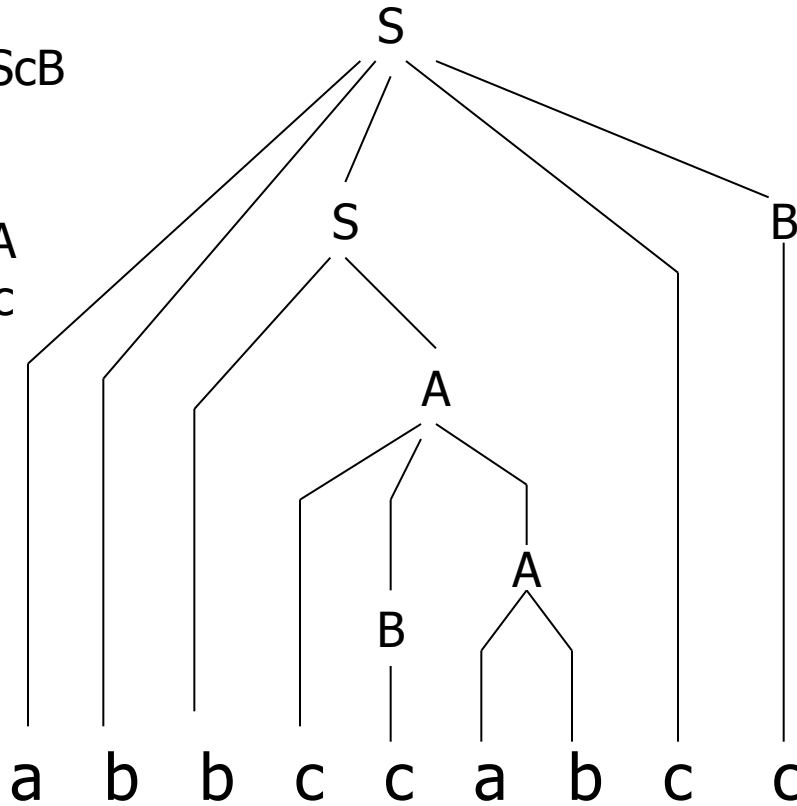
Не $A \leftarrow ab$, не $B \leftarrow c$ а именно $A \leftarrow cBA$

Идя по сентенциальной форме слева направо, хотим точно знать, где находится связка, заглянув не более, чем на k символов за нее. Свернув связку к нужному нетерминалу, получим предыдущую сентенц. форму

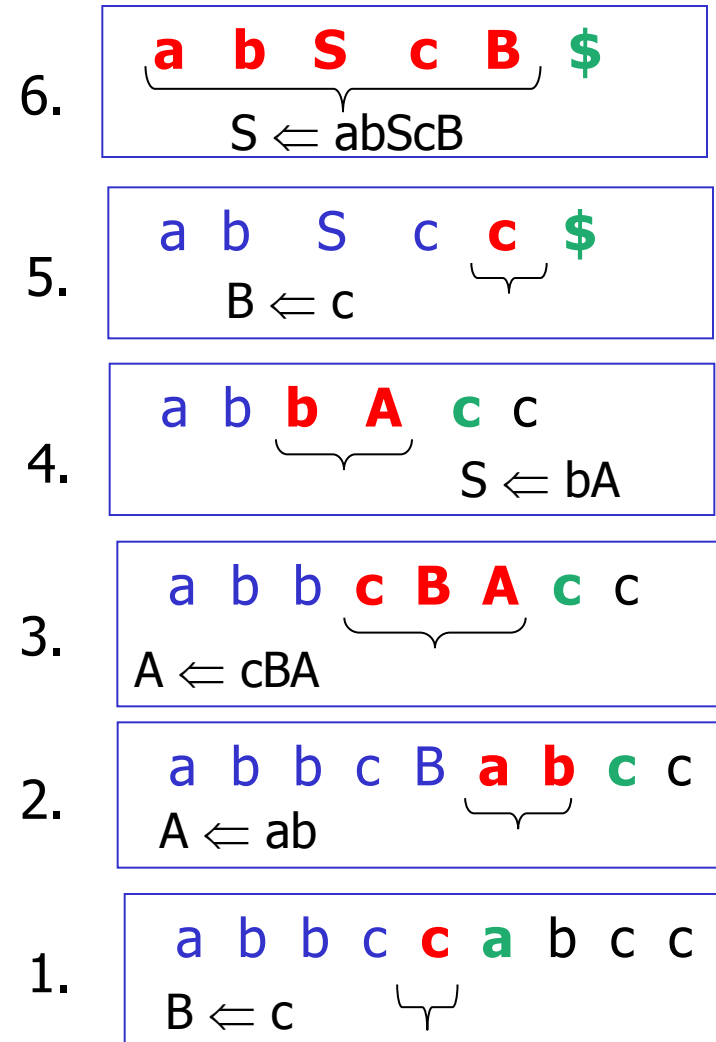
Последовательное нахождение связок = восстановление дерева вывода снизу вверх

$S \xleftarrow{1} abScB \xleftarrow{6} abScc \xleftarrow{4} abbA_{cc} \xleftarrow{4} abbcBA_{cc} \xleftarrow{3} abbcBab_{cc} \xleftarrow{6} abbccabcc$

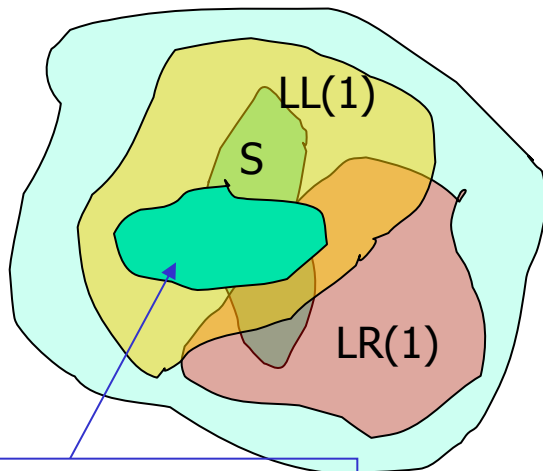
1. $S \rightarrow abScB$
2. $S \rightarrow bA$
3. $A \rightarrow ab$
4. $A \rightarrow cBA$
5. $B \rightarrow bBc$
6. $B \rightarrow c$



Если имеем алгоритм нахождения **связки** в очередной сентенциальной форме, то последовательно применяя этот алгоритм, построим дерево вывода снизу вверх



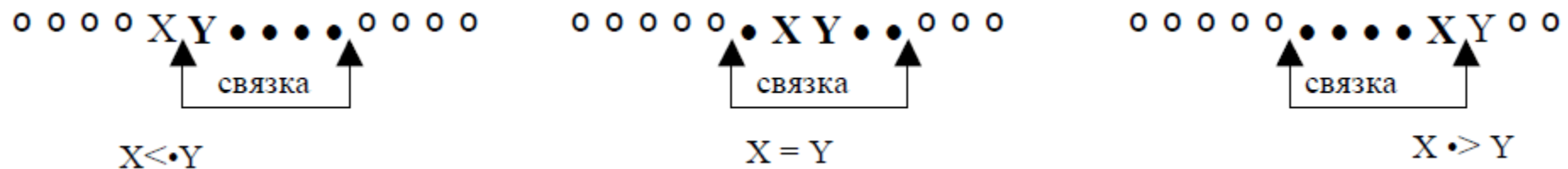
Грамматики простого предшествования



Подклассы КС-грамматик
s-грамматики
LL(k)-грамматики,
Грамматики рекурсивного спуска
Грамматики предшествования
LR(k)-грамматики,
LALR(k)-грамматики,
...

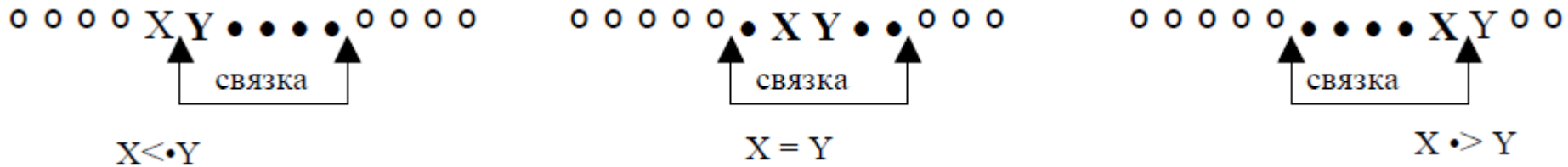
Грамматики
предшествования

Возможные расположения пары соседних символов относительно связки



- При поиске связки в сентенциальной форме достаточно найти ее границы и тот нетерминал, к которому сворачивается эта связка
- В простейшем случае несовпадающих правых частей всех продукций по самой связке однозначно определяется нетерминал, которым каждая конкретная связка заменяется
- Таким образом, выделение границ связки становится в таких грамматиках единственной проблемой. Для нахождения границ связки можно задаться вопросом о том, между какими парами символов могут проходить такие границы начала и конца произвольной связки
- Это приводит к анализу таких бинарных отношений между символами - возможными соседями в сентенциальных формах, которые могут определить возможное расположение этих пар символов относительно связки

Возможные расположения пары соседних символов относительно связки



- Обозначим эти отношения для пары (X, Y) символов, которые могут встретиться рядом в sentенциальной форме, следующим образом:
 - $X < \bullet Y$ – если символ X может стоять перед левой границей связки, начинающейся с Y ;
 - $X \doteq Y$ – если оба символа, X и Y , могут стоять внутри связки;
 - $X \bullet > Y$ – если символ X может стоять последним в связке, а его сосед Y не принадлежит связке.
- Грамматику перепишем так: вводим новый начальный нетерминал S' и в грамматику добавим новую продукцию $S' \rightarrow \#S\#$, где S – старый начальный нетерминал

G: 0. $S' \rightarrow \#S\#$
 1. $S \rightarrow a$
 2. $S \rightarrow aT$
 3. $S \rightarrow [S]$
 4. $T \rightarrow b$
 5. $T \rightarrow bT$

$S' \Rightarrow \#S\# \Rightarrow \#[S]\# \Rightarrow \#[aT]\# \Rightarrow \#[abT]\# \Rightarrow \#[abb]\#$

Три отношения сворачиваемости между парами

$S' \Rightarrow \#S\# \Rightarrow \#[S]\# \Rightarrow \#[aT]\# \leftarrow \#[a\mathbf{bT}]\# \Rightarrow \#[abb]\#$

- По этому примеру вывода видно, что должны существовать следующие отношения предшествования:

$a < \bullet b, \quad b \doteq T \quad \text{и} \quad T \bullet >],$

потому что цепочка $\#[a < \bullet b \doteq T \bullet >]\#$ с этими введенными явно отношениями предшествования может быть свернута к предыдущей сентенциальной форме $\#[aT]\#$ в этом конкретном выводе

- Такие отношения нельзя искать анализом всех возможных выводов: таких выводов бесконечно много
- Можно ли эти отношения определить непосредственно из грамматики?
- Бинарные отношения $\{<\bullet, \doteq, \bullet>\}$ для произвольной пары символов грамматики X и Y (как терминальных, так и нетерминальных), которые могут встретиться рядом при синтаксическом анализе в произвольной сентенциальной форме, должны дать ответы на три вопроса:
 1. Может ли после символа X начаться какая-нибудь связка, если в очередной сентенциальной форме вслед за X идет Y (отношение ' $<\bullet$ ')?
 2. Могут ли символы X и Y встретиться рядом в одной связке (отношение ' \doteq ')?
 3. Может ли перед символом Y закончиться какая-нибудь связка, если в очередной сентенциальной форме ему предшествует X (отношение ' $\bullet>$ ')?

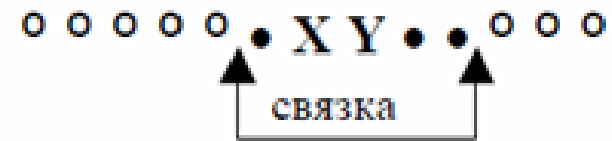
Отношение одновременной сворачиваемости - ОТНОШЕНИЕ \doteq

$$\begin{aligned} S' &\rightarrow \#S\# \\ S &\rightarrow a \mid aT \mid [S] \\ T &\rightarrow b \mid bT \end{aligned}$$

- Два символа X и Y могут встретиться рядом в одной связке (и находиться в отношении одновременной сворачиваемости $X \doteq Y$), **если и только если** существует правая часть какой-либо продукции грамматики, в которой эти два символа стоят рядом

- Для нашей грамматики это 6 пар:
 $\#S$, $S\#$, aT , $[S$, $S]$, bT ,
 что очевидно из грамматики

$\# \doteq S$, $S \doteq \#$, $a \doteq T$, $[\doteq S$, $S \doteq]$, $b \doteq T$



$X \doteq Y$

Матрица отношений предшествования

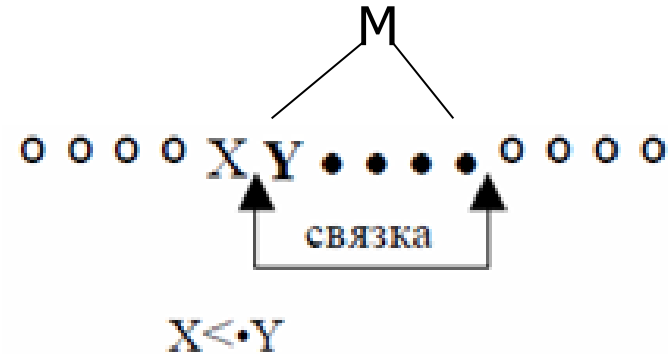
	S	T	#	[]	a	b
S			\doteq		\doteq		
T							
#	\doteq						
[\doteq						
]							
a		\doteq					
b		\doteq					

Отношение сворачиваемости ' $<\bullet$ '

$$\begin{aligned} S' &\rightarrow \#S\# \\ S &\rightarrow a \mid aT \mid [S] \\ T &\rightarrow b \mid bT \end{aligned}$$

- Отношение ' $<\bullet$ ' выполняется для пары (X, Y) , если с Y начнется цепочка, которая может быть свернута к некоторому нетерминалу M , который может после свертки встать рядом с X , а потом свернуться вместе с X при дальнейшем анализе. Следовательно, этот новый нетерминал M обязательно будет стоять рядом с X в правой части какой-нибудь продукции грамматики.
- Формально:

$$X <\bullet Y \Leftrightarrow (\exists A \rightarrow \alpha X M \beta \in R): M \Rightarrow^+ Y \gamma$$



Например, $a <\bullet b$, потому что aT стоят рядом в правиле $S \rightarrow aT$, а $T \Rightarrow^+ bT$

Далее, $[<\bullet a$ и $[<\bullet [$, потому что в G есть правило $S \rightarrow [S$, а из S выводятся цепочки, начинающиеся с a и с $[$:
 $S \Rightarrow^+ aT$ и $S \Rightarrow^+ [S$

Отношение сворачиваемости ' $<\bullet$ '

$$\begin{aligned} S' &\rightarrow \#S\# \\ S &\rightarrow a \mid aT \mid [S] \\ T &\rightarrow b \mid bT \end{aligned}$$

Формально: $X<\bullet Y \Leftrightarrow (\exists A \rightarrow \alpha X M \beta \in R): M \Rightarrow^+ Y \gamma$

Как найти все возможные отношения ' $<\bullet$ '?

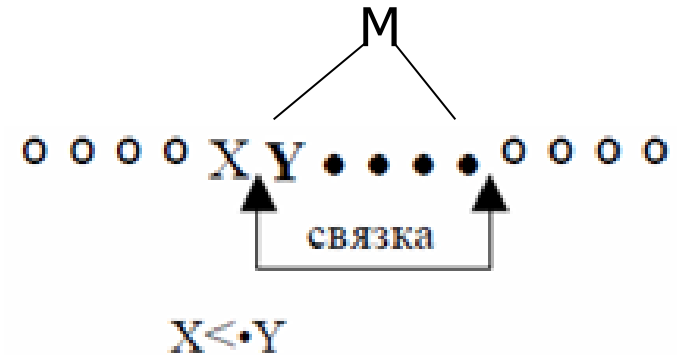
1. Находим в правилах все пары рядом стоящих **<символ>** **<нетерминал>**
2. **<символ>** будет находиться в отношении ' $<\bullet$ ' со всеми символами, с которых начинаются цепочки, выводимые из **<нетерминала>** за один шаг или более

По нашей грамматике:
пары: $\#S$, $[S$, aT , bT

Все возможные отношения ' $<\bullet$ ':

из пар $\#S$ и $[S$: $\{ \#, [\} <\bullet \{ a, [\}$

из пар aT , bT : $\{ a, b \} <\bullet \{ b \}$



	S	T	#	[]	a	b
S			\equiv		\equiv		
T							
#	\equiv			$<\bullet$		$<\bullet$	
[\equiv			$<\bullet$		$<\bullet$	
]							
a		\equiv					$<\bullet$
b		\equiv					$<\bullet$

Отношение сворачиваемости ' $\bullet>$ '

$$S' \rightarrow \#S\#$$

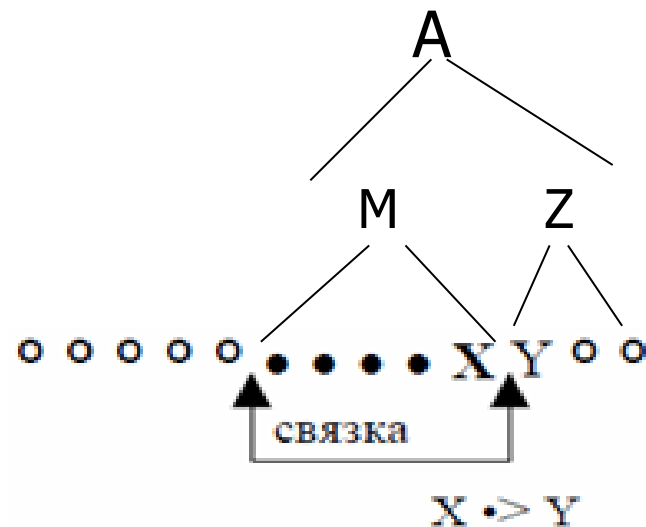
$$S \rightarrow a \mid aT \mid [S]$$

$$T \rightarrow b \mid bT$$

- Отношение ' $\bullet>$ ' выполняется для пары (X, Y) , если X кончается цепочка, которая может быть свернута к некоторому нетерминалу M , который может после свертки встать рядом с Y , а потом свернуться вместе с Y при дальнейшем анализе. Следовательно, этот новый нетерминал M обязательно будет стоять рядом с Y в правой части какой-нибудь продукции грамматики. Формально: $X \bullet > Y \Leftrightarrow (\exists A \rightarrow \alpha M Y \beta \in R): M \Rightarrow^+ \gamma X$
- При просмотре цепочки слева направо Y еще не может быть свернут к нетерминалу в составе какой-нибудь продукции и всегда является терминалом. Формально: $X \bullet > Y \Leftrightarrow (\exists A \rightarrow \alpha M Z \beta \in R): M \Rightarrow^+ \gamma X, Z \Rightarrow^* Y \delta, Y \in T$

Как найти все возможные отношения ' $\bullet>$ ' ?

1. Находим в правилах все пары рядом стоящих <нетерминал> M и <символ> Z
2. Все символы, которыми кончаются цепочки, выводимые из M , будут находиться в отношении ' $\bullet>$ ' с теми терминалами, с которых начинаются цепочки, выводимые из Z за 0 или больше шагов

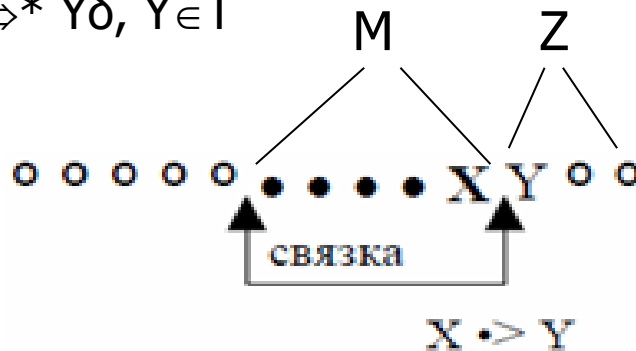


Отношение сворачиваемости ' $\bullet>$ '

$$\begin{aligned} S' &\rightarrow \#S\# \\ S &\rightarrow a \mid aT \mid [S] \\ T &\rightarrow b \mid bT \end{aligned}$$

Формально: $X\bullet>Y \Leftrightarrow (\exists A \rightarrow \alpha MZ\beta \in R): M \Rightarrow^+ \gamma X, Z \Rightarrow^* Y\delta, Y \in T$

1. Находим в правилах все пары рядом стоящих <нетерминал>M и <символ> Z
2. Все символы, которыми кончаются цепочки, выводимые из M, будут находиться в отношении ' $\bullet>$ ' с теми терминалами, с которых начинаются цепочки, выводимые из Z за 0 или больше шагов



По нашей грамматике:
пары: $S \#$; $S]$

Все возможные отношения ' $\bullet>$ ':
из пар $S\#$ и $S]$: $\{a, T,], b\} \bullet> \{\#,]\}$

	S	T	#	[]	a	b
S			\equiv		\equiv		
T			$\bullet>$		$\bullet>$		
#	\equiv			$<\bullet$		$<\bullet$	
[\equiv			$<\bullet$		$<\bullet$	
]			$\bullet>$		$\bullet>$		
a		\equiv	$\bullet>$		$\bullet>$		$<\bullet$
b		\equiv	$\bullet>$		$\bullet>$		$<\bullet$

Синтаксический анализ на основе отношений предшествования

$$\begin{aligned} S' &\rightarrow \#S\# \\ S &\rightarrow a \mid aT \mid [S] \\ T &\rightarrow b \mid bT \end{aligned}$$

- Связкой в любой sentential form будет самая левая подстрока, между символами которой выполняется отношение ' \preceq ', а левая и правая границы выделены отношениями ' $<\bullet$ ' и ' $\bullet>$ '. Например, для цепочки $\#[abb]\#$, выведенной в грамматике G , расставим отношения предшествования:

$$\# <\bullet [<\bullet a <\bullet b <\bullet \textcolor{red}{b} \bullet>] \bullet> \#$$

- Очевидно, что b является связкой, и для получения предыдущей sentential form ее нужно свернуть к нетерминалу
- Но к какому?

Этого грамматика предшествования не определяет!

Разбор:

$$\# <\bullet [<\bullet a <\bullet b <\bullet \textcolor{red}{b} \bullet>] \bullet> \#$$

	S	T	#	[]	a	b
S			\preceq		\preceq		
T			$\bullet>$		$\bullet>$		
#	\preceq			$<\bullet$		$<\bullet$	
[\preceq			$<\bullet$		$<\bullet$	
]			$\bullet>$		$\bullet>$		
a		\preceq	$\bullet>$		$\bullet>$		$<\bullet$
b		\preceq	$\bullet>$		$\bullet>$		$<\bullet$

Синтаксический анализ на основе отношений предшествования: восстановление вывода

$S' \rightarrow \#S\#$
 $S \rightarrow a \mid aT \mid [S]$
 $T \rightarrow b \mid bT$

Правый вывод восстановлен!

свертки снизу вверх!!

Разбор:

[a b b]

S

[S]

свертка $S \leftarrow [S]$

[aT]

свертка $S \leftarrow aT$

[a bT]

свертка $T \leftarrow bT$

[a b b]

свертка $T \leftarrow b$

	S	T	#	[]	a	b
S			<u>=</u>	<u>=</u>			
T			•>		•>		
#	<u>=</u>			<•		<•	
[<u>=</u>			<•		<•	
]			•>		•>		
a		<u>=</u>	•>		•>		<•
b		<u>=</u>	•>		•>		<•

Синтаксический анализ на основе отношений предшествования \rightarrow LookAhead=1

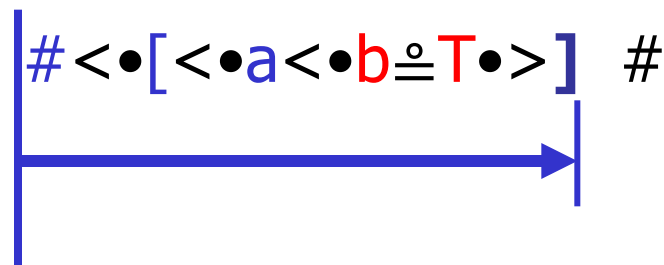
- Метод предшествования не говорит, к какому нетерминалу свернуть выделенную связку, поэтому в грамматике не должно быть нескольких разных продукций с одинаковыми правыми частями. После свертки b к T , новая сентенциальная форма имеет вид: $\#[abT]\#$, а с расставленными отношениями предшествования: $\#<\bullet[<\bullet a<\bullet b \doteq T \bullet>]\bullet>\#$.
Из расстановки этих отношений видно, что новая связка bT должна быть далее свернута к T , и т.д.

очередная сентенциальная форма
как выделить связку??

$\# \quad [\quad a \quad b \quad T \quad] \quad \#$

В грамматиках предшествования
LookAhead = 1

$\#<\bullet[<\bullet a<\bullet b \doteq T \bullet>]\bullet>\#$



В методе предшествования при просмотре сентенциальной формы слева направо связка будет найдена после анализа точно одного символа, следующего за связкой (при обнаружении отношения $\bullet>$)

Пример: решение задачи

- Задана грамматика.

Выполнить синтаксический анализ цепочки #abadaccac#
методом простого предшествования

0. $S' \rightarrow \#S\#$
1. $S \rightarrow aBSc$
2. $S \rightarrow a$
3. $B \rightarrow bSc$
4. $B \rightarrow d$

Строим матрицу отношений предшествования

1. Отношения ' $\underline{=}$ ': #S, S#, aB, BS, Sc, bS

2. Отношения ' $<\bullet$ '.

Пары #S, BS, bS, aB

Отсюда $\{\#, B, b\} <\bullet \{a\}$ и $\{a\} <\bullet \{b, d\}$

3. Отношения ' $\bullet>$ '.

Пары S#, Sc, BS

Отсюда $\{c, a\} \bullet> \{\#, c\}$ и $\{c, d\} \bullet> \{a\}$

	S	B	#	a	b	c	d
S			$\underline{=}$			$\underline{=}$	
B	$\underline{=}$			$<\bullet$			
#	$\underline{=}$			$<\bullet$			
a		$\underline{=}$	$\bullet>$		$<\bullet$	$\bullet>$	$<\bullet$
b	$\underline{=}$			$<\bullet$			
c			$\bullet>$	$\bullet>$		$\bullet>$	
d				$\bullet>$			

0. $S' \rightarrow \#S\#$
1. $S \rightarrow aBSc$
2. $S \rightarrow a$
3. $B \rightarrow bSc$
4. $B \rightarrow d$

свертка $S \Leftarrow \text{aBSc}$

свертка $S \Leftarrow a$

свертка $B \Leftarrow bSc$ свертка $S \Leftarrow \text{aBSc}$

свертка $S \Leftarrow a$

свертка $B \Leftarrow d$

свертки снизу вверх!!

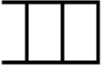
	S	B	#	a	b	c	d
S			$\underline{\underline{=}}$			$\underline{\underline{=}}$	
B	$\underline{\underline{=}}$			$<\bullet$			
#	$\underline{\underline{=}}$			$<\bullet$			
a		$\underline{\underline{=}}$	$\bullet>$		$<\bullet$	$\bullet>$	$<\bullet$
b	$\underline{\underline{=}}$			$<\bullet$			
c			$\bullet>$	$\bullet>$		$\bullet>$	
d				$\bullet>$			



Синтаксический анализ с помощью стека

- Синтаксический анализ на основе отношений предшествования удобно выполнять с помощью стека.
- В стек символ за символом записывается исходная цепочка вместе с отношениями предшествования между соседними символами до тех пор, пока отношение между парой соседних символов не станет ' $\bullet \rightarrow$ '.
- Найденная связка сворачивается к подходящему (единственному) нетерминалу (т.е. выталкивается из верхушки стека, а соответствующий нетерминал вталкивается вместо нее в стек), определяются отношения предшествования между новым нетерминалом и его соседями, и алгоритм продолжает работу до тех пор, пока цепочка $\# \doteq S \doteq \#$, представляющая начальную продукцию, не окажется в стеке

Восстановление правого вывода в грамматике предшествования

N	Стек 	Отношение между символом вверху стека и очередным символом входной цепочки	Остаток входной цепочки	Выполняемая операция	Примечание
1	\perp	$<\bullet$	$\#[abb]\#\perp$	Запись очередных символов в стек	
2	$\perp<\bullet\#\bullet[<\bullet a<\bullet b<\bullet b$	$\bullet>$	$]\#\perp$	Редукция $T \Leftarrow b$: замена b в стеке на T	Вычисление отношений между (b, T) и $(T,])$
3	$\perp<\bullet\#\bullet[<\bullet a<\bullet b \doteq T$	$\bullet>$	$]\#\perp$	Редукция $T \Leftarrow bT$: замена bT в стеке на T	Вычисление отношений между (a, T) и $(T,])$
4	$\perp<\bullet\#\bullet[<\bullet a \doteq T$	$\bullet>$	$]\#\perp$	Редукция $S \Leftarrow aT$: замена aT в стеке на S	Вычисление отношений между $([, S)$ и $(S,])$

Восстановление правого вывода в грамматике предшествования

5	$\perp < \bullet \# < \bullet \overset{\circ}{=} S$	$\overset{\circ}{=}$	$] \# \perp$] – в стек	Вычисление отношения между] и #.
6	$\perp < \bullet \# < \bullet \overset{\circ}{=} S \overset{\circ}{=}$	$\bullet >$	$\# \perp$	Редукция $S \Leftarrow [S]$: замена [S] в стеке на S	Вычисление отношения между S и #
7	$\perp < \bullet \# \overset{\circ}{=} S$	$\overset{\circ}{=}$	$\# \perp$	# – в стек	Вычисление отношения между # и \$
8	$\perp < \bullet \# \overset{\circ}{=} S \overset{\circ}{=} \#$	$\bullet >$	\perp	Редукция $S' \Leftarrow \# S \#$: замена #S# в стеке на S'	Вычисление отношения между S' и \$
9	$\perp < \bullet S'$	$\bullet >$	\perp	Допустить цепочку	

Пример: решение задачи

Задана грамматика (двусмысленная грамматика условных операторов)

0. $S' \rightarrow \# S \#$
1. $S \rightarrow i B t S$
2. $S \rightarrow i B t S e S$
3. $S \rightarrow s$
4. $B \rightarrow b$

Построить матрицу отношений предшествования

1. Отношения ' $\underline{\circ}$ '.

Пары: $\#S, S\#, iB, Bt, tS, Se, eS$

Отсюда: $\# \underline{\circ} S, S \underline{\circ} \#, i \underline{\circ} B, B \underline{\circ} t, t \underline{\circ} S, S \underline{\circ} e, e \underline{\circ} S$

2. Отношения ' $<\bullet$ '.

Пары: $iB, \#S, tS, eS$

Отсюда: $i <\bullet b$ и $\{ \#, t, e \} <\bullet \{ i, s \}$

3. Отношения ' $\bullet>$ '.

Пары: $S\#, Se, Bt$

Отсюда: $\{ s, S \} \bullet> \{ \#, e \}$ и $\{ b \} \bullet> \{ t \}$

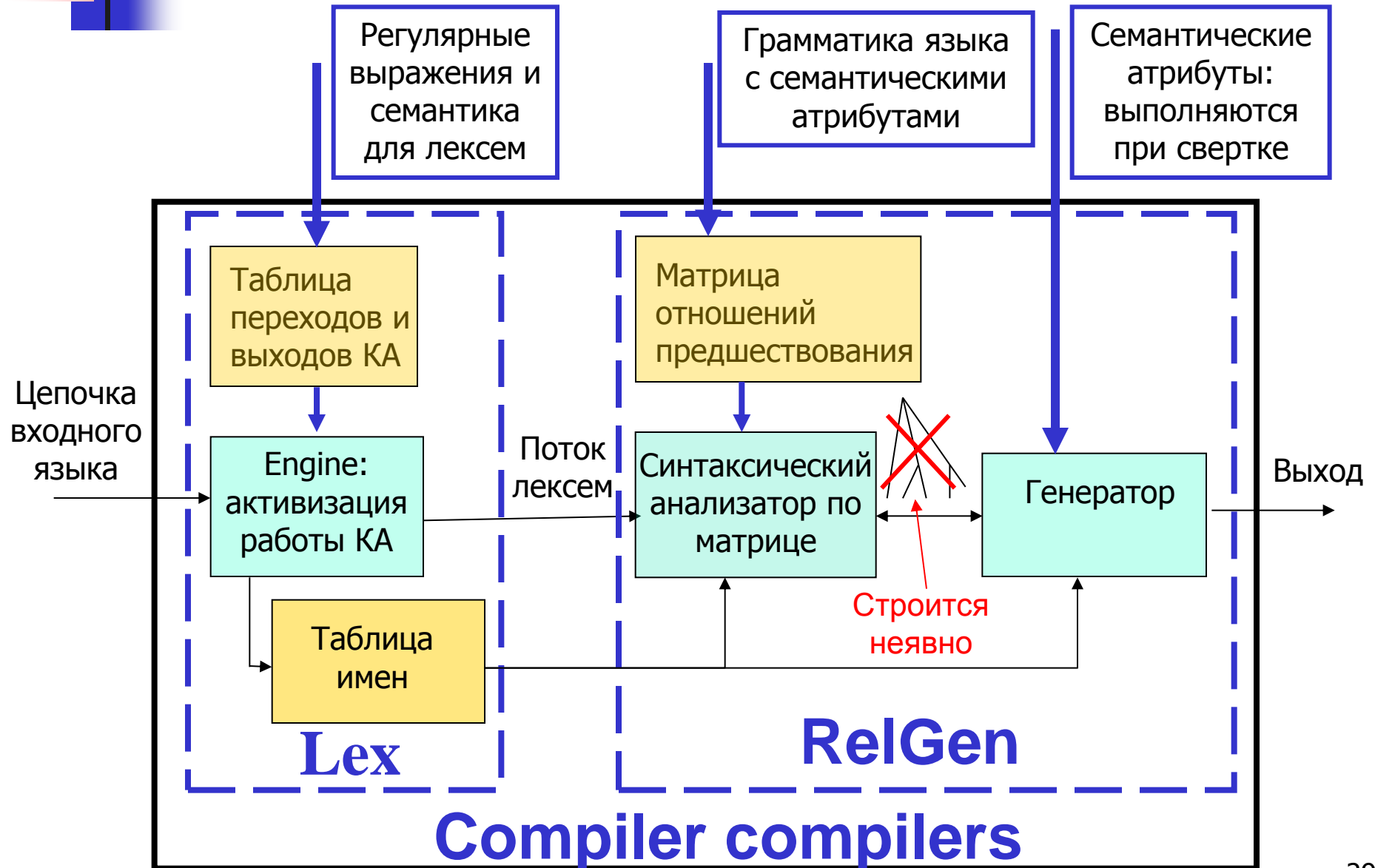
Матрица отношений предшествования неоднозначна. Невозможно выполнить синтаксический анализ произвольных цепочек методом простого предшествования

	S	B	#	i	b	t	e	s
S			$\underline{\circ}$ $\bullet>$				$\underline{\circ}$ $\bullet>$	
B						$\underline{\circ}$		
#	$\underline{\circ}$			$<\bullet$				$<\bullet$
i		$\underline{\circ}$			$<\bullet$			
b						$\bullet>$		
t	$\underline{\circ}$			$<\bullet$				$<\bullet$
e	$\underline{\circ}$			$<\bullet$				$<\bullet$
s			$\bullet>$				$\bullet>$	

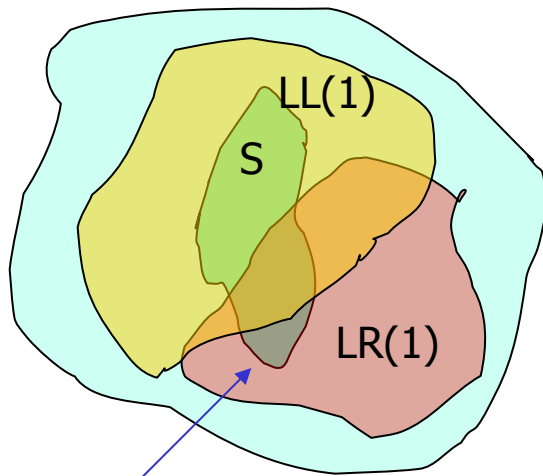
Теория грамматик простого предшествования

- Если матрица отношений предшествования неоднозначна, то выполнить синтаксический анализ этим методом нельзя
- **Определение.** КС-грамматика G является грамматикой простого предшествования, если:
 - она не содержит ϵ - продукций ;
 - она не содержит различных продукций с одинаковыми правыми частями;
 - матрица отношений предшествования этой грамматики однозначна, т.е. не содержит различных отношений для каждой пары символов
- Отношения простого предшествования для грамматики $G=(T,N,S,R)$ формально определяются так:
 - $X <\bullet Y \Leftrightarrow (\exists A \rightarrow \alpha X M \beta \in R): M \Rightarrow^+ Y \gamma ;$
 - $X \doteq Y \Leftrightarrow (\exists A \rightarrow \alpha X Y \beta \in R);$
 - $X \bullet > Y \Leftrightarrow (\exists A \rightarrow \alpha M Z \beta \in R): M \Rightarrow^+ \gamma X, Z \Rightarrow^* Y \delta, Y \in T$
- Два символа X, Y могут стоять рядом в произвольной сентенциальной форме, выводимой в грамматике G , если только для пары X, Y существует хотя бы одно отношение предшествования
- Если для пары соседних символов в процессе разбора не найдется в матрице предшествования никакого отношения предшествования, то это свидетельствует об ошибке во входной строке

Компилятор компиляторов для грамматик предшествования



LR(k) - грамматики



Грамматики LR(k)

Подклассы КС-грамматик
s-грамматики
LL(k)-грамматики,
Грамматики рекурсивного спуска
Грамматики предшествования
LR(k)-грамматики,
LALR(k)-грамматики,
...



LR(k) грамматики

- Это наиболее широкий класс грамматик, допускающих эффективный восходящий детерминированный синтаксический анализ
- LR(k)-грамматики были введены Дональдом Кнутом и являются основным классом грамматик, для которого строятся практические производственные компиляторы
- В 1975г. для LALR(1) грамматик (подкласса LR-грамматик) был разработан компилятор компиляторов YACC (Yet Another Compiler-Compiler).

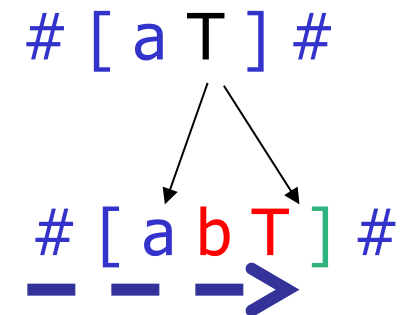
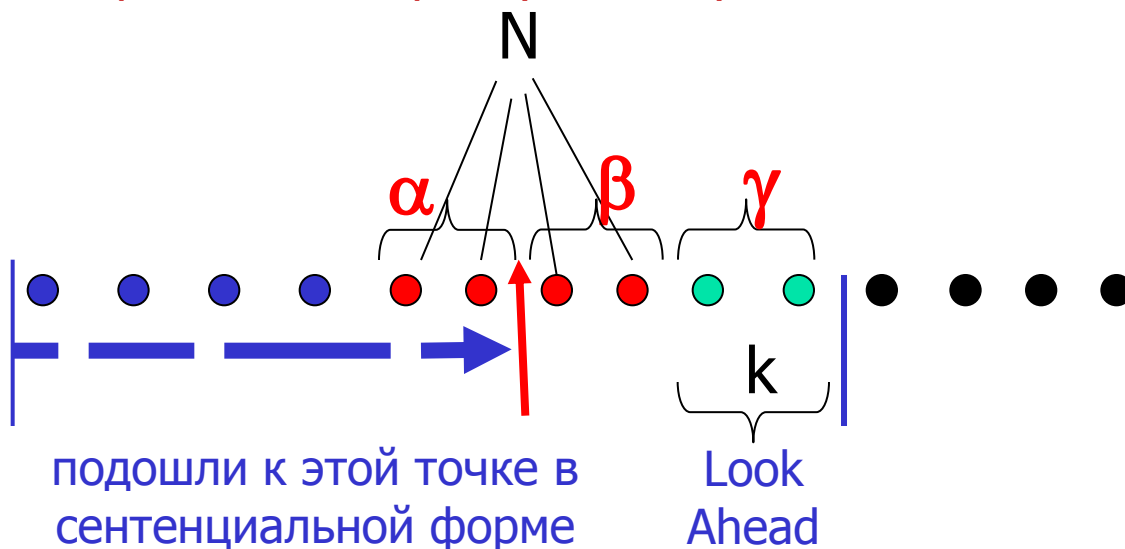
В течение уже почти 40 лет!!! этот компилятор компиляторов является основным инструментом разработки компиляторов для реальных языков программирования
Это БЕСПРЕЦЕДЕНТНЫЙ пример долгожительства в информатике

LR(k)–распознаватель

G: 0. $S' \rightarrow \#S\#$
 1. $S \rightarrow a$
 2. $S \rightarrow aT$
 3. $S \rightarrow [S]$
 4. $T \rightarrow b$
 5. $T \rightarrow bT$

$S' \Rightarrow \#S\# \Rightarrow \#[S]\# \Rightarrow \#[aT]\# \Rightarrow \#[abT]\# \Rightarrow \#[abb]\#$

- LR(k)-распознаватель, считывая sentенциальную форму символ за символом, имеет цель определить самую левую связку, а также нетерминал левой части, которым следует заменить эту связку
- LR(k)-распознаватель при поиске связки учитывает информацию не только о парах соседних символов, но информацию о всей просмотренной слева части входной цепочки. LR(k)-распознаватель принимает на вход sentенциальную форму – промежуточную цепочку вывода - и выдает ответ: **какая продукция определяет очередную связку**



Обозначение ситуации:
 $\langle N \rightarrow \alpha \bullet \beta; \lambda \rangle$

L R (k) - грамматики

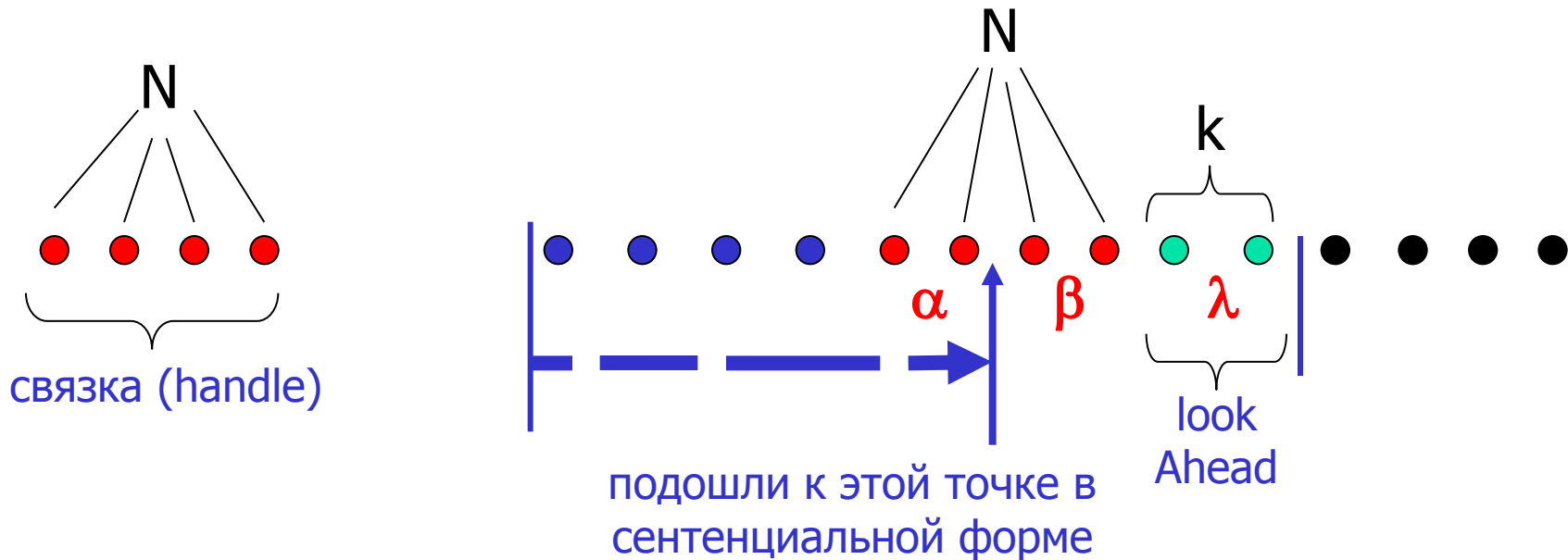
→ просмотр вперед на k символов **после** связи

→ восстанавливаем **правый** вывод

→ просматриваем цепочку **слева**

Ситуации в LR(k)–распознавателях

Ситуация – это четверка: $\langle N, \alpha, \beta, \lambda \rangle$



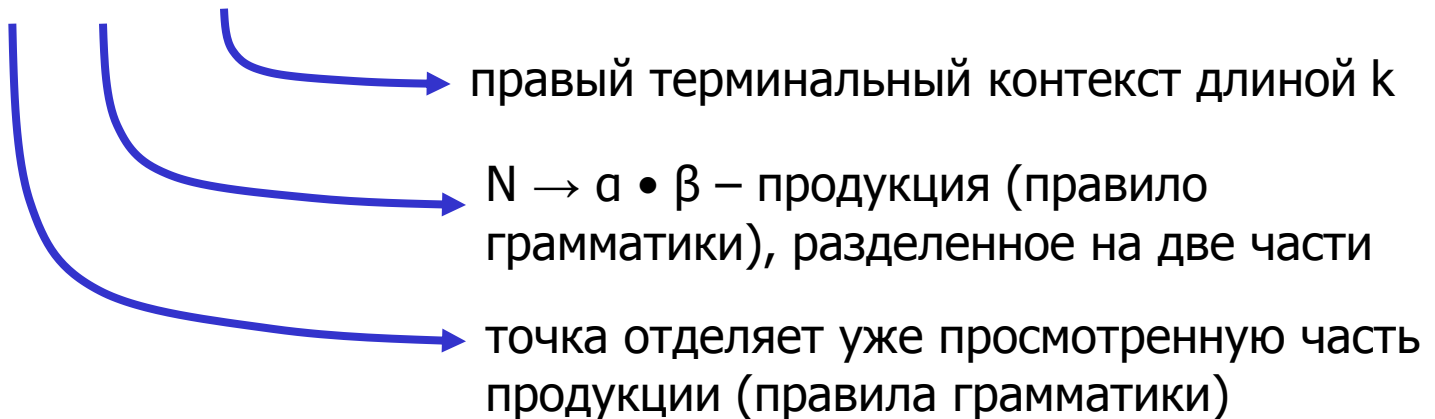
Обозначение ситуации: $\langle N \rightarrow \alpha \bullet \beta; \lambda \rangle$

Смысл ситуации: при просмотре очередной сентенциальной формы мы подошли к точке, в которой, если далее встретится цепочка β , а затем будет идти цепочка λ , то мы можем выполнить свертку $N \leftarrow \alpha\beta$

Множество ситуаций

Смысл ситуации: при просмотре очередной сентенциальной формы мы подошли к точке, в которой, если далее встретится цепочка β , а затем будет идти цепочка λ , то мы можем выполнить свертку $N \leftarrow \alpha\beta$

$\langle N \rightarrow \alpha \bullet \beta ; \lambda \rangle$ - ситуация



- **С точки зрения возможных сверток** каждую позицию внутри сентенциальной формы в процессе ее просмотра можно описать *множеством ситуаций* – множеством продукций, соответствующих этой позиции, с меткой-указателем в каждой продукции, показывающей, в каком месте продукции находится анализатор на данном шаге просмотра сентенциальной формы.



Ситуаций конечное число

$\langle N \rightarrow \alpha \bullet \beta ; \lambda \rangle$ - ситуация

Поскольку продукций у грамматики конечное количество, их наборов с указанием метки позиции внутри них – тоже конечное количество, и подмножеств таких помеченных продукций (характеристических множеств) со своими контекстами также конечное количество.

Д. Кнут ввел понятие ситуаций, показал, что для любой грамматики число ситуаций конечно, число возможных множеств ситуаций конечно, и, следовательно, каждый шаг преобразования от одной сентенциальной формы к предыдущей описывается конечным автоматом, состояниями которого являются характеристические множества

LR(k)– распознаватель является конечным автоматом

Построение LR(0)–анализатора: вперед не заглядываем

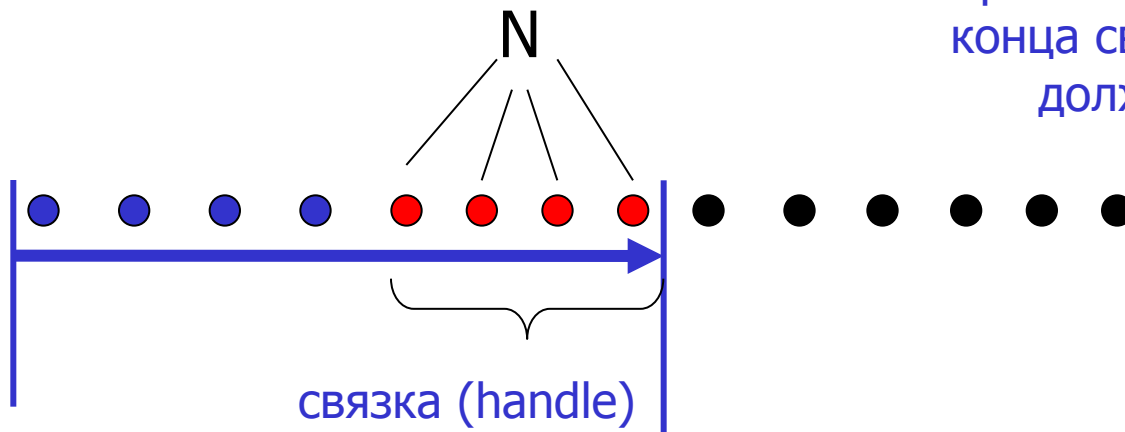
- Рассмотрим самый простой класс: LR(0)–грамматики. Построенный LR(0)-анализатор должен в каждой позиции просматриваемых сентенциальных форм указать, в каком месте возможных связок может находиться анализ, и при достижении границы связки явно ее указать

G= 0. $S' \rightarrow \#S\#$

1. $S \rightarrow aSc$

2. $S \rightarrow b$

вывод : $S' \Rightarrow \#S\# \Rightarrow \#aSc\# \Rightarrow \#aaSc\# \Rightarrow \#aabcc\#$



При $k=0$ как только мы дойдем до конца связки, распознаватель это должен сразу обнаружить

Построение LR(0)–анализатора

- $G =$
 0. $S' \rightarrow \#S\#$
 1. $S \rightarrow aSc$
 2. $S \rightarrow b$

Начальное множество ситуаций:

$S' \rightarrow \bullet \#S\#$

вывод : $S' \Rightarrow \#S\# \Rightarrow \#aSc\# \Rightarrow \#aaSc\# \Rightarrow \#aabcc\#$

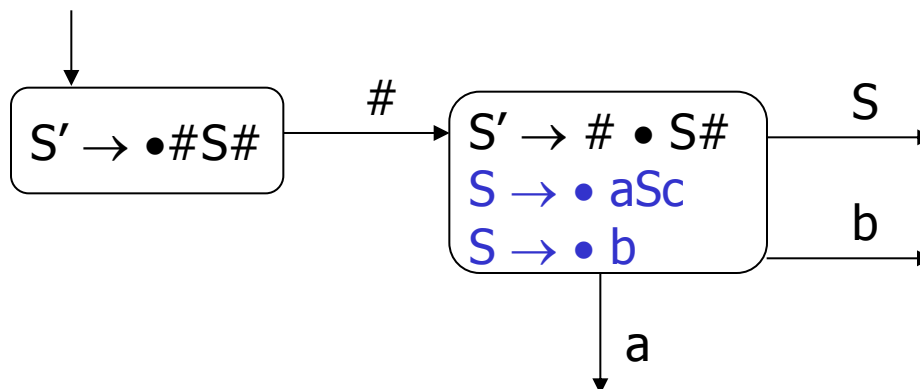
- Возможными входами LR(0)–анализатора для этой грамматики могут быть любые сентенциальные формы, $\{\#S\#, \#aSc\#, \#aaSc\#, \dots \#b\#, \#aaabccc\#, \dots\}$. В каждой такой цепочке распознаватель должен найти связку - что сворачивать, и к чему
- Распознаватель - конечный автомат. Каждым его состоянием будет характеристическое множество - множество помеченных продукций, на границе или внутри которых мы можем находиться в данный момент
- В начале анализа (в начальном состоянии) такая продукция - только одна, $S' \rightarrow \#S\#$, а находиться мы можем только **перед** ее правой частью. Итак, начальное множество ситуаций (начальное состояние) LR(0)-анализатора для этой грамматики – $[S' \rightarrow \bullet \#S\#]$.

Следующее множество ситуаций

$$\begin{array}{l} S' \rightarrow \# S \# \\ S \rightarrow a S c \\ S \rightarrow b \end{array}$$

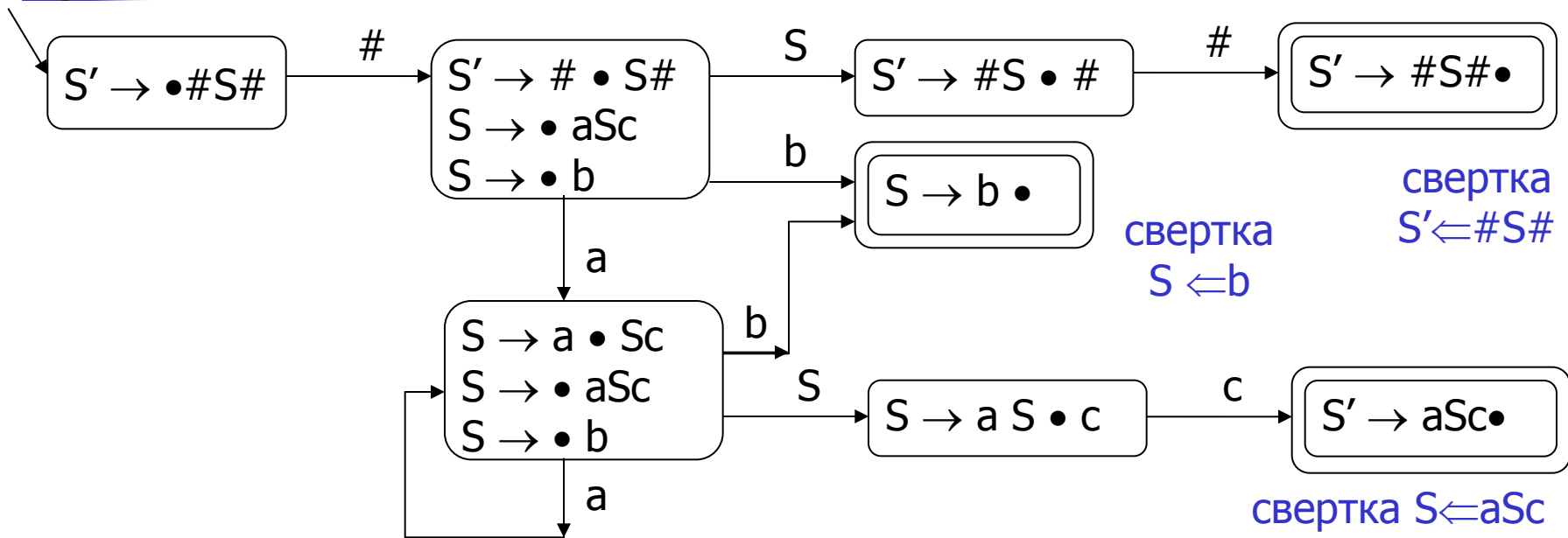
- Следующая ситуация зависит от того, какой символ в обрабатываемой сентенциальной форме мы встретим. В этой конкретной грамматике все сентенциальные формы начинаются с символа '#', на это указывает и единственный такой символ, стоящий вслед за точкой в ситуации $\langle S' \rightarrow \bullet \# S \# \rangle$
- Поэтому под воздействием этого входа '#' распознающий автомат перейдет в следующее состояние, определяемое множеством ситуаций, которое включает эту продукцию **с продвинутой меткой-указателем позиции**: $\langle S' \rightarrow \# \bullet S \# \rangle$
- Среди сентенциальных форм, которые могут обрабатываться анализатором, есть одна такая, в которой после маркера идет символ S. Но во всех других сентенциальных формах нетерминал S уже заменен в соответствии с одним из правил грамматики. Поэтому надо принять во внимание, что, возможно, LR(0)–анализатор при анализе в данной позиции может находиться не только внутри связки для продукции $S' \rightarrow \# \bullet S \#$, а **перед началом любой связки, соответствующей продукциям с нетерминалом S в левой части правил**, т.е. возможно, что к S еще подстрока внутри данной сентенциальной формы не свернута, но та подстрока, которая **может быть свернута к S, начинается именно здесь!**
- Поэтому множество ситуаций LR(0)–анализатора **в данном состоянии** включает и ситуации $S \rightarrow \bullet a S c$ и $S \rightarrow \bullet b$.
- Полное множество ситуаций в данном состоянии –
$$\{ \langle S' \rightarrow \# \bullet S \# \rangle, \langle S \rightarrow \bullet a S c \rangle, \langle S \rightarrow \bullet b \rangle \}$$

LR(0)–анализатор обрабатывает все сентенциальные формы

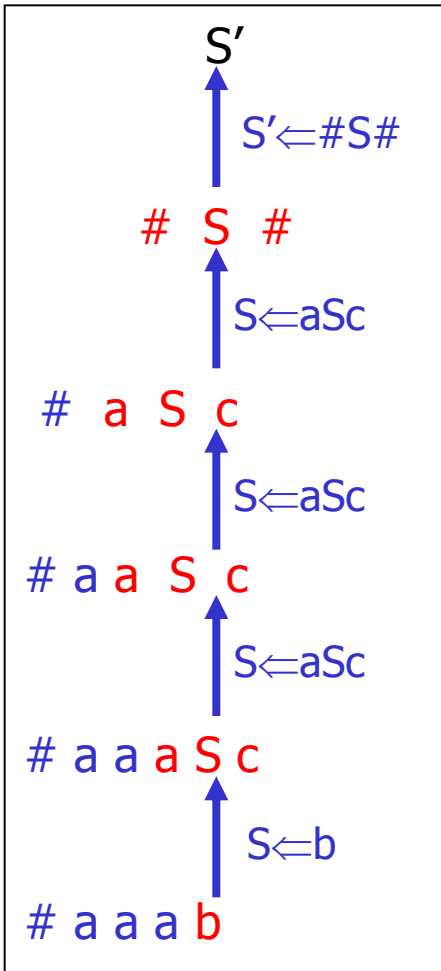
$$\begin{aligned} S' &\rightarrow \#S\# \\ S &\rightarrow aSc \\ S &\rightarrow b \end{aligned}$$
$$S' \Rightarrow \#S\# \Rightarrow \#aSc\# \Rightarrow \#aaSc\# \Rightarrow \#aabcc\#$$
$$S' \Rightarrow \#S\# \Rightarrow \#b\#$$


- Следующее состояние LR(0)–анализатора зависит от очередного входного символа анализируемой сентенциальной формы
 - Если сентенциальная форма $\#S\#$, то следующим мы встретим нетерминал S , если эта сентенциальная форма $\#b\#$, то следующим встретится терминал b , во всех остальных случаях правильной входной цепочки очередным входом будет терминал a . Но именно эти очередные символы и определяются как следующие в этом состоянии после точки, что видно из множества $[S' \rightarrow \# \bullet S \#, S \rightarrow \bullet a S c, S \rightarrow \bullet b]$.
- Каждый возможный входной символ изменяет состояние анализатора
- Входными могут быть как терминальные, так и нетерминальные символы, поскольку входом распознавателя являются сентенциальные формы

LR(0) автомат разбора

$$\begin{aligned} S' &\rightarrow \#S\# \\ S &\rightarrow aSc \\ S &\rightarrow b \end{aligned}$$


- Автомат LR(0)-разбора имеет три заключительных состояния по числу продукций грамматики: целью анализатора как раз и является определение того, в соответствии с какой из продукций нужно выполнить очередную свертку. В этих заключительных состояниях множество ситуаций состоит из одной ситуации вида $N \rightarrow a \bullet$. Такие ситуации будем называть *терминальными*
- Наличие единственной терминальной продукции в характеристическом множестве дает право принять однозначное решение о выполнении редукции $N \leftarrow a$, поскольку ситуация $N \rightarrow a \bullet$ говорит о том, что конец соответствующей связки в сентенциальной форме достигнут

$$\begin{aligned} S' &\rightarrow \#S\# \\ S &\rightarrow aSc \\ S &\rightarrow b \end{aligned}$$


Цепочка:
a a a b c c c

```
graph LR; start(( )) --> q0((q0)); q0 -- # --> q1((q1)); q1 -- S --> q2((q2)); q2 -- # --> q3(((q3))); q1 -- b --> q4(((q4))); q1 -- a --> q5((q5)); q5 -- b --> q4; q5 -- S --> q6((q6)); q5 -- a --> q5; q6 -- c --> q7(((q7))); style start fill:none,stroke:none
```

$S' \leftarrow \#S\#$

$S \leftarrow b$

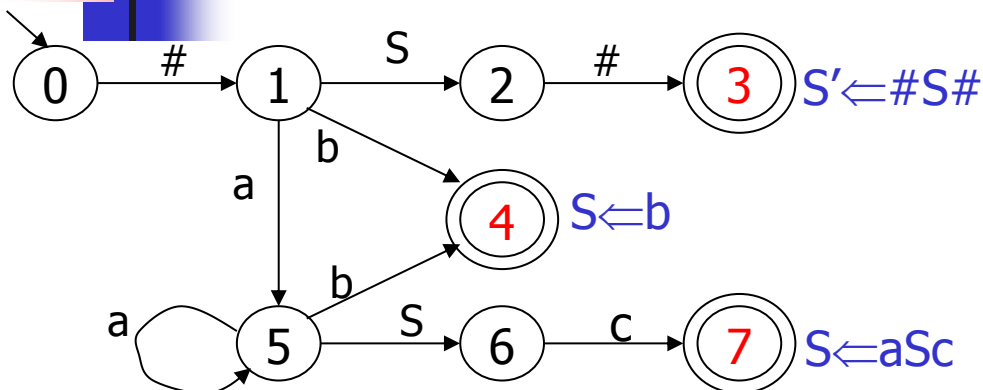
$S \leftarrow aSc$

- LR(0)–автомат принимает на вход любую sentential форму, выводимую в грамматике G . Попадание в заключительное состояние с терминальной продукцией свидетельствует о том, что при движении по sentential форме мы достигли конца связки, и необходимо выполнить редукцию: последние символы sentential формы заменить на нетерминал, стоящий в левой части этой продукции.
- **Определение.** КС-грамматика является LR(0) –грамматикой, если не более одной помеченной продукции вида $N \rightarrow a \bullet$ входит в каждое множество ситуаций построенного для нее LR(0)– анализатора. При этом условии LR(0)–анализатор позволяет однозначно принять решение о редукции связки в sentential формах.

- Конечно, бессмысленно повторять для очередной сентенциальной формы весь тот путь из начального состояния, который распознающим автоматом был проделан до границы свернутой связки в предыдущей сентенциальной форме: последовательность пройденных состояний будет повторяться
 - Разумно этот путь сохранять в стеке, подобно тому, как для грамматик предшествования в стеке сохраняется начальный префикс сентенциальной формы со вставленными между символами отношениями предшествования
- Результатом анализа в LR- распознавателе является путь в распознающем автомате; именно его и нужно запоминать в стеке как последовательность символов входной цепочки, перемежаемых номерами состояний распознающего автомата
- Вначале в стеке находится номер 0 начального состояния

Синтаксический анализ с помощью LR(0) анализатора

1. $S' \rightarrow \#S\#$
2. $S \rightarrow aSc$
3. $S \rightarrow b$



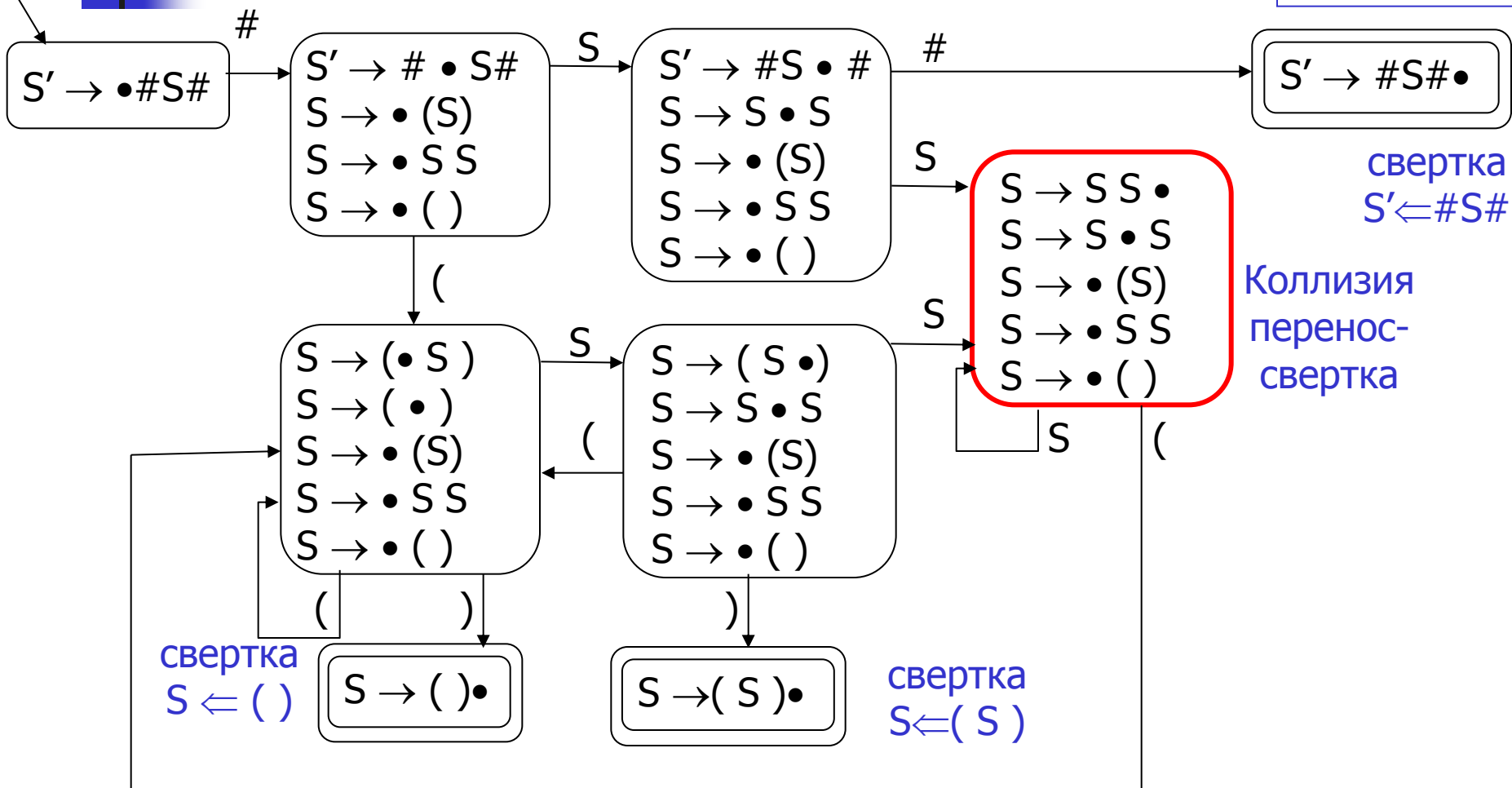
Анализ #aabccs#

Для того, чтобы не повторять весь путь сначала при анализе новой сентенциальной формы, храним в стеке и начало цепочки, и путь, т.е. последовательность состояний/входов

Стек	Тек s	Оч симв	Ост. вх цеп	Предпринимаемые действия
\perp	0	#	# a a b c c #	Перенос в стек тек сост и оч символа
$\perp 0\#$	1	a	a a b c c #	- " " -
$\perp 0\#1a$	5	a	a b c c #	- " " -
$\perp 0\#1a5a$	5	b	b c c #	- " " -
$\perp 0\#1a5a5b$	4	c	c c #	Свертка 3: $S \leftarrow b$
$\perp 0\#1a5a5S$	6	c	c c #	Перенос в стек тек сост и оч символа
$\perp 0\#1a5a5S6c$	7	c	c #	Свертка 2: $S \leftarrow aSc$
$\perp 0\#1a5S$	6	c	c #	Перенос в стек тек сост и оч символа
$\perp 0\#1a5S6c$	7	#	#	Свертка 2: $S \leftarrow aSc$

Пример: LR(0)-автомат разбора для двусмысленной грамматики

$S' \rightarrow \#S\#$
 $S \rightarrow (S)$
 $S \rightarrow S S$
 $S \rightarrow ()$



Наличие коллизии говорит о том, что грамматика – не относится к классу LR(0)



Заключение

- LR(k)-алгоритмы – это эффективные (линейные) и очень мощные восходящие алгоритмы синтаксического анализа: они мощнее всех рассматривавшихся ранее алгоритмов для подклассов КС-грамматик. Коммерческие компиляторы для современных ЯВУ строятся на основе LR-грамматик
- Восходящие алгоритмы синтаксического анализа удобны для атрибутивных семантических вычислений: вместе со сверткой (редукцией) по синтаксическому правилу выполняются семантики. Особенно это удобно для синтезированных атрибутов
- LR(0)-анализатор очень простой и удобный, но имеет ограничения: мало грамматик, для которых может быть построен такой анализатор
- В следующей лекции мы рассмотрим LR(1) анализатор, позволяющий разрешать конфликты, и подклассы грамматик SLR(k) и LALR(k), которые являются, с одной стороны, простыми, а с другой – мощными, т.е. такие анализаторы можно строить для широкого класса грамматик



Нисходящая vs (versus) восходящая стратегии

- Нисходящая стратегия распознавания (LL(k)-языки и грамматики) достаточно мощная техника, на ней основаны системы распознавания и построения трансляторов, в частности, транслятор языка Паскаль
- Восходящая стратегия распознавания (LR(k)-языки и грамматики) – очень мощная техника. Практические системы построения компиляторов в основном построены на LR(k) распознавателях
- Из Википедии:

Существуют противоречия между так называемой «Европейской школой» построения языков, которая основывается на LL-грамматиках, и «Американской школой», которая предпочитает LR-грамматики. Такие противоречия обусловлены традициями преподавания и деталями описания различных методов и инструментов в конкретных учебниках; кроме того, свое влияние оказал Н. Вирт из ETHZ, чьи исследования описывают различные методы оптимизации LL(1) распознавателей и компиляторов. «Американская школа» начата Д. Кнутом
- Мы здесь не пытаемся выполнять коррекцию/восстановление после обнаружения ошибок. Наш анализ распознает синтаксические ошибки и просто останавливается на первой из них, как это происходит в Turbo Pascal



Спасибо за внимание