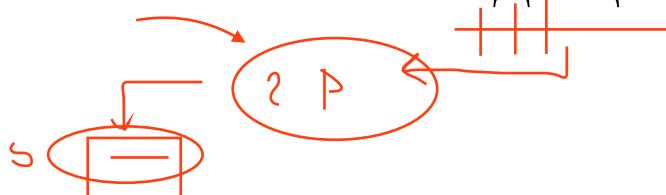


# Глава 10

## Алгоритм на основе восходящего анализа



В данном разделе будут рассмотрены алгоритмы восходящего синтаксического анализа LR-семейства, в том числе Generalized LR (GLR). Также будет рассмотрено обобщение алгоритма GLR для решения задачи поиска путей с контекстно-свободными ограничениями в графах.

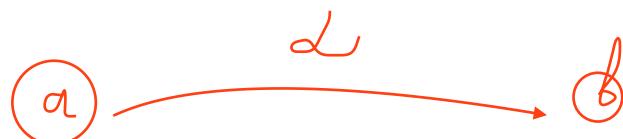
### 10.1 Восходящий синтаксический анализ

Существует большое семейство  $LR(k)$  алгоритмов — алгоритм восходящего синтаксического анализа. Основная идея, лежащая в основе семейства, заключается в следующем: входная последовательность символов считывается слева направо с попутным добавлением в стек и выполнением сворачивания на стеке — замены последовательности терминалов и нетерминалов, лежащих наверху стека, на нетерминал, если существует соответствующее правило в исходной грамматике.

Как и в случае с LL используется магазинный автомат, управляемый таблицами, построенными по грамматике. При этом, у LR анализатора есть два типа команд:

1. shift — прочитать следующий символ входной последовательности, положив его в стек, и перейти в следующее состояние;
2. reduce( $k$ ) — применить  $k$ -ое правило грамматики, правая часть которого уже лежит на стеке: снимаем со стека правую часть продукции и кладём левую часть.

123



→ А управляющая таблица выглядит следующим образом.

States	$t_0$	...	$t_a$	...	$\$$	$N_0$	...	$N_b$	...
10	...	...	$s_i$	...	$r_k$	...	...	$j$	...
...	...	...	...	...	$acc$	...	...	...	...

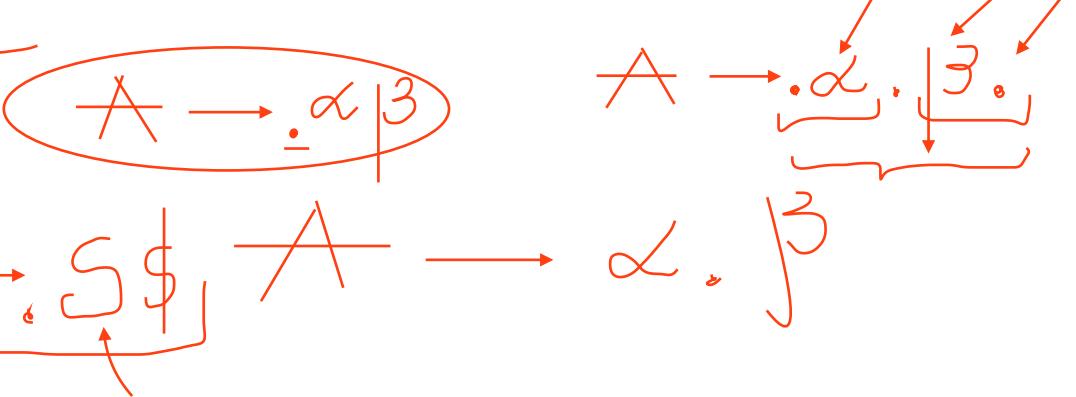
Здесь

- $s_i$  — shift: перенести соответствующий символ в стек и перейти в состояние  $i$ .
- $r_k$  — reduce( $k$ ): в стеке накопилась правая часть продукции  $k$ , пора производить свёртку.
- $j$  — goto: выполняется после reduce. Сама по себе команда reduce не переводит автомат в новое состояние. Команда goto переведёт автомат в состояние  $j$ .
- $acc$  — accept: разбор завершился успешно.

Если ячейка пустая и в процессе работы мы пропали в неё — значит произошла ошибка. Для детерминированной работы анализатора требуется, чтобы в каждой ячейке было не более одной команды. Если это не так, то говорят о возникновении конфликтов.

- С
- shift-reduce — ситуация, когда не понятно, читать ли следующий символ или выполнить reduce. Например, если правая часть одного из правил является префиксом правой части другого правила:  $N \rightarrow u, M \rightarrow wu'$ .
  - reduce-reduce — ситуация, когда не понятно, к какому правилу нужно применить reduce. Например, если есть два правила с одинаковыми правыми частями:  $N \rightarrow w, M \rightarrow w$ .

Принцип работы LR анализаторов следующий. Пусть у нас есть входная строка, LR-автомат со стеком и управляющая таблица. В начальный момент на стеке лежит стартовое состояние LR-автомата, позиция во входной строке соответствует её началу. На каждом шаге анализируется текущий символ входа и текущее состояние, в котором находится автомат, и совершается одно из действий:



- Если в управляющей таблице нет инструкции для текущего состояния автомата и текущего символа на входе, то завершаем разбор с ошибкой.
- Иначе выполняем одну из инструкций:
  - в случае acc — успешно завершаем разбор.
  - в случае shift — кладем на стек текущий символ входа, сдвигая при этом текущую позицию, и номер нового состояния. Переходим в новое состояние.
  - в случае reduce(k) — снимаем со стека  $2l$  элементов:  $l$  состояний и  $l$  терминалов/нетерминалов (где  $l$  — длина правой части  $k$ -ого правила), кладём на стек нетерминал левой части правила. Теперь на вершине стека у нас нетерминал  $N_a$ , а следующий элемент — состояние  $i$ . Если в ячейке  $(i, N_a)$  управляющей таблицы лежит состояние  $j$  то кладём его на вершину стека. Иначе завершаемся с ошибкой.

Разные алгоритмы из LR-семейства строят таблицы разными способами и, соответственно, могут избегать тех или иных конфликтов. Рассмотрим некоторых представителей.

### 10.1.1 LR(0) алгоритм

Данный алгоритм самый “слабый” из семейства — разбирает наименьший класс языков. Для построения используются LR(0) пункты.

**Определение 10.1.1.** LR(0) пункт (LR(0) item) — правило грамматики, в правой части которого имеется точка, отделяющая уже разобранную часть правила (слева от точки) от того, что еще предстоит распознать (справа от точки):  $A \rightarrow \underline{\alpha} \cdot \beta$ , где  $A \rightarrow \alpha\beta$  — правило грамматики. □

Состояние LR(0) автомата — множество LR(0) пунктов. Для того чтобы из них построить используется операция *closure* или *замыкание*.

**Определение 10.1.2.**  $\text{closure}(X) = \text{closure}(X \cup \{M \rightarrow \cdot \gamma \mid N_i \rightarrow \alpha \cdot M\beta \in X\})$  □

**Определение 10.1.3.** Ядро — исходное множество пунктов, до применения к нему замыкания.

Для перемещения точки в пункте используется функция *goto*.

**Определение 10.1.4.**  $\text{goto}(X, p) = \{N_j \rightarrow \alpha p \cdot \beta \mid N_j \rightarrow \alpha \cdot p\beta \in X\}$

Теперь мы можем построить LR(0) автомат. Первым шагом необходимо расширить грамматику: добавить к исходной грамматике правило вида  $S' \rightarrow S\$$ , где  $S$  — стартовый нетерминал исходной грамматики,  $S'$  — новый стартовый нетерминал (не использовался ранее в грамматике),  $\$$  — маркер конца строки (не входил в терминальный алфавит исходной грамматики).

Далее строим автомат по следующим принципам.

- Состояния — множества пунктов.
- Переходы между состояниями осуществляются по символам грамматики.
- Начальное состояние —  $\text{closure}(\{S' \rightarrow \cdot S\$\})$ .
- Следующее состояние по текущему состоянию  $X$  и символу  $p$  вычисляются как  $\text{closure}(\text{goto}(X, p))$

Управляющая таблица по автомату строится следующим образом.

- $acc$  в ячейку, соответствующую финальному состоянию и  $\$$
- $s_i$  в ячейку  $(j, t)$ , если в автомате есть переход из состояния  $j$  по терминалу  $t$  в состояние  $i$
- $i$  в ячейку  $(j, N)$ , если в автомате есть переход из состояния  $j$  по нетерминалу  $N$  в состояние  $i$

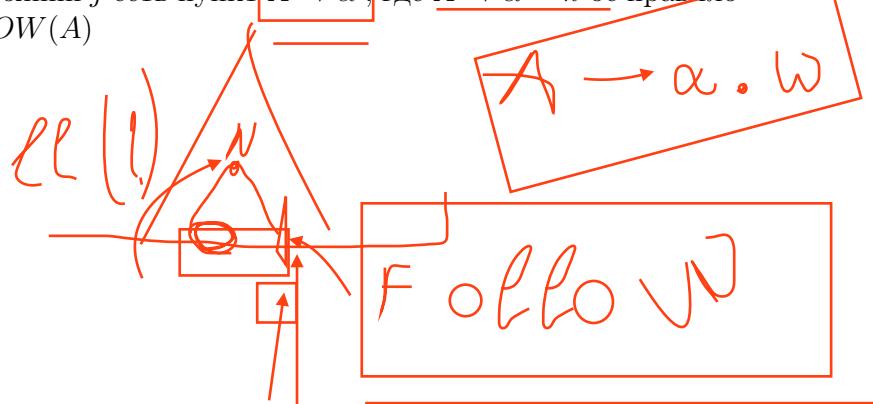
- $r_k$  в ячейку  $(j, t)$ , если в состоянии  $j$  есть пункт  $A \rightarrow \alpha \cdot$ , где  $A \rightarrow \alpha$  —  $k$ -ое правило грамматики,  $t \in FOLLOW(A)$

### 10.1.2 SLR(1) алгоритм

X.

SLR(1) анализатор отличается от LR(0) анализатора построением таблицы по автомату (автомат в точности как у LR(0)). А именно,  $r_k$  добавляется в ячейку  $(j, t)$ , если в состоянии  $j$  есть пункт  $A \rightarrow \alpha \cdot$ , где  $A \rightarrow \alpha$  —  $k$ -ое правило грамматики,  $t \in FOLLOW(A)$

$$\left\{ \begin{array}{l} N \rightarrow \alpha \cdot \\ M \rightarrow \alpha \cdot \end{array} \right.$$



### 10.1.3 CLR(1) алгоритм

Canonical LR(1), он же LR(1). Данный алгоритм является дальнейшим расширением SLR(1): к пунктам добавляются множества предпросмотра (lookahead).

**Определение 10.1.5.** Множество предпросмотра для правила  $P$  — терминалы, которые должны встретиться в выведенной строке сразу после строки, выводимой из данного правила.  $\square$

**Определение 10.1.6.** CLR пункт:  $[A \rightarrow \alpha \cdot \beta, \{t_0, \dots, t_n\}]$ , где  $t_0, \dots, t_n$  — множество предпросмотра для правила  $A \rightarrow \alpha\beta$ .  $\square$

**Определение 10.1.7.** Пусть дана грамматика  $G = \langle \Sigma, N, R, S \rangle$ .

$$\text{closure}(X) = \text{closure}(X \cup \{[B \rightarrow \cdot \delta, \text{FIRST}(\beta t_0), \dots, \text{FIRST}(\beta t_n)] \mid B \rightarrow \beta \in R, [A \rightarrow \alpha \cdot B\beta, \{t_0, \dots, t_n\}] \in \text{closure}(X)\})$$

 $\square$ 

Функция  $goto$  определяется аналогично LR(0), автомат строится по тем же принципам.

При построении управляющей таблицы усиливается правило добавления команды  $redice$ . А именно, добавляем  $r_k$  в ячейку  $(j, t_i)$ , если в состоянии  $j$  есть пункт  $[A \rightarrow \alpha \cdot, \{t_0, \dots, t_n\}]$ , где  $A \rightarrow \alpha$  —  $k$ -ое правило грамматики.

### 10.1.4 Примеры

Рассмотрим построение автоматов и таблиц для различных модификаций LR алгоритма.

Возьмем следующую грамматику:

0) $S \rightarrow aSbS$
1) $S \rightarrow \epsilon$

Расширим вышеупомянутую грамматику, добавив новый стартовый нетерминал  $S'$ , и далее будем работать с этой расширенной грамматикой:

0) $S \rightarrow aSbS$
1) $S \rightarrow \epsilon$
2) $S' \rightarrow S\$$

**Пример 10.1.1.** Пример ядра и замыкания.

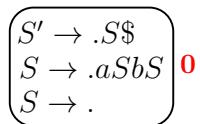
Возьмем правило 2 нашей грамматики, предположим, что мы только начинаем разбирать данное правило.

Ядром в таком случае является item исходного правила:  $S' \rightarrow .S\$$

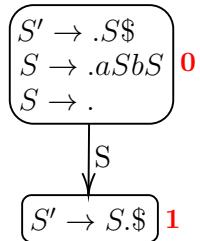
При замыкании добавятся ещё два item'a с правилами по выводу нетерминала 'S', поэтому получаем три item'a:  $S' \rightarrow .S\$$ ,  $S \rightarrow .aSbS$  и  $S \rightarrow .\varepsilon$

**Пример 10.1.2.** Пример построения LR(0)-автомата для нашей грамматики с применением замыкания.

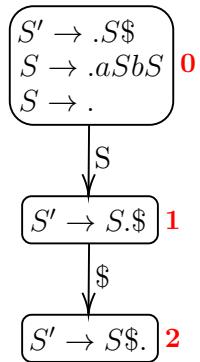
- Добавляем стартовое состояние: item правила 0 и его замыкание (вместо item'a  $S \rightarrow .\varepsilon$  будем писать  $S \rightarrow .$ ).



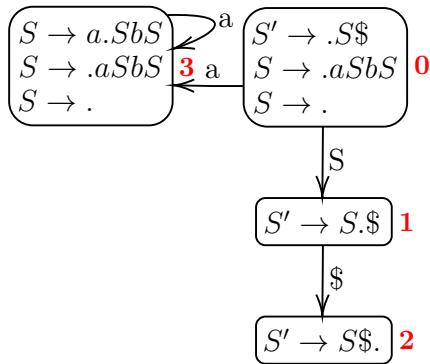
- По 'S' добавляем переход из стартового состояния в новое состояние 1.



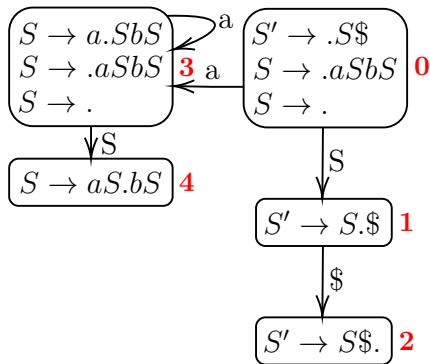
- По '\$' добавляем переход из состояния 1 в новое состояние 2.



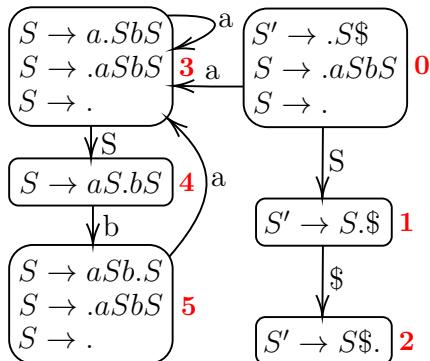
- По 'a' добавляем переход из стартового состояния в новое состояние 3 и делаем его замыкание. Также добавляем переход по 'a' из этого состояния в себя же.



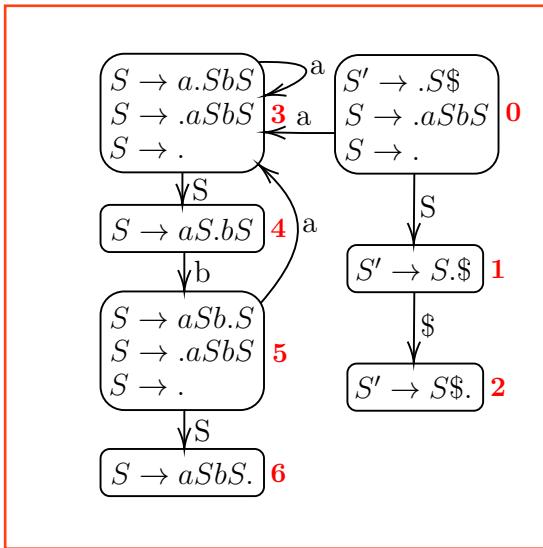
5. По 'S' добавляем переход из состояния 3 в новое состояние 4.



6. По 'b' добавляем переход из состояния 4 в новое состояние 5 и делаем его замыкание. Также добавляем переход по 'a' из этого состояния в состояние 3.



7. По 'S' добавляем переход из состояния 5 в новое состояние 6. Завершаем построение LR-автомата.



Далее будем использовать этот автомат для построения управляющей таблицы.

**Пример 10.1.3.** Пример управляющей LR(0) таблицы.

	a	b	\$	S
0	$s_3, r_1$	$r_1$	$r_1$	1
1			acc	
2	$r_2$	$r_2$	$r_2$	
3	$s_3, r_1$	$r_1$	$r_1$	4
4		$s_5$		
5	$s_3, r_1$	$r_1$	$r_1$	6
6	$r_0$	$r_0$	$r_0$	

Как видим, в данном случае в таблице присутствуют shift-reduce конфликты. В случае, когда не удается построить таблицу без конфликтов, говорят, что грамматика не LR(0).

□

**Пример 10.1.4.** Пример управляющей LR(1) таблицы. Автомат тот же, однако команды *reduce* расставляются с использованием FOLLOW.

$$FOLLOW_1(S) = \{b, \$\}$$

	a	b	\$	S
0	$s_3$	$r_1$	$r_1$	1
1			acc	
2				
3	$s_3$	$r_1$	$r_1$	4
4		$s_5$		
5	$s_3$	$r_1$	$r_1$	6
6		$r_0$	$r_0$	

В данном случае в таблице отсутствуют shift-reduce конфликты. То есть наша грамматика SLR(1), но не LR(0).

□

**Пример 10.1.5.** Пример LR-разбора входного слова abab\$ из языка нашей грамматики с использованием построенных ранее LR-автомата и управляющей таблицы.

- Начало разбора. На стеке — стартовое состояние 0.

Вход:	a	b	a	b	\$
Стек:	0				

- Выполняем shift 3: сдвигаем указатель на входе, кладем на стек 'a', новое состояние 3 и переходим в него.

Вход:	a	b	a	b	\$
Стек:	0	a	3		

- Выполняем reduce 1 (кладем на стек 'S'), кладем новое состояние 4 и переходим в него.

Вход:	a	b	a	b	\$
Стек:	0	a	3	S	4

- Выполняем shift 5: сдвигаем указатель на входе, кладем на стек 'b', новое состояние 5 и переходим в него.

Вход:	a	b	a	b	\$
Стек:	0	a	3	S	4

Стек: 

0	a	3	S	4	b	5	
---	---	---	---	---	---	---	--

5. Выполняем shift 3.

Вход: 

a	b	a	<b>b</b>	\$
---	---	---	----------	----

  
 Стек: 

0	a	3	S	4	b	5	a	3	
---	---	---	---	---	---	---	---	---	--

6. Выполняем reduce 1, кладем новое состояние 4 и переходим в него.

Вход: 

a	b	a	<b>b</b>	\$
---	---	---	----------	----

  
 Стек: 

0	a	3	S	4	b	5	a	3	S	4	
---	---	---	---	---	---	---	---	---	---	---	--

7. Выполняем shift 5.

Вход: 

a	b	a	b	<b>\$</b>
---	---	---	---	-----------

  
 Стек: 

0	a	3	S	4	b	5	a	3	S	4	b	5	
---	---	---	---	---	---	---	---	---	---	---	---	---	--

8. Выполняем reduce 1, кладем новое состояние 6 и переходим в него.

Вход: 

a	b	a	b	<b>\$</b>
---	---	---	---	-----------

  
 Стек: 

0	a	3	S	4	b	5	a	3	S	4	b	5	S	6	
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	--

9. Выполняем reduce 0 (снимаем со стека 8 элементов и кладем 'S'), оказываемся в состоянии 5 и делаем переход в новое состояние 6 с добавлением его на стек.

Вход: 

a	b	a	b	<b>\$</b>
---	---	---	---	-----------

  
 Стек: 

0	a	3	S	4	b	5	S	6	
---	---	---	---	---	---	---	---	---	--

10. Снова выполняем reduce 0, оказываемся в состоянии 0 и делаем переход в новое состояние 1 с добавлением его на стек. Заканчиваем разбор.

Вход: 

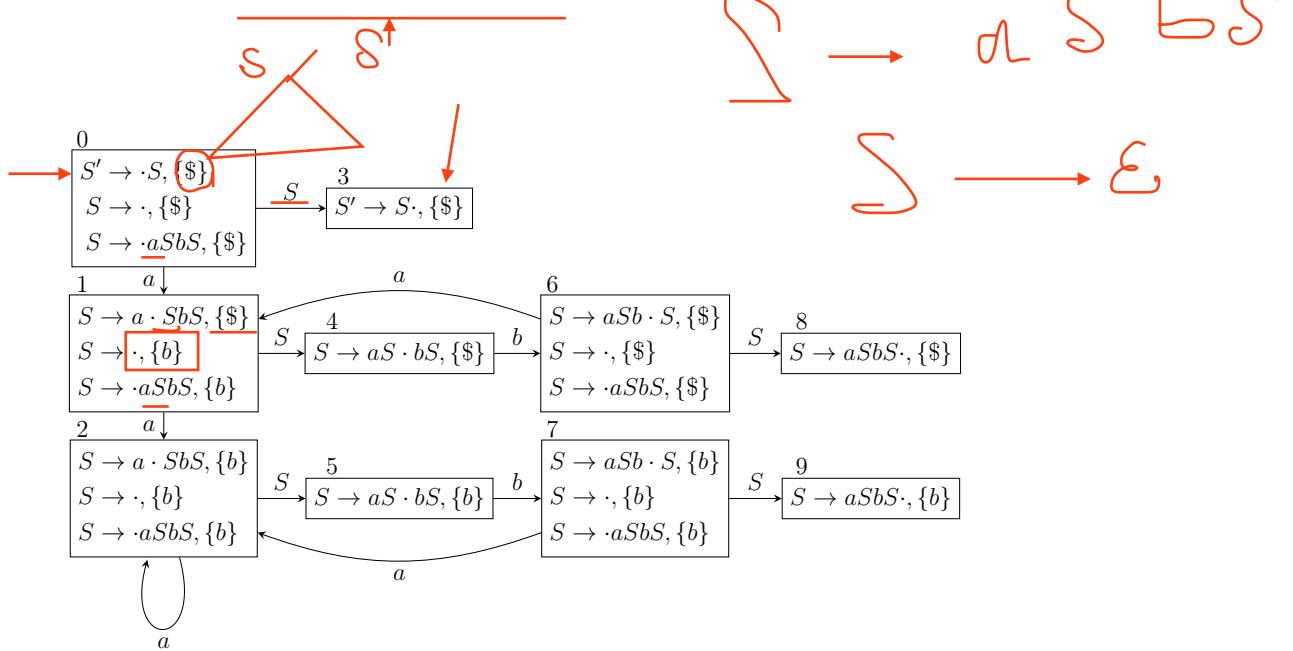
a	b	a	b	<b>\$</b>
---	---	---	---	-----------

  
 Стек: 

0	S	1	
---	---	---	--

□

**Пример 10.1.6.** Пример CLR автомата.



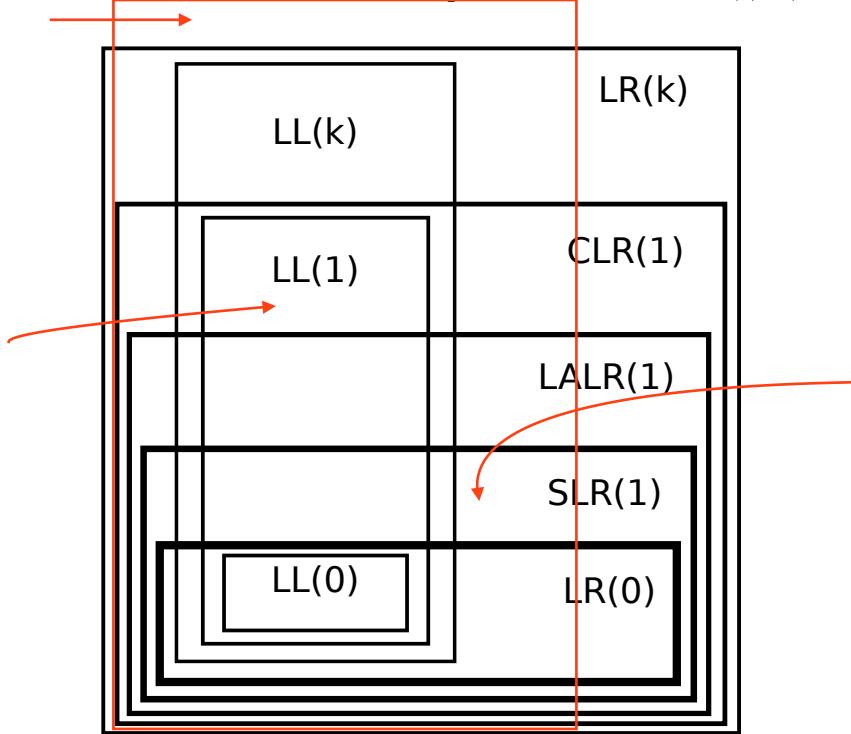
□

Существуют и другие модификации, например **LALR(1)**

На практике конфликты стараются решать ещё и на этапе генерации. Прикладные инструменты могут сгенерировать парсер по неоднозначной грамматике: из переноса или свёртки выбирать перенос, из нескольких свёрток — первую в каком-то порядке (обычно в порядке появления соответствующих продукции в грамматике).

### 10.1.5 Сравнение классов LL и LR

Иерархию языков, распознаваемых различными классами алгоритмов, можно представить следующим образом.



Из диаграммы видно, что класс языков, распознаваемых  $LL(k)$  алгоритмом уже, чем класс языков, распознаваемый  $LR(k)$  алгоритмом, при любом конечном  $k$ . Приведём несколько примеров.

1.  $L = \{a^m b^n c \mid m \geq n \geq 0\}$  является  $LR(0)$ , но для него не существует  $LL(1)$  грамматики.
2.  $L = \{a^n b^n + a^n c^n \mid n > 0\}$  является  $LR$ , но не  $LL$ .
3. Больше примеров можно найти в работе Джона Битти [10].

## 10.2 GLR и его применение для КС запросов

Алгоритм  $LR$  довольно эффективен, однако позволяет работать не со всеми КС-грамматиками, а только с их подмножеством  $LR(k)$ . Если грамматика находится за рамками допускаемого класса, некоторые ячейки управляющей таблицы могут содержать несколько значений. В этом случае грамматика отвергалась анализатором.

Чтобы допустить множественные значения в ячейках управляющей таблицы, потребуется некоторый вид недетерминизма, который даст возможность