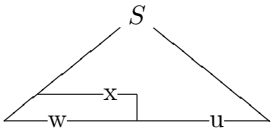
Системное программное обеспечение

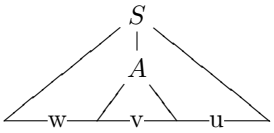
КОНСПЕКТ ЛЕКЦИЙ. ЧАСТЬ 2

LR(k)-грамматики

Если в процессе LR-разбора принять детерминированное решение о сдвиге/свертке удается, рассматривая только цепочку x и первые k символов непросмотренной части входной цепочки u (эти k символов называют аванцепочкой), то говорят, что грамматика обладает LR(k)-свойством.

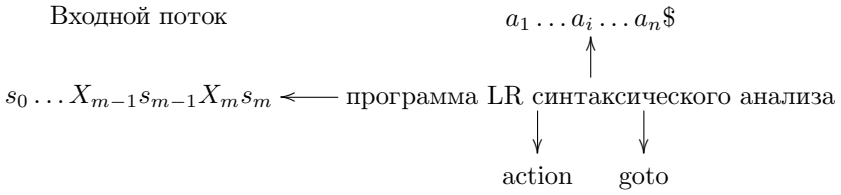


Различие между LL(k)- и LR(k)-грамматиками в терминах дерева вывода.



В случае LL(k)-грамматик однозначно определить правило, примененное к A, можно по w и первым k символам vu, а в случае LR(k)-грамматик — по w, v и первым k символам u. Это показывает, что LL(k)-языки $\subset$ LR(k)-языки.

Схема LR-анализатора



Анализатор состоит из входного потока, стека, управляющей программы, таблиц action и goto. Управляющая программа для всех LR-языков одинакова — изменяются только таблицы. Программа считывает символы из входного потока и использует стек для хранения строк вида $s_0 X_1 s_1 \dots X_m s_m$ (s_m — вершина стека), где X_i — символ грамматики, а s_i — символ состояния, который обобщает информацию, хранящуюся в стеке ниже него. Комбинация символа состояния на вершине стека и текущего входного символа используется как индекс к таблице action и определяет дальнейшее действие — перенос, свертку, допуск или ошибку. Таблица goto в качестве индекса использует комбинацию из символа грамматики и состояния и определяет новое состояние.

Конфигурация LR-анализатора — это пара, первый компонент которой — содержимое стека, второй — непросмотренная часть входной цепочки: $\langle s_0 X_1 s_1 X_2 s_2 \dots X_m s_m,\ a_i a_{i+1} \dots a_n\$\rangle$. Переход в новую конфигурацию осуществляется в зависимости от текущего входного символа a_i и состояния на вершине стека s_m, если:

1) action[s_m,a_i] = “перенос s”, то переход в конфигурацию $\langle s_0 \dots X_m s_m a_i s,\ a_{i+1} \dots a_n\$\rangle$;
2) action[s_m,a_i] = “свертка $A \to b$”, то переход в конфигурацию $\langle s_0 \dots X_{m-r} s_{m-r} A s,\ a_i a_{i+1} \dots a_n\$\rangle$, где s = goto[s_{m-r},A], $r = \#b$;
3) action[s_m,a_i] = “допуск”, то синтаксический разбор успешно завершается;
4) action[s_m,a_i] = “ошибка”, то вызывается подпрограмма восстановления после ошибки.

Таблицы для LR(1) синтаксического анализа для грамматики

(1) $E \to E + T$,
(2) $E \to T$,
(3) $T \to T * F$,
(4) $T \to F$,
(5) $F \to (E)$,
(6) $F \to t$.

Состояние	action						goto		
	t	$+$	$*$	(	)	\$	E	T	F
0	s_5			s_4			1	2	3
1		s_6				acc			
2		r_2	s_7		r_2	r_2			
3		r_4	r_4		r_4	r_4			
4	s_5			s_4			8	2	3
5		r_6	r_6		r_6	r_6			
6	s_5			s_4				9	3
7	s_5			s_4					10
8		s_6			s_{11}				
9		r_1	s_7		r_1	r_1			
10		r_3	r_3		r_3	r_3			
11		r_5	r_5		r_5	r_5			

В ней s_i — означает “перенос i”, r_j — свертку по правилу j, acc — допуск, пустая ячейка — ошибку.
Синтаксический разбор выражения $5 * (7 + 4 * 2)$:

№	Стек	Входной поток	Действие
1	0	$5 * (7 + 4 * 2)\$$	перенос 5
2	$0\ 5\ 5$	$*(7 + 4 * 2)\$$	свертка 6
3	$0\ F\ 3$	$*(7 + 4 * 2)\$$	свертка 4
4	$0\ T\ 2$	$*(7 + 4 * 2)\$$	перенос 7
5	$0\ T\ 2 * 7$	$(7 + 4 * 2)\$$	перенос 4
6	$0\ T\ 2 * 7\ (\ 4$	$7 + 4 * 2)\$$	перенос 5
7	$0\ T\ 2 * 7\ (\ 4\ 7\ 5$	$+4 * 2)\$$	свертка 6
8	$0\ T\ 2 * 7\ (\ 4\ F\ 3$	$+4 * 2)\$$	свертка 4
9	$0\ T\ 2 * 7\ (\ 4\ T\ 2$	$+4 * 2)\$$	свертка 2
10	$0\ T\ 2 * 7\ (\ 4\ E\ 8$	$+4 * 2)\$$	перенос 6
11	$0\ T\ 2 * 7\ (\ 4\ E\ 8 + 6$	$4 * 2)\$$	перенос 5
12	$0\ T\ 2 * 7\ (\ 4\ E\ 8 + 6\ 4\ 5$	$*2)\$$	свертка 6
13	$0\ T\ 2 * 7\ (\ 4\ E\ 8 + 6\ F\ 3$	$*2)\$$	свертка 4
14	$0\ T\ 2 * 7\ (\ 4\ E\ 8 + 6\ T\ 9$	$*2)\$$	перенос 7
15	$0\ T\ 2 * 7\ (\ 4\ E\ 8 + 6\ T\ 9 * 7$	$2)\$$	перенос 5
16	$0\ T\ 2 * 7\ (\ 4\ E\ 8 + 6\ T\ 9 * 7\ 2\ 5$	$)\$$	свертка 6
17	$0\ T\ 2 * 7\ (\ 4\ E\ 8 + 6\ T\ 9 * 7\ F\ 10$	$)\$$	свертка 3
18	$0\ T\ 2 * 7\ (\ 4\ E\ 8 + 6\ T\ 9$	$)\$$	свертка 1
19	$0\ T\ 2 * 7\ (\ 4\ E\ 8$	$)\$$	перенос 11
20	$0\ T\ 2 * 7\ (\ 4\ E\ 8\ )\ 11$	$\$$	свертка 5
21	$0\ T\ 2 * 7\ F\ 10$	$\$$	свертка 3
22	$0\ T\ 2$	$\$$	свертка 2
23	$0\ E\ 1$	$\$$	допуск.

При разборе строки LR(0)-языка можно вообще не использовать аванцепочку — выбор между сдвигом и сверткой делается на основании цепочки x.

На практике LR(k)-грамматики при k > 1 не применяются. На это имеются две причины:

1) очень большое число LR(k) состояний;
2) для любого языка, определяемого LR(k)-грамматикой, существует LR(1)-грамматика.

Для любого детерминированного КС-языка существует LR(1)-грамматика.

Если грамматика не является LR, то при работе анализатора будут возникать конфликты типа перенос/свертка или свертка/свертка.

Число LR(1)-состояний для практически интересных грамматик весьма велико. LR(0) свойством такие грамматики обладают редко. На практике чаще всего используются промежуточные между LR(0) и LR(1) методы, известные под названиями SLR(1) — Simple LR(1) и LALR(1) — Look Ahead LR(1). Языки SLR(1) — подмножество языков LALR(1), которые, в свою очередь, — подмножество LR(1) языков.

Построение таблиц action и goto

Основано на множествах пунктов (items), которое строится подпрограммами closure (замыкание), goto и items.

Множества пунктов будут именоваться как I с индексом, начиная с нуля.

К исходной грамматике G с начальным символом S добавляется правило $S' \to S$ и, следовательно, символ S' в этой новой грамматике G' будет начальным.

```
function closure(I);
begin
  repeat
    for каждый пункт [A → α · Bβ, a] ∈ I,
      каждое правило B → γ из G'
      и каждый терминал b из FIRST(βa) do
        добавить [B → ·γ, b] к I
  until пунктов для добавления в I больше нет;
  return I
end

function goto(I, X);
begin
  Пусть J - множество пунктов [A → αX · β, a],
    таких, что [A → α · Xβ, a] ∈ I;
  return closure(J)
end

procedure items(G');
begin
  I = {closure(S' → ·S, $)} = {I0};
  repeat
    for каждое множество пунктов In ∈ I и каждый символ
      грамматики X, такие, что goto(In, X) не пусто do
        добавить goto(In, X) в I
  until множеств пунктов для добавления в I больше нет
end
```

Вычислим множества пунктов для расширенной грамматики: $S' \to S$, $S \to CC$, $C \to cC|d$.

$I_0 = \text{closure}(S' \to \cdot S, \$)$. Сначала I_0 состоит только из одного пункта $S' \to \cdot S, \$$. Затем вычисляем closure — сопоставляем $A \to \alpha \cdot B\beta$ $S' \to \cdot S$. $\text{FIRST}(\$) = \{\$\}$. Добавляем к I_0 пункт $S \to \cdot CC, \$$. Для этого нового пункта вычисляем $\text{FIRST}(C\$) = \text{FIRST}(C) = \{c, d\}$ и добавляем к I_0 четыре пункта $C \to \cdot cC, c|d$ и $C \to \cdot d, c|d$.

Итак, $I_0 = S' \to \cdot S, \$ \quad S \to \cdot CC, \$ \quad C \to \cdot cC, c|d \quad C \to \cdot d, c|d$.

Далее перебираем все символы грамматики (S, C, c, d) с I_0.

$I_1 = \text{goto}(I_0, S) = \text{closure}(S' \to S\cdot, \$) = S' \to S\cdot, \$$.

$I_2 = \text{goto}(I_0, C) = \text{closure}(S \to C \cdot C, \$) = S \to C \cdot C, \$ \quad C \to \cdot cC, \$ \quad C \to \cdot d, \$$.

$I_3 = \text{goto}(I_0, c) = \text{closure}(C \to c \cdot C, c|d) = C \to c \cdot C, c|d \quad C \to \cdot cC, c|d \quad C \to \cdot d, c|d$.

$I_4 = \text{goto}(I_0, d) = \text{closure}(C \to d\cdot, c|d) = C \to d\cdot, c|d$.

Перебираем все символы грамматики с I_1 — все goto будут здесь пустыми.

$I_5 = \text{goto}(I_2, C) = \text{closure}(S \to CC\cdot, \$) = S \to CC\cdot, \$$.

$I_6 = \text{goto}(I_2, c) = \text{closure}(C \to c \cdot C, \$) = C \to c \cdot C, \$ \quad C \to \cdot cC, \$ \quad C \to \cdot d, \$$.

$I_7 = \text{goto}(I_2, d) = \text{closure}(C \to d\cdot, \$) = C \to d\cdot, \$$.

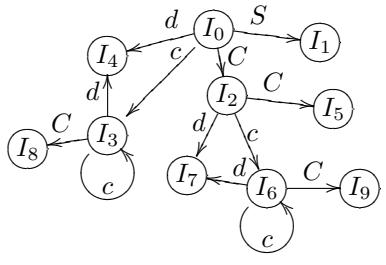
$I_8 = \text{goto}(I_3, C) = \text{closure}(C \to cC\cdot, c|d) = C \to cC\cdot, c|d$.

$\text{goto}(I_3, c) = I_3$, $\text{goto}(I_3, d) = I_4$, $\text{goto}(I_4|I_5|I_7|I_8, X) = \varnothing$, т. е. I_4, I_5, I_7 и I_8 переходов не имеют.

$I_9 = \text{goto}(I_6, C) = \text{closure}(C \to cC\cdot, \$) = C \to cC\cdot, \$$.

$\text{goto}(I_6, c) = I_6$, $\text{goto}(I_6, d) = I_7$.

Множества I являются состояниями детерминированного конечного автомата, который представляется следующей схемой (это общий факт).



По построенным множествам пунктов таблица action заполняется по следующим правилам:

1) если $[A \to \alpha \cdot a\beta, b] \in I_n$ и $\text{goto}(I_n, a) = I_j$ (a — терминал), то $\text{action}[n, a] =$ "перенос j";

2) если $[A \to \alpha\cdot, a] \in I_n$ и $A \neq S'$, то action$[n, a]$ = "свертка $A \to \alpha$";
3) если $[S' \to S\cdot, \$] \in I_n$, то action$[n, \$]$ = "допуск".

Таблица goto однозначно соответствует множествам goto: если $\text{goto}(I_n, A) = I_j$ (A — нетерминал), то $\text{goto}[n, A] = j$.

Все незаполненные таким образом ячейки таблиц action и goto помечаются как "ошибка".

Начальное состояние синтаксического анализатора (s_0) представляет собой состояние, построенное из множества $\text{closure}(S' \to \cdot S, \$) = I_0$.

Построим таблицы для грамматики $S \to CC$ (1), $C \to cC$ (2), $C \to d$ (3).

Состояние	action			goto	
	c	d	\$	S	C
0	s_3	s_4		1	2
1			acc		
2	s_6	s_7			5
3	s_3	s_4			8
4	r_3	r_3			
5			r_1		
6	s_6	s_7			9
7			r_3		
8	r_2	r_2			
9			r_2		

Построенная таким образом соединенная таблица называется *канонической* таблицей LR(1)-анализа. Если в этой таблице нет множественных (т. е. конфликтных) записей, то соответствующая ей грамматика имеет свойство LR(1).

В SLR(1)-таблице для этой грамматики — 7 состояний.

Построим каноническую LR(1)-таблицу для грамматики

$E' \to E$ (0),
$E \to E + T$ (1),
$E \to T$ (2),
$T \to T * F$ (3),
$T \to F$ (4),
$F \to (E)$ (5),
$F \to id$ (6).

Символы этой грамматики — E, T, F, id, $+$, $*$, (,), \$.

$I_0 = \text{closure}(E' \to \cdot E, \$)$:
$E' \to \cdot E, \$$
$E \to \cdot E + T, \$|+$
$E \to \cdot T, \$|+$
$T \to \cdot T * F, \$| + |*$
$T \to \cdot F, \$| + |*$
$F \to \cdot(E), \$| + |*$
$F \to \cdot id, \$| + |*$.

$I_1 = \text{goto}(I_0, E)$:
$E' \to E\cdot, \$$
$E \to E \cdot + T, \$|+$.

$I_2 = \text{goto}(I_0, T)$:
$E \to T\cdot, \$|+$
$T \to T \cdot *F, \$| + |*$

$I_3 = \text{goto}(I_0, F)$:
$T \to F\cdot, \$| + |*$.

$I_4 = \text{goto}(I_0, id)$:
$F \to id\cdot, \$| + |*$.

$I_5 = \text{goto}(I_0, \text{"("})$:
$F \to (\cdot E), \$| + |*$
$E \to \cdot E + T,)|+$
$E \to \cdot T,)|+$
$T \to \cdot T * F,)| + |*$
$T \to \cdot F,)| + |*$
$F \to \cdot(E),)| + |*$
$F \to \cdot id,)| + |*$.

$I_6 = \text{goto}(I_1, +)$:
$E \to E + \cdot T, \$|+$

$T \to \cdot T * F, \$| + |*$
$T \to \cdot F, \$| + |*$
$F \to \cdot (E), \$| + |*$
$F \to \cdot id, \$| + |*.$

$I_7 = \text{goto}(I_2, *)$:
$T \to T * \cdot F, \$| + |*$
$F \to \cdot (E), \$| + |*$
$F \to \cdot id, \$| + |*.$

$I_8 = \text{goto}(I_5, E)$:
$F \to (E \cdot), \$| + |*$
$E \to E \cdot + T,)|+.$

$I_9 = \text{goto}(I_5, T)$:
$E \to T \cdot,)|+$
$T \to T \cdot * F,)| + |*.$

$I_{10} = \text{goto}(I_5, F)$:
$T \to F \cdot,)| + |*.$

$I_{11} = \text{goto}(I_5, id)$:
$F \to id \cdot,)| + |*.$

$I_{12} = \text{goto}(I_5, "(")$:
$F \to (\cdot E),)| + |*$
$E \to \cdot E + T,)|+$
$E \to \cdot T,)|+$
$T \to \cdot T * F,)| + |*$
$T \to \cdot F,)| + |*$
$F \to \cdot (E),)| + |*$
$F \to \cdot id,)| + |*.$

$I_{13} = \text{goto}(I_6, T)$:
$E \to E + T \cdot, \$|+$
$T \to T \cdot * F, \$| + |*.$

$\text{goto}(I_6, F) = I_3.$
$\text{goto}(I_6, id) = \text{goto}(I_7, id) = I_4.$
$\text{goto}(I_6, "(") = \text{goto}(I_7, "(") = I_5.$

$I_{14} = \text{goto}(I_7, F)$:
$T \to T * F \cdot, \$| + |*.$

$I_{15} = \text{goto}(I_8, ")")$:
$F \to (E) \cdot, \$| + |*.$

$I_{16} = \text{goto}(I_8, +)$:
$E \to E + \cdot T,)|+.$
$T \to \cdot T * F,)| + |*$
$T \to \cdot F,)| + |*$
$F \to \cdot (E),)| + |*$
$F \to \cdot id,)| + |*.$

$I_{17} = \text{goto}(I_9, *)$:
$T \to T * \cdot F,)| + |*.$
$F \to \cdot (E),)| + |*$
$F \to \cdot id,)| + |*.$

$I_{18} = \text{goto}(I_{12}, E)$:
$F \to (E \cdot),)| + |*$
$E \to E \cdot + T,)|+.$

$\text{goto}(I_{12}, T) = I_9.$
$\text{goto}(I_{12}, F) = \text{goto}(I_{16}, F) = I_{10}.$
$\text{goto}(I_{12}, id) = \text{goto}(I_{16}, id) = \text{goto}(I_{17}, id) = I_{11}.$
$\text{goto}(I_{12}, "(") = \text{goto}(I_{16}, "(") = \text{goto}(I_{17}, "(") = I_{12}.$
$\text{goto}(I_{13}, *) = I_7.$

$I_{19} = \text{goto}(I_{16}, T)$:
$E \to E + T \cdot,)|+.$
$T \to T \cdot * F,)| + |*.$

$I_{20} = \text{goto}(I_{17}, F)$:
$T \to T * F \cdot,)| + |*.$

$\text{goto}(I_{18}, +) = I_{16}.$

$I_{21} = \text{goto}(I_{18}, ")")$:
$F \to (E) \cdot,)| + |*.$

$\text{goto}(I_{19}, *) = I_{17}$.
Каноническая LR(1)-таблица будет иметь следующий вид.

Состояние	action						goto		
	id	+	*	(	)	$	*E*	*T*	*F*
0	s_4			s_5			1	2	3
1		s_6				*acc*			
2		r_2	s_7			r_2			
3		r_4	r_4			r_4			
4		r_6	r_6			r_6			
5	s_{11}			s_{12}			8	9	10
6	s_4			s_5				13	3
7	s_4			s_5					14
8		s_{16}			s_{15}				
9		r_2	s_{17}		r_2				
10		r_4	r_4		r_4				
11		r_6	r_6		r_6				
12	s_{11}			s_{12}			18	9	10
13		r_1	s_7			r_1			
14		r_3	r_3			r_3			
15		r_5	r_5			r_5			
16	s_{11}			s_{12}				19	10
17	s_{11}			s_{12}					20
18		s_{16}			s_{21}				
19		r_1	s_{17}		r_1				
20		r_3	r_3		r_3				
21		r_5	r_5		r_5				

В LALR(1) или SLR(1) таблицах, использованных ранее, для этой грамматики всего 12 состояний (число состояний в SLR(1) и LALR(1) таблицах всегда одинаково).

В SLR(1) используется только левая часть пункта, а правая (для сверток $A \to \alpha$) заменяется на FOLLOW(A). В LALR(1) множества пунктов, отличающихся только правой частью объединяются, — это не может приводить к конфликтам типа сдвиг/свертка, но может порождать конфликты типа свертка/свертка. В приведенной таблице объединяются состояния 2–9, 3–10, 4–11, 5–12, 6–16, 7–17, 8–18, 13–19, 14–20, 15–21. Существует методы быстрого построения LALR(1) таблиц, не требующие построения всех LR(1) пунктов.

Программа YACC — Yet Another Compiler Compiler (в Linux используется также BISON, совместимый с YACC) предназначена для построения синтаксического анализатора детерминированного контекстно-свободного языка. Анализируемый язык описывается с помощью грамматики в виде, близком форме Бэкуса-Наура (НФБН). Результатом работы YACC'а является программа на Си, реализующая восходящий LALR(1) распознаватель.

Неоднозначная грамматика не может быть LR-грамматикой. Классический пример, демонстрирующий это связан с конструкцией "кочующего else":

$S \to \text{if}E\text{then}S|\text{if}E\text{then}S\text{else}S$.

Если LR-анализатор находится в состоянии $\langle \ldots \text{if}E\text{then}S,\ \text{else} \ldots \$\rangle$, то невозможно сделать детерминированный выбор между сдвигом и сверткой.

YACC, при отсутствии других явных указаний, разрешает конфликты между сдвигом и сверткой в пользу сдвига (что снимает проблему условного оператора if), а в конфликтах между свертками он выбирает свертку, меньшую по порядковому номеру.