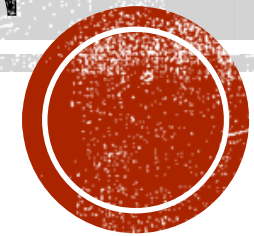
ЯЗЫКИ ПРОГРАММИРОВАНИЯ И МЕТОДЫ ТРАНСЛЯЦИИ



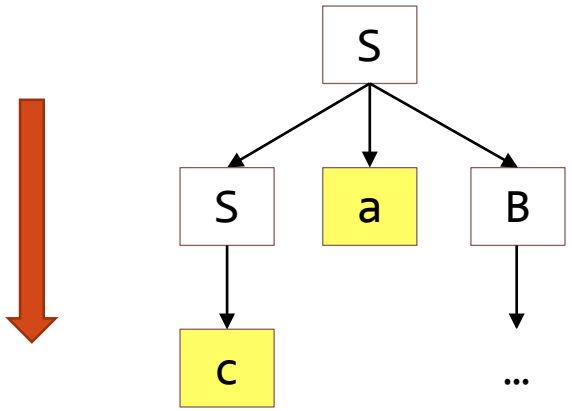
Лекции 12-13

4 мая 2018 г.

СИНТАКСИЧЕСКИЙ АНАЛИЗ

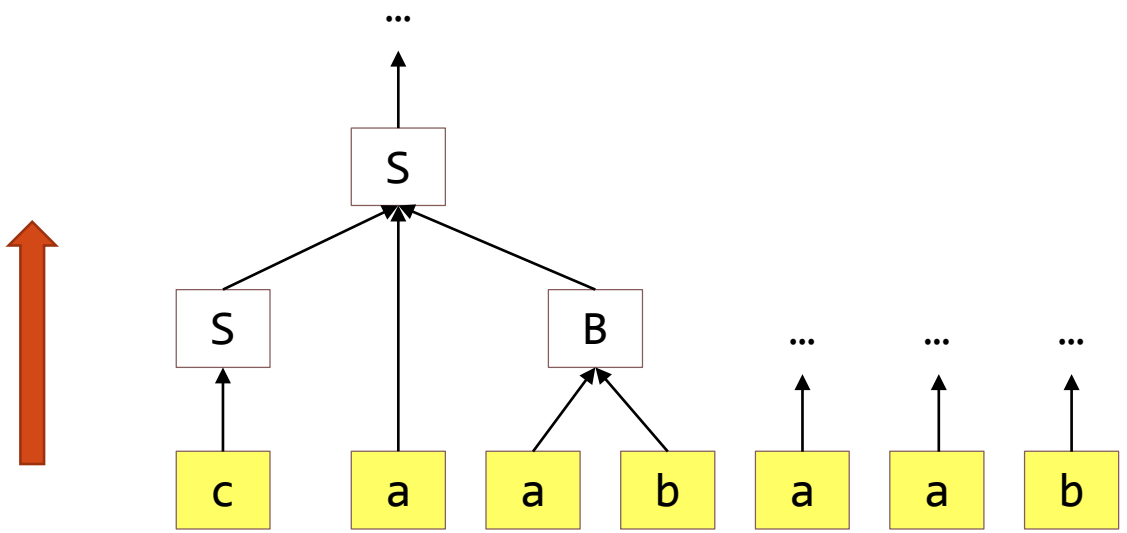
```
S -> S a B
S -> c
B -> a b
```

нисходящий



	СТЕК	ВХОД
нач.	▼S	w$
кон.	▼	$

восходящий



	СТЕК	ВХОД
нач.	▼	w$
кон.	▼S	$

ОБОСНОВАНИЕ ИСПОЛЬЗОВАНИЯ

- LR-анализаторы могут быть созданы для распознавания практически всех конструкций языков программирования, для которых может быть написана контекстно-свободная грамматика.
- Класс грамматик, которые могут быть проанализированы с использованием LR-методов, представляет собой надмножество класса грамматик, которые могут быть проанализированы с использованием предиктивных или LL-методов синтаксического анализа.

Основной недостаток: построение LR-анализатора для грамматики типичного языка программирования вручную требует очень большого объема работы.

ТИПЫ LR-АНАЛИЗАТОРОВ

- **SLR (Simple)** – анализаторы (самый простой, таблица небольшая)
- **Канонические LR** анализаторы (самый сложный, таблица большая)
- **LALR (Look Ahead)** – анализаторы с предпросмотром (средней сложности, таблица небольшая)

ВОСХОДЯЩИЙ СИНТАКСИЧЕСКИЙ АНАЛИЗ

Опр. 1: *ПС-анализ* – разновидность восходящего анализа, основывающаяся на двух операциях – Переносе и Свертке, в котором для хранения символов грамматики используется стек, а для остающейся части входной строки – входная лента.

Опр. 2: *Перенос* – процесс переноса символа с входной ленты на стек.

Опр. 3: *Свертка* – замена символов на вершине стека одним нетерминалом в соответствии с каким-либо правилом грамматики.

ВОСХОДЯЩИЙ СИНТАКСИЧЕСКИЙ АНАЛИЗ

```
S -> S a B
S -> c
B -> a b
```

```
S => S a B =>
=> S a a b =>
=> S a B a a b =>
=> S a a b a a b =>
=> c a a b a a b
```

Правый вывод, поэтому **LR**-анализ

Опр. 4: Последовательность «сворачиваемых» символов грамматики называется *основой.*

ПРИМЕР

1. S -> S a B
2. S -> c
3. B -> a b

СТЕК	ЛЕНТА	ДЕЙСТВИЕ
▼	caabaab$	сдвиг
▼c	aabaab$	свертка по правилу 2
▼S	aabaab$	сдвиг
▼Sa	abaab$	сдвиг
▼Saa	baab$	сдвиг
▼Saab	aab$	свертка по правилу 3
▼SaB	aab$	свертка по правилу 1
▼S	aab$	сдвиг
▼Sa	ab$	сдвиг
▼Saa	b$	сдвиг
▼Saab	$	свертка по правилу 3
▼SaB	$	свертка по правилу 1
▼S	$	принять

ОШИБКИ ВОСХОДЯЩЕГО ПС-РАЗБОРА

```
S -> S a B
S -> a
B -> a b
```

```
▼      aaab$
▼a     aab$
▼S     aab$
▼Sa    ab$
▼SS    ab$ свертка?
или
▼Saa   b$  сдвиг?
```

Опр. 5: Конфликт «сдвиг/свертка» (**shift/reduce**) – когда анализатор не может выбрать, сдвигать или сворачивать.

```
S -> S A
S -> a
A -> a
```

```
▼      aaab$
▼a     aab$
▼S     aab$ свертка 2?
или
▼A     aab$ свертка 3?
```

Опр. 6: Конфликт «свертка/свертка» (**reduce/reduce**) – когда анализатор не может выбрать, по какому правилу сворачивать.

УПРАЖНЕНИЯ

Упражнение 1
Для заданной грамматики и каждой конфигурации стека определите основу:

```
1. S -> S A b
2. S -> a c b
3. A -> b B c
4. A -> b c
5. B -> b a
6. B -> A c
```

(a) ▼SSAb

(b) ▼SSbbc

(c) ▼SbBc

(d) ▼Sbbc

УПРАЖНЕНИЯ

Упражнение 2

Для заданной грамматики и слова выпишите последовательность состояний стека и ленты во время ПС-разбора:
`1. S -> S a B` Слово: acbbcb
`2. S -> c`
`3. B -> a b`

УПРАЖНЕНИЯ

Упражнение 3

Для заданной грамматики и каждого слова определите, возникнет ли конфликт во время разбора и если да, то какого типа:

```
1. S -> S a b
2. S -> b A
3. A -> b b
4. A -> b A
5. A -> b b c
6. A -> c
```

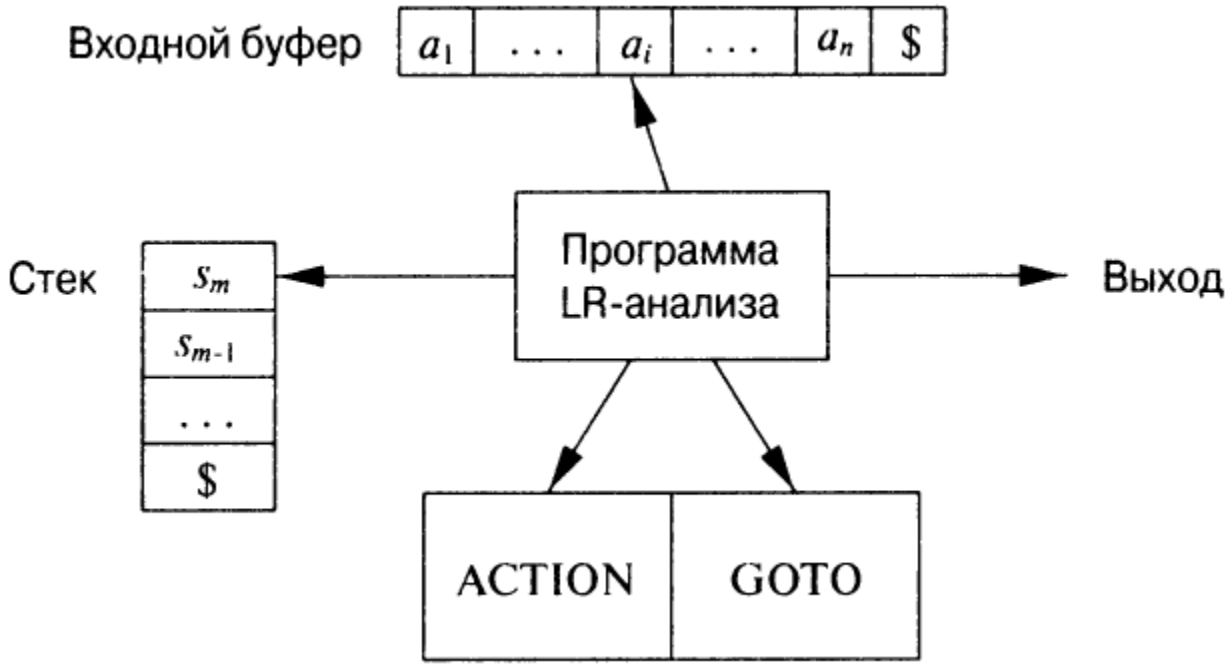
```
(a) bc
(b) bbcab
(c) bacb
```

ТАБЛИЧНЫЙ ПС-АНАЛИЗ

Опр. 7: *Табличный ПС-анализатор* – анализатор, работа которого управляется двумя таблицами: таблицей действий (**Action table**) и переходов (**Goto table**).

Опр. 8: *Таблица действий* определяет, нужно ли производить Сдвиг или Свертку, а если Свертку, то по какому правилу.

Опр. 9: Таблица переходов определяет, какой символ необходимо положить на стек после свертки.



АЛГОРИТМ РАБОТЫ ТАБЛИЧНОГО ПС-АНАЛИЗА

В начале – на стеке ▼, на ленте – слово w, заканчивающееся $

1. **В таблице Действий (Action)**, используя символ на ленте для столбца и символ на стеке для строки, находим ячейку с действием.
2. Если действие – это сдвиг S, то
   a) Кладем символ с ленты на стек и сдвигаем указатель на ленте на один символ
3. Если действие – это свертка Rn, то
   a) Находим Правило №n грамматики, по которому производится свертка
   b) Символы в правой части этого Правила должны также лежать на вершине стека – снимаем их все со стека (если это не так – ошибка)
   c) В таблице Переходов (Goto), используя нетерминал в левой части Правила для столбца и символ, лежащий сейчас на вершине стека, для строки, находим ячейку с нетерминальным символов. Кладем его на стек.
4. Если это действие – пустое, то выдаем сообщение об ошибке
5. Если это действие – Accept, то завершаем работу с успехом
6. Идем на шаг 1 и повторяем все заново.

1. Expr -> Expr + Term
2. Expr -> Term
3. Term -> Term * Factor
4. Term -> Factor
5. Factor -> (Expr)
6. Factor -> var

Goto Table

	Expr	Term	Factor
▼	Expr1	Term2	Factor4
Expr1			
Term1			
Factor3			
(	Expr5	Term2	Factor4
Expr5			
)			
+		Term1	Factor4
Term2			
*			Factor3
Factor4			
var			

Action Table

	+	*	(	)	var	$
▼			S		S	
Expr1	S					ACC
Term1	R1	S		R1		R1
Factor3	R3	R3		R3		R3
(			S		S	
Expr5	S			S		
)	R5	R5		R5		R5
+			S		S	
Term2	R2	S		R2		R2
*			S		S	
Factor4	R4	R4		R4		R4
var	R6	R6		R6		R6

УПРАЖНЕНИЯ

Упражнение 4
Для табличного анализатора с предыдущего слайда и для выражения **var + var * var** *нарисуйте последовательность состояний стека и входной ленты с указанием выбираемых на каждом шаге действий и переходов.*

LR(0)-СИТУАЦИИ И LR(0)-АВТОМАТ

Опр. 10: *LR(0)-ситуация (пункт, состояние)* грамматики **G** – это правило **G** с точкой в некоторой позиции правой части.

Пример 1:
Правило `A -> XYZ` дает четыре LR(0)-ситуации:

```
A -> ·XYZ
A -> X·YZ
A -> XY·Z
A -> XYZ·
```

Опр. 11: *Канонический набор множеств (КНМ)* **LR(0)**-ситуаций – это набор, строящийся по расширенной грамматике **G'** с использованием операций **CLOSURE** и **GOTO**.

Опр. 12: Для грамматики **G** со стартовым символом **S** *расширенная грамматика G'* представляет собой **G** с новым стартовым символом **S'** и продукцией **S' -> S**.

АЛГОРИТМ CLOSURE

Пусть I – множество ситуаций грамматики G.

1. Изначально в CLOSURE(I) добавляются все ситуации из I
2. Если [A -> α ·B β] входит в CLOSURE(I), а B -> γ является правилом G, то в CLOSURE(I) добавляется ситуация [B -> ·γ], если ее еще там нет
3. Повторяем 2 до тех пор, пока не останется ситуаций, которые можно добавить в CLOSURE(I)

Пример 2:

E′ -> E
E -> E + T | T
T -> T * F | F
F -> (E) | id

Пусть **I = {[E' -> ·E]}**, тогда **CLOSURE(I)** =
{ [E' -> ·E], [E -> ·E + T],
[E -> ·T], [T -> ·T * F],
[T -> ·F], [F -> ·(E)], [F -> ·id] }

Алгоритм GOTO

Опр. 13: Пусть I – множество ситуаций грамматики G, а X – символ грамматики G. Тогда GOTO(I, X) определяется как замыкание всех ситуаций [A -> α X · β], таких, что [A -> α ·X β] находится в I.

Пример 3:

Пусть I = {[E' -> E·], [E -> E · + T] }. Тогда GOTO(I,+) содержит ситуации:

```
{ [E -> E + ·T],
  [T -> ·T * F],
  [T -> ·F],
  [F -> ·( E )],
  [F -> ·id] }
```

Алгоритм построения КНМ LR(0)-ситуаций

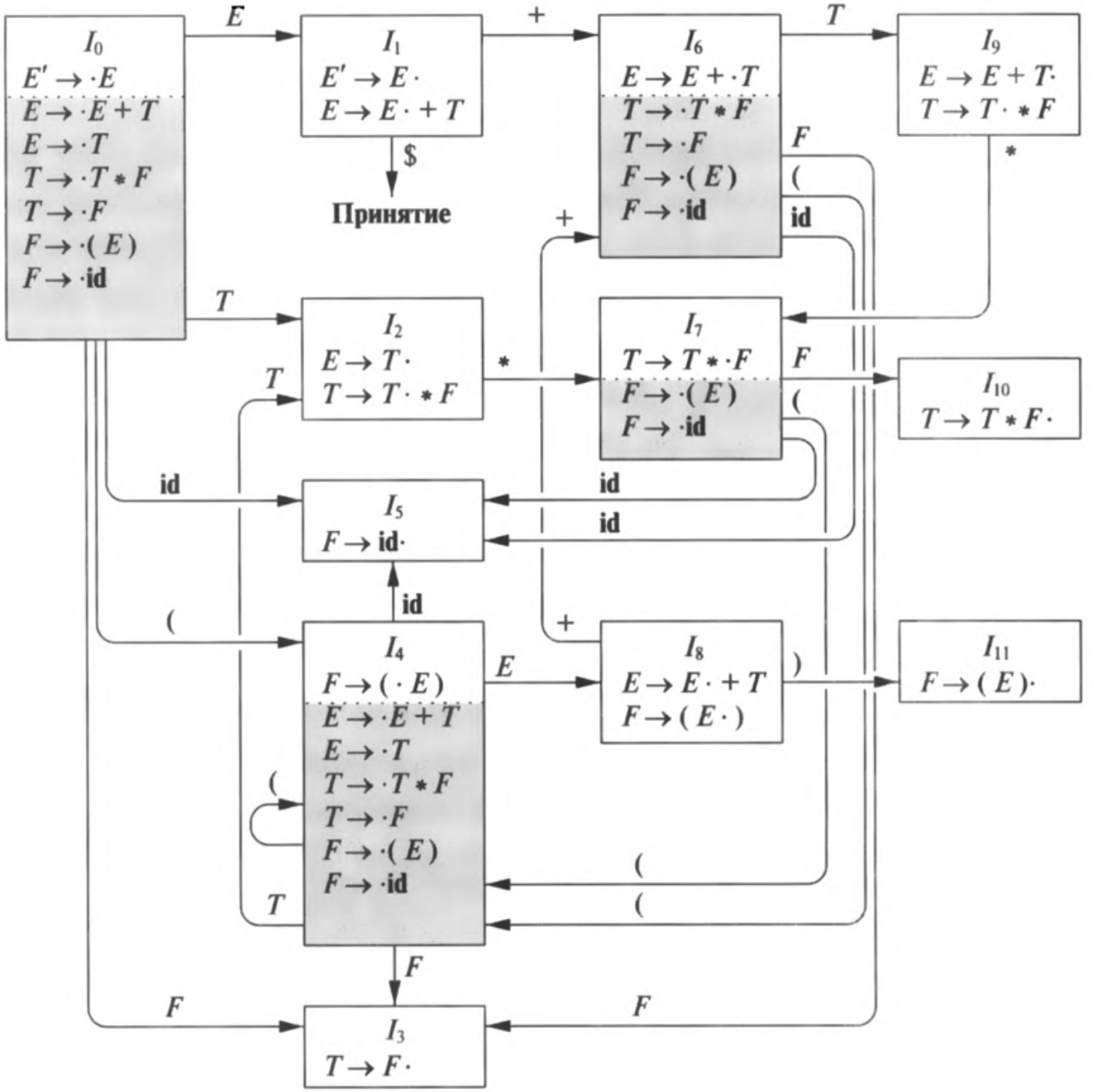
```
C = CLOSURE({[S' -> ·S]})
повторять пока нет новых множеств пунктов для добавления в C
    для каждого множества пунктов I в C
        для каждого символа X грамматики G
            если множество GOTO(I, X) не пустое и не входит в C
                добавить GOTO(I, X) в C
```

ПРИМЕР

Пример 4:

```
E’ -> E
E -> E + T | T
T -> T * F | F
F -> ( E ) | id
```

Опр. 14: КНМ вместе с функцией **GOTO** образуют **LR(0)**-*автомат*. Состояниями этого автомата являются множества **LR(0)**-ситуаций из канонического набора, а переходы определяются функцией **GOTO**.



УПРАЖНЕНИЯ

Упражнение 5

*Постройте КНМ **LR(0)**-ситуаций для следующей грамматики:*
 	
`S -> L = R | R`
`L -> * R | id`
`R -> L`

SLR-АНАЛИЗ

Опр. 15: SLR-анализатор использует LR(0)-автомат для заданной грамматики G для построения таблиц Действий и Переходов.

Пример 5:

1) E′ -> E
2) E -> E + T
3) E -> T
4) T -> T * F
5) T -> F
6) F -> (E)
7) F -> id

Строка: id * id $

Стек		Вход	Действие
0	$	id * id $	S5
0 5	$ id	* id $	R7
0 3	$ F	* id $	R5
0 2	$ T	* id $	S7
0 2 7	$ T *	id $	S5
0 2 7 5	$ T * id	$	R7
0 2 7 10	$ T * F	$	R4
0 2	$ T	$	R3
0 1	$ E	$	Accept

Алгоритм построения SLR-таблицы

1. Строим C = {I_0, I_1, …, I_n} – КНМ LR(0)-пунктов
2. Из I_i строим состояние i, для которого определяем действия:

   а) Если $[A \to \alpha \cdot a\beta] \in I_i$ и GOTO(I_i,a) = I_j, то ACTION[i, a] = «Sj». Здесь a должно быть терминалом.

   б) Если $[A \to \alpha \cdot] \in I_i$, то ACTION[i, a] = «Rn», где n – номер правила A -> α в грамматике G, для всех терминалов a из FOLLOW(A). Здесь A не должно быть равно S'

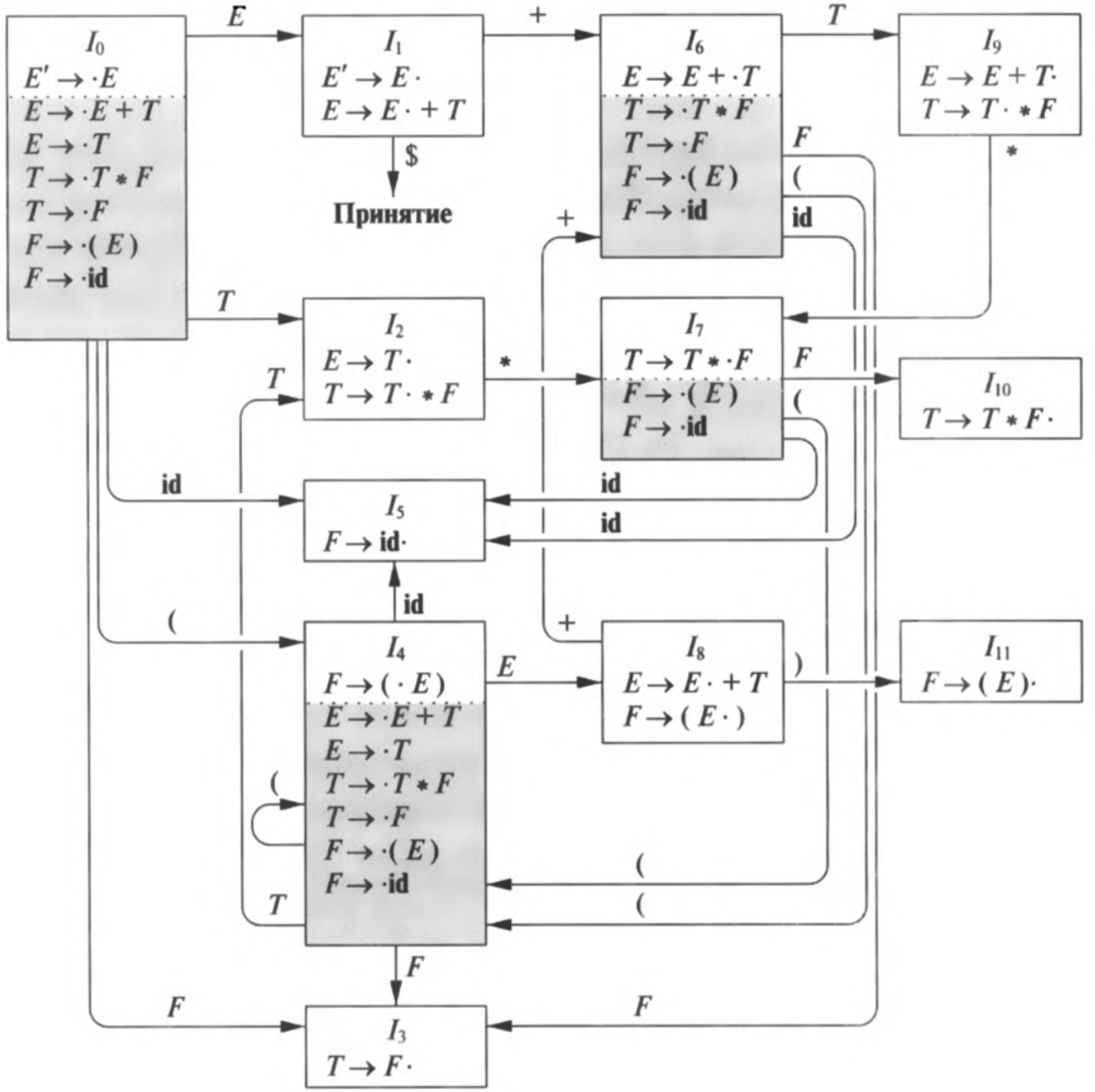
   в) Если $[S' \to S \cdot] \in I_i$, ACTION[i, $] = «Accept»

3. Переходы для состояния i строим для всех нетерминалов A: если GOTO(I_i,A)=I_j, то GOTO[i, A] = j
4. Все записи, не определенные правилами 2 и 3 – ошибочные.
5. Начальное состояние содержит ситуацию [S' -> · S]

ПРИМЕР

Пример 4:

```
E’ -> E
E -> E + T | T
T -> T * F | F
F -> ( E ) | id
```



УПРАЖНЕНИЯ

Упражнение 6
*Построить **SLR**-таблицу для грамматики из примера 4.*

Упражнение 7
*Для полученной в упр. 4 **SLR**-таблицы и строки **(id + id) * id** написать последовательность состояний стека и входной ленты, а также применяемые **SLR**-автоматом действия.*