Тема №12. Синтаксический анализ. Часть 4

- Главные рассматриваемые вопросы:
  - Канонический LR(1)-анализ
  - LALR(1)-анализ
  - Восстановление после ошибок при LR-анализе
  - Использование неоднозначных грамматики при LR-анализе
  - LR(*k*)-грамматики

Канонический LR(1)-анализ

- Можно сохранять в состоянии больший объем информации
- Он позволит отбрасывать такие некорректные свертки
- Дополнительная информация вносится в состояние переопределением ситуаций, чтобы они включали в качестве второго компонента терминальный символ
- Общий вид ситуации - [$A \to \alpha \bullet \beta, a$], где $A \to \alpha\beta$ – продукция, a – терминал или маркер конца строки $
- Это **LR(1)-ситуация**, или **LR(1)-пункт**
- В скобках указана длина второго компонента ситуации - **предпросмотра** ситуации
- Предпросмотр не влияет на ситуацию [$A \to \alpha \bullet \beta, a$], где $\beta \neq \varepsilon$, но ситуация [$A \to \alpha \bullet, a$] приводит к свертке по продукции $A \to \alpha$, если входной символ равен a

Канонический LR(1)-анализ

Свертка в соответствии с продукцией $A \to \alpha$ применяется только при входном символе a, для которого $[A \to \alpha \bullet, a]$ является LR(1)-ситуацией из состояния на вершине стека. Множество таких a всегда является подмножеством $FOLLOW(A)$, но может быть истинным подмножеством, как в предыдущем примере.

Считаем, что LR(1)-ситуация $[A \to \alpha \bullet \beta, a]$ **допустима** (*valid*) для активного префикса γ, если существует порождение $S \underset{rm}{\overset{*}{\Rightarrow}} \delta A w \underset{rm}{\Rightarrow} \delta\alpha\beta w$, где

1) $\gamma = \delta\alpha$;

2) либо a является первым символом w, либо $w = \varepsilon$, а a = \$.

В грамматике

$S \to BB$

$B \to aB \mid b$

Существует правое порождение $S \underset{rm}{\overset{*}{\Rightarrow}} aaBab \underset{rm}{\Rightarrow} aaaBab$. Мы видим, что ситуация $[B \to a \bullet b, a]$ *допустима* для активного префикса $\gamma = aaa$, если в приведенном выше определении $\delta = aa$, $A = B$, $w = ab$, $\alpha = a$ и $\beta = B$.

Существует также правое порождение $S \underset{rm}{\overset{*}{\Rightarrow}} BaB \underset{rm}{\Rightarrow} BaaB$. Из него видно, что ситуация $[B \to a \bullet B, \$]$ является допустимой для активного префикса Baa.

Канонический LR(1)-анализ

Чтобы разобраться в новом определении операции *CLOSURE* (в частности, почему *b* должно быть в *FIRST* (*β a*)), рассмотрим ситуацию вида [*A* -> *α* • *Bβ*, *a*] в множестве ситуаций, допустимых для некоторого активного префикса *γ*. Тогда существует правое порождение $S \underset{rm}{\overset{*}{\Rightarrow}} \delta Aax \underset{rm}{\Rightarrow} \delta\alpha B\beta ax$, где *γ* = δ *α*. Предположим, что *βax* порождает строку терминалов *βy*. Тогда для каждой продукции вида *B* -> *η* для некоторого *η* мы имеем порождение $S \underset{rm}{\overset{*}{\Rightarrow}} \gamma Bby \underset{rm}{\Rightarrow} \gamma\eta by$. Таким образом, [*B* -> • *Bη*, *b*] является допустимой для *γ*. При этом, что *b* может быть первым терминалом, порожденным из *β*, либо, когда *β* порождает *ε* в порождении $\beta ax \underset{rm}{\overset{*}{\Rightarrow}} by$, то *b* может представлять собой *a*. Подытоживая обе возможности, мы говорим, что *b* может быть любым терминалом в *FIRST*(*βax*), где FIRST – процедура вычисления множества первых порождаемых символов. Заметим, что *x* не может содержать первый терминал из *by*, так что *FIRST*(*βax*) = FIRST(*βa*).

Канонический LR(1)-анализ

- Алгоритм построения множеств LR(1)-ситуаций
  - ВХОД: расширенная грамматика *G'*
  - ВЫХОД: множества LR(1)-ситуаций, которые представляют собой множество ситуаций, допустимых для одного или нескольких активных префиксов *G'*

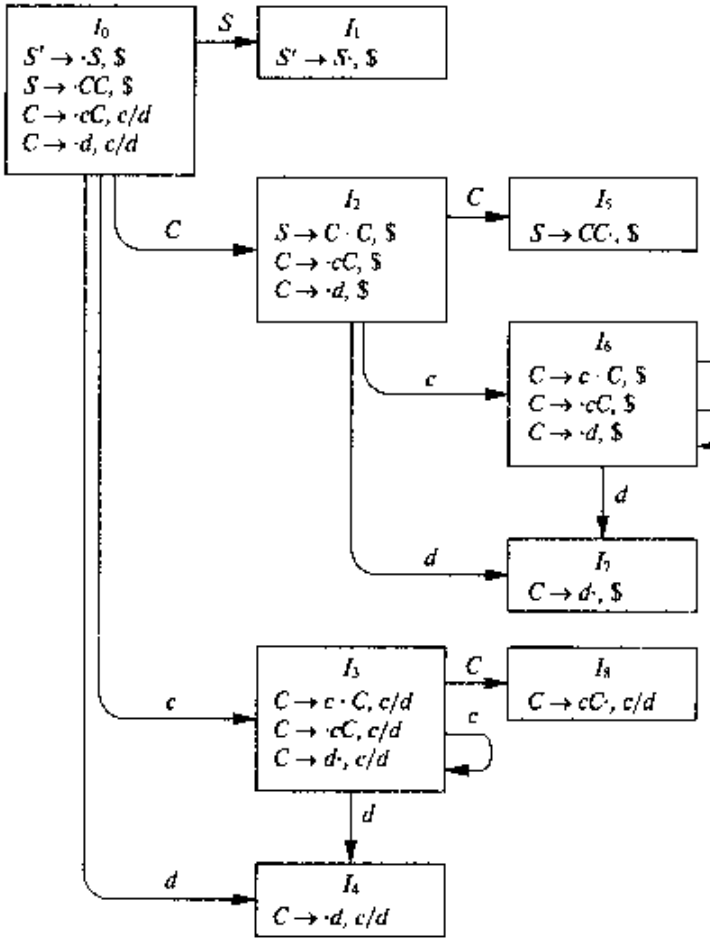
```
SetOfItems CLOSURE(I)
{
  repeat
    foreach(ситуация [A —> α • Bβ, a] из J)
      foreach(продукция B —> γ из G')
        foreach(терминал b € FIRST(βα))
          Добавить [B —> • γ, b] в множество I;
  until нет больше ситуаций для добавления в I;
  return J;
}

SetOfItems GOTO(I, X)
{
  Инициализировать J пустым множеством;
    foreach(ситуация [A -> α • Xβ, a] из I)
      Добавить ситуацию [A —> αX • β, a] во множество J;
  return CLOSURE(J);
}

void items(G')
{
  Инициализировать C множеством {CLOSURE({[S' -> • S,$]})};
  repeat
    foreach(множество ситуаций I в C)
      foreach(каждый грамматический символ X)
      if(GOTO(I, X) не пустое множество и не входит в C)
        Добавить GOTO(I, X) в C;
  until нет новых множеств ситуаций для добавления в C;
}
```

Канонический LR(1)-анализ

$$
\begin{aligned}
S' &\rightarrow S \\
S &\rightarrow C\ C \\
C &\rightarrow c\ C \mid d
\end{aligned}
$$




```
SetOfItems CLOSURE(I)
{
  repeat
    foreach(ситуация [A -> α • Bβ, a] из J)
      foreach(продукция B -> γ из G')
        foreach(терминал b ∈ FIRST(βα))
          Добавить [B -> • γ, b] в множество I;
  until нет больше ситуаций для добавления в I;
  return J;
}
```

```
SetOfItems GOTO(I, X)
{
  Инициализировать J пустым множеством;
    foreach(ситуация [A -> α • Xβ, a] из I)
      Добавить ситуацию [A -> αX • β, a] во множество J;
  return CLOSURE(J);
}
```

Канонический LR(1)-анализ

- Алгоритм построения таблиц LR(1)-анализа
  - ВХОД: расширенная грамматика *G'*
  - ВЫХОД: таблица с функциями *ACTION* и *GOTO* для *G'*
  - МЕТОД: выполнить следующие действия
    1. Построить $C' = \{I_0, I_1 \ldots, I_n\}$ – набор множеств LR(1)-ситуаций для *G'*
    2. Состояние синтаксического анализатора строится из I_i. Действие СА для состояния *i* определяется следующим образом
       - а) Если [$A \to \alpha \bullet a\beta, b$] входит в I_i и GOTO(I_i, A) = I_j, то ACTION[i, a] = «перенос j»
       - б) Если [$A \to \alpha \bullet, a$] входит в I_i и $A \neq S'$, то ACTION[i, a] = «свертка A -> α»
       - в) Если [$S' \to S \bullet$, \$] входит в I_i, то ACTION[i, \$] = «принятие»

       Если при применении указанных правил **обнаруживаются конфликтующие** действия, грамматика **не принадлежит** классу LR(1)
    3. Переходы для состояния *i* строятся для всех нетерминалов *A* с использованием следующего правила: если GOTO(I_i, A) = I_j, то GOTO[i, A] = j
    4. Все записи, не определенные (2) и (3), считаются записями «ошибка»
    5. Начальное состояние анализатора – состояние, построенное из множества ситуаций, содержащего [$S' \to S \bullet$, \$]

Канонический LR(1)-анализ

$$S \rightarrow C\ C$$
$$C \rightarrow c\ C \mid d$$

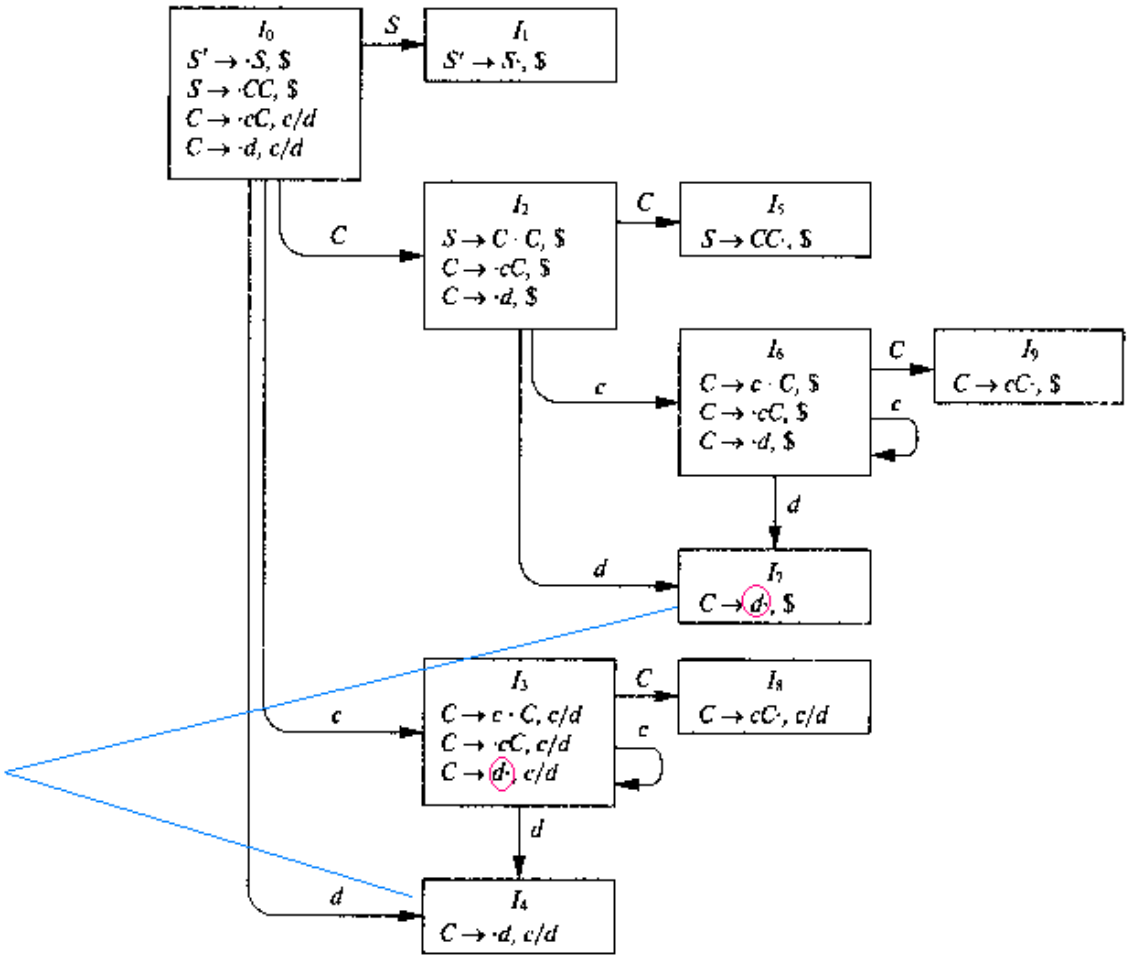
Состояние	ACTION			GOTO	
	c	d	\$	S	C
0	s3	s4		1	2
1			acc		
2	s6	s7			5
3	s3	s4			8
4	r3	r3			
5			r1		
6	s6	s7			9
7			r3		
8	r2	r2			
9			r2		

LALR(1)-анализ

- Часто используется на практике
- В ТСА значительно меньше состояний, чем у канонического LR(1)-метода
- Примерно столько же состояний, что и у SLR(1)
- Достаточно мощен для покрытия большинства конструкций практически используемых языков программирования

LALR(1)-анализ

$$
\begin{array}{lcl}
S' & \rightarrow & S \\
S & \rightarrow & C\ C \\
C & \rightarrow & c\ C \mid d
\end{array}
$$



Одинаковые первые компоненты

ЯДРО C -> d •

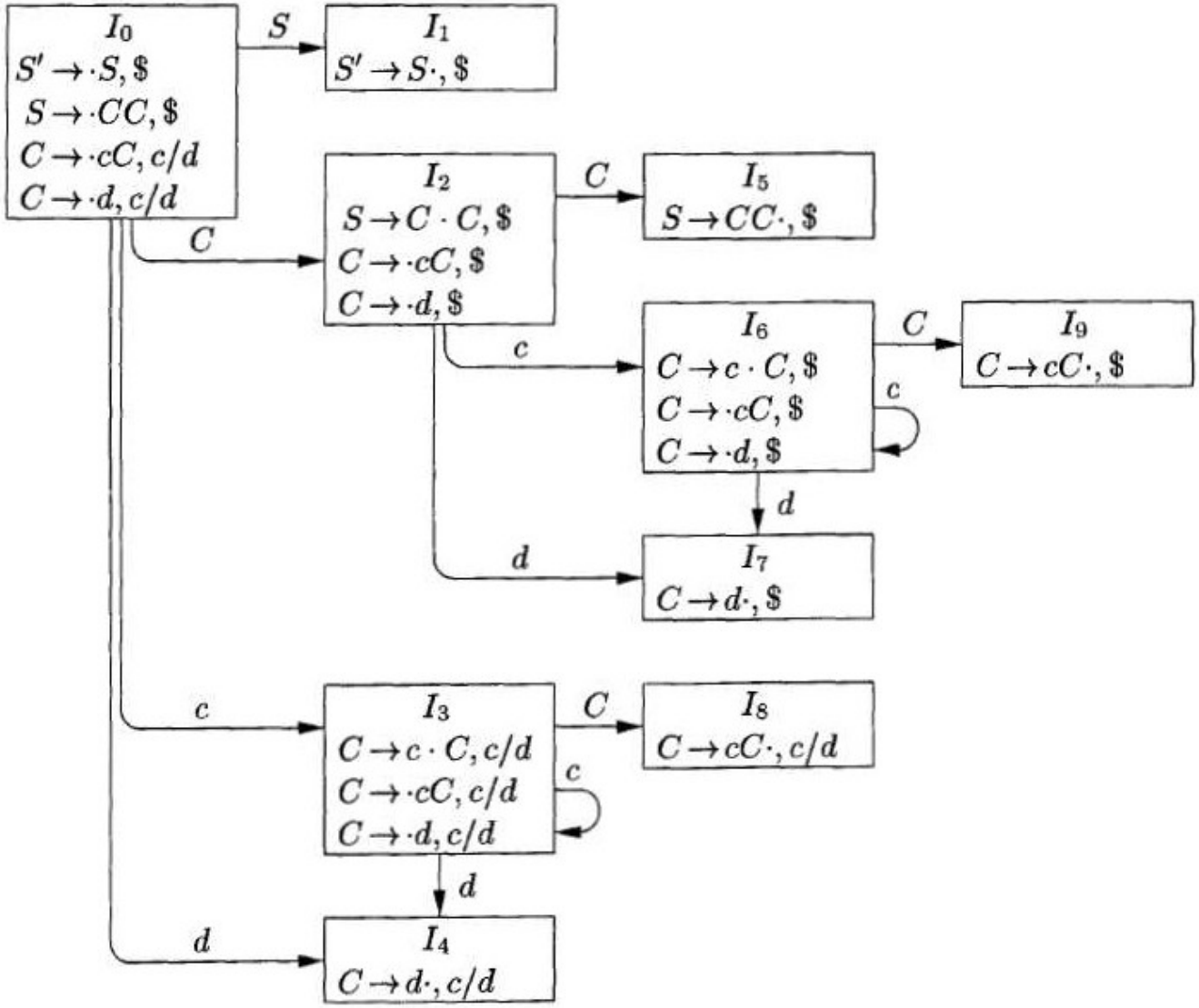
LALR(1)-анализ

- («Затратное») построение LALR(1)-таблицы
  - ВХОД: расширенная грамматика *G*'
  - ВЫХОД: таблица с функциями *ACTION* и *GOTO* для *G*'
  - МЕТОД: выполнить следующие действия
    1. Построить $C = \{I_0, I_1 ..., I_n\}$ – набор множеств LR(1)-ситуаций для *G*'
    2. Для каждого среди множества LR(1)-ситуаций, находим все множества с одним ядром и заменяем множества их объединением
    3. Пусть $C' = \{J_0, J_1,..., J_m\}$ – полученные в результате множества LR(1)-ситуаций. Функцию ACTION для состояния *i* строим из J_i.
       - а) Если [*A* -> *α* • *aβ*, *b*] входит в J_i и GOTO(J_i, *A*) = J_j, то ACTION[*i*, *a*] = «перенос *j*»
       - б) Если [*A* -> *α* •, *a*] входит в J_i и $A \neq S'$, то ACTION[*i*, *a*] = «свертка A -> *α*»
       - в) Если [*S*' -> *S* •, $] входит в I_i, то ACTION[*i*, $] = «принятие»

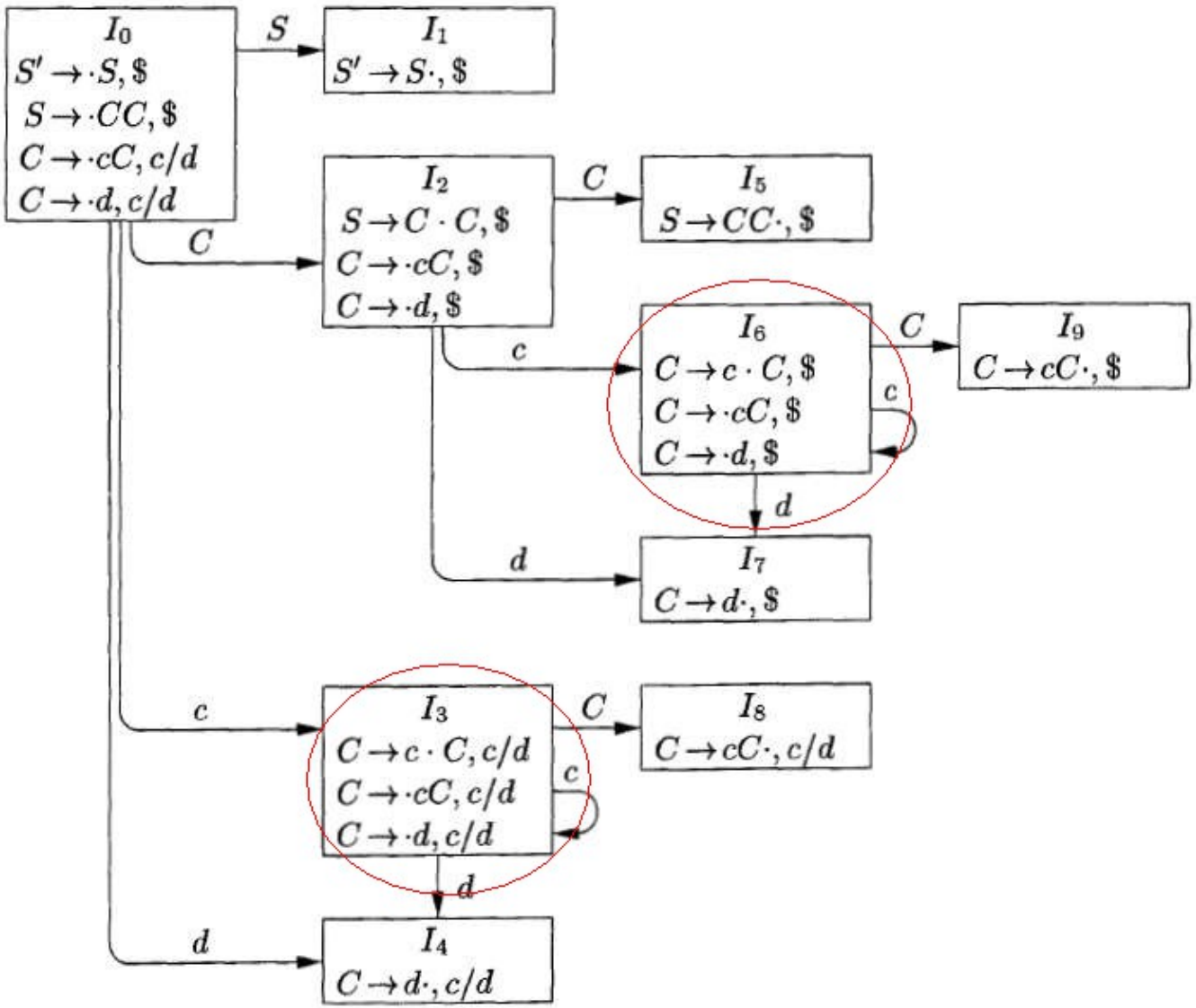
       Если при применении указанных правил **обнаруживаются конфликтующие** действия, грамматика **не LALR(1)**
    4. Если $J = I_1 \cup I_2 \cup ... \cup I_k$, то ядра GOTO($I_1$, *X*), GOTO($I_2$, *X*), ..., GOTO($I_k$, *X*) одни и те же
       - Обозначим через *K* объединение множеств ситуаций, имеющих то же ядро, что и GOTO(I_1, *X*), тогда GOTO(*J*, *X*) = *K*
    5. Все записи, не определенные (3) и (4), считаются «ошибками»
    6. Начальное состояние анализатора – состояние, построенное из множества ситуаций, содержащего [*S*' -> *S* •, $]

LALR(1)-анализ

$$
\begin{array}{lcl}
S' & \rightarrow & S \\
S & \rightarrow & C\ C \\
C & \rightarrow & c\ C \mid d
\end{array}
$$

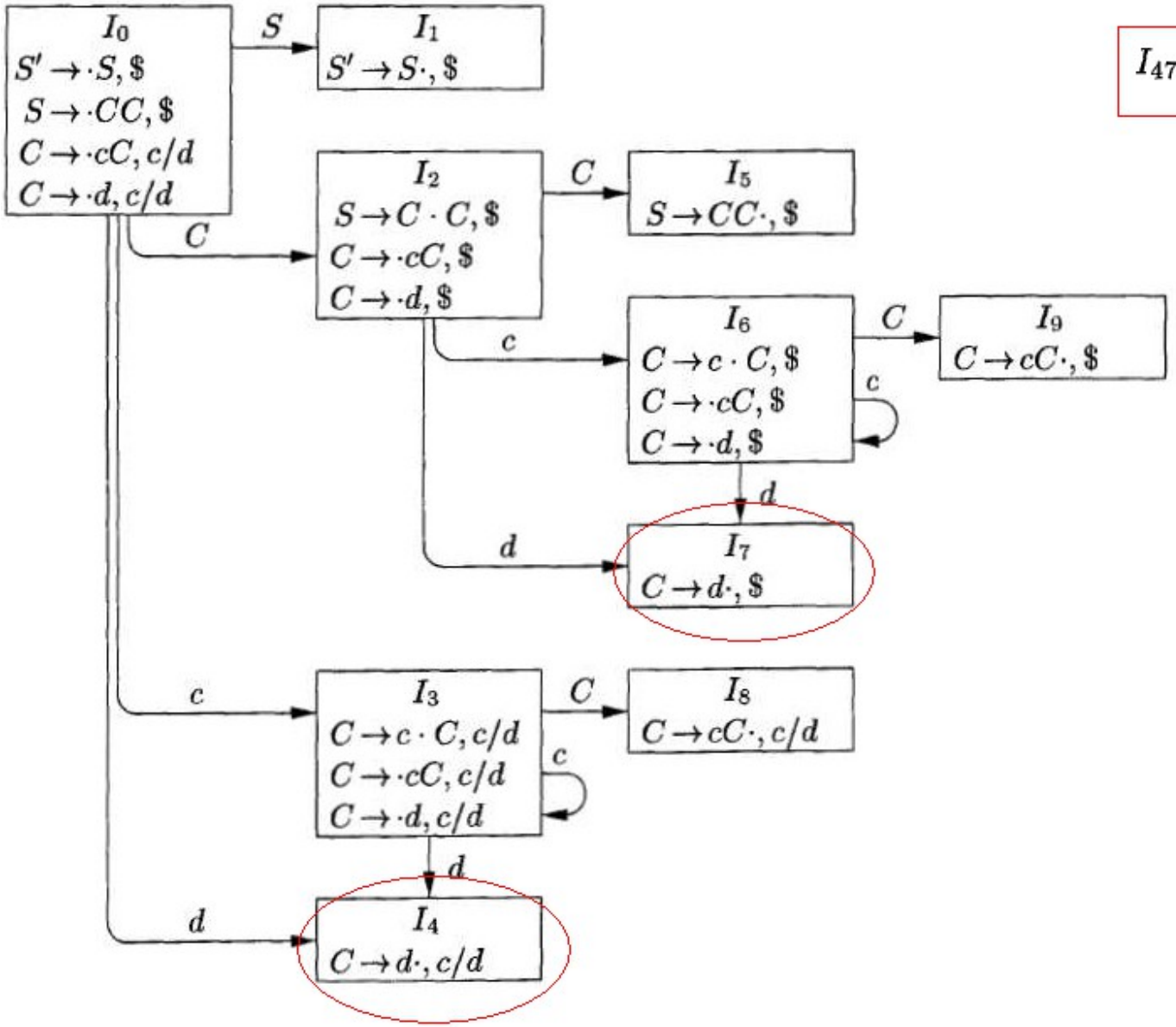


LALR(1)-анализ



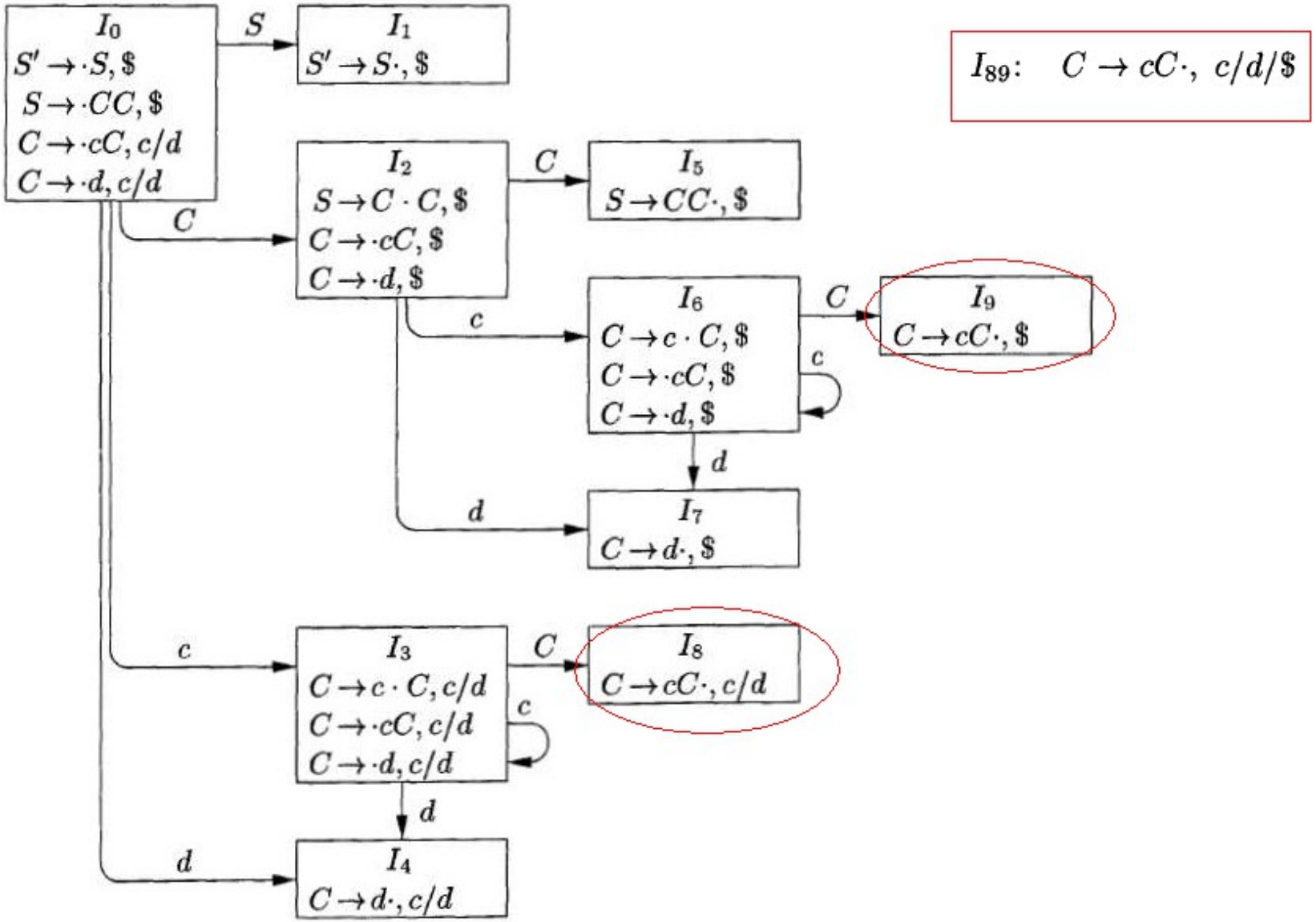
I_{36}: $C \to c\cdot C,\ c/d/\$$
$C \to \cdot cC,\ c/d/\$$
$C \to \cdot d,\ c/d/\$$

LALR(1)-анализ



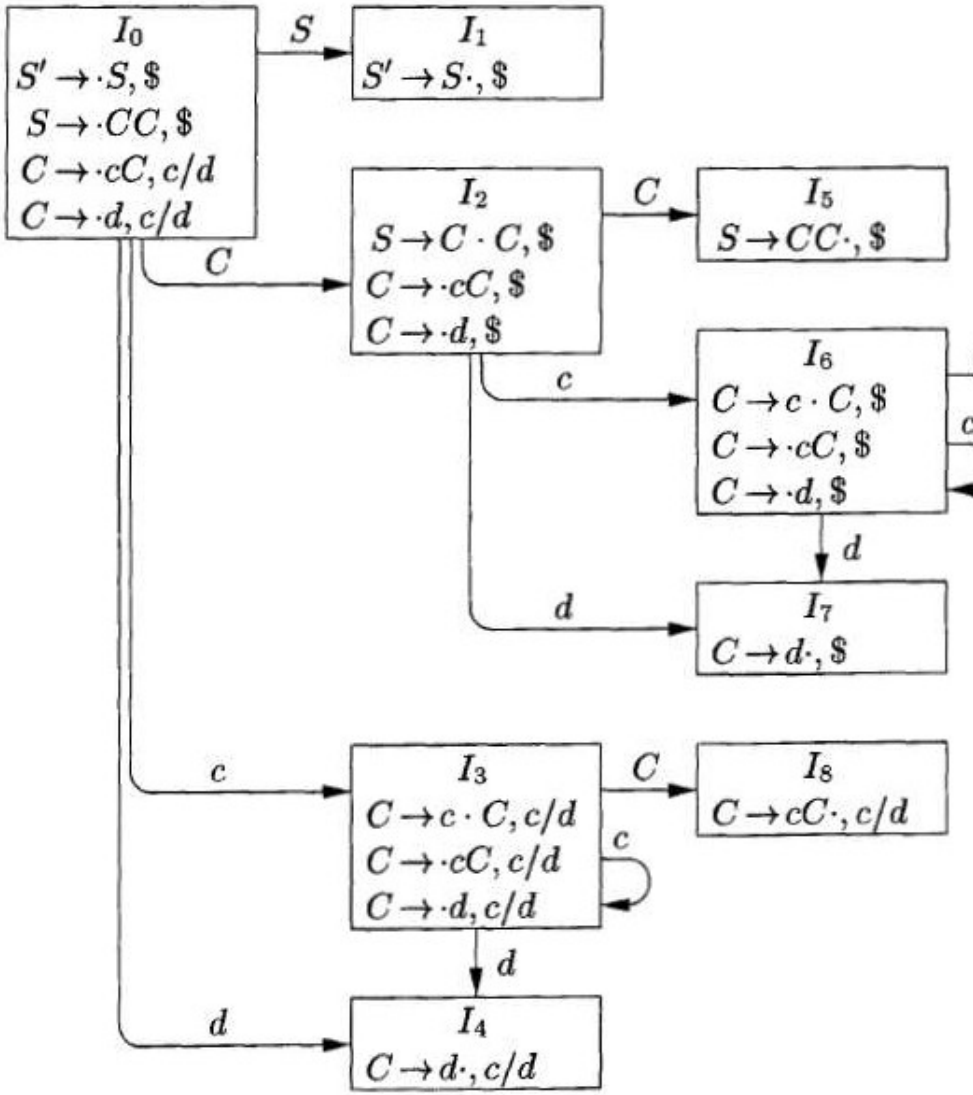
I_{47}: $C \to d\cdot,\ c/d/\$$

LALR(1)-анализ



I_{89}: $C \to cC\cdot,\ c/d/\$$

LALR(1)-анализ



I_{36}: $C \to c\cdot C,\ c/d/\$$
$C \to \cdot cC,\ c/d/\$$
$C \to \cdot d,\ c/d/\$$

I_{47}: $C \to d\cdot,\ c/d/\$$

I_{89}: $C \to cC\cdot,\ c/d/\$$

Состояние	ACTION			GOTO	
	c	*d*	$	*S*	*C*
0	s36	s47		1	2
1			acc		
2	s36	s47			5
36	s36	s47			89
47	r3	r3	r3		
5			r1		
89	r2	r2	r2		

LALR(1)-анализ

Каноническая ТСА LR-метода

Состояние	ACTION			GOTO	
	c	*d*	$	*S*	*C*
0	s3	s4		1	2
1			acc		
2	s6	s7			5
3	s3	s4			8
4	r3	r3			
5			r1		
6	s6	s7			9
7			r3		
8	r2	r2			
9			r2		

ТСА LALR-метода

Состояние	ACTION			GOTO	
	c	*d*	$	*S*	*C*
0	s36	s47		1	2
1			acc		
2	s36	s47			5
36	s36	s47			89
47	r3	r3	r3		
5			r1		
89	r2	r2	r2		

- **СТУДЕНТАМ**: проследите содержимое стека состояний анализатора для входа *cdcd* для обоих методов

LALR(1)-анализ

Каноническая ТСА LR-метода

Состояние	ACTION			GOTO	
	c	*d*	$	*S*	*C*
0	s3	s4		1	2
1			acc		
2	s6	s7			5
3	s3	s4			8
4	r3	r3			
5			r1		
6	s6	s7			9
7			r3		
8	r2	r2			
9			r2		

ТСА LALR-метода

Состояние	ACTION			GOTO	
	c	*d*	$	*S*	*C*
0	s36	s47		1	2
1			acc		
2	s36	s47			5
36	s36	s47			89
47	r3	r3	r3		
5			r1		
89	r2	r2	r2		

- **СТУДЕНТАМ**: проследите содержимое стека состояний анализатора для входа *ccd* для обоих методов

LALR(1)-анализ

$$S' \rightarrow S$$
$$S \rightarrow a\,A\,d \mid b\,B\,d \mid a\,B\,e \mid b\,A\,e$$
$$A \rightarrow c$$
$$B \rightarrow c$$

Язык: acd, ace, bed и bee

Множества ситуаций:

{[A -> c •, d],[B -> c •, e]} для префикса ac
{[A -> c •, e],[B -> c •, d]} для префикса bc

ЗДЕСЬ НЕТ КОНФЛИКТА

Объединение множеств

$$A \rightarrow c\cdot, d/e$$
$$B \rightarrow c\cdot, d/e$$

КОНФЛИКТ СВЕРТКА/СВЕРТКА:
на символах d и e вызываются
свертка A -> c и свертка B -> c

LALR(1)-анализ

```
SetOfItems CLOSURE(I)
{
  repeat
    foreach(ситуация [A —> α • Bβ, a] из J)
      foreach(продукция B —> γ из G')
        foreach(терминал b ∈ FIRST(βα))
          Добавить [B —> • γ, b] в множество I;
  until нет больше ситуаций для добавления в I;
  return J;
}
```

- Можно представить любое множество LR(0)- или LR(1)-ситуаций *I* его ядром, т.е. теми ситуациями, которые либо являются стартовыми ([*S*' -> • *S*] или [*S*' -> • *S*, $]), либо содержат точку не в начале тела продукции

- Алгоритм построения таблиц LR(1)-анализа
  - ВХОД: расширенная грамматика *G*'
  - ВЫХОД: таблица с функциями *ACTION* и *GOTO* для *G*'
  - МЕТОД: выполнить следующие действия
    1. Построить $C' = \{I_0, I_1 ..., I_n\}$ – набор множеств LR(1)-ситуаций для *G*'
    2. Состояние синтаксического анализатора строится из I_i. Действие СА для состояния *i* определяется следующим образом
       - а) Если [*A* -> *α* • *aβ*, *b*] входит в I_i и GOTO(I_i, *A*) = I_j, то ACTION[*i*, *a*] = «перенос *j*»
       - б) Если [*A* -> *α* •, *a*] входит в I_i и *A* ≠ *S*', то ACTION[*i*, *a*] = «свертка A -> *α*»
       - в) Если [*S*' -> *S* •, $] входит в I_i, то ACTION[*i*, $] = «принятие»

       Если при применении указанных правил **обнаруживаются конфликтующие** действия, грамматика **не принадлежит** классу LR(1)
    3. Переходы для состояния *i* строятся для всех нетерминалов *A* с использованием следующего правила: если GOTO(I_i, *A*) = I_j, то GOTO[*i*, *A*] = *j*
    4. Все записи, не определенные (2) и (3), считаются записями «ошибка»
    5. Начальное состояние анализатора – состояние, построенное из множества ситуаций, содержащего [*S*' -> *S* •, $]

- Можно строить ядра LALR(1)-ситуаций на основе ядер LR(0)-ситуаций при помощи процесса пропагации и спонтанной генерации символов предпросмотра
- Если имеются ядра LALR(1), то можно сгенерировать LALR(1)-таблицу путем замыкания каждого ядра с использованием функции *CLOSURE*, а затем вычислить записи таблицы, как если бы LALR(1)-множества ситуаций были каноническими LR(1)-множествами ситуаций

LALR(1)-анализ

$$
\begin{array}{lcl}
S' & \to & S \\
S & \to & L = R \mid R \\
L & \to & *R \mid \mathbf{id} \\
R & \to & L
\end{array}
$$

I_0 : $S' \to \cdot S$

I_1 : $S' \to S\cdot$

I_2 : $S \to L\cdot = R$
$R \to L\cdot$

I_3 : $S \to R\cdot$

I_4 : $L \to * \cdot R$

I_5 : $L \to \mathbf{id}\cdot$

I_6 : $S \to L = \cdot R$

I_7 : $L \to *R\cdot$

I_8 : $R \to L\cdot$

I_9 : $S \to L = R\cdot$

LALR(1)-анализ

- Для создания ядер множеств LALR(1)-ситуаций требуется назначить корректные предпросматриваемые символы ядрам LR(0)-ситуаций
- Способы, которыми предпросматриваемый символ *b* может быть назначен LR(0)-ситуации *B* -> *γ* • *δ* из некоторого множества LALR(1)-ситуаций *J* (всего – 2):
  1. Существует множество ситуаций *I* с базисной ситуацией *A* -> *α* • *β*, *a*, и *J* = GOTO (*I*, *X*), и построение GOTO(CLOSURE ({[*A* -> *α* • *β*, *a*]}) ,X) содержит *B* -> *γ* • *δ*, *b*] безотносительно к *a*
     - Символ *b* - спонтанно сгенерированный для *B* -> *γ* • *δ*
     - Частный случай: предпросматриваемый символ $ спонтанно генерируется для ситуации *S*' -> • S в начальном множестве

LALR(1)-анализ

- Для создания ядер множеств LALR(1)-ситуаций требуется назначить корректные предпросматриваемые символы ядрам LR(0)-ситуаций
- Способы, которыми предпросматриваемый символ *b* может быть назначен LR(0)-ситуации *B -> γ • δ* из некоторого множества LALR(1)-ситуаций *J* (всего – 2):

  2. a = b и GOTO(CLOSURE ({[*A -> α • β, b*]}) ,X) содержит [*B -> γ • δ, b*], т.к. одним из связанных с *A -> α • β* предпросматриваемых символов является *b*
  - В этом случае говорят о распространении («пропагации») символа предпросмотра от *A -> α • β* в ядре *I* к *B -> γ • δ* в ядре *J*
  - Пропагация не зависит от конкретного символа предпросмотра
  - Либо все символы предпросмотра распространяются от одной ситуации к другой, либо ни один из них

LALR(1)-анализ

- Пусть # – символ, отсутствующий в рассматриваемой грамматике
- Пусть $A \to \alpha \bullet \beta$ – ядро LR(0)-ситуации во множестве I
- Для каждого X вычислим

$$J = GOTO(CLOSURE(\{[A \to \alpha \bullet \beta, \#]\}), X)$$

- Для каждой базисной ситуации в J проверим ее множество символов предпросмотра
- Если # - символ предпросмотра, то предпросмотры распространяются к этому ситуации от $A \to \alpha \bullet \beta$
- Все прочие предпросмотры генерируются спонтанно
- Эти идеи изложены в приведенном на следующем слайде алгоритме, который использует тот факт, что только в базисных ситуациях J точка непосредственно следует за X
- Значит, они должны иметь вид $B \to \gamma X \bullet \delta$

LALR(1)-анализ

- ВХОД: ядро *K* множества LR(0)-ситуаций *I* и грамматический символ *X*
- ВЫХОД: предпросмотры, спонтанно генерируемые ситуациями из *I* для базисных ситуаций в GOTO(*I*,*X*), а также ситуации из *I*, от которых предпросмотры распространяются к базисным ситуациям в GOTO(*I*, X)
- МЕТОДИКА:

```
foreach(каждая ситуация A -> α • β из K)
{
  J = CLOSURE({[A -> α • β,#]}));
  if ([B -> γ •X δ, a] € J и a ≠ #)
    Заключаем, что символ предпросмотра a спонтанно
        генерируется для ситуации B —> γX • δ в GOTO(I, X);
  if([B -> γ •X δ, #] € J)
    Заключаем, что символы предпросмотра распространяются
        от A -> α • β из I к B -> γX • δ в GOTO(I, X);
}
```

LALR(1)-анализ

- **Эффективное вычисление ядер наборов множеств LALR(1)-ситуаций**
- ВХОД: расширенная грамматика *G*'
- ВЫХОД: ядра наборов множеств LALR(1)-ситуаций для *G*'
- МЕТОДИКА: следующие действия:
  1. Строим ядра множеств LR(0)-ситуаций для *G*
  2. По ядру каждого множества LR(0)-ситуаций и символу *X* определяем, какие символы предпросмотра спонтанно генерируются для базисных ситуаций в *GOTO*(*I*,*X*) и из каких ситуаций *I* символы предпросмотра распространяются на базисные ситуации *GOTO*(*I*,*X*)
  3. Инициализируем таблицу, которая для каждой базисной ситуации в каждом множестве дает связанные с ними предпросмотры
  4. Повторяем проходы по базисным ситуациям во всех множествах, и при посещении ситуации *i* с помощью таблицы, построенной в п.2, ищем базисные ситуации, на которые *i* распространяет свои предпросмотры

LALR(1)-анализ

$S \rightarrow L = R \mid R$

$L \rightarrow *R \mid \mathbf{id}$

$R \rightarrow L$

Пополненная грамматика

$S' \rightarrow \cdot S, \#$ $\quad L \rightarrow \cdot * R, \#/=$

$S \rightarrow \cdot L = R, \#$ $\quad L \rightarrow \cdot \mathbf{id}, \#/=$

$S \rightarrow \cdot R, \#$ $\quad R \rightarrow \cdot L, \#$

CLOSURE({[S' -> • S, #]})

$S' \rightarrow \cdot S$

$S' \rightarrow S\cdot$

$S \rightarrow L\cdot = R$

$R \rightarrow L\cdot$

$S \rightarrow R\cdot$

$L \rightarrow * \cdot R$

$I_5: \; L \rightarrow \mathbf{id}\cdot$

$I_6: \; S \rightarrow L = \cdot R$

$I_7: \; L \rightarrow *R\cdot$

$I_8: \; R \rightarrow L\cdot$

$I_9: \; S \rightarrow L = R\cdot$

Ядра LR(0)-ситуаций

$S \rightarrow L\cdot = R$ в I_2 $\quad L \rightarrow \mathbf{id}\cdot$ в I_5

$S \rightarrow R\cdot$ в I_3 $\quad R \rightarrow L\cdot$ в I_2

От	К
$I_0: \; S' \rightarrow \cdot S$	$I_1: \; S' \rightarrow S\cdot$
	$I_2: \; S \rightarrow L\cdot = R$
	$I_2: \; R \rightarrow L\cdot$
	$I_3: \; S \rightarrow R\cdot$
	$I_4: \; L \rightarrow * \cdot R$
	$I_5: \; L \rightarrow \mathbf{id}\cdot$
$I_2: \; S \rightarrow L\cdot = R$	$I_6: \; S \rightarrow L = \cdot R$
$I_4: \; L \rightarrow * \cdot R$	$I_4: \; L \rightarrow * \cdot R$
	$I_5: \; L \rightarrow \mathbf{id}\cdot$
	$I_7: \; L \rightarrow *R\cdot$
	$I_8: \; R \rightarrow L\cdot$
$I_6: \; S \rightarrow L = \cdot R$	$I_4: \; L \rightarrow * \cdot R$
	$I_5: \; L \rightarrow \mathbf{id}\cdot$
	$I_8: \; R \rightarrow L\cdot$
	$I_9: \; S \rightarrow L = R\cdot$

Пропагация символов предпросмотра

LALR(1)-анализ

Множество ситуация	Символы предпросмотра			
	Изначально	Проход 1	Проход 2	Проход 3
$I_0: S' \to \cdot S$	\$	\$	\$	\$
$I_1: S' \to S\cdot$		\$	\$	\$
$I_2: S \to L\cdot = R$		\$	\$	\$
$R \to L\cdot$		\$	\$	\$
$I_3: S \to R\cdot$		\$	\$	\$
$I_4: L \to *\cdot R$	=	= /\$	= /\$	= /\$
$I_5: L \to \mathbf{id}\cdot$	=	= /\$	= /\$	= /\$
$I_6: S \to L = \cdot R$			\$	\$
$I_7: L \to *R\cdot$		=	= /\$	= /\$
$I_8: R \to L\cdot$		=	= /\$	= /\$
$I_9: S \to L = R\cdot$				\$

Обработка ошибок при LR-анализе

- Восстановление после ошибок «в режиме паники»
  - Стек просматривается от вершины до состояния *s* с записью в *GOTO*-части таблицы для некоторого нетерминала *A*
  - После этого пропускается нуль или несколько символов входного потока, пока не найден символ *a*, который при отсутствии ошибки может идти за *A*
  - Затем анализатор переносит в стек состояние *GOTO*(*s*, *A*) и продолжает выполнение СА
  - При выборе нетерминала *A* возможны несколько вариантов
    - Обычно это нетерминалы, представляющие крупные фрагменты программы, такие как выражение, оператор или блок
    - Например, если *A* – нетерминал *stmt*, то *a* может быть символом ‘;’ или ‘}’, помечающим конец оператора

Обработка ошибок при LR-анализе

- Восстановление на уровне фразы реализуется путем проверки каждой ошибочной записи в таблице LR-анализа и принятия решения о том, какая наиболее вероятная ошибка могла привести к данной ситуации
- После этого можно построить подходящую функцию восстановления после ошибки
- Возможно, при этом придется изменить вершину стека и/или первые символы входного потока способом, соответствующим данной записи ошибки
- При разработке функций обработки ошибок для анализатора можно заполнить каждую пустую запись таблицы действия указателем на подпрограмму, которая будет выполнять действия, выбранные для данного случая разработчиком транслятора

Обработка ошибок при LR-анализе

$E \rightarrow E + E \mid E * E \mid (E) \mid \mathbf{id}$

Состояние	ACTION						GOTO
	id	+	*	(	)	$	E
0	s3			s2			1
1		s4	s5			acc	
2	s3			s2			6
3		r4	r4		r4	r4	
4	s3			s2			7
5	s3			s2			8
6		s4	s5		s9		
7		r1	s5		r1	r1	
8		r2	r2		r2	r2	
9		r3	r3		r3	r3	

Без функций обработки ошибок

Состояние	ACTION						GOTO
	id	+	*	(	)	$	E
0	s3	e1	e1	s2	e2	e1	1
1	e3	s4	s5	e3	e2	acc	
2	s3	e1	e1	s2	e2	e1	6
3	r4	r4	r4	r4	r4	r4	
4	s3	e1	e1	s2	e2	e1	7
5	s3	e1	e1	s2	e2	e1	8
6	e3	s4	s5	e3	s9	e4	
7	r1	r1	s5	r1	r1	r1	
8	r2	r2	r2	r2	r2	r2	
9	r3	r3	r3	r3	r3	r3	

С функциями обработки ошибок

Обработка ошибок при LR-анализе

$E \rightarrow E + E \mid E * E \mid (E) \mid \mathbf{id}$

Состояние	ACTION						GOTO
	id	+	*	(	)	$	E
0	s3	e1	e1	s2	e2	e1	1
1	e3	s4	s5	e3	e2	acc	
2	s3	e1	e1	s2	e2	e1	6
3	r4	r4	r4	r4	r4	r4	
4	s3	e1	e1	s2	e2	e1	7
5	s3	e1	e1	s2	e2	e1	8
6	e3	s4	s5	e3	s9	e4	
7	r1	r1	s5	r1	r1	r1	
8	r2	r2	r2	r2	r2	r2	
9	r3	r3	r3	r3	r3	r3	

e1: Эта подпрограмма вызывается из состояний 0, 2, 4 и 5; все они ожидают начало операнда – **id** или левую скобку. Вместо этого обнаруживается оператор + или * либо окончание входного потока.

Поместить в стек состояние 3 (переход из состояний 0, 2, 4 и 5 при входном символе **id**).
Вывести сообщение "Отсутствует операнд".

Обработка ошибок при LR-анализе

$E \rightarrow E + E \mid E * E \mid (E) \mid \mathbf{id}$

Состояние	ACTION						GOTO
	id	+	*	(	)	$	*E*
0	s3	e1	e1	s2	e2	e1	1
1	e3	s4	s5	e3	e2	acc	
2	s3	e1	e1	s2	e2	e1	6
3	r4	r4	r4	r4	r4	r4	
4	s3	e1	e1	s2	e2	e1	7
5	s3	e1	e1	s2	e2	e1	8
6	e3	s4	s5	e3	s9	e4	
7	r1	r1	s5	r1	r1	r1	
8	r2	r2	r2	r2	r2	r2	
9	r3	r3	r3	r3	r3	r3	

e2: Эта подпрограмма вызывается из состояний 0, 1, 2, 4 и 5 при обнаружении правой скобки.

Удалить правую скобку из входного потока.
Вывести сообщение “Несбалансированная правая скобка”.

Обработка ошибок при LR-анализе

$E \rightarrow E + E \mid E * E \mid (E) \mid \mathbf{id}$

Состояние	ACTION						GOTO
	id	+	*	(	)	$	E
0	s3	e1	e1	s2	e2	e1	1
1	e3	s4	s5	e3	e2	acc	
2	s3	e1	e1	s2	e2	e1	6
3	r4	r4	r4	r4	r4	r4	
4	s3	e1	e1	s2	e2	e1	7
5	s3	e1	e1	s2	e2	e1	8
6	e3	s4	s5	e3	s9	e4	
7	r1	r1	s5	r1	r1	r1	
8	r2	r2	r2	r2	r2	r2	
9	r3	r3	r3	r3	r3	r3	

e3: Эта подпрограмма вызывается из состояний 1 и 6, когда ожидается оператор, а обнаруживается **id** или правая скобка.

Поместить в стек состояние 4 (соответствующее символу +).
Вывести сообщение “Отсутствует оператор”.

Обработка ошибок при LR-анализе

$E \rightarrow E + E \mid E * E \mid (E) \mid \mathbf{id}$

Состояние	ACTION						GOTO
	id	+	*	(	)	$	E
0	s3	e1	e1	s2	e2	e1	1
1	e3	s4	s5	e3	e2	acc	
2	s3	e1	e1	s2	e2	e1	6
3	r4	r4	r4	r4	r4	r4	
4	s3	e1	e1	s2	e2	e1	7
5	s3	e1	e1	s2	e2	e1	8
6	e3	s4	s5	e3	s9	e4	
7	r1	r1	s5	r1	r1	r1	
8	r2	r2	r2	r2	r2	r2	
9	r3	r3	r3	r3	r3	r3	

e4: Эта подпрограмма вызывается из состояния 6 при обнаружении конца входного потока

Поместить в стек состояние 9 (для правой скобки).
Вывести сообщение “Отсутствует правая скобка”.

Обработка ошибок при LR-анализе

$E \rightarrow E + E \mid E * E \mid (E) \mid \mathbf{id}$

Состояние	ACTION						GOTO
	id	+	*	(	)	$	E
0	s3	e1	e1	s2	e2	e1	1
1	e3	s4	s5	e3	e2	acc	
2	s3	e1	e1	s2	e2	e1	6
3	r4	r4	r4	r4	r4	r4	
4	s3	e1	e1	s2	e2	e1	7
5	s3	e1	e1	s2	e2	e1	8
6	e3	s4	s5	e3	s9	e4	
7	r1	r1	s5	r1	r1	r1	
8	r2	r2	r2	r2	r2	r2	
9	r3	r3	r3	r3	r3	r3	

id+)

Стек	Символы	Входной поток	Действия
0		**id**+)$	
0 3	**id**	+)$	
0 1	E	+)$	
0 1 4	E+	)$	"Несбалансированная правая скобка"
			e2 удаляет правую скобку
0 1 4	E+	$	"Отсутствует операнд"
			e1 помещает в стек состояние 3
0 1 4 3	E + **id**	$	
0 1 4 7	E+	$	
0 1	E+	$	

Использование неоднозначных грамматики при LR-анализе

$E \to E + E \mid E * E \mid (E) \mid \mathbf{id}$

Состояние	ACTION						GOTO
	id	+	*	(	)	$	E
0	s3			s2			1
1		s4	s5			acc	
2	s3			s2			6
3		r4	r4		r4	r4	
4	s3			s2			7
5	s3			s2			8
6		s4	s5		s9		
7		r1	s5		r1	r1	
8		r2	r2		r2	r2	
9		r3	r3		r3	r3	

Эквивалентная однозначная грамматика

$$E \to E + T \mid T$$
$$T \to T * F \mid F$$
$$F \to (E) \mid \mathbf{id}$$

I_0: $E' \to \cdot E$
$E \to \cdot E + E$
$E \to \cdot E * E$
$E \to \cdot (E)$
$E \to \cdot \mathbf{id}$

I_1: $E' \to E \cdot$
$E \to E \cdot + E$
$E \to E \cdot * E$

I_2: $E \to (\cdot E)$
$E \to \cdot E + E$
$E \to \cdot E * E$
$E \to \cdot (E)$
$E \to \cdot \mathbf{id}$

I_3: $E \to \mathbf{id} \cdot$

I_4: $E \to E + \cdot E$
$E \to \cdot E + E$
$E \to \cdot E * E$
$E \to \cdot (E)$
$E \to \cdot \mathbf{id}$

I_5: $E \to E * \cdot E$
$E \to \cdot E + E$
$E \to \cdot E * E$
$E \to \cdot (E)$
$E \to \cdot \mathbf{id}$

I_6: $E \to (E \cdot)$
$E \to E \cdot + E$
$E \to E \cdot * E$

I_7: $E \to E + E \cdot$
$E \to E \cdot + E$
$E \to E \cdot * E$

I_8: $E \to E * E \cdot$
$E \to E \cdot + E$
$E \to E \cdot * E$

I_9: $E \to (E) \cdot$

Использование неоднозначных грамматики при LR-анализе

stmt → **if** *expr* **then** *stmt* **else** *stmt*
| **if** *expr* **then** *stmt*
| **other**

$S' \rightarrow S$

$S \rightarrow i\ S\ e\ S \mid i\ S \mid a$

Состояние	ACTION				GOTO
	i	e	a	\$	S
0	s2		s3		1
1				acc	
2	s2		s3		4
3		r3		r3	
4		s5		r2	
5	s2		s3		6
6		r1		r1	

I_0: $S' \rightarrow \cdot S$
$S \rightarrow \cdot iSeS$
$S \rightarrow \cdot iS$
$S \rightarrow \cdot a$

I_1: $S' \rightarrow S\cdot$

I_2: $S \rightarrow i \cdot SeS$
$S \rightarrow i \cdot S$
$S \rightarrow \cdot iSeS$
$S \rightarrow \cdot iS$
$S \rightarrow \cdot a$

I_3: $S \rightarrow a\cdot$

I_4: $S \rightarrow iS \cdot eS$

I_5: $S \rightarrow iSe \cdot S$
$S \rightarrow \cdot iSeS$
$S \rightarrow \cdot iS$
$S \rightarrow \cdot a$

I_6: $S \rightarrow iSeS\cdot$

Использование неоднозначных грамматики при LR-анализе

	Стек	Символы	Вход	Действия
(1)	0		$i\ i\ a\ e\ a\$$	Перенос
(2)	0 2	i	$i\ a\ e\ a\$$	Перенос
(3)	0 2 2	$i\ i$	$a\ e\ a\$$	Перенос
(4)	0 2 2 3	$i\ i\ a$	$e\ a\$$	Перенос
(5)	0 2 2 4	$i\ i\ S$	$e\ a\$$	Свертка по $S \rightarrow a$
(6)	0 2 2 4 5	$i\ i\ S\ e$	$a\$$	Перенос
(7)	0 2 2 4 5 3	$i\ i\ S\ e\ a$	$\$$	Свертка по $S \rightarrow a$
(8)	0 2 2 4 5 6	$i\ i\ S\ e\ S$	$\$$	Свертка по $S \rightarrow iSeS$
(9)	0 2 4	$i\ S$	$\$$	Свертка по $S \rightarrow iS$
(10)	0 1	S	$\$$	Принятие

Альтернативная грамматика

stmt → *matched_stmt*
| *open_stmt*

matched_stmt → **if** *expr* **then** *matched_stmt* **else** *matched_stmt*
| **other**

open_stmt → **if** *expr* **then** *stmt*
| **if** *expr* **then** *matched_stmt* **else** *open_stmt*

LR(k)-грамматики, GLR-анализ, IELR-анализ

- Доказано, что любая LR(k) грамматика имеет эквивалентную ей LR(1)-грамматики (проблема в ее поиске)
- Поэтому в большинстве случаев не имеет смысла работать с $k > 1$
- Пример LR(2)-грамматики:

```
S : 'a' L ',' S | 'a' L;
L : 'x' ',' L   | 'x';
```

- **Студентам**: какой язык описывается данной грамматикой?
- *Hint: See lecture notes*

LR(k)-грамматики, GLR-анализ, IELR-анализ

- Если мощности LALR-, и даже канонического LR(1)-анализа недостаточно, можно воспользоваться обобщенным алгоритмом LR(1)-анализа
- Это GLR-анализ, подробнее, см. http://www.cs.rhul.ac.uk/research/languages/publica
- Альтернатива – IELR-анализ, подробнее, см. http://www.sciencedirect.com/science/article/pii/S0167642309001191

Дополнительные источники

1. Backus, J.W. Revised Report on the Algorithmic Language ALGOL 60. / J.W. Backus et al., in P. Naur editions. // Commun. ACM, vol. 6, no. 1, p. 1-17, January 1960
2. Standard ECMA-262. ECMAScript Language Specification – http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-2
3. Standard ECMA-334. C# Language Specification – http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-3
4. Grune, D. Parsing Techniques: a practical guide. / D. Grune, C. Jakobs. – Ellis Horwood, Chichester, 1990. – 334 p
5. Knuth, D. E. Semantics of context-free languages/ D.E. Knuth // Mathematical Systems Theory 2:2, 1968. – P. 127-145
6. Wirth, N. The design of a Pascal compiler. / N. Wirth // Software – Practice and Experience 1:4, 1971, p. 309-333
7. Хомский, Н. «Три модели для описания языка». / Н. Хомский // Кибернетический сборник. – М. ИЛ, 1961. – Вып. 2. – С. 237-266
8. Мозговой, М.В. Классика программирования: алгоритмы, языки, автоматы, компиляторы Практический подход / М.В. Мозговой. – СПб.: Наука и техника, 2006. – 320 с.
9. Гросс, М. Теория формальных грамматик. / М. Гросс, А. Лантен. – М.: Мир, 1971. – 296 с.
10. Waite, W.M. Compiler Construction / W.M. Waite, G. Goos. – Berlin: Springer Verlag, 1996. – 372 p.