11. Синтаксический анализ. Часть 3

Разделы:

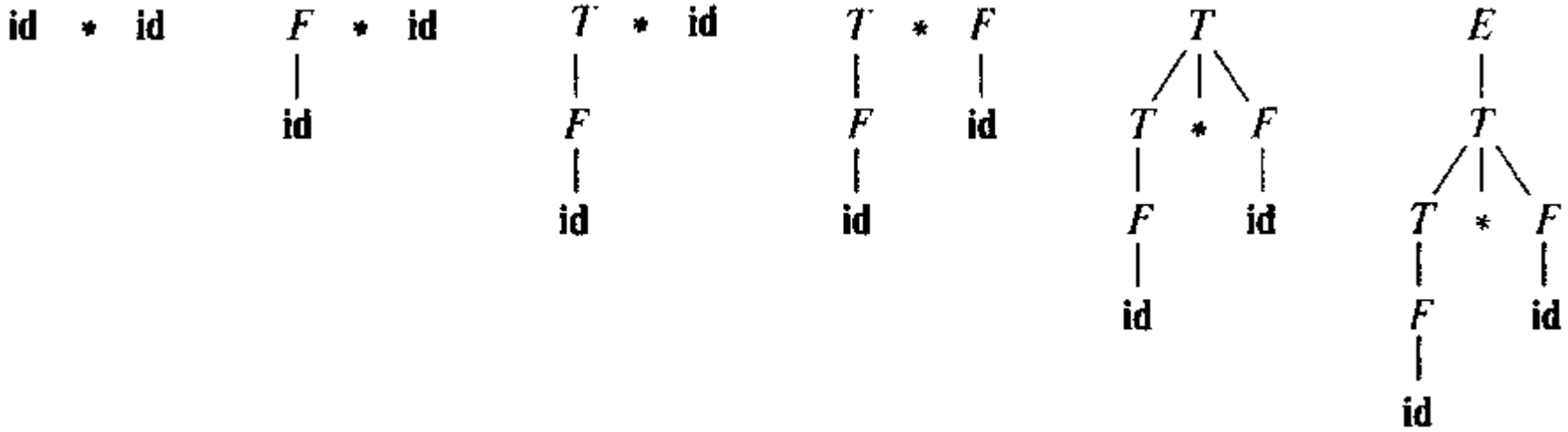
- Метод «перенос-свертка»
- LR(0)-автоматы
- Общий алгоритм LR(1)-анализа
- Таблицы SLR-анализа
- Активные префиксы

ПС-анализ

$$E \rightarrow E + T \mid T$$

$$T \rightarrow T * F \mid F$$

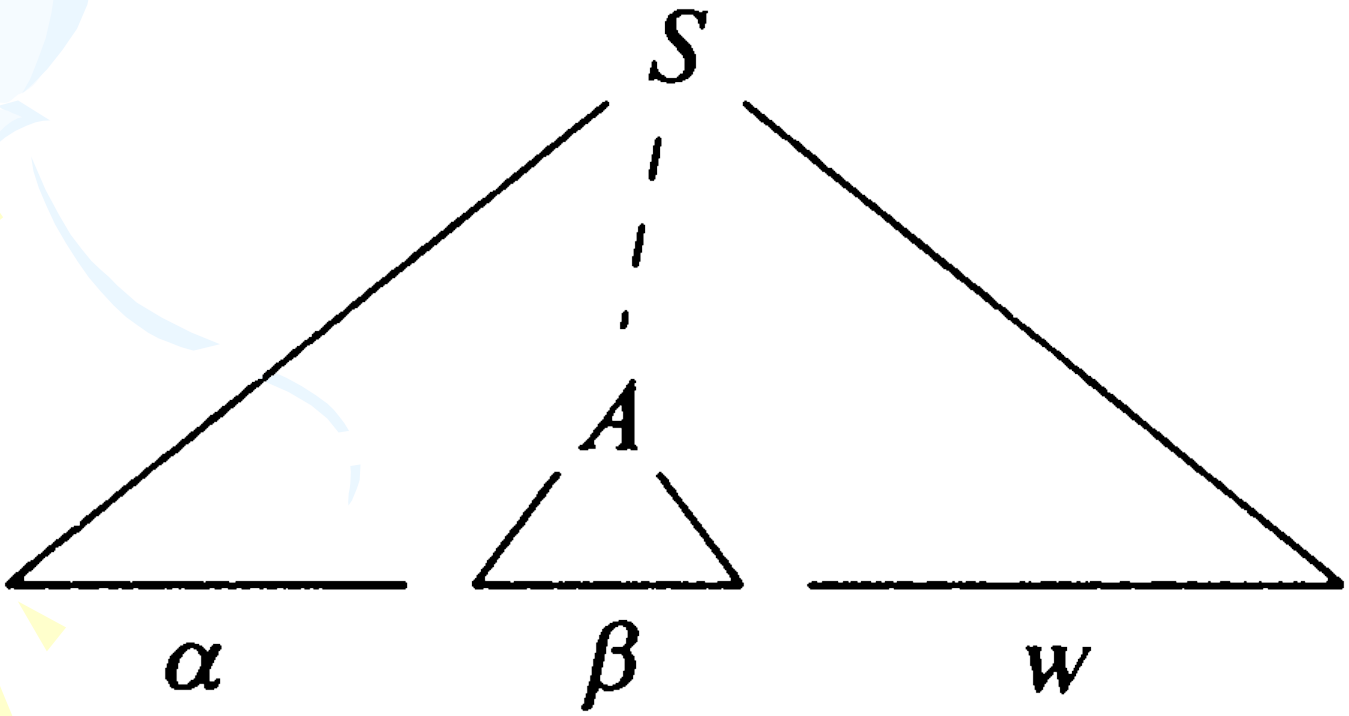
$$F \rightarrow (E) \mid \mathbf{id}$$



$$E \Rightarrow_r T \Rightarrow_r T*F \Rightarrow_r T*id \Rightarrow_r F*id \Rightarrow_r id*id$$

ПС-анализ

Правая сентенциальная форма	Основа	Сворачивающая продукция
$\mathbf{id}_1 * \mathbf{id}_2$	$\mathbf{id}_1$	$F \to \mathbf{id}$
$F * \mathbf{id}_2$	F	$T \to F$
$T * \mathbf{id}_2$	$\mathbf{id}_2$	$F \to \mathbf{id}$
$T * F$	$T * F$	$E \to T * F$



ПС-анализ

- Начинаем с анализируемой строки терминалов
- Если w – строка грамматики, то пусть $w = \gamma_n$, где γ_n – n-я правосентенциальная форма некоторого неизвестного правого порождения
  $S =>_r \gamma_0 =>_r \gamma_1 =>_r \gamma_2 =>_r^* \ldots =>_r \gamma_{n-1} =>_r \gamma_n =>_r w$
- Для его воссоздания мы находим основу β_n в γ_n и заменяем ее LHS продукции A_n -> β_n для получения γ_{n-1}
- Затем мы повторяем описанный процесс, т.е. находим в γ_{n-1} основу β_{n-1} и свертываем ее для получения γ_{n-2}
- Если после очередного шага СФ содержит только S, то мы прекращаем процесс и сообщаем об успешном завершении анализа
- Обращенная последовательность продукций, использованных в свертках, представляет собой правое порождение входной строки

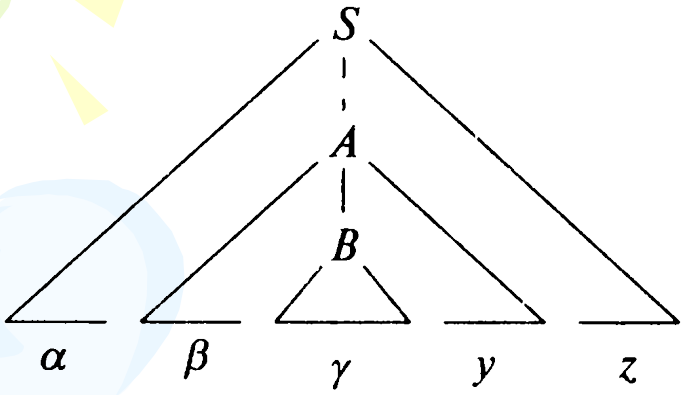
ПС-анализ

Стек	Вход	Действие
$	id_1 * $id_2$$	Перенос
$$id_1$	* $id_2$$	Свертка по F -> id
$$F$	* $id_2$$	Свертка по T -> F
$$T$	* $id_2$$	Перенос
$$T$*	$id_2$$	Перенос
$$T$*$id_2$	$	Свертка по F -> id
$$T$*$F$	$	Свертка по T -> T*F
$$T$	$	Свертка по E -> T
$$E$	$	Принятие

- **Перенос** (*shift*) – перемещение очередного входного символа на вершину стека
- **Свертка** (*reduce*) - следующая операция
  - На вершине стека должна располагаться RHS сворачиваемой строки
  - Определяется левый конец строки в стеке и принимается решение о том, каким нетерминалом будет заменена строка.
- **Принятие** (*accept*) - объявление об успешном завершении ВСА
- Обнаружение синтаксической **ошибки** (*error*) и вероятный вызов подпрограммы восстановления после ошибки

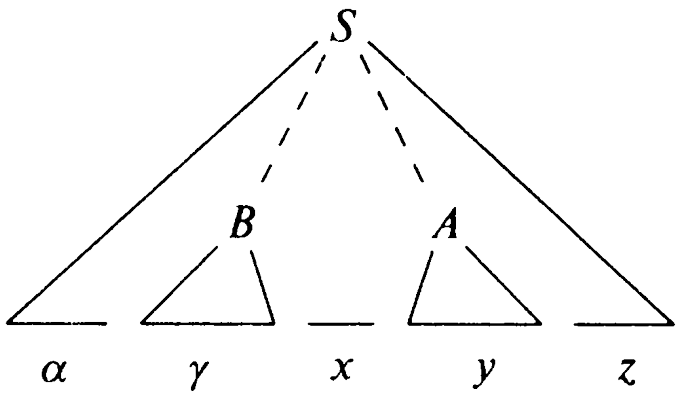
ПС-анализ

1. $S \Rightarrow_r \alpha Az \Rightarrow_r \alpha\beta Byz \Rightarrow_r \alpha\beta\gamma yz$.
2. $S \Rightarrow_r \alpha BxAz \Rightarrow_r \alpha Bxyz \Rightarrow_r \alpha\gamma xyz$.



Случай (1)

Стек	Вход
...	...
$\$\alpha\beta\gamma$	$yz\$$
$\$\alpha\beta B$	$yz\$$
$\$\alpha\beta By$	$z\$$
$\$\alpha A$	$z\$$
...	



Случай (2)

Стек	Вход
...	...
$\$\alpha\gamma$	$xyz\$$
$\$\alpha B$	$xyz\$$
$\$\alpha Bxy$	$z\$$
...	

ПС-анализ

- Любой ПС-анализатор может достичь конфигурации, в которой синтаксический анализатор, обладая информацией о содержимом стека и очередных *k* входных символах, не может принять решение о том, что следует выполнить:
  - перенос или свертку (это **конфликт «перенос/свертка»**)
  - либо какое именно из нескольких сверток должно быть выполнено (**конфликт «свертка/свертка»**)

ПС-анализ

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

Стек
...**if** *expr* **then** *stmt*

Вход
else…$

ПС-анализ

stmt	→	**id** (*parameter_list*)
stmt	→	*expr* := *expr*
parameter_list	→	*parameter_list* , *parameter*
parameter_list	→	*parameter*
parameter	→	**id**
expr	→	**id** (*expr_list*)
expr	→	**id**
expr_list	→	*expr_list* , *expr*
expr_list	→	*expr*

Стек
...**id** (**id**

Вход
, ***id***)…

Стек
... **procid** (**id**

Вход
, ***id***)…

LR(0)-автоматы

- **LR(*k*)-анализ**
- *L* - просмотр входной строки слева направо, *R* – построение обратного правого порождения, а *k* – количество предпросматриваемых символов входного потока, необходимое для принятия решения
- Неформально КСГ является LR, если синтаксический анализатор, работающий слева направо методом переноса-свертки, способен распознавать основы правых СФ при их появлении на вершине стека
- Причины популярности LR-анализа:
  - Можно создавать анализаторы для распознавания основных конструкций языков программирования
  - Это наиболее общий метод ПС-анализа без возврата, который не уступает другим, более примитивным методам
  - LR-анализатор может обнаруживать ошибки на первом же некорректном входном символе
  - Класс LR-грамматик представляет собой истинное надмножество грамматик, анализируемых LL-методом

LR(0)-автоматы

Стек	Вход	Действие
$	$id_1 * id_2$	Перенос
$$id_1$	* id_2	Свертка по $F \to id$
$$F$	* id_2	Свертка по $T \to F$
$$T$	* id_2	Перенос
$$T*$	id_2	Перенос
$$T*id_2$	$	Свертка по $F \to id$
$$T*F$	$	Свертка по $T \to T*F$
$$T$	$	Свертка по $E \to T$
$$E$	$	Принятие

- Множество *LR*(0)-ситуаций

$$A \to \cdot XYZ$$

$$A \to X \cdot YZ$$

$$A \to XY \cdot Z$$

$$A \to XYZ \cdot$$

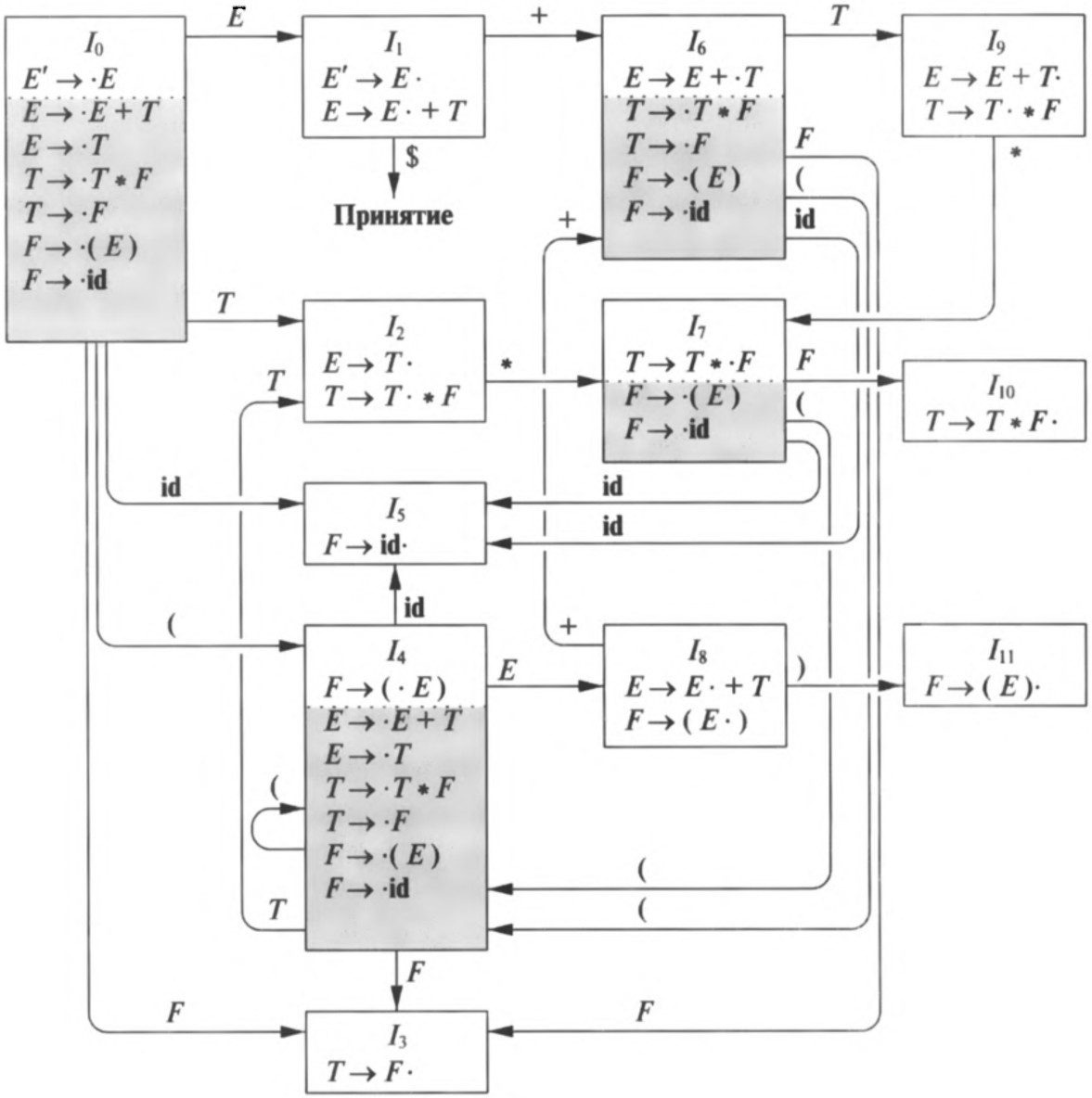
$A \to \varepsilon$

$A \to \cdot$

LR(0)-автоматы

- Один набор множеств LR(0)-ситуаций - **канонический набор LR(0)**
- Обеспечивает основу для построения ДКА, который используется для принятия решений в процессе СА
- Такой автомат называется **LR(0)-автоматом**
- Каждое состояние LR(0)-автомата представляет множество ситуаций в каноническом наборе LR(0)

LR(0)-автоматы



LR(0)-автоматы

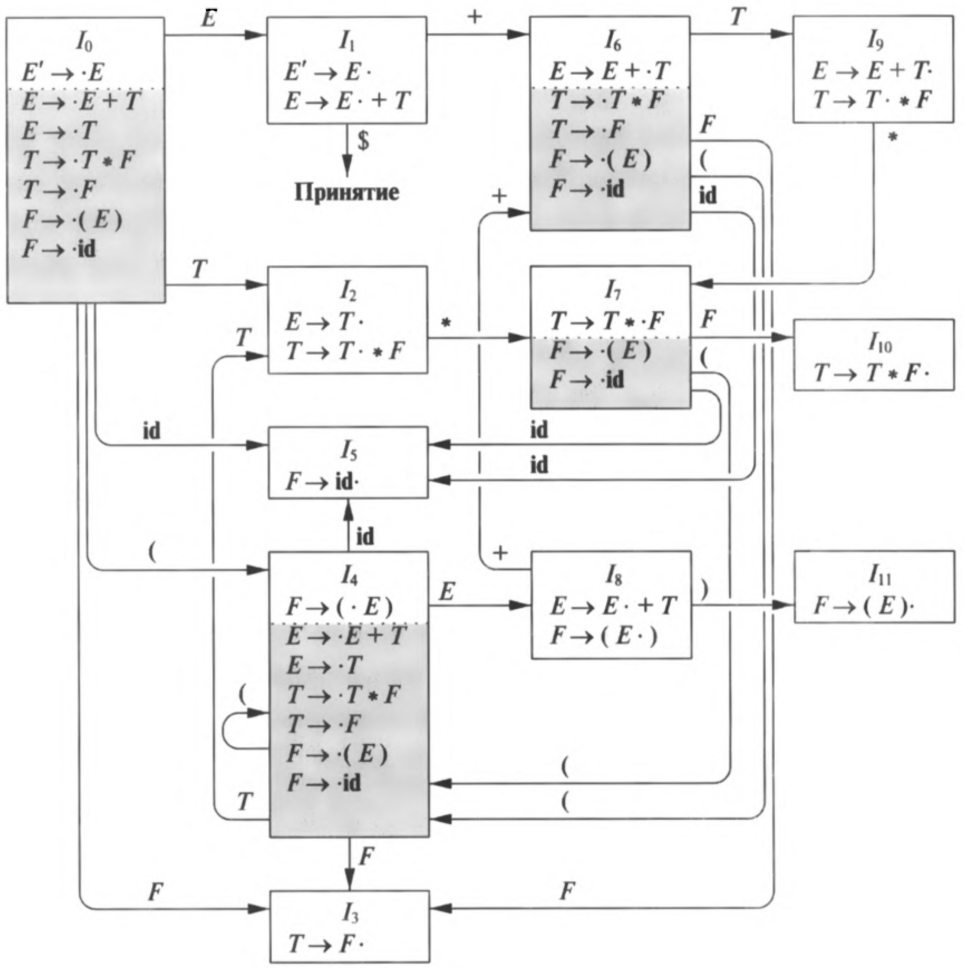
- Для построения канонического LR(0)-набора мы определяем пополненную грамматику и две функции, *CLOSURE* и *GOTO*
- Если *G* – грамматика с аксиомой *S*, то пополненная грамматика *G'* представляет собой *G* с новой аксиомой *S'* и продукцией *S'* -> *S*
- Назначение новой аксиомы – указать синтаксическому анализатору, когда следует прекратить анализ и сообщить о принятии входной строки
  - принятие осуществляется тогда и только тогда, когда синтаксический анализатор выполняет свертку с использованием продукции *S'* -> *S*

LR(0)-автоматы

- Если *I* – множество ситуаций грамматики *G*, то *CLOSURE*(*I*) - множество ситуаций, построенное из *I* по двум правилам
  1. Изначально в *CLOSURE*(*I*) добавляются все элементы *I*
  2. Если *A* -> *α*•*Bβ* входит в *CLOSURE*(*I*), а *B* -> *γ* является продукцией, то в *CLOSURE*(*I*) добавляется ситуация *B* -> • *γ*, если ее там еще нет
     - Это правило применяется до тех пор, пока не останется ситуаций для добавления в *CLOSURE*(*I*).
- Неформально, *A* -> *α*•*Bβ* в *CLOSURE*(*I*) указывает, что в некоторой точке процесса синтаксического анализа мы полагаем, что далее во входной строке мы можем встретить подстроку, порождаемую из *Bβ*
- Эта подстрока имеет префикс, порождаемый из *B* путем применения одной из *B*-продукций
- Мы добавляем ситуации для всех *B*-продукций
  - если *B* -> *γ* является продукцией, то мы включаем *B* -> • *γ* в *CLOSURE*(*I*)

LR(0)-автоматы

$$
\begin{aligned}
E' &\rightarrow E \\
E &\rightarrow E + T \mid T \\
T &\rightarrow T * F \mid F \\
F &\rightarrow (E) \mid \mathbf{id}
\end{aligned}
$$



LR(0)-автоматы

```
SetOfItems CLOSURE(I)
{
  J = I;
  repeat
    foreach (пункт A –> a • Bβ из J)
      foreach (продукция B –> γ из G)
        if (B -> • γ не входит в J)
          Добавить B -> • γ в J;
  until больше нет пунктов для добавления в J за один проход;
  return J;
}
```

- Удобный способ реализации функции *closure* состоит в поддержании булева массива *added*, индексированного нетерминалами грамматики *G*
- *added*[*B*] устанавливается равным *true*, если (и когда) мы добавляем ситуацию *B* -> • *γ* для каждой *B*-продукции *B* -> *γ*

LR(0)-автоматы

- Если одна *B*-продукция добавляется в замыкание *I* с точкой на левом конце, то в замыкание будут аналогичным образом добавлены все *B*-продукции
- Значит, при некоторых условиях нет необходимости в перечислении ситуаций *B* -> • *γ*, добавленных в *I* функцией *CLOSURE*
- Достаточно списка нетерминалов *B*, продукции которых были добавлены таким способом
- Множество ситуаций разделим на два класса
  1. **Базисные ситуации**, или **ситуации ядра** (*kernel items*): начальная ситуация *S*' -> • *S* и все ситуации, у которых точки расположены не у левого края
  2. **Небазисные** (*nonkernel items*) **ситуации**, у которых точки расположены слева, за исключением *S*' -> • *S*

LR(0)-автоматы

- Вторая функция - *GOTO*(*I*, *X*), где *I* – множество ситуаций, а *X* – символ грамматики
- *GOTO*(*I*,*X*) определяется как замыкание множества всех ситуаций [*A* -> *aX* • *β*], таких, что [*A* -> *aX* • *β*] находится в *I*
- Неформально, функция *GOTO* используется для определения переходов в LR(0)-автомате грамматики
- Состояния автомата соответствуют множествам ситуаций, и *GOTO*(*I*, *X*) указывает переход из состояния *I* при входном символе *X*

LR(0)-автоматы

- Если *I* – множества из двух ситуаций {[*E*' -> *E* •], [*E* -> *E* • + *T*]}, то *GOTO*(*I*, +) содержит ситуации:

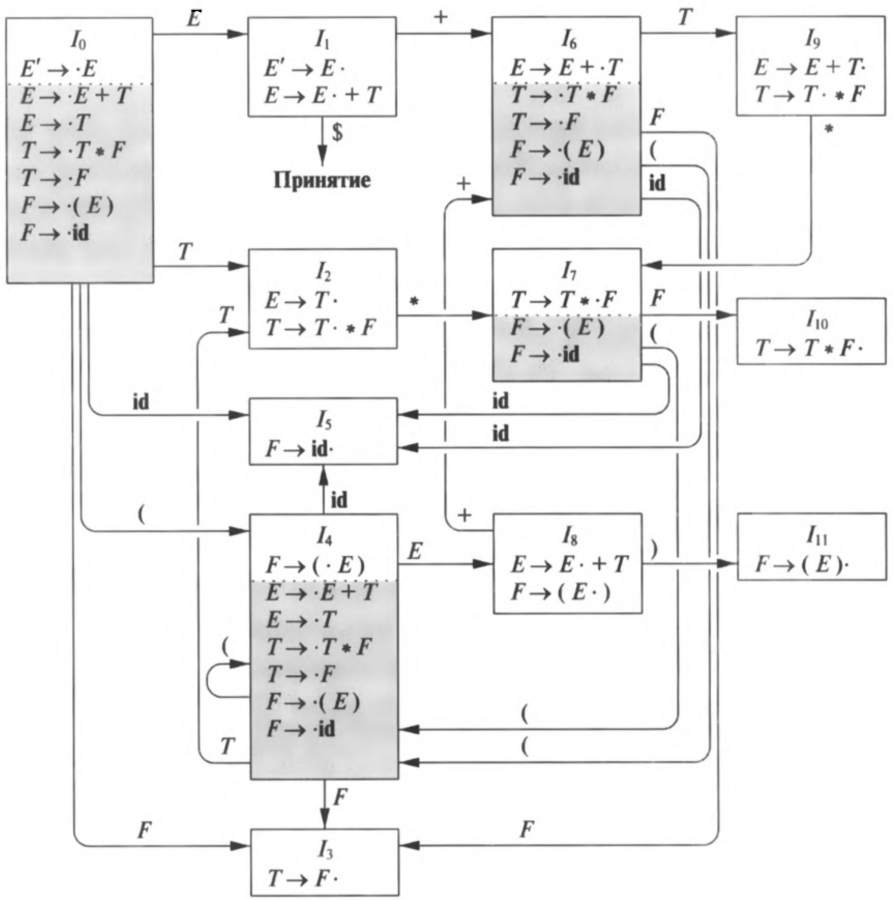
$$
\begin{aligned}
E &\rightarrow E + \cdot T \\
T &\rightarrow \cdot T * F \\
T &\rightarrow \cdot F \\
F &\rightarrow \cdot (E) \\
F &\rightarrow \cdot \mathbf{id}
\end{aligned}
$$

LR(0)-автоматы

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

LR(0)-автоматы

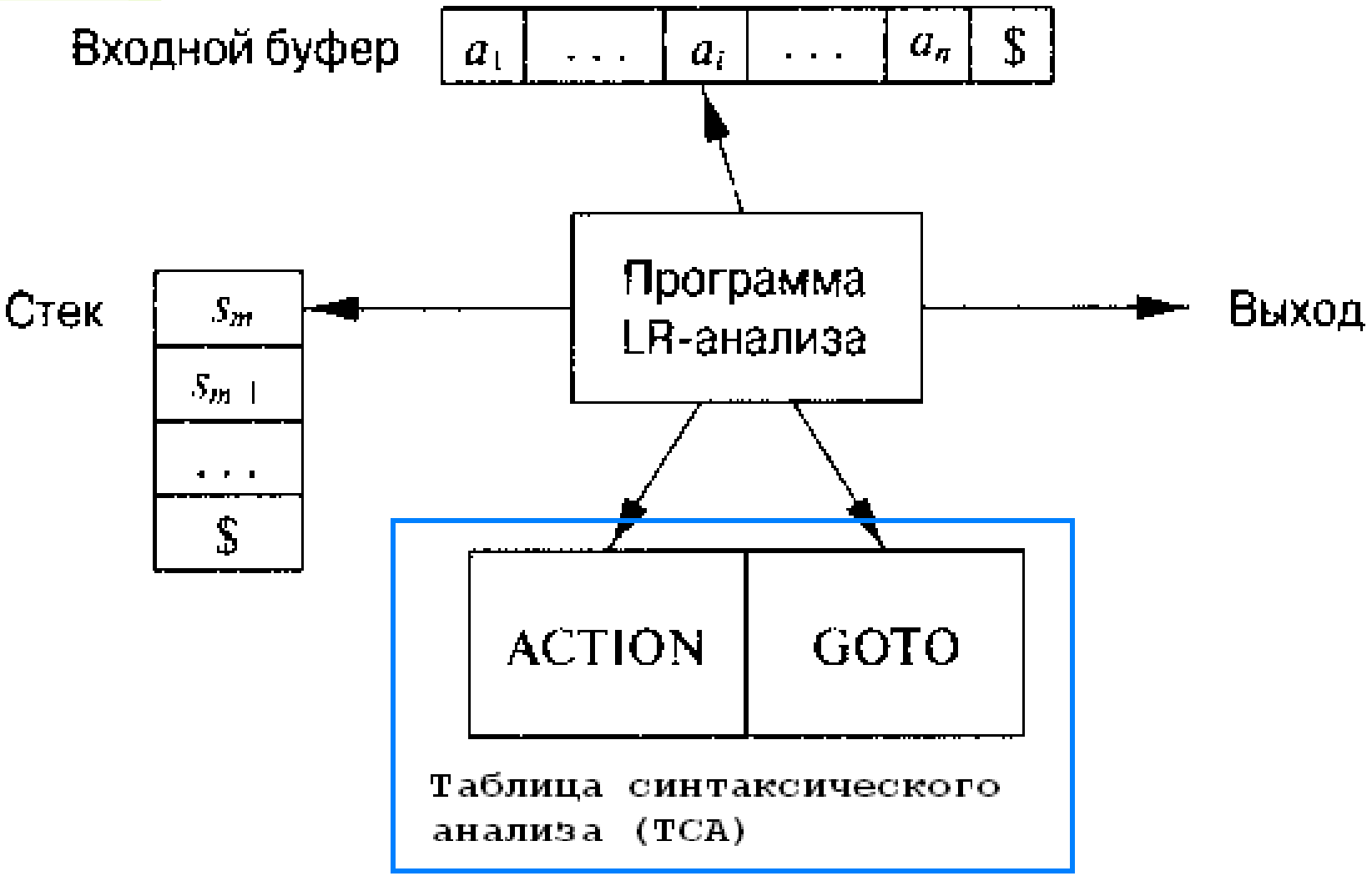
- Стартовое состояние LR(0)-автомата – *CLOSURE*({[*S*' -> • *S*]})
  - *S*' – новая аксиома
- Все состояния являются принимающими
- Под *j* мы понимаем состояние, соответствующее множеству ситуаций *Ij*
- Пусть строка *γ* из символов грамматики переводит LR(0)-автомат из состояния 0 в некоторое состояние *j*
- Тогда выполним **перенос** символа *a*, если состояние *j* имеет для него переход
- В противном случае выбирается **свертка**
  - Ситуация в состоянии *j* говорит нам, какую продукцию следует для этого использовать



LR(0)-автоматы

Строка	Стек	Символы	Вход	Действие
(1)	0	\$	**id** $*$ **id**\$	Перенос в 5
(2)	0 5	\$**id**	$*$**id**\$	Свертка по $F \to$ **id**
(3)	0 3	\$$F$	$*$**id**\$	Свертка по $T \to F$
(4)	0 2	\$$T$	$*$**id**\$	Перенос в 7
(5)	0 2 7	\$$T*$	**id**\$	Перенос в 5
(6)	0 2 7 5	\$$T*$ **id**	\$	Свертка по $F \to$ **id**
(7)	0 2 7 10	\$$T*F$	\$	Свертка по $T \to T*F$
(8)	0 2	\$$T$	\$	Свертка по $E \to T$
(9)	0 1	\$$E$	\$	Принятие

LR-анализ


Входной буфер
a_1 . . . a_i . . . a_n $
Стек
S_m
S_{m-1}
. . .
$
Программа LR-анализа
Выход
ACTION
GOTO
Таблица синтаксического анализа (ТСА)

LR-анализ

- ТСА состоит из двух частей

1) Функция *ACTION* принимает в качестве аргумента состояние *i* и терминал *a* (или маркер конца входной строки)
   - Значение *ACTION*[*i*, *a*] может быть одного из следующих видов:
     1а) **Перенос** в состояние *j*
     1б) **Свертка** согласно продукции *A* -> *β*
     1в) **Принятие**
     1г) **Ошибка**
2) Функция *GOTO*, определенная на множествах ситуаций, распространяется на состояния
   - если *GOTO*[*Ii*, *A*] = *Ij*, то *GOTO* отображает также состояние *i* и нетерминал *A* на состояние *j*

LR-анализ

- Конфигурация LR-анализатора - пара ($s_0 s_1 \ldots s_m$, $a_i a_{i+1} \ldots a_n$\$)
- Эта конфигурация представляет собой правую СФ $X_1 X_2 \ldots X_m a_i a_{i+1} \ldots a_n$
- Стартовое состояние синтаксического анализатора s_0 не представляет символ грамматики, а служит маркером дна стека
- Очередной шаг анализатора из конфигурации определяется текущим входным символом a_i и состоянием на вершине стека s_m путем обращения к записи $ACTION[s_m, a_i]$ в таблице действий

LR-анализ

- В результате обращения к $ACTION[s_m, a_i]$ получаются конфигурации (всего – 4):
  1. Если $ACTION[...]$ = перенос s, то анализатор выполняет перенос в стек состояния s и очередной конфигурацией становится $(s_0 s_1 ... s_m s, a_{i+1} ... a_n \$)$

LR-анализ

- В результате обращения к $ACTION[s_m, a_i]$ получаются конфигурации(всего – 4):
  2. Если $ACTION[...]$ = свертка $A \to \beta$, то анализатор выполняет свертку и его конфигурацией становится $(s_0 s_1 ... s_{m-r} s, a_i a_{i+1} ... a_n \$)$
     - $r = |\beta|$, а $s = GOTO[s_{m-r}, A]$
     - Анализатор вначале снимает r символов состояний с вершины стека, что переносит на вершину стека состояние s_{m-r}, после чего на вершину стека помещается s, запись из $GOTO[s_{m-r}, A]$
     - Текущий входной символ не изменяется
     - В LR-анализаторах последовательность символов $X_{m-r+1} ... Xm$ всегда соответствует β, RHS продукции свертки

LR-анализ

- В результате обращения к *ACTION*[s_m, a_i] получаются конфигурации(всего – 4):

3. Если *ACTION*[...] = принятие, то синтаксический анализ завершается
4. Если *ACTION*[...] = ошибка, то анализатор обнаруживает ошибку и вызывает подпрограмму восстановления после ошибки

LR-анализ

- Алгоритм LR-анализа
  - ВХОД: входная строка *w* и таблица LR-анализа с функциями *ACTION* и *GOTO* для грамматики *G*
  - ВЫХОД: если *w* принадлежит *L*(*G*), то шаги сверток восходящего синтаксического анализа *w*; в противном случае – указание на ошибку
  - МЕТОДИКА: изначально в стеке анализатора находится начальное состояние s_0, а во входном буфере – *w*$
  - Далее выполняется программа (псевдокод – на следующем слайде)

LR-анализ

```
Пусть a — первый символ w$
while (!0) /* Бесконечный цикл */
{
  Пусть s — состояние на вершине стека.
  if (ACTION[s, a] = перенос t)
  {
     Внести t в стек.
     Присвоить a очередной входной символ.
  }
  else if (ACTION[s, a] = свертка A —> β)
  {
    Снять |β| символов со стека.
    Теперь на вершине стека находится состояние t.
    Внести в стек GOTO [t, A].
    Вывести продукцию A —> β.
  }
  else if(ACTION[s, a] = принятие)
  {
    break; /* Синтаксический анализ завершен */
  }
  else
  {
    Вызов подпрограммы восстановления после ошибки.
  }
}
```

LR-анализ

JFLAP : (SLR1-1.jff)

File Input Test Convert Help

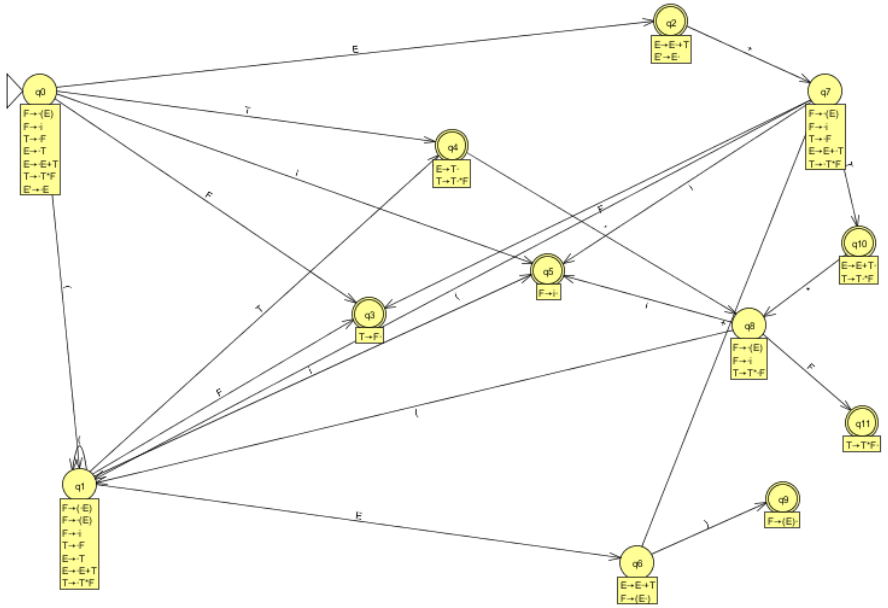
Editor | Build SLR(1) Parse

Do Selected | Do Step | Do All | Next | Parse

E'	⟶	E
E	⟶	E+T
E	⟶	T
T	⟶	T*F
T	⟶	F
F	⟶	(E)
F	⟶	i

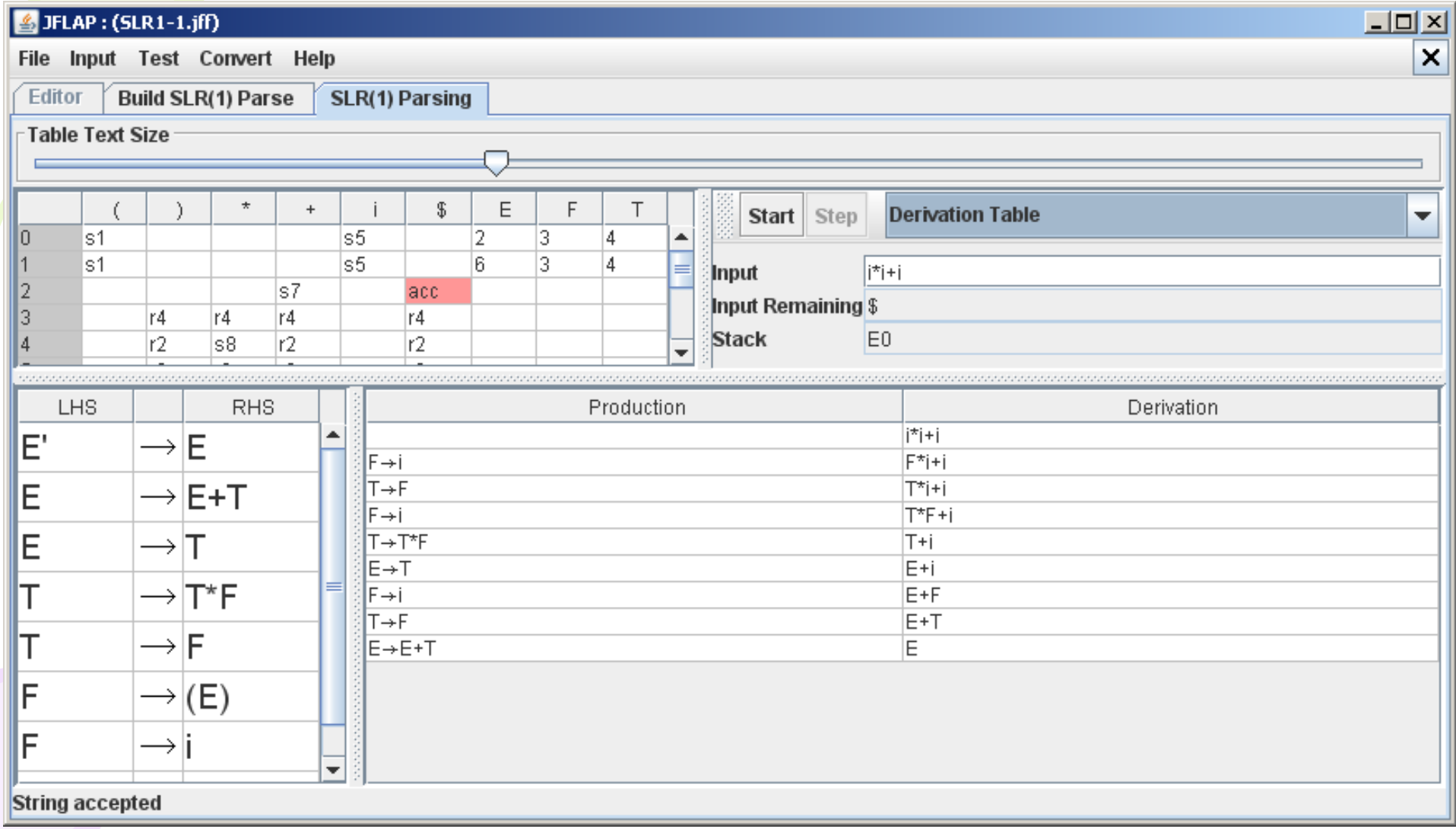
Parse table complete. Press "parse" to use it.

	FIRST	FOLLOW
E	{ (, i }	{ $, +,) }
F	{ (, i }	{ $, *, +,) }
T	{ (, i }	{ $, *, +,) }



	(	)	*	+	i	$	E	F	T
0	s1				s5		2	3	4
1	s1				s5		6	3	4
2				s7		acc			
3		r4	r4	r4		r4			
4		r2	s8	r2		r2			
5		r6	r6	r6		r6			
6		s9		s7					
7	s1				s5			3	10
8	s1				s5			11	
9		r5	r5	r5		r5			
10		r1	s8	r1		r1			
11		r3	r3	r3		r3			

LR-анализ


JFLAP : (SLR1-1.jff)
File Input Test Convert Help
Editor Build SLR(1) Parse SLR(1) Parsing
Table Text Size
( ) * + i $ E F T
0 s1 s5 2 3 4
1 s1 s5 6 3 4
2 s7 acc
3 r4 r4 r4 r4
4 r2 s8 r2 r2
Start Step Derivation Table
Input i*i+i
Input Remaining $
Stack E0
LHS RHS
E' → E
E → E+T
E → T
T → T*F
T → F
F → (E)
F → i
Production Derivation
i*i+i
F→i F*i+i
T→F T*i+i
F→i T*F+i
T→T*F T+i
E→T E+i
F→i E+F
T→F E+T
E→E+T E
String accepted

LR-анализ

- ТСА, построенные методом SLR(1) - **SLR-таблицы**
- LR-анализатор, использующий SLR-таблицы, – **SLR-анализатор**
- SLR-метод начинается с LR(0)-ситуаций и LR(0)-автомата
  - Для данной грамматики *G* мы строим ее расширение *G'* с новой аксиомой *S'*
  - Для *G'* строится канонический набор *C* множеств ситуаций *G'* вместе с функцией *GOTO*
  - Затем строятся записи *ACTION* и *GOTO* в ТСА с использованием алгоритма, который требует знания *FOLLOW*(*A*) для каждого нетерминала *A* грамматики

LR-анализ

- ВХОД: расширенная грамматика *G*'
- ВЫХОД: функции таблицы SLR-анализа *ACTION* и *GOTO* для *G*'
- *Методика*: выполняются действия (всего – 5):

1. Строится $C = \{I0, I1, ..., In\}$ – набор множеств LR(0)-ситуаций для *G*'

LR-анализ

- ВХОД: расширенная грамматика *G*'
- ВЫХОД: функции таблицы SLR-анализа *ACTION* и *GOTO* для *G*'
- *Методика*: выполняются действия (всего – 5):

2. Из *Ii* строится состояние *i*, а действие СА для состояния *i* определяем так:

   а) Если [*A* -> *α* • *aβ*] принадлежит *Ii* и *GOTO*(*Ii*, *a*) = *Ij*, то устанавливаем *ACTION*[*i*, *a*] = «перенос *j*

   б) Если [*A* -> *α* •] принадлежит *Ii*, то устанавливаем *ACTION*[*i*, *a*] = «свертка *A* -> *a*» для всех *a* из *FOLLOW*(*A*)

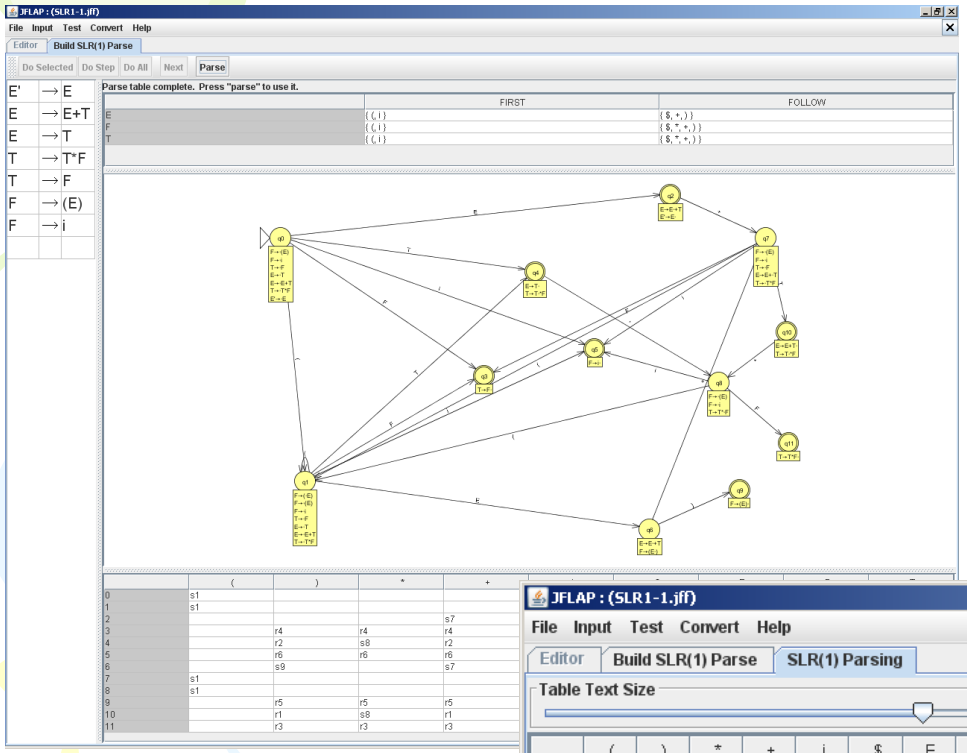
   в) Если [*S*' -> *S* •] принадлежит *Ii*, то устанавливаем *ACTION*[*i*, $] = «принятие»

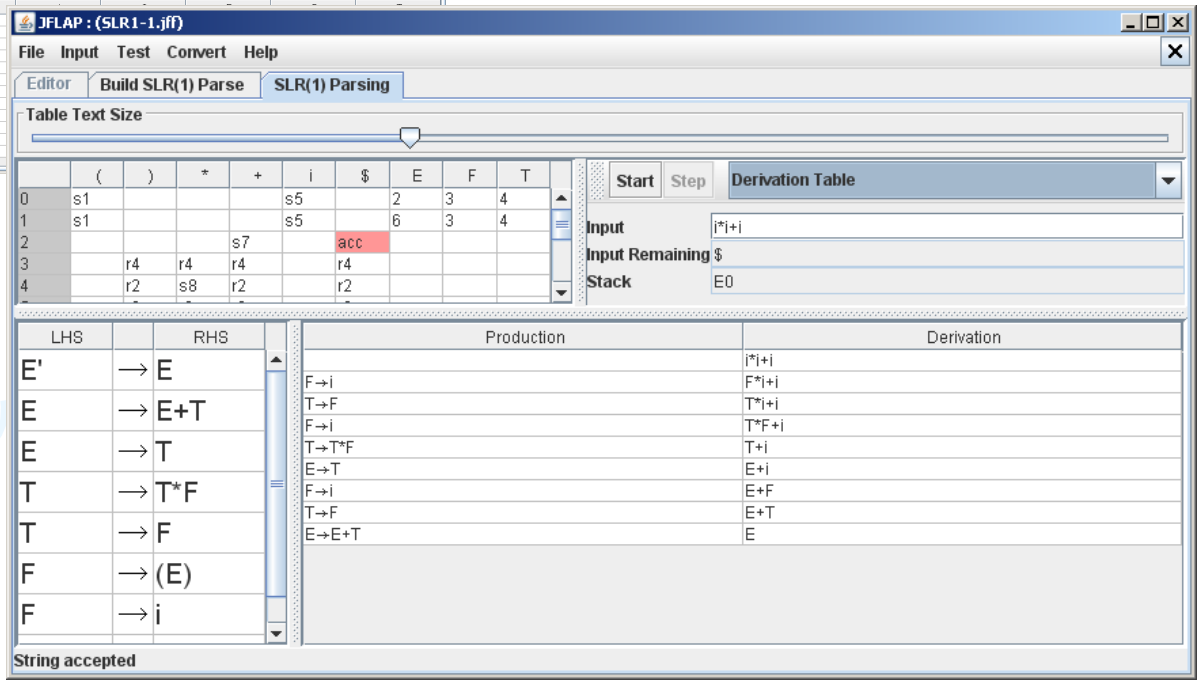
LR-анализ

- ВХОД: расширенная грамматика *G'*
- ВЫХОД: функции таблицы SLR-анализа *ACTION* и *GOTO* для *G'*
- *Методика*: выполняются действия (всего – 5):

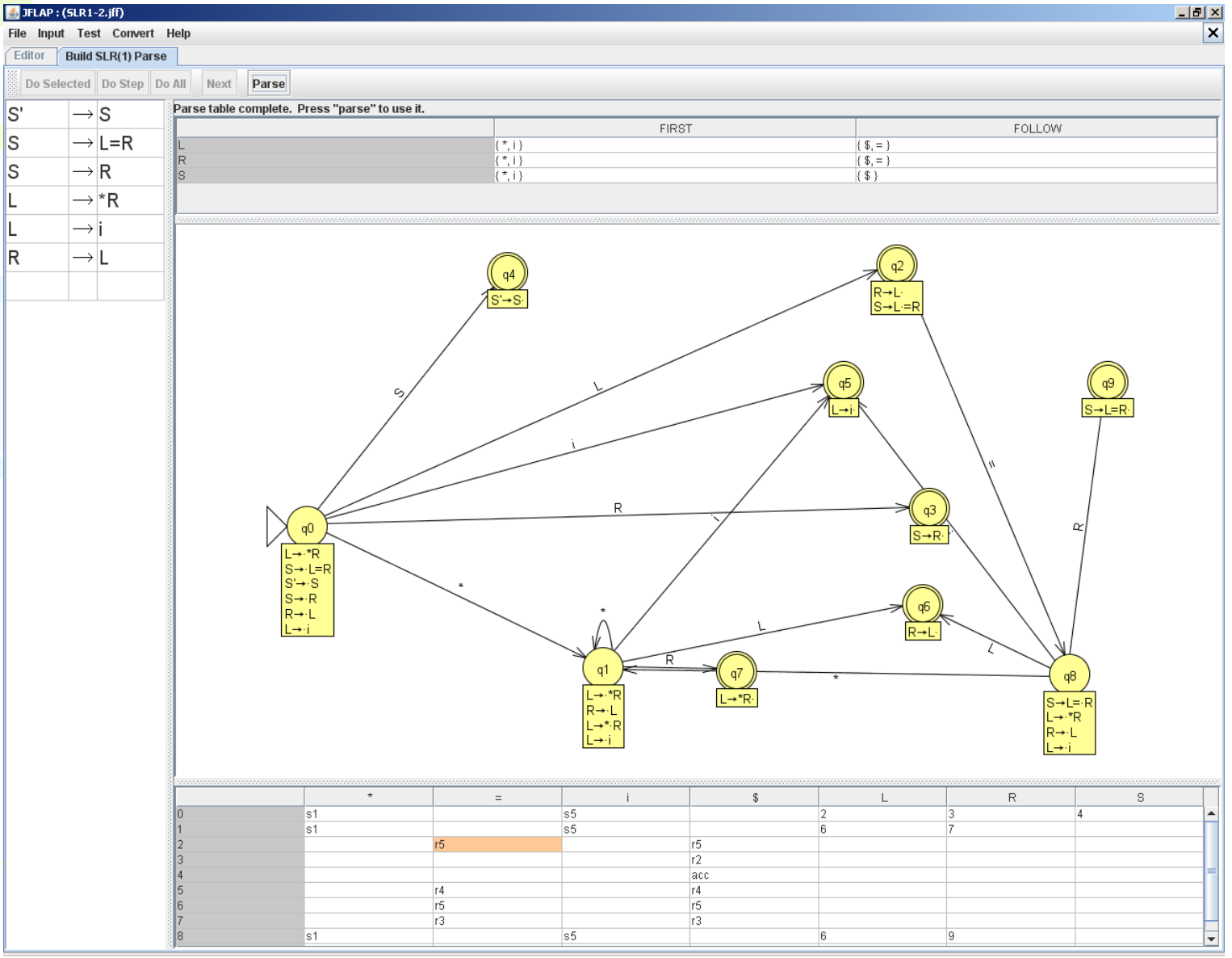
3. Переходы для состояния *i* строим для всех нетерминалов *A* с использованием следующего правила: если *GOTO*(*Ii*, *A*) = *Ij*, то *GOTO*[*i*, *A*] = *j*
4. Все записи, не определенные правилами 2 и 3, получают значение «ошибка»
5. Начальное состояние синтаксического анализатора строится из множества ситуаций, содержащего [*S'* -> • *S*]

LR-анализ


JFLAP : (SLR1-1.jff)
File Input Test Convert Help
Editor Build SLR(1) Parse
Do Selected Do Step Do All Next Parse
Parse table complete. Press "parse" to use it.
E' → E
E → E+T
E → T
T → T*F
T → F
F → (E)
F → i
FIRST
FOLLOW


JFLAP : (SLR1-1.jff)
File Input Test Convert Help
Editor Build SLR(1) Parse SLR(1) Parsing
Table Text Size
Start Step Derivation Table
Input i*i+i
Input Remaining $
Stack E0
LHS RHS
E' → E
E → E+T
E → T
T → T*F
T → F
F → (E)
F → i
Production Derivation
String accepted

Построение таблиц SLR-анализа



Активные префиксы

- Содержимое стека должно быть префиксом правосентенциальной формы
- Если в стеке хранится *a*, а оставшаяся часть входной строки – *x*, то последовательность сверток должна привести *ax* в *S*
  - В терминах порождений $S \Rightarrow^* ax$
- Предположим, например, $E \Rightarrow^* F * i \Rightarrow (E) * i$
- В процессе СА в стеке хранятся {(, (*E* и (*E*)}, но в нем не должно быть (*E*)*, т.к. (*E*) - основа, которую анализатор должен свернуть в *F* до переноса *

Активные префиксы

- Префиксы правосентенциальных форм, которые могут находиться в стеке ПС-анализатора - **активные префиксы**
- Активный префикс является префиксом правосентенциальной формы, не выходящим за пределы правого конца крайней справа основы СФ
- К концу активного префикса всегда можно добавить терминальные символы для получения правосентенциальной формы
- SLR-анализ основан на том факте, что LR(0)-автомат распознает активные префиксы

Активные префиксы

- Мы говорим, что ситуация $A \to \beta 1 \bullet \beta 2$ **допустима** для активного префикса $\alpha\beta 1$, если существует правое порождение $S' \Rightarrow^* \alpha A w \Rightarrow \alpha\beta 1\beta 2 w$
- Тот факт, что $A \to \beta 1 \bullet \beta 2$ допустима для $\alpha\beta 1$, многое говорит о том, что следует выбрать – перенос или свертку – при обнаружении $\alpha\beta 1$ на вершине стека
- Если $\beta 2 \neq \varepsilon$, то основа еще не полностью перенесена в стек и очередное действие анализатора – перенос
- Если $\beta 2 = \varepsilon$, то $A \to \beta 1$ – основа, и анализатор должен выполнить свертку

Активные префиксы

$$E' \underset{rm}{\Rightarrow} E \underset{rm}{\Rightarrow} E+T \underset{rm}{\Rightarrow} E+T*F$$

$$E' \underset{rm}{\Rightarrow} E \underset{rm}{\Rightarrow} E+T \underset{rm}{\Rightarrow} E+T*F \underset{rm}{\Rightarrow} E+T*(E)$$

$$E' \underset{rm}{\Rightarrow} E \underset{rm}{\Rightarrow} E+T \underset{rm}{\Rightarrow} E+T*F \underset{rm}{\Rightarrow} E+T*\mathbf{id}$$

- НКА *N* для распознавания активных префиксов может быть построен путем рассмотрения ситуаций в качестве состояний
- Существует переход из *A* -> *a* • *Xβ* в *A* -> *aX* • *β*, помеченный *X*, и переход из *A* -> *a* • *Bβ* в *B* -> • *γ*, помеченный *ε*

Активные префиксы

- *CLOSURE*(*I*) для множества ситуаций (или состояний *N*) *I* в точности представляет собой *ε*-замыкание множества состояний НКА
- Таким образом, *GOTO*(*I*, *X*) дает переход из *I* для символа *X* в ДКА, построенном из *N* при помощи метода конструирования подмножеств

Дополнительные источники

- LR(0) - http://ru.wikipedia.org/wiki/LR(0)
- SLR(1) - http://ru.wikipedia.org/wiki/SLR(1)
- Ахо, А. Компиляторы: принципы, технологии и инструментарий, 2 издание / А. Ахо, М.Лам, Р. Сети, Дж. Ульман. – М.: Издательский дом «Вильямс», 2008. – 1184 с.
- LR-анализатор - http://ru.wikipedia.org/wiki/LR
- The Lex & Yacc Page - http://dinosaur.compilertools.net/