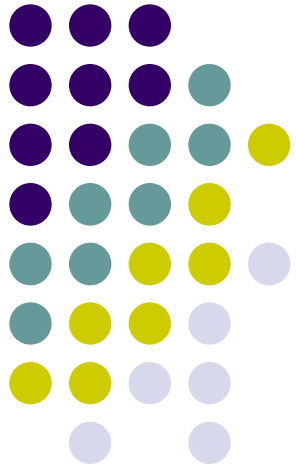
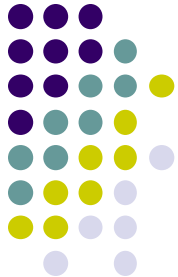
Теория языков программирования и методы трансляции

Тема №5

Восходящий синтаксический анализ (down-top)



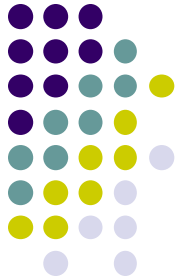
Вопросы



В данной теме описывается восходящий синтаксический анализ и его реализация. В частности, будут рассмотрены следующие вопросы.

- Этапы восходящего синтаксического анализа и критерии принятия решений.
- Использование таблицы синтаксического анализа для направления процесса синтаксического анализа.
- Ключевые свойства восходящего синтаксического анализа.
- Генератор синтаксических анализаторов, именуемый YACC, и использование данного средства для создания различных простых синтаксических инструментов.

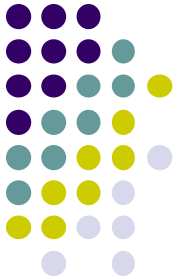
Восходящий синтаксический анализ. Основные понятия.



Задача синтаксического анализа состоит в нахождении порождения конкретного выражения с использованием данной грамматики. При восходящем синтаксическом анализе искомым обычно является *правое порождение*.

При восходящем синтаксическом анализе мы начинаем с имеющегося предложения языка, которое впоследствии *сворачивается* в символ предложения.

Восходящий синтаксический анализ. Основные понятия.



Рассмотрим язык $\{x^n y^m | n,m>0\}$, генерируемый следующими продукциями:

$S \to XY$

$X \to xX$

$X \to x$

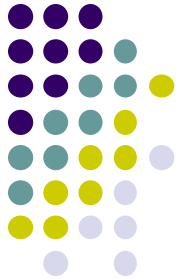
$Y \to yY$

$Y \to y$

предложение языка *xxxyy* можно сгенерировать при помощи *правого* порождения порождения

$S \Rightarrow XY \Rightarrow XyY \Rightarrow Xyy \Rightarrow xXyy \Rightarrow xxXyy \Rightarrow xxxyy$

Восходящий синтаксический анализ. Основные понятия.

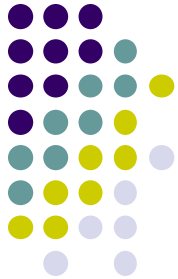


При восходящем синтаксическом анализе этапы порождения определяются не в показанном, а в *противоположном* порядке, а правый *синтаксический анализ,* который соответствует приведенному выше порождению, можно записать в следующем виде.

$$xxxyy \Rightarrow xxXyy \Rightarrow xXyy \Rightarrow Xyy \Rightarrow XyY \Rightarrow XY \Rightarrow S$$

На каждом этапе применяется продукция грамматики; правая часть продукции заменяется ее левой частью, которая состоит из одного символа.

Восходящий синтаксический анализ. Основные понятия.

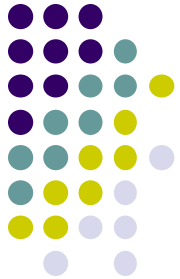


Пусть предложение

xxxyy

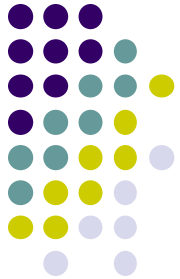
читается слева направо, тогда совсем не очевидно, почему первый *x* не заменяется *X* на первом этапе синтаксического анализа, используя соответствующую продукцию грамматики, или почему первый *y* подобным образом на четвертом этапе не заменяется *Y*. Очевидно, что для выполнения синтаксического анализа требуется некоторая дополнительная информация, отличная от предоставляемой продукциями грамматики.

Восходящий синтаксический анализ. Критерии принятия решений.



При восходящем синтаксическом анализе правые части продукций не распознаются, пока не будут полностью считаны, следовательно, существует необходимость хранения частично распознанных правых частей продукций, пока они не будут заменены соответствующими левыми частями. Для запоминания частично распознанных строк подходящей структурой является *стек*. Таким образом, подробное описание процесса восходящего синтаксического анализа включает отображение содержимого этого стека, а также информации, указанной в связи с нисходящим синтаксическим анализом

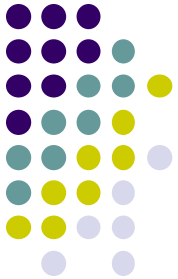
Восходящий синтаксический анализ. Критерии принятия решений.



Этапы процесса синтаксического анализа могут **рассматриваться** как состоящие из двух типов действий.

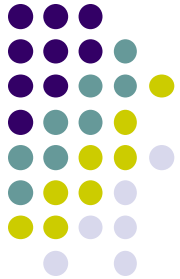
1. Перемещение последнего считанного символа в стек – действие *переноса* (shift action).
2. Замена строки наверху стека посредством применения продукции грамматики – действие *свертки* (reduce action).

Восходящий синтаксический анализ. Критерии принятия решений.



Входная строка	Стек	Продукция	Сентенциальная форма	Перенос(S)/Свертка(R)
$xxxyy$			$xxxyy$	
~~x~~$xxyy$	x		$xxxyy$	(S)
~~xx~~xyy	xx		$xxxyy$	(S)
~~xxx~~yy	xxx		$xxxyy$	(S)
~~xxx~~yy	xxX	$X \to x$	$xxXyy$	(R)
~~xxx~~yy	xX	$X \to xX$	$xXyy$	(R)
~~xxx~~yy	X	$X \to xX$	Xyy	(R)
~~$xxxy$~~y	Xy		Xyy	(S)
~~$xxxyy$~~	Xyy		Xyy	(S)
~~$xxxyy$~~	XyY	$Y \to y$	XyY	(R)
~~$xxxyy$~~	XY	$Y \to yY$	XY	(R)
~~$xxxyy$~~	S	$S \to XY$	S	(R)

Восходящий синтаксический анализ. Критерии принятия решений.

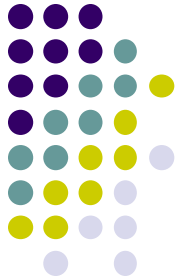


В приведенном выше примере не ясно, когда должны применяться действия переноса и свертки и как производится выбор, если возможными являются несколько операций свертки.

Необходимым условием действия свертки является наличие правой части некоторой продукции на вершине стека, в противном случае производится перенос, и на вершине стека появится следующий символ.

Появление правой части продукции на вершине стека не является *достаточным* условием для применения свертки. Если на первых этапах синтаксического анализа на вершине стека появляется *x,* он *не* сворачивается в *X* по *неясным,* на первый взгляд, причинам.

Восходящий синтаксический анализ. Основные понятия.

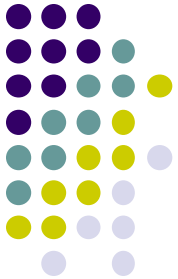


В строке символов на вершине стека возможно будут определены правые части более одной продукции, так что на определенном этапе синтаксического анализа могут существовать две или более возможных свертки.

Если, в определенный момент кажутся возможными действия свертки и переноса, говорят, что имеет место *конфликт перенос/свертка* (shift-reduce conflict). Если возможными кажутся несколько операций свертки, говорят, что имеет место *конфликт свертка/свертка* (reduce-reduce conflict). С целью выработки детерминированного метода синтаксического анализа могут применяться различные стратегии разрешения названных конфликтов. На практике данные конфликты разрешаются с использованием следующей информации

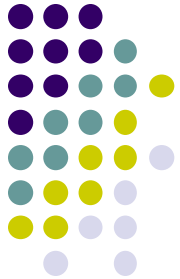
- Предшествующая история синтаксического анализа.
- Информация, полученная путем предпросмотра.

Восходящий синтаксический анализ. Основные понятия.



Как и при нисходящем синтаксическом анализе, для разрешения конфликтов обычно используется один символ предпросмотра. Кроме того, для разрешения конфликтов может использоваться информация, касающаяся истории синтаксического анализа. В приведенном выше примере символом предпросмотра, определяющим применение продукции $X \rightarrow x$ был y; подобным образом, символ предпросмотра $\perp$ (маркер конца) определяет применение такой продукции: $Y \rightarrow y$

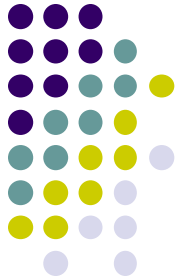
Восходящий синтаксический анализ. *LR*(*k*)-грамматика.



Грамматика, все конфликты которой, возникающие при восходящем синтаксическом анализе слева направо, могут быть разрешены с использованием фиксированного объема информации, касающейся уже проведенного анализа, и конечного числа символов предпросмотра, называется *LR*(*k*)-*грамматикой.* Здесь L означает чтение слева (L*eft*) направо, R – правые порождения (R*ightmost*), а *k* обозначает количество символов предпросмотра. *Языком LR*(*k*) называется язык, который можно сгенерировать посредством *LR(k)* -грамматики,

Если требуется только один символ предпросмотра, грамматика и язык относятся к классу *LR*(1).

Восходящий синтаксический анализ. Критерии принятия решений.



Рассмотрим пример. Пусть имеется грамматика со следующими продукциями.

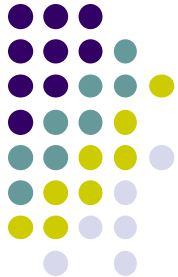
1. $E \rightarrow E+T$
2. $E \rightarrow T$
3. $T \rightarrow T*F$
4. $T \rightarrow F$
5. $F \rightarrow (E)$
6. $F \rightarrow x$

Здесь E – символ предложения. Грамматика может использоваться в качестве основы для восходящего синтаксического анализа

Восходящий синтаксический анализ. Критерии принятия решений.

Входная строка	Стек	Продукция	Сентенциальная форма	Перенос(S)/Свертка(R)
x+x+x*x			x+x+x*x	
~~x~~+x+x*x	x		x+x+x*x	(S)
~~x~~+x+x*x	F	F→x	F+x+x*x	(R)
~~x~~+x+x*x	T	T→F	T+x+x*x	(R)
~~x~~+x+x*x	E	E→T	E+x+x*x	(R)
~~x~~+x+x*x	E+		E+x+x*x	(S)
~~x+x~~+x*x	E+x		E+x+x*x	(S)
~~x+x~~+x*x	E+F	F→x	E+F+x*x	(R)
~~x+x~~+x*x	E+T	T→F	E+T+x*x	(R)
~~x+x~~+x*x	E	E→E+T	E+x*x	(R)
~~x+x+~~x*x	E+		E+x*x	(S)
~~x+x+~~x*x	E+x		E+x*x	(S)
~~x+x+x~~*x	E+F	F→x	E+F*x	(R)
~~x+x+x~~*x	E+T	T→F	E+T*x	(R)
~~x+x+x*~~x	E+T*		E+T*x	(S)
~~x+x+x*x~~	E+T*x		E+T*x	(S)
~~x+x+x*x~~	E+T*F	F→x	E+T*F	(R)
~~x+x+x*x~~	E+T	T→T*F	E+T	(R)
~~x+x+x*x~~	E	E→E+T	E	(R)

Восходящий синтаксический анализ. Критерии принятия решений.



Первый *x,* помещаемый в стек, сворачивается в *F*, затем в *T*, затем в *E,* тогда как второй и третий *x* сворачиваются в *F*, потом в *T*, а четвертый *x* – только в *F,* Первый и второй символы *x* имеют одинаковые символы предпросмотра, и различная их трактовка основана на предшествующей истории синтаксического анализа. Для третьего и четвертого символов *x* символы предпросмотра отличаются (* и $\perp$, соответственно), и снова имеем различную трактовку — и снова, основанную на истории синтаксического анализа. Критерий принятия решения относительно предпринимаемого действия – переноса или свертки (или выбора из нескольких возможных сверток) – может содержаться в таблице, называемой таблицей синтаксического анализа.

Поскольку таблица синтаксического анализа создается инструментальным средством, наподобие YACC, программисту нет нужды понимать принципы ее формирования. Впрочем, некоторые вопросы ее использования все же стоит рассмотреть.

Восходящий синтаксический анализ. Таблица синтаксического анализа.

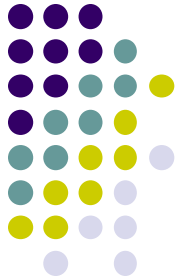
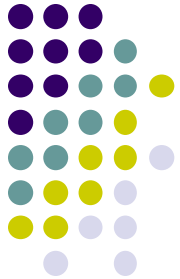


Таблица синтаксического анализа, которая используется в восходящем синтаксическом анализе, является прямоугольной, каждому состоянию анализатора (всегда конечное число) соответствует одна строка, а каждому терминалу и нетерминалу грамматики соответствует один столбец.

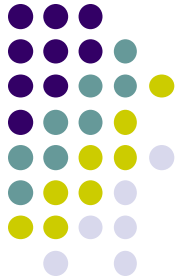
Восходящий синтаксический анализ. Таблица синтаксического анализа.



В начале процесса синтаксического анализа анализатор находится в состоянии 1, а входным символом является первый введенный символ. Каждый шаг анализа определяется позицией таблицы, соответствующей *текущему состоянию,* и *входным символом***.** Позиция таблицы может принадлежать к одному из двух типов.

1. Позиция *переноса* вида *Sm,* вынуждающая анализатор выполнить действие переноса и изменить текущее состояние на состояние *m*.
2. Позиция *свертки* вида *Rn,* вынуждающая анализатор выполнить действие свертки, используя продукцию *n*.

Восходящий синтаксический анализ. Таблица синтаксического анализа.



Пустые позиции таблицы соответствуют синтаксическим ошибкам во вводе, и, при желании, с каждой такой позицией можно соотнести индивидуальное сообщение об ошибке. На практике таблицы синтаксического анализа могут быть очень большими, но они часто сжимаются, в основном, за счет удаления различных пустых позиций и увеличения времени обращения к элементам таблицы. В любом случае точные сообщения об ошибках, предоставляемые анализатором, часто немногого стоят, поскольку анализатор не предполагает, *что будет делать пользователь при обнаружении ошибки.*

Восходящий синтаксический анализ. Таблица синтаксического анализа.

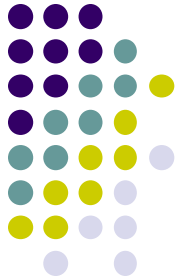
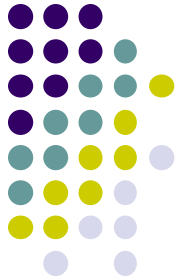


Таблица синтаксического анализа представляет зависимую от языка часть синтаксического анализатора, остальная часть анализатора является полностью или преимущественно независимой от языка и состоит из *драйверной программы*, которая интерпретирует данные в таблице синтаксического анализа и выполняет подходящие действия. Если зависимая от языка часть анализатора (таблица синтаксического анализа) может быть довольно большой, независимая от языка часть, скорее всего, невелика и переносится с одного компьютера на другой.

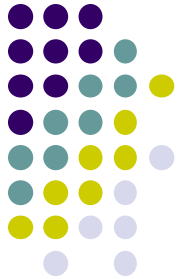
Восходящий синтаксический анализ. Таблица синтаксического анализа.



На каждом этапе синтаксического анализа анализатор находится в одном из конечного числа состояний, и это состояние плюс *входной символ* (либо символ предпросмотра, либо только что свернутый нетерминал) определяют элемент в таблице синтаксического анализа. Предполагая отсутствие синтаксических ошибок, этот элемент – действие переноса или свертки. В начале синтаксического анализа анализатор находится в состоянии 1, а входной символ – это первый символ анализируемого предложения. Если позиция таблицы определяет *действие переноса,* имеют место следующие операции:

- Символ, соответствующий столбцу, в котором находится данная позиция таблицы, заносится в стек символов.
- Анализатор переходит в стек, который определяется позицией переноса, и *это* состояние заносится в стек состояний.
- Если входной символ является терминалом, он принимается, и входным символом становится следующий терминал предложения (или маркер конца).

Восходящий синтаксический анализ. Таблица синтаксического анализа.



Если позиция таблицы определяет *действие свертки,* имеют место следующие операции.

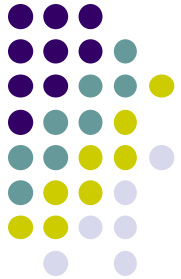
- Из стека символов удаляются *n* символов и из стека состояний удаляются *n* состояний, где *n* – число символов в правой части продукции, фигурирующей в свертке.
- Анализатор переходит в состояние на вершине стека состояний.
- Входной символ становится символом в левой части продукции, определенной в позиции свертки.

Восходящий синтаксический анализ. Таблица синтаксического анализа.

Таблица 1

	E	*T*	*F*	+	*	(	)	*x*	⊥
1	S2	S5	S8			S9		S12	
2				S3					
3		S4	S8			S9		S12	
4				R1	S6		R1		R1
5				R2	S6		R2		R2
6			S7			S9		S12	
7				R3	R3		R3		R3
8				R4	R4		R4		R4
9	S10	S5	S8			S9		S12	
10				S3			S11		
11				R5	R5		R5		R5
12				R6	R6		R6		R6

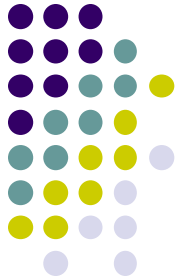
Восходящий синтаксический анализ. Таблица синтаксического анализа.



Проследим анализ следующего предложения. $x + x + x*x$

Входная строка	Стек состояний	Стек символов	Сентенциальная форма
$x + x + x*x$	1		$x + x + x*x$
~~x~~$+ x + x*x$	1,12	x	$x + x + x*x$
~~x~~$+ x + x*x$	1		$x + x + x*x$
~~x~~$+ x + x*x$	1,8	F	$F+ x + x*x$
~~x~~$+ x + x*x$	1		$F+ x + x*x$
~~x~~$+ x + x*x$	1,5	T	$T+ x + x*x$
~~x~~$+ x + x*x$	1		$T+ x + x*x$
~~x~~$+ x + x*x$	1,2	E	$E+ x + x*x$
~~$x +$~~ $x + x*x$	1,2,3	$E+$	$E+ x + x*x$
~~$x + x$~~ $+ x*x$	1,2,3,12	$E+x$	$E+ x + x*x$
~~$x + x$~~ $+ x*x$	1,2,3	$E+$	$E+ x + x*x$
~~$x + x$~~ $+ x*x$	1,2,3,8	$E+F$	$F+ x + x*x$
……	……	… … … … .	……
~~$x + x$~~ $+ x*x$	1,2	E	$E + x*x$
……	……	… … … … .	……
~~$x + x + x*x$~~	1,2	E	E

Восходящий синтаксический анализ. Таблица синтаксического анализа.



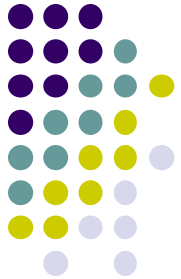
Рассмотрев, как синтаксический анализатор использует одноименную таблицу, пришло время определить принципы ее формирования из контекстно-свободной грамматики. Вначале проиллюстрируем роль состояний, рассмотрев создание таблицы синтаксического анализа для простой грамматики, генерирующей конечный язык.

Рассмотрим грамматику со следующими продукциями (*P* – символ предложения).

1. $P \rightarrow bD;Se$
2. $D \rightarrow d;d$
3. $S \rightarrow s;s$

bd;d;s;se

Восходящий синтаксический анализ. Таблица синтаксического анализа.



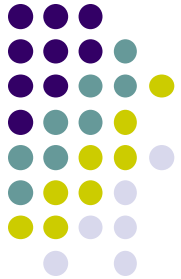
Как обычно, предполагается, что синтаксический анализатор изначально находится в состоянии 1. Для отображения этого факта следует несколько изменить форму записи грамматики

1. $P \rightarrow_1 bD;Se$
2. $D \rightarrow d;d$
3. $S \rightarrow s;s$

После считывания символа b грамматика будет находиться в состоянии 2.

1. $P \rightarrow_1 b_2 D;Se$
2. $D \rightarrow d;d$
3. $S \rightarrow s;s$

Восходящий синтаксический анализ. Таблица синтаксического анализа.



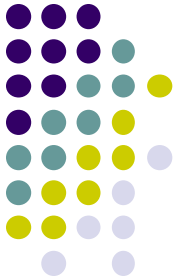
Анализатор находится в состоянии 2 до распознавания символа *D*, так что соответствующее обозначение помещаем и в начало продукции для нетерминала *D*

1. $P \rightarrow_1 b_2 D; Se$
2. D $\rightarrow_2 d; d$
3. S $\rightarrow s; s$

Состояния 3 и 4 также соответствуют считыванию символов продукции 1. Поскольку анализатор находится в состоянии 4 до распознавания символа *S*, оно также соответствует началу продукции для *S*.

1. $P \rightarrow_1 b_2 D_3;_4 Se$
2. D $\rightarrow_2 d; d$
3. S $\rightarrow_4 s; s$

Восходящий синтаксический анализ. Таблица синтаксического анализа.

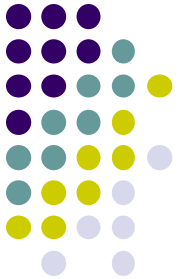


Подобным образом вводим еще восемь состояний, в которых может находиться синтаксический анализатор.

1. $P \rightarrow {}_1b_2D_3;{}_4S_5e_6$
2. $D \rightarrow {}_2d_7;{}_8d_9$
3. $S \rightarrow {}_4s_{10};{}_{11}s_{12}$

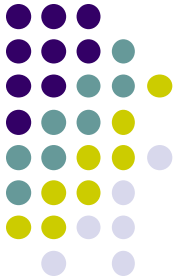
Вся информация, необходимая для направления процесса анализа, содержится в представленной выше аннотированной грамматике, но удобнее ее представить в табличной форме.

Восходящий синтаксический анализ. Таблица синтаксического анализа.



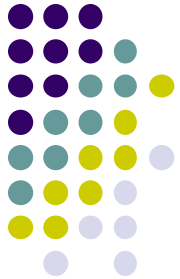
Позиции переноса заносятся в таблицу легко. Например, из начала правой части продукции 1, состояние 1, при входном символе *b* очевидным является перенос в состояние 2, другие позиции переноса также очевидны. Рассмотрим теперь позиции свертки. Состояние в конце продукции – это состояние, в котором должна происходить свертка, так что из аннотированной версии продукции 1 видим, что в состоянии 6 должна иметь место свертка согласно продукции 1. Действия свертки вносятся в каждый столбец состояния свертки, *если* в рассматриваемой строке нет действий переноса; по этой причине действия переноса всегда заносятся в таблицу до действий свертки. Позднее будет рассмотрен вопрос о том, что происходит при возникновении конфликта между действиями переноса и свертки (конфликт перенос/свертка) или между двумя действиями свертки (конфликт свертка/свертка).

Восходящий синтаксический анализ. Таблица синтаксического анализа.



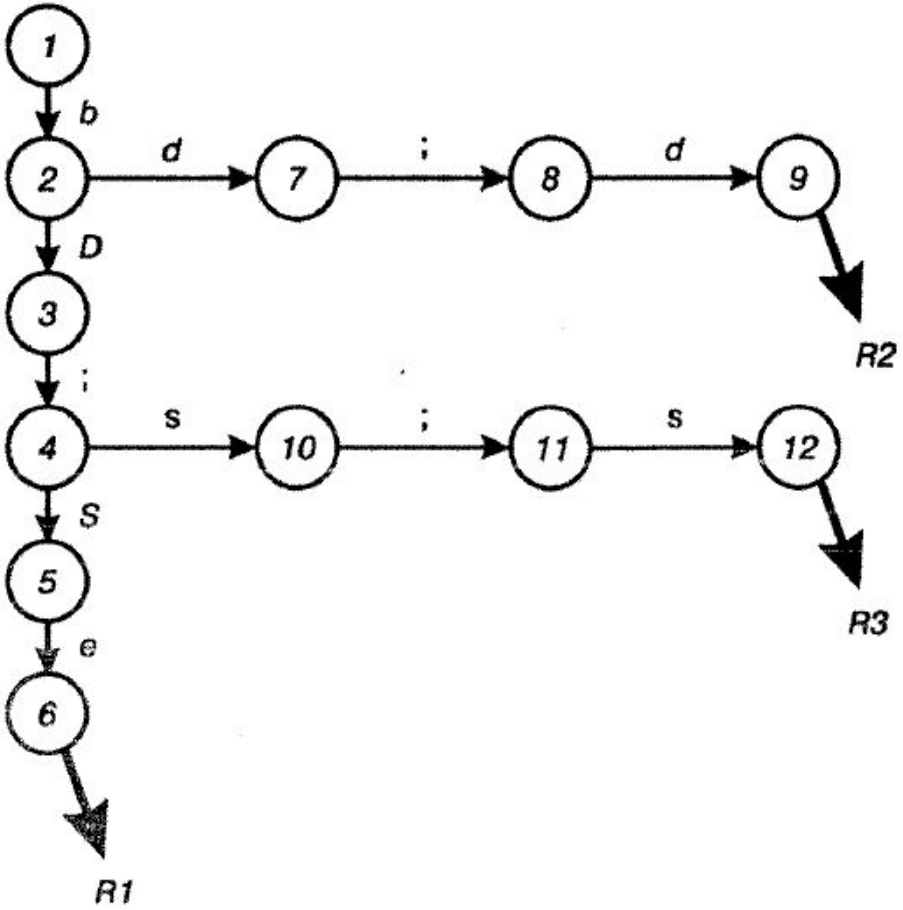
	P	***D***	***S***	***b***	***e***	***d***	***;***	***s***	⊥
1				S2					
2		S3				S7			
3							S4		
4			S5					S10	
5					S6				
6	R1	R1	R1	R1	R1	R1	R1	R1	R1
7							S8		
8						S9			
9	R2	R2	R2	R2	R2	R2	R2	R2	R2
10							S11		
11								S12	
12	R3	R3	R3	R3	R3	R3	R3	R3	R3

Восходящий синтаксический анализ. Таблица синтаксического анализа.

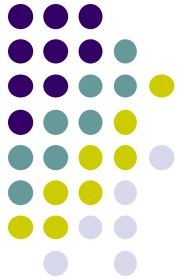


Другим способом представления таблицы синтаксического анализа является ориентированный граф. Это представление называется *характеристическим конечным автоматом* (characteristic finite state machine) далее ХКА. Если передать управление синтаксическим анализом данному конечному автомату и соотнести с каждым состоянием номер его вершины, то при действии *переноса* это номер будет заносится в стек состояний. При действии *свертки* поведение анализатора будет несколько иным. В этом случае из стека будет извлечено требуемое число состояний и проходится такое число обратно по конечному автомату. Как и ранее следующий входной символ – это символ в правой части продукции, только что использованной при свертке. ХКА формируется аналогично тому, как в грамматику добавлялись аннотации.

Восходящий синтаксический анализ. Таблица синтаксического анализа.



Восходящий синтаксический анализ. Таблица синтаксического анализа.

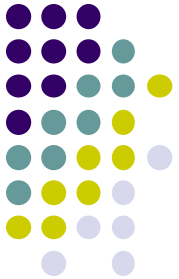


В большинстве случаев действия свертки поместить в таблицу не так просто, как может показаться из приведенного примера для иллюстрации вернемся к рассмотренной выше грамматике.

1. $E \rightarrow E+T$
2. $E \rightarrow T$
3. $T \rightarrow T*F$
4. $T \rightarrow F$
5. $F \rightarrow (E)$
6. $F \rightarrow x$

Аннотирование грамматики происходит следующим образом. В первую очередь устанавливается состояние 1.

Восходящий синтаксический анализ. Таблица синтаксического анализа.

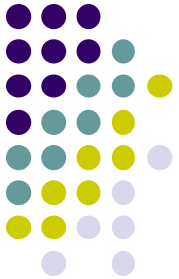


1. $E \rightarrow_1 E+T$
2. $E \rightarrow_1 T$
3. $T \rightarrow_1 T*F$
4. $T \rightarrow_1 F$
5. $F \rightarrow_1 (E)$
6. $F \rightarrow_1 x$

Каждая позиция, в которую было помещено состояние 1 – это пример *конфигурации* грамматики. По определению, *конфигурация* – это позиция в правой части продукции перед первым символом, после последнего символа или между двумя любыми символами.

Конфигурации, соответствующие одному состоянию, неразличимы с точки зрения синтаксического анализатора. Далее вводим в грамматику состояние 2, которое соответствует единственной конфигурации.

Восходящий синтаксический анализ. Таблица синтаксического анализа.

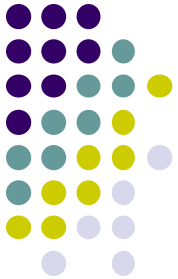


1. $E \rightarrow_1 E_2 + T$
2. $E \rightarrow_1 T$
3. $T \rightarrow_1 T * F$
4. $T \rightarrow_1 F$
5. $F \rightarrow_1 (E)$
6. $F \rightarrow_1 x$

Состояние 3, поскольку оно появляется перед нетерминалом, соответствует нескольким конфигурациям.

1. $E \rightarrow_1 E_2 +_3 T$
2. $E \rightarrow_1 T$
3. $T \rightarrow_{1,3} T * F$
4. $T \rightarrow_{1,3} F$
5. $F \rightarrow_{1,3} (E)$
6. $F \rightarrow_{1,3} x$

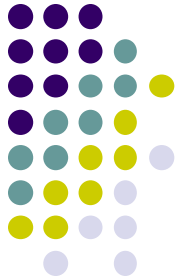
Восходящий синтаксический анализ. Таблица синтаксического анализа.



На первый взгляд может показаться странным наличие двух состояний, соответствующих одной конфигурации грамматики (например, оба состояния 1 и 3 появляются в начале правой части продукции 3). В то же время, если учитывать историю синтаксического анализа, два состояние *различимы,* поскольку состояние 3 всегда появляется после символа +, а состояние 1 – нет. Вводим далее состояние 4.

1. $E \rightarrow_1 E_2 +_3 T_4$
2. $E \rightarrow_1 T$
3. $T \rightarrow_{1,3} T_4 * F$
4. $T \rightarrow_{1,3} F$
5. $F \rightarrow_{1,3} (E)$
6. $F \rightarrow_{1,3} x$

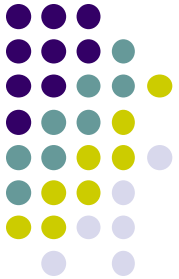
Восходящий синтаксический анализ. Таблица синтаксического анализа.



Состояние 4 появляется в двух местах: в конце продукции 1, где оно соответствует действию свертки, и поскольку оно определено как состояние, в которое переходит состояние 3 при считывании символа, после *T* в продукции 3. Уже видно, что состояние 4 приведет к некоторому конфликту перенос/свертка.

По подобной причине состояние 5 так же появляется в двух местах в конце продукции 2 (действие свертки) и в продукции 3 (перенос). Отметим что в продукции 3 состояния, идущие перед *T* в правой части, появляются в порядке, соответствующем состояниям, идущим после *T,* т.е. после состояния 1 идет состояние 5, а после 3 – состояние 4.

Восходящий синтаксический анализ. Таблица синтаксического анализа.

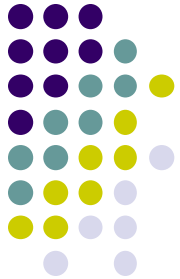


1. $E \rightarrow_{1} E_{2} +_{3} T_{4}$
2. $E \rightarrow_{1} T_{5}$
3. $T \rightarrow_{1,3} T_{5,4} * F$
4. $T \rightarrow_{1,3} F$
5. $F \rightarrow_{1,3} (E)$
6. $F \rightarrow_{1,3} x$

Состояние 6 появляется в нескольким местах, поскольку оно предшествует нетерминалу и далее просто вводятся состояния 7 и 8.

1. $E \rightarrow_{1} E_{2} +_{3} T_{4}$
2. $E \rightarrow_{1} T_{5}$
3. $T \rightarrow_{1,3} T_{5,4} *_{6} F_{7}$
4. $T \rightarrow_{1,3} F_{8}$
5. $F \rightarrow_{1,3,6} (E)$
6. $F \rightarrow_{1,3,6} x$

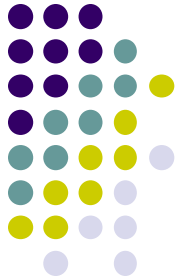
Восходящий синтаксический анализ. Таблица синтаксического анализа.



Состояние 9 несколько сложнее, так как оно появляется перед нетерминалом, поэтому должно находиться в начале правил для этого нетерминала и т.д. рекурсивно.

1. $E \rightarrow_{1,9} E_2 +_3 T_4$
2. $E \rightarrow_{1,9} T_5$
3. $T \rightarrow_{1,3,9} T_{5,4} *_6 F_7$
4. $T \rightarrow_{1,3,9} F_8$
5. $F \rightarrow_{1,3,6,9} (_9 E)$
6. $F \rightarrow_{1,3,6,9} x$

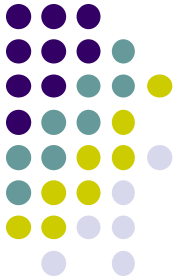
Восходящий синтаксический анализ. Таблица синтаксического анализа.



Состояние 10 следует вводить аккуратно. Состояние 10 следует после *E* в состоянии 9, таким образом, появляется после *E* в продукциях 1 и 5. В то же время, в продукциях 2 и 3 уже не нужно вводить новые состояния, которые бы "помнили", появилось это состояние из продукции в состоянии 1 или в состоянии 9. По этой кажущейся несколько условной причине любое новое состояние, введенное после *T* в данных продукциях, будет соответствовать набору конфигураций, идентичному существующему в уже введенном состоянии.

1. $E \rightarrow_{1,9} E_{2,10} +_{3} T_{4}$
2. $E \rightarrow_{1,9} T_{5}$
3. $T \rightarrow_{1,3,9} T_{5,4} *_{6} F_{7}$
4. $T \rightarrow_{1,3,9} F_{8}$
5. $F \rightarrow_{1,3,6,9} (_{9} E_{10})$
6. $F \rightarrow_{1,3,6,9} x$

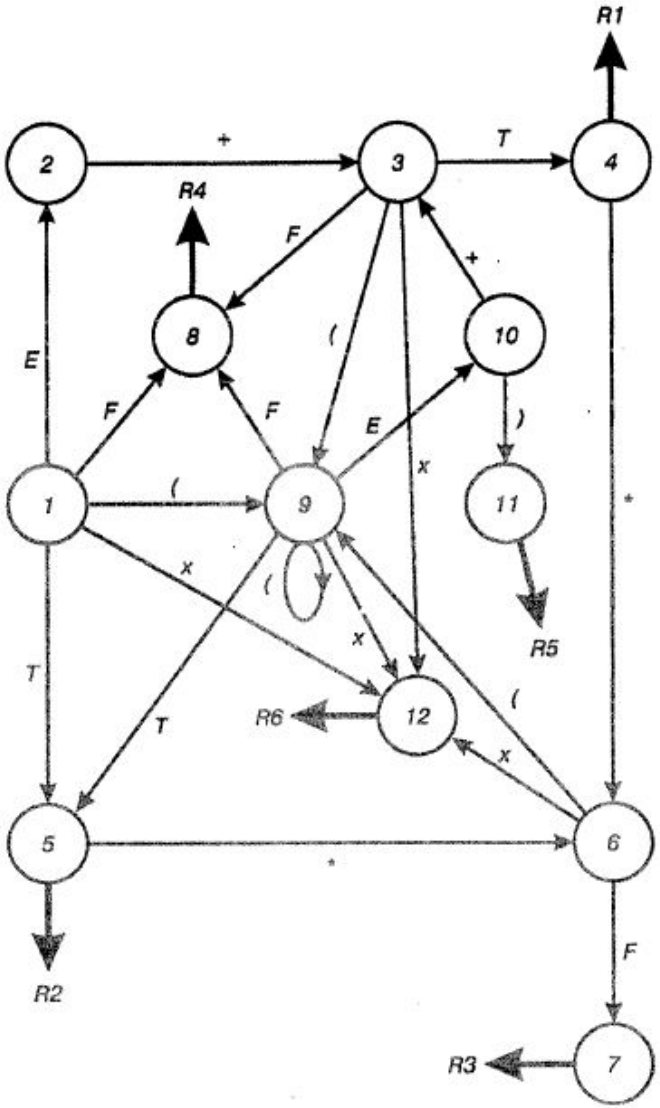
Восходящий синтаксический анализ. Таблица синтаксического анализа.



В продукциях 4-6 уже не требуются новые состояния, следующие за 8 (по причинам, сходным с приведенными выше для продукций 2 и 3). Оставшиеся состояния грамматики вводятся без каких-либо проблем.

1. $E \rightarrow_{1,9} E_{2,10} +_{3} T_{4}$
2. $E \rightarrow_{1,9} T_{5}$
3. $T \rightarrow_{1,3,9} T_{5,4} *_{6} F_{7}$
4. $T \rightarrow_{1,3,9} F_{8}$
5. $F \rightarrow_{1,3,6,9} (_{9} E_{10})_{11}$
6. $F \rightarrow_{1,3,6,9} x_{12}$

Восходящий синтаксический анализ. Таблица синтаксического анализа.



Восходящий синтаксический анализ. Таблица синтаксического анализа.

Таблица 2

	E	*T*	*F*	+	*	(	)	*x*	⊥
1	S2	S5	S8			S9		S12	
2				S3					
3		S4	S8			S9		S12	
4					S6				
5					S6				
6			S7			S9		S12	
7									
8									
9	S10	S5	S8			S9		S12	
10				S3			S11		
11									
12									

Переносы легко получаются из грамматики или из ХКА.

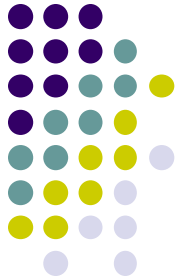
Восходящий синтаксический анализ. Таблица синтаксического анализа.

Таблица 3

	E	*T*	*F*	+	*	(	)	*x*	⊥
1	S2	S5	S8			S9		S12	
2				S3					
3		S4	S8			S9		S12	
4					S6				
5					S6				
6			S7			S9		S12	
7	R3	R3	R3	R3	R3	R3	R3	R3	R3
8	R4	R4	R4	R4	R4	R4	R4	R4	R4
9	S10	S5	S8			S9		S12	
10				S3			S11		
11	R5	R5	R5	R5	R5	R5	R5	R5	R5
12	R6	R6	R6	R6	R6	R6	R6	R6	R6

Кроме того, в таблицу просто (не порождая конфликтов) вводятся некоторые свертки

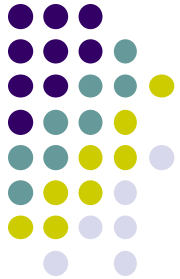
Восходящий синтаксический анализ. Таблица синтаксического анализа.



Замечаем, что свертки в состояниях **4** и **5** не являются очевидными, поскольку в строках, которые соответствуют данным состояниям, уже имеются несколько переносов. Данная грамматика не относится к классу LR(0), и при определении нужного действия (перенос или свертка) следует принимать во внимание символы предпросмотра. В первой ситуацию, возникающей в состоянии 4, могут иметь место как перенос, так и свертка. Из таблицы 3 или ХКА видим, что перенос возможен, только если символ предпросмотра — *. Таким образом, делаем вывод, какие символы предпросмотра соответствуют свертке.

Свертка, если она имеет место, будет происходить в символ *E,* так что мы рассматриваем символы, которые могут следовать за *E.*

Восходящий синтаксический анализ. Таблица синтаксического анализа.



Из продукции 1 получаем, что за *E* может следовать символ +, а из продукции 5 находим символ “)”. Кроме того, за *E* также может следовать символ ⊥, поскольку *E* – это символ предложения; таким образом, единственные символы, которые могут идти за E – это [+, ⊥,)]. Итак, действия свертки в состоянии 4 помещаются только в те столбцы таблицы синтаксического анализа**,** что соответствуют трем указанным символам.

Рассмотрим теперь состояние 5. Снова символом предпросмотра для действия *переноса* является *, а символы предпросмотра, соотнесенные с действием *свертки*, – это [+, ⊥,)] (последователи символа *E*). Теперь можно поместить действия свертки для продукций 1 и 2 в соответствующие столбцы для состояний 4 и 5, в результате чего получим таблицу 4.

Восходящий синтаксический анализ. Таблица синтаксического анализа.

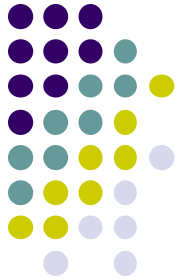
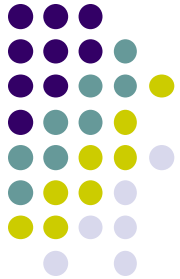


Таблица 4

	E	*T*	*F*	+	*	(	)	*x*	⊥
1	S2	S5	S8			S9		S12	
2				S3					
3		S4	S8			S9		S12	
4				R1	S6		R1		R1
5				R2	S6		R2		R2
6			S7			S9		S12	
7	R3	R3	R3	R3	R3	R3	R3	R3	R3
8	R4	R4	R4	R4	R4	R4	R4	R4	R4
9	S10	S5	S8			S9		S12	
10				S3			S11		
11	R5	R5	R5	R5	R5	R5	R5	R5	R5
12	R6	R6	R6	R6	R6	R6	R6	R6	R6

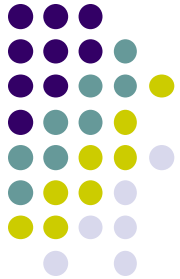
Поскольку в данных продукциях отсутствуют действия переноса, все конфликты перенос/свертка разрешены.

Восходящий синтаксический анализ. Таблица синтаксического анализа.



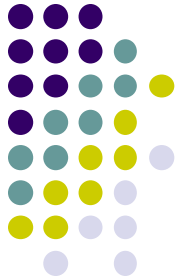
Хотя строго это и не обязательно, позиции других действий свертки также можно вычислить, приняв во внимание символы предпросмотра. Например, свертка согласно продукции 3 будет уместной, только если символ предпросмотра — это символ, который может следовать за *T*. Поскольку за *T* может следовать символ *, а любой последователь *E* может быть последователем *T*, полное множество символов-последователей имеет вид [*, +, ⊥,)]. Таким образом, действие **R3** в состоянии **7** уместно только в этих столбцах. Подобным образом, для состояния **8** (где свертка также выполняется в *T*) действие **R4** появляется только в столбцах, которые соответствуют множеству [*, +, ⊥,)].

Восходящий синтаксический анализ. Таблица синтаксического анализа.



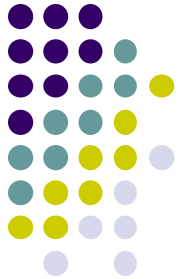
В состояниях 11 и 12 свертка производится в символ *F*, и снова рассматривая символы, которые могут следовать за *F* и *T* (согласно продукции 4), получаем множество [*, +, ⊥,)]. Таким образом, действия **R5** и **R6** должны появляться только в этих столбцах. Окончательно получаем табл. 1, которая приводилась ранее без объяснения принципов формирования.

Восходящий синтаксический анализ. SLR(1)-грамматика.



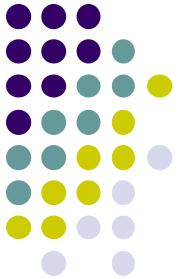
Если для внесения в таблицу синтаксического анализа действий свертки учитываются все возможные символы-последователи нетерминалов левой части продукции, таблица называется *простой* (simple) *LR(1)-таблицей* или *SLR(1)-таблицей*, а использованный алгоритм именуется *алгоритмом SLR(1)*. Если созданная таким образом таблица не содержит конфликтов, то грамматика, на основе которой получена таблица, называется *SLR(1)-грамматикой*. Очевидно, все LR(0)-грамматики относятся к классу SLR(l), хотя, как можно видеть из рассматриваемого примера, не все SLR(1)-грамматики относятся к классу LR(0). Отметим также, что даже если грамматика не является SLR(l), оставшиеся конфликты, возможно, удастся разрешить каким-то иным способом, и грамматику можно будет назвать LR(1).

Восходящий синтаксический анализ. LALR(1)-грамматика.



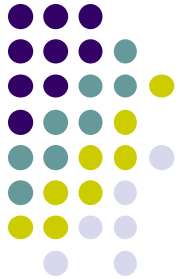
Как было показано ранее, алгоритм LR(0) успешен, если вообще нет нужды рассматривать символы-последователи, и все действия свертки можно поместить во все столбцы. Алгоритм SLR(1) успешен, если конфликты, выявленные алгоритмом LR(0), разрешаются с использованием символов-последователей описанным выше способом. В то же время иногда даже этого подхода недостаточно для разрешения всех конфликтов, и требуется более общий подход. В этом случае оставшиеся конфликты пытаются разрешить посредством алгоритма LALR(l) (**LALR(1)** — lookahead LR(1), LR(1) с предпросмотром)**.** Алгоритм **LALR(1)** ограничивает число символов-последователей, которые нужно рассматривать при конкретной свертке, используя контекстную (т.е. касающуюся состояний) информацию для определения только тех последователей, которые корректны в рассматриваемом состоянии.

Восходящий синтаксический анализ. LALR(1)-грамматика.



Все конфликты может не разрешить даже алгоритм LALR(1). Это, разумеется, объясняться тем, что грамматика не относится к классу LR(1), так что нельзя создать бесконфликтную синтаксическую таблицу описанного выше типа. Например, к классу LR(1) не относятся неоднозначные грамматики**.** В то же время существуют грамматики, являющиеся LR(1), но не LALR(1). Для этих грамматик таблица синтаксического анализа формируется путем введения в аннотированную грамматику дополнительных состояний в тех местах, где мы этого не делали в приведенном выше примере, т.е. когда одному набору конфигураций грамматики соответствует более одного состояния. Такие состояния бывают иногда нужны, возможно, их даже окажется много, что значительно увеличит таблицу синтаксического анализа, К счастью, описанной трактовки требуют лишь несколько грамматик, представляющих скорее академический, чем практический интерес, поэтому соответствующие примеры приводиться не будут.

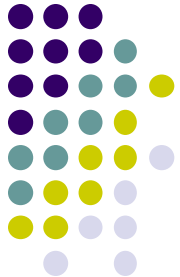
Восходящий синтаксический анализ.



Рассмотрим общий алгоритм формирования таблицы LR(1)-анализа, Он имеет несколько этапов.

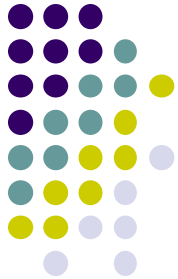
1. Для создания таблицы пытаемся использовать алгоритм LR(0). Если он успешен (т.е. конфликты отсутствуют), алгоритм прекращается.
2. При неудаче пытаемся применить алгоритм SLR(l). Если он увенчался успехом, алгоритм прекращается.
3. При неудаче пробуем алгоритм LALR(l). Если он достиг цели, алгоритм прекращается.
4. При неудаче испытываем алгоритм LR(1).

Восходящий синтаксический анализ.



Основная идея описанного подхода: первым применяется простейший алгоритм, далее сложность алгоритмов идет по возрастающей. Реально к классу LR(0) принадлежат немногие языки программирования, но многие языки принадлежат к классу SLR(l), оставшиеся практически наверняка будут LALR(l). Состояния LR(0)-, SLR(l)- и LALR(1)-грамматик, будут почти одинаковыми, но в LR(1)-грамматике их будет значительно больше. Последнее замечание: не имеет значения, какой алгоритм применяется для создания таблицы синтаксического анализа, синтаксический анализатор все таблицы использует одинаково.

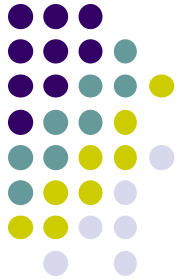
Восходящий синтаксический анализ. Особенности LR-анализа.



На данном этапе стоит упомянуть некоторые теоретические результаты.

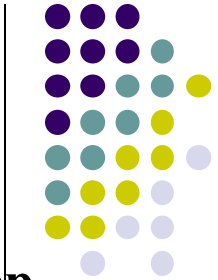
- Существует алгоритм определения, относится ли грамматика к классу LR(k) при данном k.
- Не существует алгоритма определения, существует ли k, при котором данная грамматика относится к классу LR(k). В общем случае
  данная задача не решается.
- Любой язык, относящийся к классу LR(k) при данном k, также относится к классу LR(1).

Восходящий синтаксический анализ. Особенности LR-анализа.



Впрочем, ни один из приведенных результатов не является особенно важным с практической точки зрения. Третий, например, означает, что с точки зрения языка нет смысла рассматривать более одного символа предпросмотра, поскольку если при некотором k для языка существует грамматика LR(k), также имеется и грамматика LR(1). Менее очевидно (и не доказывается), что на практике грамматика обычно является LR(l). LR(k) грамматики легко преобразовываются в LR(1).

Восходящий синтаксический анализ. Введение в YACC.



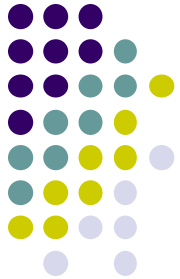
YACC (Yet Another Compiler-Compiler — "еще один компилятор компиляторов") был разработан в Bell Laboratories и используется для создания LR-анализатора из любой LALR(1)-грамматики. Это средство полностью совместимо с Lex, фактически оно появилось на несколько лет раньше Lex, гак что в первые годы использования YACC пользователи должны были вручную создавать лексические анализаторы для применения с автоматически создаваемыми синтаксическими анализаторами. Впрочем, сейчас этого уже не требуется, так что ниже будет показано, как Lex и YACC используются совместно. Вход YACC всегда имеет следующий вид:

объявления
%%
правила
%%
функции, определенные пользователем

В то же время, разделы *объявления* и *функции, определенные пользователем,* могут быть пустыми. Если раздел *функции, определенные пользователем,* пуст, второй разделитель *%%* можно опустить, так что минимальный вход YACC имеет следующий вид:

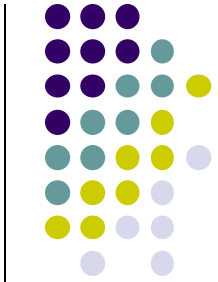
%%
правила

Восходящий синтаксический анализ. Введение в YACC.



Выход YACC – это программа на языке C, компилировать которую можно обычными средствами. В настоящее время доступны версии YACC, поддерживающие различные языки, такие как Turbo Pascal от Borland. YACC весьма подобен Lex по многим пунктам, хотя, конечно, поддерживает более общие языковые конструкции. Анализируемый язык выражается как контекстно-свободная грамматика, при этом используется форма записи, подобная применяющейся в контекстно-свободных грамматиках, хотя и с некоторыми расширениями. Эти расширения не увеличивают выразительной силы формы записи в том смысле, что она может использоваться для описания языков, которые нельзя описать посредством контекстно-свободной грамматики, но они упрощают описание некоторых языков, а также иногда сокращают описание языка.

Восходящий синтаксический анализ. Введение в YACC.



```
%left '+' '-'
%left '*' '/'
%%
exp : exp '+' exp
      | exp '-' exp
      | exp '*' exp
      | exp '/' exp
      | '(' exp ')'
    | num
```

Появление left перед каждой парой операторов обозначает, что они являются левоассоциативными. Значит, выражение

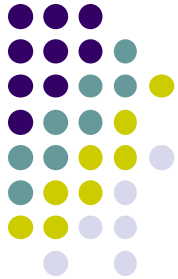
3 + 4 + 5

нужно вычислять следующим образом:

((3 + 4)+5)

Если имеются правила, которые позволяют разрешить неоднозначность. Такие правила называются *правила разрешения неоднозначности* (disambiguating rules).

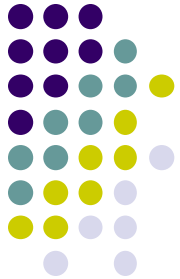
Восходящий синтаксический анализ. Введение в YACC.



Приведенный выше пример можно расширить, включив унарные операторы, приоритет которых может отличаться от приоритетов бинарных операторов, представленных теми же символами. Если нужно включить унарный минус с приоритетом выше, чем у операторов умножения, вход YACC будет иметь следующий вид.

```
%left '+' '-'
%left '*' '/'
%left UMINUS
%%
exp : exp '+' exp
        | exp '-' exp
        | exp '*' exp
        | exp '/' exp
        | '-' exp   %prec UMINUS
        | '(' exp ')'
    | num
```

Восходящий синтаксический анализ. Введение в YACC.



Как и для Lex, действия пишутся на C и обычно (хотя и не всегда) располагаются в конце синтаксических правил, таким образом соответствуя приведению выражений. Например, к синтаксическому правилу

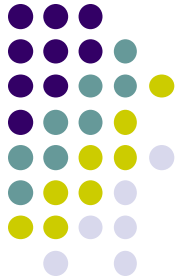
```
exp : exp '+' exp
```

можно добавить действие, в результате чего получится следующее.

```
exp : exp '+' exp {$$=$1+$3;}
```

Здесь код C замкнут в фигурные скобки. Переменные со знаком доллара – это отличительная особенность YACC, которая является крайне полезной. $n – это численное значение (атрибут), соотнесенное с n-м символом правой части продукции, а $$ — численное значение, которое будет соотнесено с символом в левой части. Таким образом, в приведенном примере значение переменной со знаком доллара, соотнесенное с выражением в левой части правила, будет равно сумме значений переменных со знаком доллара, соотнесенных с первым и третьим символами правой части правила. Подобным образом значения метут передаваться от правых частей продукций левым частям или (с другой точки зрения) от основания синтаксического дерева к вершине.

Вход YACC, включающий комментарии, которые выделены знаками
/* и */, имеет следующий вид.



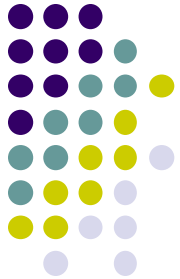
```
%token NUMBER
%left ‘+’ ‘-’
%left ‘*’ ‘/’
%left UMINUS
/*приоритет унарного минуса выше приоритета других операторов*/
%% /*раздел правил*/
S : exp
     {printf("%d\n", $1);};
exp : exp ‘+’ exp
     {$$=$1+$3;}
     | exp ‘-’ exp
     {$$=$1-$3;}
     | exp ‘*’ exp
     {$$=$1*$3;}
     | exp ‘/’ exp
     {if ($3 == 0) yyerror (“divide by 0”) else $$=$1/$3;}
     | '-' exp     %prec UMINUS
     {$$=-$2;}
     | ‘(’ exp ‘)’
     {$$=$2;}
     | NUMBER;
%%
#include "lex.yy.c"
yyerror(s)
char *s;
{printf(“%s\n”, s);}

main()
{return yyparse();
}
```

Восходящий синтаксический анализ. Введение в YACC.

Для связи с лексическим анализатором, созданным Lex**,** перед пользовательскими функциями "включается" lex.yy.c. Пользователь должен сам поддерживать свою версию функций **yyerror** и **main** (которая должна возвращать значение, полученное при вызове **yyparse** ()). Эти функции могут быть простыми или сложными, по желанию. Отметим также, что терминалы в грамматике, которые не соответствуют одному знаку, "объявляются" как %token. Это обеспечивает связь с лексическим анализатором. Терминалы, состоящие из одного знака, могут передаваться Lex автоматически, без необходимости явного "распознавания ".
Вход Lex для распознавателя выражений будет иметь следующий вид.

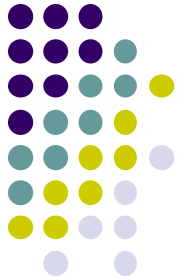
```
%%
[0-9]+   {yylval = atoi   (yytext);   return NUMBER;}
[ \t]      ;   /*игнорировать  пробелы*/
return yytext[0];
```

Восходящий синтаксический анализ. Введение в YACC.



Здесь yytext — массив, содержащий знаки символа, с которым только что было найдено соответствие. Для конвертирования этой строки в целое число используется функция C, именуемая atoi, а для обеспечения связи с синтаксическим анализатором возвращается символ number. Значение числа присваивается целому yylval, посредством которого это значение и передается синтаксическому анализатору. Отметим, что переменные, имеющие специальное значение, в YACC обычно начинаются с yy. Это помогает избежать путаницы с пользовательскими переменными.

Восходящий синтаксический анализ. Введение в YACC.



Предполагая, что входами Lex и YACC являются, соответственно, nums.l и nums.y, для генерации синтаксического анализатора потребуется следующий ввод.

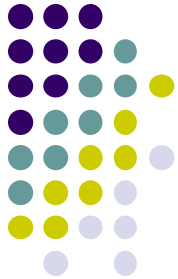
```
lex nums.l
yacc nums.y
cc -o nums y.tab.c -ll -ly
```

Опция -o указывает, что синтаксический анализатор должен быть создан в файле nums, а опции -ll и -ly гарантируют включение библиотек Lex и YACC. Для выполнения программы синтаксического анализа требуется ввести следующее.

```
nums < dat
```

Здесь dat содержит данные (выражение), значение которых требуется вычислить.

Восходящий синтаксический анализ. Использование YACC.

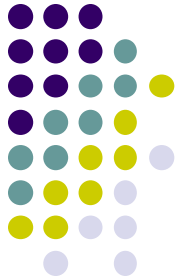


Больше узнать о том, как YACC формирует таблицу синтаксического анализа, можно, запустив YACC в "подробном режиме" (verbose mode), после чего получить достаточно подробное описание в следующем файле y.output. В среде Unix для получения этого файла нужно вызвать YACC с опцией -v.

```
yacc  -v nums.y
```

Файл y.output содержит подробности относительно состояний синтаксического анализатора

Восходящий синтаксический анализ. Использование YACC.



Как упоминалось выше, интересная и широко известная неоднозначность появляется при **использовании** оператора If, следующим образом определенного в C.

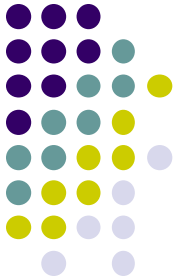
```
if-statement : if '('expression')' statement
        | if '('expression')' statement else statement;
```

При таком определении может возникнуть следующее неоднозначное выражение.

```
if (x==3) if (y==5) z = 6; else w = 7;
```

Неоднозначность заключается в том, что неясно, к какому if относится else – к первому или второму. Легко показать, что приведенное выше предложение имеет два левых и два правых порождения и два синтаксических дерева. Неоднозначность может быть исключена следующим способом.

Восходящий синтаксический анализ. Использование YACC.



```
%token IF, ELSE, EXP
%start statement
%%
statement:  if_statement
         |;
if-statement: IF '('EXP')' statement
         | IF '('EXP')' statement EXP statement
```