нетерминалы, из которых не выводится непустого слова, несложно удалить из грамматики.)

Замечания.

- Приведенный алгоритм использует `S` как стек (все действия производятся с левого конца).
- Действия двух последних вариантов внутри цикла не приводят к чтению очередного символа со входа, поэтому их можно заранее предвычислить для каждого нетерминала и каждого символа `Next`. После этого на каждом шаге цикла будет читаться очередной символ входа.
- При практической реализации удобно составить таблицу, в которой записаны варианты действий в зависимости от входного символа и первого символа `S`, и небольшую программу, выполняющую действия в соответствии с этой таблицей.

13.3.9. При проверке того, относится ли данная грамматика к типу LL(1), необходимо вычислить Послед(Т) и Нач(Т) для всех нетерминалов Т. Как это сделать?

Решение. Если в грамматике есть, например, правило $\mathsf{K} \to \mathsf{L}\,\mathsf{M}\,\mathsf{N}$, то

$$
\begin{array}{rcll}
\text{Нач}(\mathsf{L}) & \subset & \text{Нач}(\mathsf{K}), & \\
\text{Нач}(\mathsf{M}) & \subset & \text{Нач}(\mathsf{K}), & \text{если из } \mathsf{L} \text{ выводимо пустое слово,}\\
\text{Нач}(\mathsf{N}) & \subset & \text{Нач}(\mathsf{K}), & \text{если из } \mathsf{L} \text{ и } \mathsf{M} \text{ выводимо пустое слово,}\\
\text{Послед}(\mathsf{K}) & \subset & \text{Послед}(\mathsf{N}), & \\
\text{Послед}(\mathsf{K}) & \subset & \text{Послед}(\mathsf{M}), & \text{если из } \mathsf{N} \text{ выводимо пустое слово,}\\
\text{Послед}(\mathsf{K}) & \subset & \text{Послед}(\mathsf{L}), & \text{если из } \mathsf{M} \text{ и } \mathsf{N} \text{ выводимо пустое слово,}\\
\text{Нач}(\mathsf{N}) & \subset & \text{Послед}(\mathsf{M}), & \\
\text{Нач}(\mathsf{M}) & \subset & \text{Послед}(\mathsf{L}), & \\
\text{Нач}(\mathsf{N}) & \subset & \text{Послед}(\mathsf{L}), & \text{если из } \mathsf{M} \text{ выводимо пустое слово,}
\end{array}
$$

Подобные правила позволяют постепенно порождать множества Нач(Т), а затем и Послед(Т), для всех терминалов и нетерминалов Т. При этом началом служит

$$\texttt{EOI} \in \text{Послед}(\mathsf{K})$$

для начального нетерминала К и

$$z \in \text{Нач}\,(z)$$

для любого терминала z. Порождение заканчивается, когда применение правил перестаёт давать новые элементы множеств Нач(Т) и Послед(Т).

14. Синтаксический разбор слева направо (LR)

Сейчас мы рассмотрим еще один метод синтаксического разбора, называемый LR(1)-разбором, а также некоторые упрощённые его варианты.

14.1. LR-процессы

Два отличия LR(1)-разбора от LL(1)-разбора: во-первых, строится не левый вывод, а правый, во-вторых, он строится не с начала, а с конца. (Вывод в КС-грамматике называется *правым*, если на каждом шаге замене подвергается самый правый нетерминал.)

14.1.1. Доказать, что если слово, состоящее из терминалов, выводимо, то оно имеет правый вывод.

Нам будет удобно смотреть на правый вывод «задом наперёд». Определим понятие LR-*процесса над словом* A. В этом процессе, помимо A, будет участвовать и другое слово S, которое может содержать как терминалы, так и нетерминалы. Вначале слово S пусто. В ходе LR-процесса разрешены два вида действий:

(1) можно перенести первый символ слова A (его называют *очередным* символом и обозначают `Next`) в конец слова S, удалив его из A (это действие называют *сдвигом*);

(2) если правая часть одного из правил грамматики оказалась концом слова S, то разрешается заменить её на нетерминал, стоящий в левой части этого правила; при этом слово A не меняется. (Это действие называют *свёрткой*, или *приведением*.

Отметим, что LR-процесс не является детерминированным: в одной и той же ситуации могут быть разрешены разные действия.

Говорят, что LR-процесс на слове A *успешно завершается*, если слово A становится пустым, а в слове S остается единственный нетерминал — начальный нетерминал грамматики.

14.1.2. Доказать, что для любого слова A (из терминалов) успешно завершающийся LR-процесс существует тогда и только тогда, когда слово A выводимо в грамматике. В ходе доказательства установить взаимно однозначное соответствие между правыми выводами и успешно завершающимися LR-процессами.

Решение. При сдвиге слово SA не меняется, при свёртке слово SA подвергается преобразованию, обратному шагу вывода. Этот вывод будет правым, так как сворачивается конец S, а в A все символы — терминальные. Таким образом, каждому LR-процессу соответствует правый вывод. Обратное соответствие: пусть дан правый вывод. Представим себе, что за последним нетерминалом в слове стоит перегородка. Применив к этому нетерминалу правило грамматики, мы должны сдвинуть перегородку влево (если правая часть правила кончается на терминал). Разбивая этот сдвиг на отдельные шаги, получим процесс, в точности обратный LR-процессу.

Поскольку в ходе LR-процесса все изменения в слове S происходят с правого конца, слово S называют *стеком* LR-процесса.

Задача построения правого вывода для данного слова сводится, таким образом, к правильному выбору очередного шага LR-процесса. Нам нужно решить, будем ли мы делать сдвиг или свёртку, и если свёртку, то по какому правилу — ведь подходящих правил может быть несколько. В LR(1)-алгоритме это решение принимается на основе S и первого символа слова A; если используется только S, то говорят о LR(0)-алгоритме. (Точные определения смотри ниже.)

Пусть фиксирована грамматика, в которой из любого нетерминала можно вывести какое-либо слово из терминалов. (Это ограничение мы будет всегда предполагать выполненным.)

Пусть $\mathtt{K} \to \mathtt{U}$ — одно из правил грамматики ($\mathtt{K}$ — нетерминал, $\mathtt{U}$ — слово из терминалов и нетерминалов). Определим множество слов (из терминалов и нетерминалов), называемое *левым контекстом* правила $\mathtt{K} \to \mathtt{U}$. (Обозначение: ЛевКонт($\mathtt{K} \to \mathtt{U}$).) По определению в него входят все слова, которые являются содержимым стека непосредственно перед свёрткой $\mathtt{U}$ в $\mathtt{K}$ в ходе некоторого успешно завершающегося LR-процесса.

14.1.3. Переформулировать это определение на языке правых выводов.

Решение. Рассмотрим все правые выводы вида

$$\langle\text{начальный нетерминал}\rangle \leadsto \mathtt{XKA} \to \mathtt{XUA},$$

где A — слово из терминалов, X — слово из терминалов и нетерминалов. Все возникающие при этом слова XU и образуют левый контекст правила K → U. Чтобы убедиться в этом, следует вспомнить, что мы предполагаем, что из любого нетерминала можно вывести какое-то слово из терминалов, так что правый вывод слова XUA может быть продолжен до правого вывода какого-то слова из терминалов.

14.1.4. Все слова из ЛевКонт(K → U) кончаются, очевидно, на U. Доказать, что если у всех них этот конец U отбросить, то полученное множество слов не зависит от того, какое из правил для нетерминала K выбрано. (Это множество обозначается Лев(K).)

Решение. Из предыдущей задачи ясно, что Лев(K) — это всё, что может появиться в правых выводах левее самого правого нетерминала K.

14.1.5. Доказать, что в предыдущей фразе можно отбросить слова «самого правого»: Лев(K) — это всё то, что может появляться в правых выводах левее любого вхождения нетерминала K.

Решение. Продолжив построение правого вывода, все нетерминалы справа от K можно заменить на терминалы (а слева от K при этом ничего не изменится).

14.1.6. Построить грамматику, содержащую для каждого нетерминала K исходной грамматики нетерминал ⟨ЛевK⟩, причем следующее свойство должно выполняться для любого нетерминала K исходной грамматики: в новой грамматике из ⟨ЛевK⟩ выводимы все элементы Лев(K) и только они. (При этом терминалы и нетерминалы исходной грамматики являются терминалами новой.)

Решение. Пусть P — начальный нетерминал грамматики. Тогда в новой грамматике будет правило

⟨ЛевP⟩ → (пустое слово)

Для каждого правила исходной грамматики, например, правила

K → L t M N (L, M, N — нетерминалы, t — терминал),

в новую грамматику мы добавим правила

⟨ЛевL⟩ → ⟨ЛевK⟩

$$\begin{aligned} \langle\text{ЛевM}\rangle &\to \langle\text{ЛевK}\rangle\,\mathtt{L\,t} \\ \langle\text{ЛевN}\rangle &\to \langle\text{ЛевK}\rangle\,\mathtt{L\,t\,M} \end{aligned}$$

и аналогично поступим с другими правилами. Смысл новых правил таков: пустое слово может появиться слева от P; если слово X может появиться слева от K, то X может появиться слева от L, XLt может появиться слева от M, XLtM — слева от N. Индукцией по длине правого вывода легко проверить, что все, что может появиться слева от какого-то нетерминала, появляется в соответствии с этими правилами.

14.1.7. Почему в предыдущей задаче важно, что мы рассматриваем только правые выводы?

Ответ. В противном случае следовало бы учитывать преобразования, происходящие внутри слова, стоящего слева от K.

14.1.8. Для данной грамматики построить алгоритм, который по любому слову выясняет, каким из множеств Лев(K) оно принадлежит.

(Замечание для знатоков. Существование такого алгоритма — и даже конечного автомата, то есть индуктивного расширения с конечным числом значений, см. 1.3, — вытекает из предыдущей задачи, так как построенная в ней грамматика имеет специальный вид: в правых частях всего один нетерминал, причем он стоит у левого края. Тем не менее мы приведём явное построение.)

Решение. Будем называть ситуацией данной грамматики одно из её правил, в правой части которого отмечена одна из позиций (до первой буквы, между первой и второй буквой, ... , после последней буквы). Например, правило

$$\mathtt{K} \to \mathtt{L\,t\,M\,N}$$

(K, L, M, N — нетерминалы, t — терминал) порождает пять ситуаций

$$\mathtt{K} \to \_\mathtt{L\,t\,M\,N} \quad \mathtt{K} \to \mathtt{L}\_\mathtt{t\,M\,N} \quad \mathtt{K} \to \mathtt{L\,t}\_\mathtt{M\,N} \quad \mathtt{K} \to \mathtt{L\,t\,M}\_\mathtt{N} \quad \mathtt{K} \to \mathtt{L\,t\,M\,N}\_$$

(позиция указывается знаком подчеркивания).

Будем говорить, что слово S *согласовано* с ситуацией $\mathtt{K} \to \mathtt{U}\_\mathtt{V}$, если S кончается на U, то есть $\mathtt{S} = \mathtt{TU}$ при некотором T, и, кроме того, T принадлежит Лев(K). (Смысл этого определения примерно таков: в стеке

S подготовлена часть U для будущей свёртки UV в K.) В этих терминах ЛевКонт(K $\to$ X) — это множество всех слов, согласованных с ситуацией K $\to$ X _, а Лев(K) — это множество всех слов, согласованных с ситуацией K $\to$ _X (где K $\to$ X — любое правило для нетерминала K).

Эквивалентное определение в терминах LR-процесса: S согласовано с ситуацией K $\to$ U _V, если существует успешный LR-процесс, в котором события развиваются так:

- в ходе процесса в стеке появляется слово S, и оно оканчивается на U;
- некоторое время S не затрагивается, а справа от него появляется V;
- UV сворачивается в K;
- процесс продолжается и успешно завершается.

14.1.9. Доказать эквивалентность этих определений.

[Указание. Если S = TU и T принадлежит Лев(K), то можно получить в стеке сначала T, потом U, потом V, потом свернуть UV в K и затем успешно завершить процесс. (Мы используем несколько раз тот факт, что из любого нетерминала что-то да выводится: благодаря этому мы можем добавить в стек любое слово.)]

Наша цель — построение алгоритма, распознающего принадлежность произвольного слова к Лев(K). Рассмотрим функцию, сопоставляющую с каждым словом S (из терминалов и нетерминалов) множество всех согласованных с ним ситуаций. Это множество назовём *состоянием, соответствующим слову S*. Будем обозначать его Сост(S). Достаточно показать, что функция Сост(S) индуктивна, то есть что значение Сост(SJ), где J — терминал или нетерминал, может быть вычислено, если известно Сост(S) и символ J. (Мы видели ранее, как принадлежность к Лев(K) выражается в терминах этой функции.) Значение Сост(SJ) вычисляется по таким правилам:

(1) Если слово S согласовано с ситуацией K $\to$ U _V, причём слово V начинается на букву J, то есть V = JW, то SJ согласовано с ситуацией K $\to$ UJ _W.

Это правило полностью определяет все ситуации с непустой левой половиной (то есть не начинающиеся с подчёркивания), согласованные с SJ. Осталось определить, для каких нетерминалов K слово SJ принадлежит Лев(K). Это делается по двум правилам:

(2) Если ситуация L → U_V согласована с SJ (согласно правилу (1)), а V начинается на нетерминал K, то SJ принадлежит Лев(K).

(3) Если SJ входит в Лев(L) для некоторого L, причём L → V — правило грамматики и V начинается на нетерминал K, то SJ принадлежит Лев(K).

Заметим, что правило (3) можно рассматривать как аналог правила (2): в указанных в (3) предположениях ситуация L → _V согласована с SJ, а V начинается на нетерминал K.

Корректность этих правил в общем-то очевидна, если хорошенько подумать. Единственное, что требует некоторых пояснений — это то, почему с помощью правил (2) и (3) обнаружатся *все* терминалы K, для которых SJ принадлежит Лев(K). Попытаемся это объяснить. Рассмотрим правый вывод, в котором SJ стоит слева от K. Откуда мог взяться в нем нетерминал K? Если правило, которое его породило, породило также и конец слова SJ, то принадлежность SJ к Лев(K) будет обнаружена по правилу (2). Если же K было первой буквой слова, порождённого каким-то другим нетерминалом L, то — благодаря правилу (3) — достаточно установить принадлежность SJ к Лев(L). Осталось применить те же рассуждения к L и так далее.

В терминах LR-процесса то же самое можно сказать так. Сначала нетерминал K может участвовать в нескольких свёртках, не затрагивающих SJ (они соответствуют применению правила (3)), но затем он обязан подвергнуться свёртке, затрагивающей SJ (что соответствует применению правила (2)).

Осталось выяснить, какие ситуации согласованы с пустым словом, то есть для каких нетерминалов K пустое слово принадлежит Лев(K). Это определяется по следующим правилам:

(1) начальный нетерминал таков;

(2) если K таков и K $\to$ V — правило грамматики, причем слово V начинается с нетерминала L, то и L таков.

14.1.10. Проделать описанный анализ для грамматики

$$
\begin{array}{lcl}
\mathtt{E} & \to & \mathtt{E+T} \\
\mathtt{E} & \to & \mathtt{T} \\
\mathtt{T} & \to & \mathtt{T*F} \\
\mathtt{T} & \to & \mathtt{F} \\
\mathtt{F} & \to & \mathtt{x} \\
\mathtt{F} & \to & \mathtt{(\,E\,)}
\end{array}
$$

(задающей тот же язык, что и грамматика примера 3, с. 228).

Решение. Множества Сост(S) для различных S приведены в таблице. Знак равенства означает, что множества ситуаций, являющиеся значениями функции Сост(S) на словах, стоящих слева и справа от знака равенства, одинаковы.

Правило определения Сост(SJ), если известны Сост(S) и J (здесь S — слово из терминалов и нетерминалов, J — терминал или нетерминал), таково:

> надо найти Сост(S) в правой колонке, взять соответствующее ему слово T в левой колонке, приписать к нему J и взять множество, стоящее напротив слова TJ (если слово TJ в таблице отсутствует, то Сост(SJ) пусто).

14.2. LR(0)-грамматики

Напомним, что наша основная цель — это поиск вывода заданного слова, или, другими словами, поиск успешного LR-процесса над ним. Во всех рассматриваемых нами грамматиках успешный LR-процесс (над данным словом) единствен. Искать этот единственный успешный процесс мы

Слово S	Сост(S)
пустое	E → _E+T E → _T T → _T*F T → _F F → _x F → _(E)
E	E → E_+T
T	E → T_ T → T_*F
F	T → F_
x	F → x_
(	F → (_E) E → _E+T E → _T T → _T*F T → _F F → _x F → _(E)
E+	E → E+_T T → _T*F T → _F F → _x F → _(E)
T*	T → T*_F F → _x F → _(E)
(E	F → (E_) E → E_+T
(T	= T
(F	= F
(x	= x
((	= (
E+T	E → E+T_ T → T_*F
E+F	= F
E+x	= x
E+(	= (
T*F	T → T*F_
T*x	= x
T*(	= (
(E)	F → (E)_
(E+	= E+
E+T*	= T*

будем постепенно: в каждый момент мы смотрим, какой шаг возможен следующим. Для этого на грамматику надо наложить дополнительные требования, и сейчас мы рассмотрим простейший случай так называемых LR(0)-грамматик. Мы уже знаем:

> (1) В успешном LR-процессе возможна свёртка по правилу `K` $\to$ `U` при содержимом стека `S` тогда и только тогда, когда `S` принадлежит ЛевКонт(`K` $\to$ `U`) или, другими словами, когда слово `S` согласовано с ситуацией `K` $\to$ `U_`.

Аналогичное утверждение про сдвиг гласит:

> (2) В успешном LR-процессе при содержимом стека `S` возможен сдвиг с очередным символом `a` тогда и только тогда, когда `S` согласовано с некоторой ситуацией `K` $\to$ `U_aV`.

14.2.1. Докажите это.

[Указание. Пусть произошел сдвиг и к стеку `S` добавилась буква `a`. Рассмотрите первую свёртку, затрагивающую эту букву.]

Рассмотрим некоторую грамматику и произвольное слово `S` из терминалов и нетерминалов. Если множество Сост(`S`) содержит ситуацию, в которой справа от подчёркивания стоит терминал, то говорят, что *для слова S возможен сдвиг*. Если в Сост(`S`) есть ситуация, в которой справа от подчёркивания ничего нет, то говорят, что *для слова S возможна свёртка* (по соответствующему правилу). Говорят, что для слова `S` возникает *конфликт типа сдвиг/свёртка*, если возможны и сдвиг, и свёртка. Говорят, что для слова `S` возникает *конфликт типа свёртка/свёртка*, если есть несколько правил, по которым возможна свёртка.

Грамматика называется LR(0)-*грамматикой*, если в ней нет конфликтов типа сдвиг/свёртка и свёртка/свёртка ни для одного слова `S`.

14.2.2. Является ли приведённая выше грамматика LR(0)-грамматикой?

Решение. Нет, не является. Для слов `T` и `E+T` имеются конфликты типа сдвиг/свёртка.

14.2.3. Являются ли LR(0)-грамматиками такие:

(а) T → 0 (б) T → 0
T → T1 T → 1T
T → TT2 T → 2TT
T → TTT3 T → 3TTT

Решение. Являются, см. соответствующие таблицы (конфликтов нет).

Слово S	Сост(S)
пустое	T → _0 T → _T1 T → _TT2 T → _TTT3
0	T → 0_
T	T → T_1 T → T_T2 T → T_TT3
	T → _0 T → _T1 T → _TT2 T → _TTT3
T1	T → T1_
TT	T → TT_2 T → TT_T3
	T → T_1 T → T_T2 T → T_TT3
	T → _0 T → _T1 T → _TT2 T → _TTT3
TT2	T → TT2_
TTT	T → TTT_3 T → TT_2 T → TT_T3
	T → T_1 T → T_T2 T → T_TT3
	T → _0 T → _T1 T → _TT2 T → _TTT3
TT0	= 0
TTT3	T → TTT3_
TTT2	= TT2
TTTT	= TTT
TTT0	= 0

(а)

Эта задача показывает, что LR(0)-грамматики могут быть как леворекурсивными, так и праворекурсивными.

14.2.4. Пусть дана LR(0)-грамматика. Доказать, что у любого слова существует не более одного правого вывода. Построить алгоритм проверки выводимости в LR(0)-грамматике.

Решение. Пусть дано произвольное слово. Будем строить LR-процесс над ним по шагам. Пусть текущее состояние стека LR-процесса равно S.

Слово S	Сост(S)
пустое	T → _0 T → _1T T → _2TT T → _3TTT
0	T → 0_
1	T → 1_T
	T → _0 T → _1T T → _2TT T → _3TTT
2	T → 2_TT
	T → _0 T → _1T T → _2TT T → _3TTT
3	T → 3_TTT
	T → _0 T → _1T T → _2TT T → _3TTT

(б), начало

Нам надо решить, делать сдвиг или свёртку (и если свёртку, то по какому правилу). Согласно определению LR(0)-грамматики, в нашем состоянии S возможен либо только сдвиг, либо только свёртка (причем лишь по одному правилу). Таким образом, поиск возможных продолжений LR-процесса происходит детерминированно (на каждом шаге можно определить, какое действие только и возможно).

14.2.5. Что произойдет, если анализируемое слово не имеет вывода в данной грамматике?

Ответ. Либо на некотором шаге не будет возможен ни сдвиг, ни свёртка, либо все возможные сдвиги будет иметь неподходящий очередной символ.

Замечания. 1. При реализации этого алгоритма нет необходимости каждый раз заново вычислять множество Сост(S) для текущего значения S. Эти множества можно также хранить в стеке (в каждый момент хранятся множества Сост(T) для всех начал T текущего слова S).

2. На самом деле само слово S можно не хранить — достаточно хранить множества ситуаций Сост(T) для всех его начал T (включая само S).

В алгоритме проверки выводимости в LR(0)-грамматике мы используем не всю информацию, которую могли бы. В этом алгоритме для каждого состояния известно заранее, что в нем возможен только сдвиг или только свёртка (причем в последнем случае известно, по какому правилу). Более изощрённый алгоритм мог бы принимать решение о выборе между

Слово S	Сост(S)
1T	T → 1T_
10	= 0
11	= 1
12	= 2
13	= 3
2T	T → 2T_T
	T → _0 T → _1T T → _2TT T → _3TTT
20	= 0
21	= 1
22	= 2
23	= 3
3T	T → 3T_TT
	T → _0 T → _1T T → _2TT T → _3TTT
30	= 0
31	= 1
32	= 2
33	= 3
2TT	T → 2TT_
2T0	= 0
2T1	= 1
2T2	= 2
2T3	= 3
3TT	T → 3TT_T
	T → _0 T → _1T T → _2TT T → _3TTT
3T0	= 0
3T1	= 1
3T2	= 2
3T3	= 3
3TTT	T → 3TTT_
3TT0	= 0
3TT1	= 1
3TT2	= 2
3TT3	= 3

(б), окончание

сдвигом и свёрткой, посмотрев на очередной символ (`Next`). Глядя на состояние, можно сказать, при каких значениях `Next` возможен сдвиг (это те терминалы, которые в ситуациях этого состояния стоят непосредственно за подчёркиванием). Сложнее воспользоваться информацией о символе `Next` для решения вопроса о том, возможна ли свёртка. Для этого есть упрощённый метод (грамматики, к которым он применим, называют SLR(1)-грамматиками [сокращение от Simple LR(1)]) и полный метод (более сложный, но использующий всю возможную информацию; грамматики, к которым он применим, называют LR(1)-грамматиками). Есть и промежуточный класс грамматик, называемый LALR(1).

14.3. SLR(1)-грамматики

Напомним, что для любого нетерминала `K` мы определяли (с. 247) множество Послед(`K`) тех терминалов, которые могут стоять непосредственно за `K` в выводимом (из начального нетерминала) слове; в это множество добавляют также символ `EOI`, если нетерминал `K` может стоять в конце выводимого слова.

14.3.1. Доказать, что если в данный момент LR-процесса последний символ стека `S` равен `K`, причем процесс этот может в дальнейшем успешно завершиться, то `Next` принадлежит Послед(`K`).

Решение. Этот факт является непосредственным следствием определения (вспомним соответствие между правыми выводами и LR-процессами).

Рассмотрим некоторую грамматику, произвольное слово `S` из терминалов и нетерминалов и терминал `x`. Если множество Сост(`S`) содержит ситуацию, в которой справа от подчёркивания стоит терминал `x`, то говорят, что *для пары* $\langle \mathtt{S}, \mathtt{x}\rangle$ *возможен сдвиг*. Если в Сост(`S`) есть ситуация $\mathtt{K} \to \mathtt{U\_}$, причем `x` принадлежит Послед(`K`), то говорят, что *для пары* $\langle \mathtt{S}, \mathtt{x}\rangle$ SLR(1)-*возможна свёртка* (по правилу $\mathtt{K} \to \mathtt{U}$). Говорят, что для пары $\langle \mathtt{S}, \mathtt{x}\rangle$ возникает SLR(1)-*конфликт типа сдвиг/свёртка*, если возможны и сдвиг, и свёртка. Говорят, что для пары $\langle \mathtt{S}, \mathtt{x}\rangle$ возникает SLR(1)-*конфликт типа свёртка/свёртка*, если есть несколько правил, по которым возможна свёртка.

Грамматика называется SLR(1)-грамматикой, если в ней нет SLR(1)-конфликтов типа сдвиг/свёртка и свёртка/свёртка ни для одной пары

$\langle \mathtt{S}, \mathtt{x} \rangle$.

14.3.2. Пусть дана SLR(1)-грамматика. Доказать, что у любого слова существует не более одного правого вывода. Построить алгоритм проверки выводимости в SLR(1)-грамматике.

Решение. Аналогично случаю LR(0)-грамматик, только при выборе между сдвигом и свёрткой учитывается очередной символ (Next).

14.3.3. Проверить, является ли приведенная выше на с. 259 грамматика (с нетерминалами E, T и F) SLR(1)-грамматикой.

Решение. Да, является, так как оба конфликта, мешающие ей быть LR(0)-грамматикой, разрешаются с учетом очередного символа: и для слова T, и для слова E+T сдвиг возможен только при Next $=$ *, а символ * не принадлежит ни Послед(E) $= \{\mathtt{EOI}, +,)\}$, ни Послед(T) $= \{\mathtt{EOI}, +, *,)\}$, и поэтому при Next $=$ * свёртка невозможна.

14.4. LR(1)-грамматики, LALR(1)-грамматики

Описанный выше SLR(1)-подход используют не всю возможную информацию при выяснении того, возможна ли свёртка. Именно, он отдельно проверяет, возможна ли свёртка при данном состоянии стека S и отдельно — возможна ли свёртка по данному правилу при данном символе Next. Между тем эти проверки не являются независимыми: обе могут дать положительный ответ, но тем не менее свёртка при стеке S и очередном символе Next невозможна. В LR(1)-подходе этот недостаток устраняется.

LR(1)-подход состоит вот в чём: все наши определения и утверждения модифицируются так, чтобы учесть, какой символ стоит справа от разворачиваемого нетерминала (другими словами, чему равен Next при свёртке).

Пусть $\mathtt{K} \to \mathtt{U}$ — одно из правил грамматики, а t — некоторый терминал или спецсимвол EOI (который мы домысливаем в конце входного слова). Определим множество ЛевКонт$(\mathtt{K} \to \mathtt{U}, \mathtt{t})$ как множество всех слов, которые являются содержимым стека непосредственно перед свёрткой U в K в ходе успешного LR-процесса, при условии Next $=$ t (в момент свёртки).

Если отбросить у всех слов из ЛевКонт$(\mathtt{K} \to \mathtt{U})$ их конец U, то получится множество всех слов, которые могут появиться в правых выводах перед нетерминалом K, за которым стоит символ t. Это множество (не зависящее от того, какое из правил $\texttt{K} \to \texttt{U}$ для нетерминала K выбрано) мы будем обозначать Лев(K, t).

14.4.1. Написать грамматику для порождения множеств Лев(K, t).

Решение. Её нетерминалами будут символы $\langle$ЛевK t$\rangle$ для каждого нетерминала K и для каждого терминала t (а также для t = EOI). Её правила таковы. Пусть P — начальный нетерминал исходной грамматики. Тогда в новой грамматике будет правило

$$\langle\text{ЛевP EOI}\rangle \to \qquad \text{(пустое слово).}$$

Каждое правило исходной грамматики порождает несколько правил новой. Например, для правила

$$\texttt{K} \to \texttt{L u M N}$$

(L, M, N — нетерминалы, u — терминал) в новую грамматику мы добавим правила

$$\langle\text{ЛевL u}\rangle \to \langle\text{ЛевK x}\rangle$$

(для всех терминалов x);

$$\langle\text{ЛевM s}\rangle \to \langle\text{ЛевK y}\rangle\,\texttt{L u}$$

(для всех s, которые могут начинать слова, выводимые из N, и для всех y, а также для всех пар s = y, если из N выводимо пустое слово);

$$\langle\text{ЛевN s}\rangle \to \langle\text{ЛевK s}\rangle\,\texttt{L u M}$$

(для всех терминалов s).

14.4.2. Как меняется определение ситуации?

Решение. Ситуацией называется пара

[ситуация в старом смысле, терминал или EOI]

14.4.3. Как изменится определение согласованности?

Решение. Слово S из терминалов и нетерминалов согласовано с ситуацией [K → U_V, t] (здесь t — терминал или EOI), если S кончается на U, то есть S = TU, и, кроме того, T принадлежит Лев(K, t).

14.4.4. Каковы правила для индуктивного вычисления множества Сост(S) ситуаций, согласованных с данным словом S?

Ответ.

> (1) Если слово S согласовано с ситуацией [K → U_V, t], причём слово V начинается на букву J, то есть V = JW, то слово SJ согласовано с ситуацией [K → UJ_W, t].

Это правило полностью определяет все ситуации с непустой левой половиной (то есть не начинающиеся с подчёркивания), согласованные с SJ. Осталось определить, для каких нетерминалов K и терминалов t слово SJ принадлежит Лев(K, t). Это делается по двум правилам:

> (2) Если ситуация [L → U_V, t] согласована с SJ (согласно правилу (1)), а V начинается на нетерминал K, то SJ принадлежит Лев(K, s) для всех терминалов s, которые могут начинать слова, выводимые из слова V \ K (слово V без первой буквы K), а также для s = t, если из V \ K выводится пустое слово.
>
> (3) Если SJ входит в Лев(L, t) для некоторых L и t, причем L → V — правило грамматики и V начинается на нетерминал K, то SJ принадлежит Лев(K, s) для всех терминалов s, которые могут начинать слова, выводимые из V \ K, а также для s = t, если из V \ K выводится пустое слово.

14.4.5. Дать определения конфликтов сдвиг/свёртка и свёртка/свёртка по аналогии с данными выше.

Решение. Пусть дана некоторая грамматика. Пусть S — произвольное слово из терминалов и нетерминалов. Если множество Сост(S) содержит ситуацию, в которой справа от подчёркивания стоит терминал t, то говорят, что для пары ⟨S, t⟩ возможен сдвиг. (Это определение не изменилось по сравнению с SLR(1)-случаем — вторые компоненты пар из Сост(S) не учитываются.)

Если в Сост(S) есть ситуация, в которой справа от подчёркивания ничего нет, а вторым членом пары является терминал t, то говорят, что для пары ⟨S, t⟩ LR(1)-*возможна свёртка* (по соответствующему правилу). Говорят, что для пары ⟨S, t⟩ возникает LR(1)-*конфликт типа сдвиг/свёртка*, если возможны и сдвиг, и свёртка. Говорят, что для пары $\langle \mathtt{S}, \mathtt{t} \rangle$ возникает LR(1)-*конфликт типа свёртка/свёртка*, если есть несколько правил, по которым возможна свёртка.

Грамматика называется LR(1)-*грамматикой*, если в ней нет LR(1)-конфликтов типа сдвиг/свёртка и свёртка/свёртка ни для одной пары $\langle \mathtt{S}, \mathtt{t} \rangle$

14.4.6. Построить алгоритм проверки выводимости слова в LR(1)-грамматике.

Решение. Как и раньше, на каждом шаге LR-процесса можно однозначно определить, какой шаг только и может быть следующим.

Полезно (в частности, для LALR(1)-разбора, смотри ниже) понять, как связаны понятия LR(0) и LR(1)-согласованности.

14.4.7. Сформулировать и доказать соответствующее утверждение.

Ответ. Пусть фиксирована некоторая грамматика. Слово $\mathtt{S}$ из терминалов и нетерминалов является LR(0)-согласованным с ситуацией $\mathtt{K} \to \mathtt{U\_V}$ тогда и только тогда, когда оно LR(1)-согласовано с парой $[\mathtt{K} \to \mathtt{U\_V}, \mathtt{t}]$ для некоторого терминала $\mathtt{t}$ (или для $\mathtt{t} = \mathtt{EOI}$). То же самое другими словами: Лев($\mathtt{K}$) есть объединение Лев($\mathtt{K}, \mathtt{t}$) по всем $\mathtt{t}$. В последней форме это совсем ясно.

Замечание. Таким образом, функция Сост($\mathtt{S}$) в LR(1)-смысле является расширением функции Сост($\mathtt{S}$) в LR(0)-смысле: $\text{Сост}_{\text{LR(0)}}(\mathtt{S})$ получается из $\text{Сост}_{\text{LR(1)}}(\mathtt{S})$, если во всех парах выбросить вторые члены.

Теперь мы можем дать определение LALR(1)-грамматики. Пусть фиксирована некоторая грамматика, $\mathtt{S}$ — слово из нетерминалов и терминалов, $\mathtt{t}$ — некоторый терминал (или $\mathtt{EOI}$). Будем говорить, что для пары $\langle \mathtt{S}, \mathtt{t} \rangle$ LALR(1)-возможна свёртка по некоторому правилу, если существует другое слово $\mathtt{S}_1$ с $\text{Сост}_{\text{LR(0)}}(\mathtt{S}_0) = \text{Сост}_{\text{LR(0)}}(\mathtt{S}_1)$, причем для пары $\langle \mathtt{S}_1, \mathtt{t} \rangle$ LR(1)-возможна свёртка по рассматриваемому правилу. Далее определяются конфликты (естественным образом), и грамматика называется LALR(1)-грамматикой, если конфликтов нет.

14.4.8. Доказать, что всякая SLR(1)-грамматика является LALR(1)-грамматикой, а всякая LALR(1)-грамматика является LR(1)-грамматикой.

[Указание. Это — простое следствие определений.]

14.4.9. Построить алгоритм проверки выводимости в LALR(1)-грамматике, который хранит в стеке меньше информации, чем соответствующий LR(1)-алгоритм.

[Указание. Достаточно хранить в стеке множества $\text{Сост}_{LR(0)}(S)$, поскольку согласно определению LALR(1)-возможность свёртки ими определяется. (Так что сам алгоритм ничем не отличается от SLR(1)-случая, кроме таблицы возможных свёрток.)]

14.4.10. Привести пример LALR(1)-грамматики, не являющейся SLR(1)-грамматикой.

14.4.11. Привести пример LR(1)-грамматики, не являющейся LALR(1)-грамматикой.

14.5. Общие замечания о разных методах разбора

Применение этих методов на практике имеет свои хитрости и тонкости, которых мы не касались. (Например, таблицы следует хранить по возможности экономно.) Часто оказывается также, что для некоторого входного языка наиболее естественная грамматика не является LL(1)-грамматикой, но является LR(1)-грамматикой, а также может быть заменена на LL(1)-грамматику без изменения языка. Какой из этих вариантов выбрать, не всегда ясно. Дилетантский совет: если Вы сами проектируете входной язык, то не следует выпендриваться и употреблять одни и те же символы для разных целей — и тогда обычно несложно написать LL(1)-грамматику или рекурсивный анализатор. Если же входной язык задан заранее с помощью LR(1)-грамматики, не являющейся LL(1)-грамматикой, то лучше её не трогать, а разбирать как есть. При этом могут оказаться полезные средства автоматического порождения анализаторов, наиболее известными из которых являются `yacc` (UNIX) и `bison` (GNU).

Большое количество полезной и хорошо изложенной информации о теории и практике синтаксического разбора имеется в книге: Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman. Compilers: principles, techniques and tools. Addison-Wesley (1985).

Книги для чтения

А. Ахо, Дж. Хопкрофт, Дж. Ульман. Построение и анализ вычислительных алгоритмов. М.:Мир, 1979.

Н. Вирт. Систематическое программирование. Введение. М.:Мир, 1977.

Н. Вирт. Алгоритмы + структуры данных = программы. М.:Мир, 1985.

Д. Грис. Наука программирования. М.:Мир, 1984.

М. Гэри, Д. Джонсон. Вычислительные машины и труднорешаемые задачи. М.:Мир, 1982.

Э. Дейкстра. Дисциплина программирования. М.:Мир, 1978.

А.Г. Кушниренко, Г.В. Лебедев. Программирование для математиков. М.:Наука, 1988.

В. Липский. Комбинаторика для программистов. М.:Мир, 1988.

Э. Рейнгольд, Ю. Нивергельт, Н. Део. Комбинаторные алгоритмы. Теория и практика. М.:Мир, 1980.

A. Aho, R. Sethi, J. Ullman. Compilers: principles, techniques and tools. Addison-Wesley, 1986

T. Cormen, C. Leiserson, R. Rivest. Introduction to algorithms. Cambridge (Mass.): The MIT Press, 1990.

Предметный указатель