Лингвистические основы информатики
Лекция 16
Обработчик синтаксических ошибок LL(k)-грамматики. Рекурсивный спуск

Ю. В. Нагребецкая

Уральский федеральный университет
Институт естественных наук и математики
Департамент математики, механики и компьютерных наук
Направления: Математика и компьютерные науки
Компьютерная безопасность
(6 семестр)

Пример 4 грамматики арифметических выражений

Рассмотрим (однозначную) грамматику G арифметических выражений (пример 2, лекция 3):

$$S \to S + A \mid A$$
$$A \to A * B \mid B$$
$$B \to (S) \mid x$$

Пример 4 грамматики арифметических выражений

Рассмотрим (однозначную) грамматику G арифметических выражений (пример 2, лекция 3):

$$S \to S + A \mid A$$
$$A \to A * B \mid B$$
$$B \to (S) \mid x$$

Она, очевидно, содержит левую рекурсию (почему?). Устраним ее, получим грамматику G':

Пример 4 грамматики арифметических выражений

Рассмотрим (однозначную) грамматику G арифметических выражений (пример 2, лекция 3):

$$S \to S + A \mid A$$
$$A \to A * B \mid B$$
$$B \to (S) \mid x$$

Она, очевидно, содержит левую рекурсию (почему?). Устраним ее, получим грамматику G':

$$S \to AS'$$
$$S' \to +AS' \mid \varepsilon$$
$$A \to BA'$$
$$A' \to *BA' \mid \varepsilon$$
$$B \to (S) \mid x$$

Пример 4. множества *FIRST* и *FOLLOW*

Найдем множества *FIRST* и *FOLLOW* для грамматики G' (фигурные скобки, обозначающие множества будем опускать).

нетерминалы	*FIRST*	*FOLLOW*
S	$(, x$	$), \dashv$
S'	$+, \varepsilon$	$), \dashv$
A	$(, x$	$+,), \dashv$
A'	$*, \varepsilon$	$+,), \dashv$
B	$(, x$	$*$

правые части правил	*FIRST*
AS'	$(, x$
$+AS'$	$+$
ε	ε
BA'	$(, x$
$*BA'$	$*$
(S)	$($
x	x

Пример 4. Множество $SELECT$ для грамматики G'

Найдем множество $SELECT$ для грамматики G'.

$A \to \gamma$	$FIRST(\gamma)$	$SELECT(A \to \gamma)$
$S \to AS'$	$(, x$	$(, x$
$S' \to +AS'$	$+$	$+$
$S' \to \varepsilon$	ε	$), \dashv$
$A \to BA'$	$(, x$	$(, x$
$A \to *BA'$	$*$	$*$
$A' \to \varepsilon$	ε	$+,), \dashv$
$B \to (S)$	$($	$($
$B \to x$	x	x

Пример 4. Таблица переходов δ МП-автомата $\mathcal{M}'$

Построим таблицу нисходящего анализатора — таблицу переходов δ МП-автомата $\mathcal{M}'$ (круглые скобки мы опускаем).

	x	+	*	(	)	$\dashv$
S	$AS', \_$			$AS', \_$		
S'		$+AS', \_$			$\varepsilon, \_$	$\varepsilon, \_$
A	$BA', \_$			$BA', \_$		
A'		$\varepsilon, \_$	$*BA', \_$		$\varepsilon, \_$	$\varepsilon, \_$
B	$x, \_$			$(S), \_$		
x	$\varepsilon, \rightarrow$					
$+$		$\varepsilon, \rightarrow$				
$*$			$\varepsilon, \rightarrow$			
$($				$\varepsilon, \rightarrow$		
$)$					$\varepsilon, \rightarrow$	
∇						$\vee$

Пример 4. Таблица переходов δ' МП-автомата $\mathcal{M}''$

Упростим МП-автомат $\mathcal{M}'$ по лемме 3 Лекции 14, получим МП-автомат $\mathcal{M}''$.

	x	+	*	(	)	$\dashv$
S	$AS', \_$			$AS', \_$		
S'		$AS', \rightarrow$			$\varepsilon, \_$	$\varepsilon, \_$
A	$BA', \_$			$BA', \_$		
A'		$\varepsilon, \_$	$BA', \rightarrow$		$\varepsilon, \_$	$\varepsilon, \_$
B	$\varepsilon, \rightarrow$			$S), \rightarrow$		
)					$\varepsilon, \rightarrow$	
∇						$\vee$

Пример 4. Протокол разбора цепочки нисх. анализ.

Рассмотрим сначала левосторонний вывод цепочки правильной цепочки $w = x + x * x$ в грамматике G:

$S \Rightarrow \underline{S + A} \Rightarrow \underline{A} + A \Rightarrow \underline{B} + A \Rightarrow \underline{x} + A \Rightarrow x + \underline{A * B} \Rightarrow x + \underline{B} * B \Rightarrow x + \underline{x} * B \Rightarrow x + x * \underline{x}$

Потом рассмотрим левосторонний вывод цепочки этой цепочки $w = x + x * x$ в грамматике G':

$S \Rightarrow \underline{AS'} \Rightarrow \underline{BA'S'} \Rightarrow \underline{xA'}S' \Rightarrow x\underline{\varepsilon}S' = xS' \Rightarrow x\underline{+AS'} \Rightarrow x + \underline{BA'}S' \Rightarrow x + \underline{x}A'S' \Rightarrow x + x * \underline{B}A'S' \Rightarrow x + x * \underline{x}A'S' \Rightarrow x + x * x\underline{\varepsilon}S' = x + x * xS' \Rightarrow x + x * x\underline{\varepsilon} = x + x * \underline{x}$

Пример 4. Протокол разбора цепочки нисходящего анализа

И рассмотрим протокол вывода этой цепочки в грамматике G':

	стек	позиция указателя
1	$S\nabla$	$\diamond$x+x*x$\dashv$
2	$AS'\nabla$	$\diamond$x+x*x$\dashv$
3	$BA'S'\nabla$	$\diamond$x+x*x$\dashv$
4	$A'S'\nabla$	x$\diamond$+x*x$\dashv$
5	$S'\nabla$	x$\diamond$+x*x$\dashv$
6	$AS'\nabla$	x+$\diamond$x*x$\dashv$
7	$BA'S'\nabla$	x+$\diamond$x*x$\dashv$
8	$A'S'\nabla$	x+x$\diamond$*x$\dashv$
9	$BA'S'\nabla$	x+x*$\diamond$x$\dashv$
10	$A'S'\nabla$	x+x*x$\diamond\dashv$
11	$S'\nabla$	x+x*x$\diamond\dashv$
12	∇	x+x*x$\diamond\dashv\vee$

Обработка синтаксических ошибок

- Хороший синтаксический анализатор должен сообщать о наличии ошибки с указанием места в программе, где эта ошибка была обнаружена, и предполагаемого типа ошибки.

Обработка синтаксических ошибок

- Хороший синтаксический анализатор должен сообщать о наличии ошибки с указанием места в программе, где эта ошибка была обнаружена, и предполагаемого типа ошибки.
- Но этого недостаточно. Анализатор должен уметь исправить ошибку так, чтобы стало возможным продолжение анализа входной цепочки для обнаружения последующих ошибок (или установления их отсутствия).

Обработка синтаксических ошибок

- Хороший синтаксический анализатор должен сообщать о наличии ошибки с указанием места в программе, где эта ошибка была обнаружена, и предполагаемого типа ошибки.
- Но этого недостаточно. Анализатор должен уметь исправить ошибку так, чтобы стало возможным продолжение анализа входной цепочки для обнаружения последующих ошибок (или установления их отсутствия).
- Рассматриваемый анализатор обнаруживает ошибку в тот момент, когда не может продолжить работу из-за отсутствия в МП-автомате команды, левая часть которой есть текущая пара (входной символ, верхний символ стека).

Обработка синтаксических ошибок

- Хороший синтаксический анализатор должен сообщать о наличии ошибки с указанием места в программе, где эта ошибка была обнаружена, и предполагаемого типа ошибки.
- Но этого недостаточно. Анализатор должен уметь исправить ошибку так, чтобы стало возможным продолжение анализа входной цепочки для обнаружения последующих ошибок (или установления их отсутствия).
- Рассматриваемый анализатор обнаруживает ошибку в тот момент, когда не может продолжить работу из-за отсутствия в МП-автомате команды, левая часть которой есть текущая пара (входной символ, верхний символ стека).
- В соответствующую пустую клетку управляющей таблицы можно поместить ссылку на подпрограмму обработки возникшей ошибки.

Обработка синтаксических ошибок

Для примера разберем типы синтаксических ошибок для грамматики G'. Их четыре:

- e_1: отсутствует операнд (например, два оператора подряд), вставить операнд x;

Обработка синтаксических ошибок

Для примера разберем типы синтаксических ошибок для грамматики G'. Их четыре:

- e_1: отсутствует операнд (например, два оператора подряд), вставить операнд x;
- e_2: отсутствует оператор (два операнда подряд), вставить $+$;

Обработка синтаксических ошибок

Для примера разберем типы синтаксических ошибок для грамматики G'. Их четыре:

- e_1: отсутствует операнд (например, два оператора подряд), вставить операнд x;
- e_2: отсутствует оператор (два операнда подряд), вставить $+$;
- e_3: нет), добавить);

Обработка синтаксических ошибок

Для примера разберем типы синтаксических ошибок для грамматики G'. Их четыре:

- e_1: отсутствует операнд (например, два оператора подряд), вставить операнд x;
- e_2: отсутствует оператор (два операнда подряд), вставить $+$;
- e_3: нет), добавить);
- e_4: лишняя), удалить её.

Обработка синтаксических ошибок

Для примера разберем типы синтаксических ошибок для грамматики G'. Их четыре:

- e_1: отсутствует операнд (например, два оператора подряд), вставить операнд x;
- e_2: отсутствует оператор (два операнда подряд), вставить $+$;
- e_3: нет), добавить);
- e_4: лишняя), удалить её.
- e_5: ожидается $+$, заменить * на $+$.

Обработка синтаксических ошибок (продолжение)

Для того, чтобы сделать обработчик ошибок, для каждого нетерминала грамматики ведем понятие множества PREVIOUS, двойственное к понятию FOLLOW.

Обработка синтаксических ошибок (продолжение)

Для того, чтобы сделать обработчик ошибок, для каждого нетерминала грамматики ведем понятие множества PREVIOUS, двойственное к понятию FOLLOW.

- Пусть $A \in \Gamma$. Множеством $PREVIOUS(A)$ назовем такое подмножество множества $\Sigma \cup \{\vdash\}$, что

Обработка синтаксических ошибок (продолжение)

Для того, чтобы сделать обработчик ошибок, для каждого нетерминала грамматики ведем понятие множества PREVIOUS, двойственное к понятию FOLLOW.

- Пусть $A \in \Gamma$. Множеством $PREVIOUS(A)$ назовем такое подмножество множества $\Sigma \cup \{\vdash\}$, что
  1. терминал $a \in \Sigma$ принадлежит множеству $PREVIOUS(A)$, если и только если $S \Rightarrow^* \alpha a A \beta$ для некоторых $\alpha, \beta \in (\Sigma \cup \Gamma)^*$.

Обработка синтаксических ошибок (продолжение)

Для того, чтобы сделать обработчик ошибок, для каждого нетерминала грамматики ведем понятие множества PREVIOUS, двойственное к понятию FOLLOW.

- Пусть $A \in \Gamma$. Множеством $PREVIOUS(A)$ назовем такое подмножество множества $\Sigma \cup \{\vdash\}$, что
  1. терминал $a \in \Sigma$ принадлежит множеству $PREVIOUS(A)$, если и только если $S \Rightarrow^* \alpha a A \beta$ для некоторых $\alpha, \beta \in (\Sigma \cup \Gamma)^*$.
  2. $\vdash \in PREVIOUS(A)$, если и только если $S \Rightarrow^* A\alpha$.

Обработка синтаксических ошибок (продолжение)

Для того, чтобы сделать обработчик ошибок, для каждого нетерминала грамматики ведем понятие множества PREVIOUS, двойственное к понятию FOLLOW.

- Пусть $A \in \Gamma$. Множеством $PREVIOUS(A)$ назовем такое подмножество множества $\Sigma \cup \{\vdash\}$, что
  1. терминал $a \in \Sigma$ принадлежит множеству $PREVIOUS(A)$, если и только если $S \Rightarrow^* \alpha a A \beta$ для некоторых $\alpha, \beta \in (\Sigma \cup \Gamma)^*$.
  2. $\vdash \in PREVIOUS(A)$, если и только если $S \Rightarrow^* A\alpha$.
  3. $\vdash \in PREVIOUS(S)$, поскольку можно считать, что $S \Rightarrow^* S$.

Обработка синтаксических ошибок (продолжение)

Для того, чтобы сделать обработчик ошибок, для каждого нетерминала грамматики ведем понятие множества PREVIOUS, двойственное к понятию FOLLOW.

- Пусть $A \in \Gamma$. Множеством $PREVIOUS(A)$ назовем такое подмножество множества $\Sigma \cup \{\vdash\}$, что
  1. терминал $a \in \Sigma$ принадлежит множеству $PREVIOUS(A)$, если и только если $S \Rightarrow^* \alpha a A \beta$ для некоторых $\alpha, \beta \in (\Sigma \cup \Gamma)^*$.
  2. $\vdash \in PREVIOUS(A)$, если и только если $S \Rightarrow^* A\alpha$.
  3. $\vdash \in PREVIOUS(S)$, поскольку можно считать, что $S \Rightarrow^* S$.
- Распространим отображение $PREVIOUS : \Gamma \to 2^\Sigma$ (что такое 2^Σ) с множества Γ на множество $\Sigma \cup \Gamma$ естественным образом:

Обработка синтаксических ошибок (продолжение)

Для того, чтобы сделать обработчик ошибок, для каждого нетерминала грамматики ведем понятие множества PREVIOUS, двойственное к понятию FOLLOW.

- Пусть $A \in \Gamma$. Множеством $PREVIOUS(A)$ назовем такое подмножество множества $\Sigma \cup \{\vdash\}$, что
  1. терминал $a \in \Sigma$ принадлежит множеству $PREVIOUS(A)$, если и только если $S \Rightarrow^* \alpha a A \beta$ для некоторых $\alpha, \beta \in (\Sigma \cup \Gamma)^*$.
  2. $\vdash \in PREVIOUS(A)$, если и только если $S \Rightarrow^* A\alpha$.
  3. $\vdash \in PREVIOUS(S)$, поскольку можно считать, что $S \Rightarrow^* S$.
- Распространим отображение $PREVIOUS : \Gamma \to 2^\Sigma$ (что такое 2^Σ) с множества Γ на множество $\Sigma \cup \Gamma$ естественным образом:
  1. для любого $b \in \Sigma$ терминал $a \in \Sigma$ принадлежит множеству $PREVIOUS(b)$, если и только если $S \Rightarrow^* \alpha a b \beta$ для некоторых $\alpha, \beta \in (\Sigma \cup \Gamma)^*$.

Обработка синтаксических ошибок (продолжение)

Для того, чтобы сделать обработчик ошибок, для каждого нетерминала грамматики ведем понятие множества PREVIOUS, двойственное к понятию FOLLOW.

- Пусть $A \in \Gamma$. Множеством $PREVIOUS(A)$ назовем такое подмножество множества $\Sigma \cup \{\vdash\}$, что
  1. терминал $a \in \Sigma$ принадлежит множеству $PREVIOUS(A)$, если и только если $S \Rightarrow^* \alpha a A \beta$ для некоторых $\alpha, \beta \in (\Sigma \cup \Gamma)^*$.
  2. $\vdash \in PREVIOUS(A)$, если и только если $S \Rightarrow^* A\alpha$.
  3. $\vdash \in PREVIOUS(S)$, поскольку можно считать, что $S \Rightarrow^* S$.
- Распространим отображение $PREVIOUS : \Gamma \to 2^\Sigma$ (что такое 2^Σ) с множества Γ на множество $\Sigma \cup \Gamma$ естественным образом:
  1. для любого $b \in \Sigma$ терминал $a \in \Sigma$ принадлежит множеству $PREVIOUS(b)$, если и только если $S \Rightarrow^* \alpha a b \beta$ для некоторых $\alpha, \beta \in (\Sigma \cup \Gamma)^*$.
  2. $\vdash \in PREVIOUS(a)$, если и только если $S \Rightarrow^* b\alpha$.

Обработка синтаксических ошибок. Понятие множества $\overline{FOLLOW}$. Связь множеств $\overline{FOLLOW}$ и $PREVIOUS$.

- Распространим аналогично отображение $FOLLOW : \Gamma \to 2^{\Sigma}$ с множества Γ на множество $\Sigma \cup \Gamma$:

Обработка синтаксических ошибок. Понятие множества $\overline{FOLLOW}$. Связь множеств $\overline{FOLLOW}$ и $PREVIOUS$.

- Распространим аналогично отображение $FOLLOW : \Gamma \to 2^{\Sigma}$ с множества Γ на множество $\Sigma \cup \Gamma$:
  1. для любого $b \in \Sigma$ терминал $a \in \Sigma$ принадлежит множеству $\overline{FOLLOW}(b)$, если и только если $S \Rightarrow^* \alpha b a \beta$ для некоторых $\alpha, \beta \in (\Sigma \cup \Gamma)^*$.

Обработка синтаксических ошибок. Понятие множества $\overline{FOLLOW}$. Связь множеств $\overline{FOLLOW}$ и $PREVIOUS$.

- Распространим аналогично отображение $FOLLOW : \Gamma \to 2^{\Sigma}$ с множества Γ на множество $\Sigma \cup \Gamma$:
  1. для любого $b \in \Sigma$ терминал $a \in \Sigma$ принадлежит множеству $\overline{FOLLOW}(b)$, если и только если $S \Rightarrow^* \alpha b a \beta$ для некоторых $\alpha, \beta \in (\Sigma \cup \Gamma)^*$.
  2. для любого $b \in \Sigma$ символ $\dashv \in \overline{FOLLOW}(b)$, если и только если $S \Rightarrow^* \alpha b$.

Обработка синтаксических ошибок. Понятие множества $\overline{FOLLOW}$. Связь множеств $\overline{FOLLOW}$ и $PREVIOUS$.

- Распространим аналогично отображение $FOLLOW : \Gamma \to 2^{\Sigma}$ с множества Γ на множество $\Sigma \cup \Gamma$:
  1. для любого $b \in \Sigma$ терминал $a \in \Sigma$ принадлежит множеству $\overline{FOLLOW}(b)$, если и только если $S \Rightarrow^* \alpha b a \beta$ для некоторых $\alpha, \beta \in (\Sigma \cup \Gamma)^*$.
  2. для любого $b \in \Sigma$ символ $\dashv \in \overline{FOLLOW}(b)$, если и только если $S \Rightarrow^* \alpha b$.
- Чтобы не путать полученное отображение с отображением FOLLOW, определенным только для нетерминалов, обозначим это новое отображение через $\overline{FOLLOW}$.

Обработка синтаксических ошибок. Понятие множества $\overline{FOLLOW}$. Связь множеств $\overline{FOLLOW}$ и $PREVIOUS$.

- Распространим аналогично отображение $FOLLOW : \Gamma \to 2^{\Sigma}$ с множества Γ на множество $\Sigma \cup \Gamma$:
  1. для любого $b \in \Sigma$ терминал $a \in \Sigma$ принадлежит множеству $\overline{FOLLOW}(b)$, если и только если $S \Rightarrow^* \alpha b a \beta$ для некоторых $\alpha, \beta \in (\Sigma \cup \Gamma)^*$.
  2. для любого $b \in \Sigma$ символ $\dashv \in \overline{FOLLOW}(b)$, если и только если $S \Rightarrow^* \alpha b$.
- Чтобы не путать полученное отображение с отображением FOLLOW, определенным только для нетерминалов, обозначим это новое отображение через $\overline{FOLLOW}$.
- Получится тогда такой алгоритм вычисления $\overline{FOLLOW}$ (см. след. слайд).

Алгоритм построения множества $\overline{FOLLOW}$

Вход: КС-грамматика $G = (\Sigma, \Gamma, P, S)$.

Выход: Набор множеств $\overline{FOLLOW}(X)$ для всех $X \in \Gamma \cup \Sigma$

- for $X \in \Gamma \cup \Sigma$:

Алгоритм построения множества $\overline{FOLLOW}$

Вход: КС-грамматика $G = (\Sigma, \Gamma, P, S)$.

Выход: Набор множеств $\overline{FOLLOW}(X)$ для всех $X \in \Gamma \cup \Sigma$

- for $X \in \Gamma \cup \Sigma$:
  $\overline{FOLLOW}(X) = \emptyset$

Алгоритм построения множества $\overline{FOLLOW}$

Вход: КС-грамматика $G = (\Sigma, \Gamma, P, S)$.

Выход: Набор множеств $\overline{FOLLOW}(X)$ для всех $X \in \Gamma \cup \Sigma$

- for $X \in \Gamma \cup \Sigma$:
  $\overline{FOLLOW}(X) = \emptyset$
- $\overline{FOLLOW}(S) = \{\dashv\}$

Алгоритм построения множества $\overline{FOLLOW}$

Вход: КС-грамматика $G = (\Sigma, \Gamma, P, S)$.

Выход: Набор множеств $\overline{FOLLOW}(X)$ для всех $X \in \Gamma \cup \Sigma$

- for $X \in \Gamma \cup \Sigma$:
  $\overline{FOLLOW}(X) = \emptyset$
- $\overline{FOLLOW}(S) = \{\dashv\}$
- while (все множества $\overline{FOLLOW}(A)$ для всех $A \in \Gamma$ не стабилизировались):

Алгоритм построения множества $\overline{FOLLOW}$

Вход: КС-грамматика $G = (\Sigma, \Gamma, P, S)$.

Выход: Набор множеств $\overline{FOLLOW}(X)$ для всех $X \in \Gamma \cup \Sigma$

- for $X \in \Gamma \cup \Sigma$:
  $\overline{FOLLOW}(X) = \emptyset$
- $\overline{FOLLOW}(S) = \{\dashv\}$
- while (все множества $\overline{FOLLOW}(A)$ для всех $A \in \Gamma$ не стабилизировались):
  for $(A \to X_1 X_2 \dots X_n) \in P$:

Алгоритм построения множества $\overline{FOLLOW}$

Вход: КС-грамматика $G = (\Sigma, \Gamma, P, S)$.

Выход: Набор множеств $\overline{FOLLOW}(X)$ для всех $X \in \Gamma \cup \Sigma$

- for $X \in \Gamma \cup \Sigma$:
  $\overline{FOLLOW}(X) = \emptyset$
- $\overline{FOLLOW}(S) = \{\dashv\}$
- while (все множества $\overline{FOLLOW}(A)$ для всех $A \in \Gamma$ не стабилизировались):
  for $(A \to X_1 X_2 \dots X_n) \in P$:
  1. $P = P \cup (X_{n+1} \to \varepsilon)$

Алгоритм построения множества $\overline{FOLLOW}$

Вход: КС-грамматика $G = (\Sigma, \Gamma, P, S)$.

Выход: Набор множеств $\overline{FOLLOW}(X)$ для всех $X \in \Gamma \cup \Sigma$

- for $X \in \Gamma \cup \Sigma$:
  $\overline{FOLLOW}(X) = \emptyset$
- $\overline{FOLLOW}(S) = \{\dashv\}$
- while (все множества $\overline{FOLLOW}(A)$ для всех $A \in \Gamma$ не стабилизировались):
  for $(A \to X_1 X_2 \dots X_n) \in P$:
  1. $P = P \cup (X_{n+1} \to \varepsilon)$
  2. $ann = true$

Алгоритм построения множества $\overline{FOLLOW}$

Вход: КС-грамматика $G = (\Sigma, \Gamma, P, S)$.

Выход: Набор множеств $\overline{FOLLOW}(X)$ для всех $X \in \Gamma \cup \Sigma$

- for $X \in \Gamma \cup \Sigma$:
  $\overline{FOLLOW}(X) = \emptyset$
- $\overline{FOLLOW}(S) = \{\dashv\}$
- while (все множества $\overline{FOLLOW}(A)$ для всех $A \in \Gamma$ не стабилизировались):
  for $(A \to X_1 X_2 \dots X_n) \in P$:
  1. $P = P \cup (X_{n+1} \to \varepsilon)$
  2. $ann = true$
  3. for i in range(n,0,-1):

Алгоритм построения множества $\overline{FOLLOW}$

Вход: КС-грамматика $G = (\Sigma, \Gamma, P, S)$.

Выход: Набор множеств $\overline{FOLLOW}(X)$ для всех $X \in \Gamma \cup \Sigma$

- for $X \in \Gamma \cup \Sigma$:
  $\overline{FOLLOW}(X) = \emptyset$
- $\overline{FOLLOW}(S) = \{\dashv\}$
- while (все множества $\overline{FOLLOW}(A)$ для всех $A \in \Gamma$ не стабилизировались):
  for $(A \to X_1 X_2 \dots X_n) \in P$:
  1. $P = P \cup (X_{n+1} \to \varepsilon)$
  2. $ann = true$
  3. for i in range(n,0,-1):
     1. $\overline{FOLLOW}(X_i) = \overline{FOLLOW}(X_i) \cup (FIRST(X_{i+1} \dots X_{n+1}) \backslash \{\varepsilon\})$

Алгоритм построения множества $\overline{FOLLOW}$

Вход: КС-грамматика $G = (\Sigma, \Gamma, P, S)$.

Выход: Набор множеств $\overline{FOLLOW}(X)$ для всех $X \in \Gamma \cup \Sigma$

- for $X \in \Gamma \cup \Sigma$:
  $\overline{FOLLOW}(X) = \emptyset$
- $\overline{FOLLOW}(S) = \{\dashv\}$
- while (все множества $\overline{FOLLOW}(A)$ для всех $A \in \Gamma$ не стабилизировались):
  for $(A \to X_1 X_2 \dots X_n) \in P$:
  1. $P = P \cup (X_{n+1} \to \varepsilon)$
  2. $ann = true$
  3. for i in range(n,0,-1):
     1. $\overline{FOLLOW}(X_i) = \overline{FOLLOW}(X_i) \cup (FIRST(X_{i+1} \dots X_{n+1}) \backslash \{\varepsilon\})$
     2. if ann:
        $\overline{FOLLOW}(X_i) = \overline{FOLLOW}(X_i) \cup \overline{FOLLOW}(A)$

Алгоритм построения множества $\overline{FOLLOW}$

Вход: КС-грамматика $G = (\Sigma, \Gamma, P, S)$.

Выход: Набор множеств $\overline{FOLLOW}(X)$ для всех $X \in \Gamma \cup \Sigma$

- for $X \in \Gamma \cup \Sigma$:
  $\overline{FOLLOW}(X) = \emptyset$
- $\overline{FOLLOW}(S) = \{\dashv\}$
- while (все множества $\overline{FOLLOW}(A)$ для всех $A \in \Gamma$ не стабилизировались):
  for $(A \to X_1 X_2 \dots X_n) \in P$:
  1. $P = P \cup (X_{n+1} \to \varepsilon)$
  2. $ann = true$
  3. for i in range(n,0,-1):
     1. $\overline{FOLLOW}(X_i) = \overline{FOLLOW}(X_i) \cup (FIRST(X_{i+1} \dots X_{n+1}) \backslash \{\varepsilon\})$
     2. if ann:
        $\overline{FOLLOW}(X_i) = \overline{FOLLOW}(X_i) \cup \overline{FOLLOW}(A)$
     3. $ann = ann \wedge (\varepsilon \in FIRST(X_i))$

Алгоритм построения множества $\overline{FOLLOW}$

Вход: КС-грамматика $G = (\Sigma, \Gamma, P, S)$.

Выход: Набор множеств $\overline{FOLLOW}(X)$ для всех $X \in \Gamma \cup \Sigma$

- for $X \in \Gamma \cup \Sigma$:
  $\overline{FOLLOW}(X) = \emptyset$
- $\overline{FOLLOW}(S) = \{\dashv\}$
- while (все множества $\overline{FOLLOW}(A)$ для всех $A \in \Gamma$ не стабилизировались):
  for $(A \to X_1 X_2 \dots X_n) \in P$:
  1. $P = P \cup (X_{n+1} \to \varepsilon)$
  2. $ann = true$
  3. for i in range(n,0,-1):
     1. $\overline{FOLLOW}(X_i) = \overline{FOLLOW}(X_i) \cup (FIRST(X_{i+1} \dots X_{n+1}) \backslash \{\varepsilon\})$
     2. if ann:
        $\overline{FOLLOW}(X_i) = \overline{FOLLOW}(X_i) \cup \overline{FOLLOW}(A)$
     3. $ann = ann \wedge (\varepsilon \in FIRST(X_i))$

Реверс контекстно-свободного языка

- Из теории автоматов известно, что реверс (что такое реверс?) регулярного языка тоже является регулярным.

Реверс контекстно-свободного языка

- Из теории автоматов известно, что реверс (что такое реверс?) регулярного языка тоже является регулярным.
- Пусть $H = (\Gamma, \Sigma, P, S)$ — произвольная КС-грамматика.

Реверс контекстно-свободного языка

- Из теории автоматов известно, что реверс (что такое реверс?) регулярного языка тоже является регулярным.
- Пусть $H = (\Gamma, \Sigma, P, S)$ — произвольная КС-грамматика.
- Обозначим через $H^{reverse}$ грамматику, каждое правило $A \to X_n X_{n-1} \dots X_2 X_1$ которой получено из правила $A \to X_1 X_2 \dots X_{n-1} X_n$ грамматики H реверсом ее правой части.

Реверс контекстно-свободного языка (продолжение)

Утверждение 1 (упр. 1) (о КС грамматике реверса языка)

- Дерево вывода грамматики $H^{reverse}$ зеркально симметрично дереву вывода грамматики H.

Реверс контекстно-свободного языка (продолжение)

Утверждение 1 (упр. 1) (о КС грамматике реверса языка)

- Дерево вывода грамматики $H^{reverse}$ зеркально симметрично дереву вывода грамматики H.
- Левостороннему выводу $S \Rightarrow_H^* w$ слова $w \in L(H)$ однозначно соответствует правосторонний вывод $S \Rightarrow_{H^{reverse}}^* w^{reverse}$ слова $w^{reverse} \in L(H^{reverse})$, который получен реверсом каждого слова из вывода $S \Rightarrow_H^* w$.

Реверс контекстно-свободного языка (продолжение)

Утверждение 1 (упр. 1) (о КС грамматике реверса языка)

- Дерево вывода грамматики $H^{reverse}$ зеркально симметрично дереву вывода грамматики H.
- Левостороннему выводу $S \Rightarrow_H^* w$ слова $w \in L(H)$ однозначно соответствует правосторонний вывод $S \Rightarrow_{H^{reverse}}^* w^{reverse}$ слова $w^{reverse} \in L(H^{reverse})$, который получен реверсом каждого слова из вывода $S \Rightarrow_H^* w$.
- Язык $L(H^{reverse})$ является реверсом языка $L(H)$.

Реверс контекстно-свободного языка (продолжение)

Утверждение 1 (упр. 1) (о КС грамматике реверса языка)

- Дерево вывода грамматики $H^{reverse}$ зеркально симметрично дереву вывода грамматики H.
- Левостороннему выводу $S \Rightarrow_H^* w$ слова $w \in L(H)$ однозначно соответствует правосторонний вывод $S \Rightarrow_{H^{reverse}}^* w^{reverse}$ слова $w^{reverse} \in L(H^{reverse})$, который получен реверсом каждого слова из вывода $S \Rightarrow_H^* w$.
- Язык $L(H^{reverse})$ является реверсом языка $L(H)$.

Доказательство. (1) очевидно, (2) следует из (1), а (3) из (2).

Множество $PREVIOUS$ для грамматики G'

Утверждение 2 (упр. 2) (о связи множеств $\overline{FOLLOW}$ и $PREVIOUS$)

Пусть H — LL(1)-грамматика. Тогда для любого $X \in \Gamma \cup \Sigma$ множество $PREVIOUS(X)$ в грамматике H равно множеству $\overline{FOLLOW}(X)$ в грамматике $H^{reverse}$ с учетом замены $\dashv$ на $\vdash$.

Множество $PREVIOUS$ для грамматики G'

Утверждение 2 (упр. 2) (о связи множеств $\overline{FOLLOW}$ и $PREVIOUS$)

Пусть H — LL(1)-грамматика. Тогда для любого $X \in \Gamma \cup \Sigma$ множество $PREVIOUS(X)$ в грамматике H равно множеству $\overline{FOLLOW}(X)$ в грамматике $H^{reverse}$ с учетом замены $\dashv$ на $\vdash$.

Указание: Использовать утверждение 1.

Множество *PREVIOUS* для грамматики G'

Утверждение 2 (упр. 2) (о связи множеств $\overline{FOLLOW}$ и *PREVIOUS*)

Пусть H — LL(1)-грамматика. Тогда для любого $X \in \Gamma \cup \Sigma$ множество $PREVIOUS(X)$ в грамматике H равно множеству $\overline{FOLLOW}(X)$ в грамматике $H^{reverse}$ с учетом замены $\dashv$ на $\vdash$.

Указание: Использовать утверждение 1.

Найдем множества *PREVIOUS* для всех нетерминалов и терминала), которое присутствует в управляющей таблице МП-автомата $\mathcal{M}''$.

Множество *PREVIOUS* для грамматики G'

Для этого найдем грамматику $(G')^{reverse}$:

$$
\begin{aligned}
S &\to S'A \\
S' &\to S'A+ \mid \varepsilon \\
A &\to A'B \\
A' &\to A'B* \mid \varepsilon \\
B &\to)S(\mid x
\end{aligned}
$$

Множество *PREVIOUS* для грамматики G'

Для этого найдем грамматику $(G')^{reverse}$:

$$S \to S'A$$
$$S' \to S'A+ \mid \varepsilon$$
$$A \to A'B$$
$$A' \to A'B* \mid \varepsilon$$
$$B \to)S(\mid x$$

Найдем множество $PREVIOUS(X)$ в грамматике (G'), которое по утверждению 2 для любого $X \in \Gamma \cup \Sigma$ равно множеству $\overline{FOLLOW}(X)$ в грамматике $(G')^{reverse}$:

Множество $PREVIOUS$ для грамматики G'

X	$PREVIOUS(X)$ в G' равно $\overline{FOLLOW}(X)$ в грамматике $(G')^{reverse}$
S	$\vdash, ($
S'	$), x$
A	$\vdash, +, ($
A'	$), x$
B	$\vdash, *, +, ($
$)$	$x,)$

Множество $PREVIOUS$ для грамматики G'

X	$PREVIOUS(X)$ в G' равно $\overline{FOLLOW}(X)$ в грамматике $(G')^{reverse}$
S	$\vdash, ($
S'	$), x$
A	$\vdash, +, ($
A'	$), x$
B	$\vdash, *, +, ($
$)$	$x,)$

Заметим, что из данных нетерминалов выводятся следующие цепочки:

Множество *PREVIOUS* для грамматики G'

X	$PREVIOUS(X)$ в G' равно $\overline{FOLLOW}(X)$ в грамматике $(G')^{reverse}$
S	$\vdash, ($
S'	$), x$
A	$\vdash, +, ($
A'	$), x$
B	$\vdash, *, +, ($
$)$	$x,)$

Заметим, что из данных нетерминалов выводятся следующие цепочки:

$$S \Rightarrow^* A + A + \ldots + A$$

Множество $PREVIOUS$ для грамматики G'

X	$PREVIOUS(X)$ в G' равно $\overline{FOLLOW}(X)$ в грамматике $(G')^{reverse}$
S	$\vdash, ($
S'	$), x$
A	$\vdash, +, ($
A'	$), x$
B	$\vdash, *, +, ($
$)$	$x,)$

Заметим, что из данных нетерминалов выводятся следующие цепочки:

$$S \Rightarrow^* A + A + \ldots + A$$

$$S' \Rightarrow^* +A + \ldots + A \mid \varepsilon$$

Множество $PREVIOUS$ для грамматики G'

X	$PREVIOUS(X)$ в G' равно $\overline{FOLLOW}(X)$ в грамматике $(G')^{reverse}$
S	$\vdash, ($
S'	$), x$
A	$\vdash, +, ($
A'	$), x$
B	$\vdash, *, +, ($
$)$	$x,)$

Заметим, что из данных нетерминалов выводятся следующие цепочки:

$$S \Rightarrow^* A + A + \ldots + A$$
$$S' \Rightarrow^* +A + \ldots + A \mid \varepsilon$$
$$A \Rightarrow^* B * B * \ldots * B$$
$$A' \Rightarrow^* *B * \ldots * B \mid \varepsilon$$
$$B \Rightarrow^* (S) \mid x$$

Построение обработчика ошибок для грамматики G'

- Пусть наверху стека лежит нетерминал S, тогда на ленте при применении соответствующего правила $S \to \alpha$ ожидаются терминалы x, ((см. таблицу переходов МП-автомата $\mathcal{M}''$).

Построение обработчика ошибок для грамматики G'

- Пусть наверху стека лежит нетерминал S, тогда на ленте при применении соответствующего правила $S \to \alpha$ ожидаются терминалы x, ((см. таблицу переходов МП-автомата $\mathcal{M}''$).
  Поскольку, $PREVIOUS(S) = \{\vdash, (\}$, значит, на ленте до вывода из S могут быть только эти терминалы.

Построение обработчика ошибок для грамматики G'

- Пусть наверху стека лежит нетерминал S, тогда на ленте при применении соответствующего правила $S \to \alpha$ ожидаются терминалы x, ((см. таблицу переходов МП-автомата $\mathcal{M}''$).
  Поскольку, $PREVIOUS(S) = \{\vdash, (\}$, значит, на ленте до вывода из S могут быть только эти терминалы.
  Если на ленте указатель показывает на терминал $+$, то значит, мы оказываемся в ситуации, когда на ленте должны быть рядом терминалы $\vdash x$ или ((, а на самом деле мы имеем терминалы $\vdash +$, или $(+$, и, значит, — это ошибка e_1.
- Аналогично, если на ленте указатель показывает на терминал $*$ — это ошибка e_1.

Построение обработчика ошибок для грамматики G'

- Пусть наверху стека лежит нетерминал S, тогда на ленте при применении соответствующего правила $S \to \alpha$ ожидаются терминалы x, ((см. таблицу переходов МП-автомата $\mathcal{M}''$).
  Поскольку, $PREVIOUS(S) = \{\vdash, (\}$, значит, на ленте до вывода из S могут быть только эти терминалы.
  Если на ленте указатель показывает на терминал $+$, то значит, мы оказываемся в ситуации, когда на ленте должны быть рядом терминалы $\vdash x$ или ((, а на самом деле мы имеем терминалы $\vdash +$, или $(+$, и, значит, — это ошибка e_1.
- Аналогично, если на ленте указатель показывает на терминал $*$ — это ошибка e_1.
- Если на ленте указатель на терминале $\dashv$, значит, на ленте имеем пару терминалов $\vdash\dashv$ или $(\dashv$, и, следовательно, мы снова имеем ошибку e_1.

Построение обработчика ошибок для грамматики G'

- Пусть наверху стека лежит нетерминал S, тогда на ленте при применении соответствующего правила $S \to \alpha$ ожидаются терминалы x, ((см. таблицу переходов МП-автомата $\mathcal{M}''$).
  Поскольку, $PREVIOUS(S) = \{\vdash, (\}$, значит, на ленте до вывода из S могут быть только эти терминалы.
  Если на ленте указатель показывает на терминал $+$, то значит, мы оказываемся в ситуации, когда на ленте должны быть рядом терминалы $\vdash x$ или ((, а на самом деле мы имеем терминалы $\vdash +$, или $(+$, и, значит, — это ошибка e_1.
- Аналогично, если на ленте указатель показывает на терминал $*$ — это ошибка e_1.
- Если на ленте указатель на терминале $\dashv$, значит, на ленте имеем пару терминалов $\vdash\dashv$ или $(\dashv$, и, следовательно, мы снова имеем ошибку e_1.
- Если на ленте указатель на терминале), значит, на ленте имеем пару терминалов $\vdash)$ или $()$, и, следовательно, мы имеем ошибку e_4.

Построение обработчика ошибок для грамматики G'

- Пусть наверху стека лежит нетерминал S', тогда ожидаются терминалы $+$, $)$, $\dashv$ на ленте при выводе из S'.

Построение обработчика ошибок для грамматики G'

- Пусть наверху стека лежит нетерминал S', тогда ожидаются терминалы $+$, $)$, $\dashv$ на ленте при выводе из S'.
  Поскольку, $PREVIOUS(S') = \{), x\}$, значит, на ленте до вывода из S' могут быть только эти терминалы.

Построение обработчика ошибок для грамматики G'

- Пусть наверху стека лежит нетерминал S', тогда ожидаются терминалы $+$, $)$, $\dashv$ на ленте при выводе из S'.
  Поскольку, $PREVIOUS(S') = \{), x\}$, значит, на ленте до вывода из S' могут быть только эти терминалы.
  Значит, если указатель показывает на терминалы x, $($, то значит, на ленте $)x$, $)($, xx, или $x($, и значит, — это ошибка e_2.
- Если указатель показывает на терминал $*$, то значит, на ленте $)*$ или $x*$, значит, — это ошибка e_6.

Построение обработчика ошибок для грамматики G'

- Пусть наверху стека лежит нетерминал A, тогда ожидаются терминалы x, (на ленте при выводе из A.

Построение обработчика ошибок для грамматики G'

- Пусть наверху стека лежит нетерминал A, тогда ожидаются терминалы x, (на ленте при выводе из A.
- Поскольку, $PREVIOUS(A) = \{\vdash, +, (\}$, значит, если указатель показывает на терминалы $+, *, \dashv$, то на ленте $\vdash +$, $\vdash *$, $\vdash \dashv$, $++$, $+*$, $+ \dashv$ или $(+$, $(*$, $(\dashv$. И поэтому — это ошибка e_1.

Построение обработчика ошибок для грамматики G'

- Пусть наверху стека лежит нетерминал A, тогда ожидаются терминалы x, (на ленте при выводе из A.
- Поскольку, $PREVIOUS(A) = \{\vdash, +, (\}$, значит, если указатель показывает на терминалы $+, *, \dashv$, то на ленте $\vdash +$, $\vdash *$, $\vdash\dashv$, $++$, $+*$, $+\dashv$ или $(+$, $(*$, $(\dashv$. И поэтому — это ошибка e_1.
- Если указатель показывает на терминал), то значит, на ленте $\vdash)$, $+)$, $()$, значит, это ошибка e_4.

Построение обработчика ошибок для грамматики G'

- Пусть наверху стека лежит нетерминал A', тогда ожидаются терминалы $+$, $*$, $)$, $\dashv$ на ленте при выводе из A'.

Построение обработчика ошибок для грамматики G'

- Пусть наверху стека лежит нетерминал A', тогда ожидаются терминалы $+$, $*$, $)$, $\dashv$ на ленте при выводе из A'.
- $PREVIOUS(A') = \{), x\}$. Значит, если указатель показывает на терминалы $x, (,$ то на ленте $)x$, $)($, xx, или $x($. Здесь ситуация полностью аналогична ситуации для нетерминала S', и это ошибка e_2.

Построение обработчика ошибок для грамматики G'

- Пусть наверху стека лежит нетерминал A', тогда ожидаются терминалы $+$, $*$, $)$, $\dashv$ на ленте при выводе из A'.
- $PREVIOUS(A') = \{), x\}$. Значит, если указатель показывает на терминалы $x, (,$ то на ленте $)x$, $)($, xx, или $x($. Здесь ситуация полностью аналогична ситуации для нетерминала S', и это ошибка e_2.
- Если наверху стека лежит нетерминал B, то ожидаются терминалы x, $($ на ленте при выводе из B.

Построение обработчика ошибок для грамматики G'

- Пусть наверху стека лежит нетерминал A', тогда ожидаются терминалы $+$, $*$, $)$, $\dashv$ на ленте при выводе из A'.
- $PREVIOUS(A') = \{), x\}$. Значит, если указатель показывает на терминалы $x, (,$ то на ленте $)x$, $)($, xx, или $x($. Здесь ситуация полностью аналогична ситуации для нетерминала S', и это ошибка e_2.
- Если наверху стека лежит нетерминал B, то ожидаются терминалы x, $($ на ленте при выводе из B.
- $PREVIOUS(B) = \{\vdash, *, +, (\}$.

Построение обработчика ошибок для грамматики G'

- Пусть наверху стека лежит нетерминал A', тогда ожидаются терминалы $+$, $*$, $)$, $\dashv$ на ленте при выводе из A'.
- $PREVIOUS(A') = \{), x\}$. Значит, если указатель показывает на терминалы $x, (,$ то на ленте $)x$, $)($, xx, или $x($. Здесь ситуация полностью аналогична ситуации для нетерминала S', и это ошибка e_2.
- Если наверху стека лежит нетерминал B, то ожидаются терминалы x, $($ на ленте при выводе из B.
- $PREVIOUS(B) = \{\vdash, *, +, (\}$.
- Значит, если указатель показывает на терминалы $+, *, \dashv$, то на ленте $\vdash +$, $\vdash *$, или $\vdash\dashv$, $*+$, $**$, $* \dashv$, $++$, $+*$, $+ \dashv$, $(+$, $(*$ или $(\dashv$. Это ошибка e_2.

Построение обработчика ошибок для грамматики G'

- Пусть наверху стека лежит нетерминал A', тогда ожидаются терминалы $+$, $*$, $)$, $\dashv$ на ленте при выводе из A'.
- $PREVIOUS(A') = \{), x\}$. Значит, если указатель показывает на терминалы $x, (,$ то на ленте $)x$, $)($, xx, или $x($. Здесь ситуация полностью аналогична ситуации для нетерминала S', и это ошибка e_2.
- Если наверху стека лежит нетерминал B, то ожидаются терминалы x, $($ на ленте при выводе из B.
- $PREVIOUS(B) = \{\vdash, *, +, (\}$.
- Значит, если указатель показывает на терминалы $+, *, \dashv$, то на ленте $\vdash +$, $\vdash *$, или $\vdash\dashv$, $*+$, $**$, $* \dashv$, $++$, $+*$, $+ \dashv$, $(+$, $(*$ или $(\dashv$. Это ошибка e_2.
- А если указатель показывает на терминал $)$, то на ленте $\vdash)$, $*)$, $+)$ или $()$. Это ошибка e_4.

Построение обработчика ошибок для грамматики G'

- Наконец, если наверху стека лежит терминал), то ожидается только терминал).

Построение обработчика ошибок для грамматики G'

- Наконец, если наверху стека лежит терминал), то ожидается только терминал).
- Его нужно обязательно поставить, иначе мы будем иметь незакрытую правую скобку.

Построение обработчика ошибок для грамматики G'

- Наконец, если наверху стека лежит терминал), то ожидается только терминал).
- Его нужно обязательно поставить, иначе мы будем иметь незакрытую правую скобку.
- Таким образом, имеем ошибку e_3 для всех пустых клеток управляющей таблицы в строке для терминала).

Обработка синтаксических ошибок. Пример 5 (продолжение)

Рассмотрим полученный обработчик ошибок.

	x	+	*	(	)	⊣
S	$AS', \_$	e_1	e_1	$AS', \_$	e_4	e_1
S'	e_2	$AS', \rightarrow$	e_6	e_2	$\varepsilon, \_$	$\varepsilon, \_$
A	$BA', \_$	e_1	e_1	$BA', \_$	e_4	e_1
A'	e_2	$\varepsilon, \_$	$BA', \rightarrow$	e_2	$\varepsilon, \_$	$\varepsilon, \_$
B	$\varepsilon, \rightarrow$	e_1	e_1	$S), \rightarrow$	e_4	e_1
)	e_3	e_3	e_3	e_3	$\varepsilon, \rightarrow$	e_3
∇						$\vee$

Пример 5. Протокол разбора ошибочной цепочки

Теперь рассмотрим протокол разбора ошибочной цепочки)xx+⊣ в грамматике G':

	стек	позиция указателя	ошибка
1	$S\nabla$	◇)xx+⊣	e_4
2	$S\nabla$	◇xx+⊣	
3	$AS'\nabla$	◇xx+⊣	
4	$BA'S'\nabla$	◇xx+⊣	
5	$A'S'\nabla$	x◇x+⊣	e_2
6	$A'S'\nabla$	x◇+x+⊣	
7	$S'\nabla$	x◇+x+⊣	
8	$AS'\nabla$	x+◇x+⊣	
9	$BA'S'\nabla$	x+◇x+⊣	
10	$A'S'\nabla$	x+x◇+⊣	
11	$S'\nabla$	x+x◇+⊣	

Пример 5. Протокол разбора ошибочной цепочки (продолжение)

	стек	позиция указателя	ошибка
12	$AS'\nabla$	x+x+$\diamond$ $\dashv$	e_1
13	$AS'\nabla$	x+x+$\diamond$x$\dashv$	
14	$BA'S'\nabla$	x+x+$\diamond$x$\dashv$	
15	$A'S'\nabla$	x+x+x$\diamond$ $\dashv$	
16	$S'\nabla$	x+x+x$\diamond$ $\dashv$	
17	∇	x+x+x$\diamond$ $\dashv$ $\vee$	

Пример 5. Протокол разбора ошибочной цепочки (продолжение)

	стек	позиция указателя	ошибка
12	$AS'\nabla$	x+x+$\diamond$ $\dashv$	e_1
13	$AS'\nabla$	x+x+$\diamond$x$\dashv$	
14	$BA'S'\nabla$	x+x+$\diamond$x$\dashv$	
15	$A'S'\nabla$	x+x+x$\diamond$ $\dashv$	
16	$S'\nabla$	x+x+x$\diamond$ $\dashv$	
17	∇	x+x+x$\diamond$ $\dashv$ $\vee$	

Заметим, что нашей задачей было не восстановление цепочки, а нахождение позиции ошибки и характеризация ее!

Определение множества $FIRST_k(\alpha)$

Класс LL(1)-грамматик можно обобщить до класса LL(k)-грамматик.

Определение множества $FIRST_k(\alpha)$

Класс LL(1)-грамматик можно обобщить до класса LL(k)-грамматик.

Определение

Пусть G – КС-грамматика, $\alpha \in (\Sigma \cup \Gamma)^*$, $k \in \mathbb{N}$. Тогда $FIRST_k(\alpha)$ — это подмножество множества Σ^*, состоящее из всех цепочек u таких, что

1. либо $|u| = k$ и $\alpha \Rightarrow u\beta$ для некоторого $\beta \in (\Sigma \cup \Gamma)^*$,
2. либо $|u| < k$ и $\alpha \Rightarrow u$.

Определение множества $FIRST_k(\alpha)$

Класс LL(1)-грамматик можно обобщить до класса LL(k)-грамматик.

Определение

Пусть G – КС-грамматика, $\alpha \in (\Sigma \cup \Gamma)^*$, $k \in \mathbb{N}$. Тогда $FIRST_k(\alpha)$ — это подмножество множества Σ^*, состоящее из всех цепочек u таких, что

1. либо $|u| = k$ и $\alpha \Rightarrow u\beta$ для некоторого $\beta \in (\Sigma \cup \Gamma)^*$,
2. либо $|u| < k$ и $\alpha \Rightarrow u$.

- Из определения следует, что $FIRST_1(\alpha) = FIRST(\alpha)$ для любого $\alpha \in (\Sigma \cup \Gamma)^*$, т.е. действительно $FIRST_k(\alpha)$ является обобщением множества $FIRST(\alpha)$.

Определение множества $FIRST_k(\alpha)$

Класс LL(1)-грамматик можно обобщить до класса LL(k)-грамматик.

Определение

Пусть G – КС-грамматика, $\alpha \in (\Sigma \cup \Gamma)^*$, $k \in \mathbb{N}$. Тогда $FIRST_k(\alpha)$ — это подмножество множества Σ^*, состоящее из всех цепочек u таких, что

1. либо $|u| = k$ и $\alpha \Rightarrow u\beta$ для некоторого $\beta \in (\Sigma \cup \Gamma)^*$,
2. либо $|u| < k$ и $\alpha \Rightarrow u$.

- Из определения следует, что $FIRST_1(\alpha) = FIRST(\alpha)$ для любого $\alpha \in (\Sigma \cup \Gamma)^*$, т.е. действительно $FIRST_k(\alpha)$ является обобщением множества $FIRST(\alpha)$.
- Очевидно, если $\alpha \in \Sigma^*$, то $FIRST_k(\alpha)$ — это префикс длины k слова α при $|\alpha| \geq k$, и $\alpha \dashv$ при $|\alpha| < k$.

Определение множества $FIRST_k(\alpha)$

Класс LL(1)-грамматик можно обобщить до класса LL(k)-грамматик.

Определение

Пусть G – КС-грамматика, $\alpha \in (\Sigma \cup \Gamma)^*$, $k \in \mathbb{N}$. Тогда $FIRST_k(\alpha)$ — это подмножество множества Σ^*, состоящее из всех цепочек u таких, что

1. либо $|u| = k$ и $\alpha \Rightarrow u\beta$ для некоторого $\beta \in (\Sigma \cup \Gamma)^*$,
2. либо $|u| < k$ и $\alpha \Rightarrow u$.

- Из определения следует, что $FIRST_1(\alpha) = FIRST(\alpha)$ для любого $\alpha \in (\Sigma \cup \Gamma)^*$, т.е. действительно $FIRST_k(\alpha)$ является обобщением множества $FIRST(\alpha)$.
- Очевидно, если $\alpha \in \Sigma^*$, то $FIRST_k(\alpha)$ — это префикс длины k слова α при $|\alpha| \geq k$, и $\alpha \dashv$ при $|\alpha| < k$.
- Пример 1. $FIRST_3(abaa) = \{aba\}$, $FIRST_3(ab) = \{ab \dashv\}$.

Определение LL(k)-грамматики

Определение

КС-грамматика G называется LL(k)-грамматикой, если из того, что

1. $S \Rightarrow^* wA\alpha \Rightarrow w\beta\alpha \Rightarrow^* wu$
2. $S \Rightarrow^* wA\alpha \Rightarrow w\gamma\alpha \Rightarrow^* wv$
3. $FIRST_k(u) = FIRST_k(v)$,

где $w, u, v \in \Sigma^*$, $\alpha, \beta, \gamma \in (\Sigma \cup \Gamma)^*$, следует $\beta = \gamma$.

Определение LL(k)-грамматики

Определение

КС-грамматика G называется LL(k)-грамматикой, если из того, что

1. $S \Rightarrow^* wA\alpha \Rightarrow w\beta\alpha \Rightarrow^* wu$
2. $S \Rightarrow^* wA\alpha \Rightarrow w\gamma\alpha \Rightarrow^* wv$
3. $FIRST_k(u) = FIRST_k(v)$,

где $w, u, v \in \Sigma^*$, $\alpha, \beta, \gamma \in (\Sigma \cup \Gamma)^*$, следует $\beta = \gamma$.

Вспомним свойство LL(1)-грамматик.

Определение LL(k)-грамматики

Определение

КС-грамматика G называется LL(k)-грамматикой, если из того, что

1. $S \Rightarrow^* wA\alpha \Rightarrow w\beta\alpha \Rightarrow^* wu$
2. $S \Rightarrow^* wA\alpha \Rightarrow w\gamma\alpha \Rightarrow^* wv$
3. $FIRST_k(u) = FIRST_k(v)$,

где $w, u, v \in \Sigma^*$, $\alpha, \beta, \gamma \in (\Sigma \cup \Gamma)^*$, следует $\beta = \gamma$.

Вспомним свойство LL(1)-грамматик.

Свойство LL(1)-грамматики

Пусть КС-грамматика G является LL(1)-грамматикой, Тогда из того, что

Определение LL(k)-грамматики

Определение

КС-грамматика G называется LL(k)-грамматикой, если из того, что

1. $S \Rightarrow^* wA\alpha \Rightarrow w\beta\alpha \Rightarrow^* wu$
2. $S \Rightarrow^* wA\alpha \Rightarrow w\gamma\alpha \Rightarrow^* wv$
3. $FIRST_k(u) = FIRST_k(v)$,

где $w, u, v \in \Sigma^*$, $\alpha, \beta, \gamma \in (\Sigma \cup \Gamma)^*$, следует $\beta = \gamma$.

Вспомним свойство LL(1)-грамматик.

Свойство LL(1)-грамматики

Пусть КС-грамматика G является LL(1)-грамматикой, Тогда из того, что

1. $S \Rightarrow^* wA\alpha \Rightarrow w\beta\alpha \Rightarrow^* wu$
2. $S \Rightarrow^* wA\alpha \Rightarrow w\gamma\alpha \Rightarrow^* wv$
3. $FIRST(u) = FIRST(v)$.

где $w, u, v \in \Sigma^*$, $\alpha, \beta, \gamma \in (\Sigma \cup \Gamma)^*$, следует $\beta = \gamma$.

Свойства LL(k)-грамматик

Отсюда следует, что LL(k)-грамматика действительно является обобщением LL(1)-грамматики.

Свойства LL(k)-грамматик

Отсюда следует, что LL(k)-грамматика действительно является обобщением LL(1)-грамматики.

Лемма 1 об LL(k)-грамматики

$LL(1) \subseteq LL(2) \subseteq \ldots \subseteq LL(k) \subseteq \ldots$

Свойства LL(k)-грамматик

Отсюда следует, что LL(k)-грамматика действительно является обобщением LL(1)-грамматики.

Лемма 1 об LL(k)-грамматики

$LL(1) \subseteq LL(2) \subseteq \ldots \subseteq LL(k) \subseteq \ldots$

Доказательство следует из определения LL(k-1) и LL(k) грамматик и того простого факта, что из равенства $FIRST_k(u) = FIRST_k(v)$ для $u, v \in (\Sigma \cup \Gamma)^*$ следует равенство $FIRST_{k-1}(u) = FIRST_{k-1}(v)$.

Свойства LL(k)-грамматик (продолжение))

Пример 2. Очевидно, грамматика $G_2 : S \to a^k b \mid a^k c$ принадлежит классу LL(k+1), но не принадлежит классу LL(k). Таким образом, справедлива

Свойства LL(k)-грамматик (продолжение))

Пример 2. Очевидно, грамматика $G_2 : S \to a^k b \mid a^k c$ принадлежит классу LL(k+1), но не принадлежит классу LL(k). Таким образом, справедлива

Лемма 2 об LL(k)-грамматике

$LL(1) \subset LL(2) \subset \ldots \subset LL(k) \subset \ldots$

LL-грамматики

Определение LL-грамматики

Грамматика называется LL-грамматикой, если она является LL(k)-грамматикой для некоторого k (см. рис.1).

LL-грамматики

Определение LL-грамматики

Грамматика называется LL-грамматикой, если она является LL(k)-грамматикой для некоторого k (см. рис.1).

Пример 3. Для грамматики $G_3 : S \to A \mid B, A \to aAb \mid 0, B \to aBbb \mid 1$ выполняется

LL-грамматики

Определение LL-грамматики

Грамматика называется LL-грамматикой, если она является LL(k)-грамматикой для некоторого k (см. рис.1).

Пример 3. Для грамматики $G_3 : S \to A \mid B, A \to aAb \mid 0, B \to aBbb \mid 1$ выполняется

1. $S \Rightarrow^* S \Rightarrow A \Rightarrow^* u = a^k 0 b^k$
2. $S \Rightarrow^* S \Rightarrow B \Rightarrow^* v = a^k 1 b^{2k}$
3. $FIRST_k(u) = FIRST_k(v) = \{a^k\}$.

И при этом не $\beta \neq \gamma$.

Определение LL-грамматики

Грамматика называется LL-грамматикой, если она является LL(k)-грамматикой для некоторого k (см. рис.1).

Пример 3. Для грамматики $G_3 : S \to A \mid B, A \to aAb \mid 0, B \to aBbb \mid 1$ выполняется

1. $S \Rightarrow^* S \Rightarrow A \Rightarrow^* u = a^k 0 b^k$
2. $S \Rightarrow^* S \Rightarrow B \Rightarrow^* v = a^k 1 b^{2k}$
3. $FIRST_k(u) = FIRST_k(v) = \{a^k\}$.

И при этом не $\beta \neq \gamma$.
Таким образом, грамматика G_3 не является LL(k) грамматикой ни для какого k, поскольку не удовлетворяет определению LL(k) грамматики для $w = \alpha = \varepsilon$, $A = S$, $\beta = A$, $\gamma = B$.

LL-грамматики

Определение LL-грамматики

Грамматика называется LL-грамматикой, если она является LL(k)-грамматикой для некоторого k (см. рис.1).

Пример 3. Для грамматики $G_3 : S \to A \mid B, A \to aAb \mid 0, B \to aBbb \mid 1$ выполняется

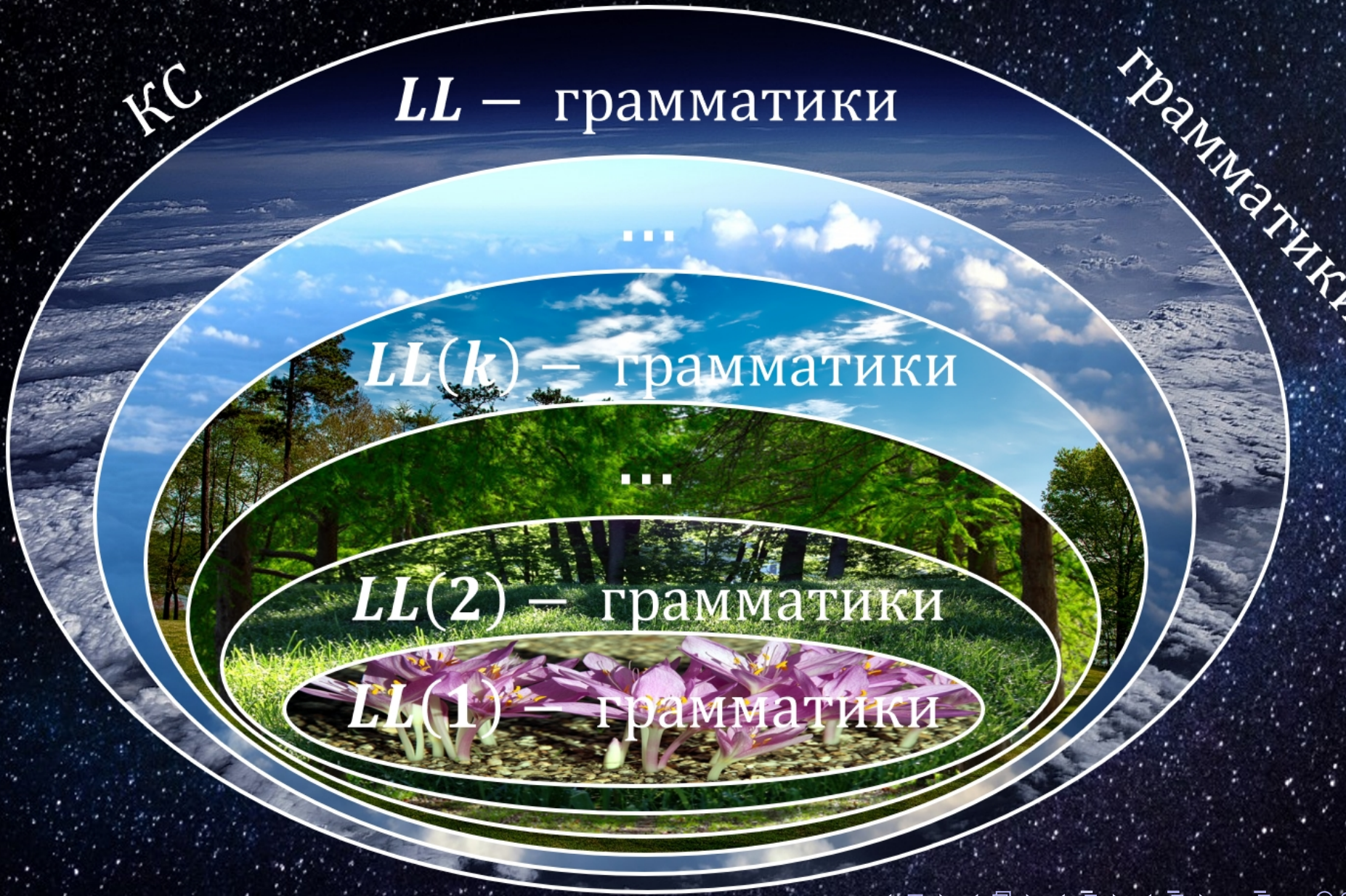
1. $S \Rightarrow^* S \Rightarrow A \Rightarrow^* u = a^k 0 b^k$
2. $S \Rightarrow^* S \Rightarrow B \Rightarrow^* v = a^k 1 b^{2k}$
3. $FIRST_k(u) = FIRST_k(v) = \{a^k\}$.

И при этом не $\beta \neq \gamma$.

Таким образом, грамматика G_3 не является LL(k) грамматикой ни для какого k, поскольку не удовлетворяет определению LL(k) грамматики для $w = \alpha = \varepsilon$, $A = S$, $\beta = A$, $\gamma = B$.

Способы построения анализаторов для LL(k)-грамматик известны, но при k>1 управляющие таблицы становятся громоздкими и редко используются на практике.

LL-грамматики. Иллюстрация



Метод рекурсивного спуска

- Рассмотрим общий метод нисходящего анализа, позволяющий, в принципе, использовать любые нелеворекурсивные грамматики.

Метод рекурсивного спуска

- Рассмотрим общий метод нисходящего анализа, позволяющий, в принципе, использовать любые нелеворекурсивные грамматики.
- Основная идея метода рекурсивного спуска состоит в следующем. Для каждого нетерминала A пишется своя процедура A, которая напрямую работает с входной цепочкой, распознавая в ней фрагмент, выводимый из A.

Метод рекурсивного спуска

- Рассмотрим общий метод нисходящего анализа, позволяющий, в принципе, использовать любые нелеворекурсивные грамматики.
- Основная идея метода рекурсивного спуска состоит в следующем. Для каждого нетерминала A пишется своя процедура A, которая напрямую работает с входной цепочкой, распознавая в ней фрагмент, выводимый из A.
- Эта процедура осуществляет выбор правила вывода и обработку этого правила, при которой, в частности, рекурсивно вызываются процедуры для нетерминалов в правой части правила.

Метод рекурсивного спуска

- Рассмотрим общий метод нисходящего анализа, позволяющий, в принципе, использовать любые нелеворекурсивные грамматики.
- Основная идея метода рекурсивного спуска состоит в следующем. Для каждого нетерминала A пишется своя процедура A, которая напрямую работает с входной цепочкой, распознавая в ней фрагмент, выводимый из A.
- Эта процедура осуществляет выбор правила вывода и обработку этого правила, при которой, в частности, рекурсивно вызываются процедуры для нетерминалов в правой части правила.
- Если обработка завершена успешно (получен вывод фрагмента входной цепочки), то процедура завершается, передавая управление вызвавшей ее процедуре.

Метод рекурсивного спуска

- Для LL(1) грамматики в случае успешного приятия цепочки w рекурсивному спуску соответствует левосторонний вывод $S \Rightarrow^* w$ этой цепочки, т.е. обход в глубину дерева вывода T_w (см. утверждение о связи вывода и дерева вывода). В частности, строится дерево вывода.

Метод рекурсивного спуска

- Для LL(1) грамматики в случае успешного приятия цепочки w рекурсивному спуску соответствует левосторонний вывод $S \Rightarrow^* w$ этой цепочки, т.е. обход в глубину дерева вывода T_w (см. утверждение о связи вывода и дерева вывода). В частности, строится дерево вывода.
- При этом если запускается очередная рекурсивная процедура $A()$, то указатель на ленте показывает на один из элементов множества $FIRST(A)$ в случае, когда нетерминал A — не аннулирующий, и на один из элементов множества $FOLLOW(A)$ в противном случае.

Метод рекурсивного спуска

- Для LL(1) грамматики в случае успешного приятия цепочки w рекурсивному спуску соответствует левосторонний вывод $S \Rightarrow^* w$ этой цепочки, т.е. обход в глубину дерева вывода T_w (см. утверждение о связи вывода и дерева вывода). В частности, строится дерево вывода.
- При этом если запускается очередная рекурсивная процедура $A()$, то указатель на ленте показывает на один из элементов множества $FIRST(A)$ в случае, когда нетерминал A — не аннулирующий, и на один из элементов множества $FOLLOW(A)$ в противном случае.
- Если не будет такого совпадения, то цепочка w не будет принята.

Метод рекурсивного спуска

- Для LL(1) грамматики в случае успешного приятия цепочки w рекурсивному спуску соответствует левосторонний вывод $S \Rightarrow^* w$ этой цепочки, т.е. обход в глубину дерева вывода T_w (см. утверждение о связи вывода и дерева вывода). В частности, строится дерево вывода.
- При этом если запускается очередная рекурсивная процедура $A()$, то указатель на ленте показывает на один из элементов множества $FIRST(A)$ в случае, когда нетерминал A — не аннулирующий, и на один из элементов множества $FOLLOW(A)$ в противном случае.
- Если не будет такого совпадения, то цепочка w не будет принята.
- Приведем алгоритм рекурсивного спуска для LL(1) грамматики. Для простоты будем считать, что она не упрощена, как в лемме 3 лекции 14 (см., например, МП-автомат $\mathcal{M}$ на слайде 7).

Алгоритм рекурсивного спуска для LL(1) грамматики

Вход: КС-грамматика $G = (\Sigma, \Gamma, P, S)$, цепочка $w \in \Sigma$, $w = w_1 w_2 \dots w_n$
Выход: Сообщение о том, что $w \in L(G)$ и дерево вывода T цепочки $w \in \Sigma$, если $w \in L(G)$ и сообщение об ошибке, если $w \notin L(G)$

Алгоритм рекурсивного спуска для LL(1) грамматики

Вход: КС-грамматика $G = (\Sigma, \Gamma, P, S)$, цепочка $w \in \Sigma$, $w = w_1 w_2 \dots w_n$
Выход: Сообщение о том, что $w \in L(G)$ и дерево вывода T цепочки $w \in \Sigma$, если $w \in L(G)$ и сообщение об ошибке, если $w \notin L(G)$

1. def $S()$ # рекурсивная функция для нетериминала S

Алгоритм рекурсивного спуска для LL(1) грамматики

Вход: КС-грамматика $G = (\Sigma, \Gamma, P, S)$, цепочка $w \in \Sigma$, $w = w_1 w_2 \dots w_n$
Выход: Сообщение о том, что $w \in L(G)$ и дерево вывода T цепочки $w \in \Sigma$, если $w \in L(G)$ и сообщение об ошибке, если $w \notin L(G)$

1. def $S()$ # рекурсивная функция для нетериминала S
   def $A()$ # рекурсивная функция для нетериминала A
   def $B()$ # рекурсивная функция для нетериминала B ...

Алгоритм рекурсивного спуска для LL(1) грамматики

Вход: КС-грамматика $G = (\Sigma, \Gamma, P, S)$, цепочка $w \in \Sigma$, $w = w_1 w_2 \dots w_n$
Выход: Сообщение о том, что $w \in L(G)$ и дерево вывода T цепочки $w \in \Sigma$, если $w \in L(G)$ и сообщение об ошибке, если $w \notin L(G)$

1. def $S()$ # рекурсивная функция для нетериминала S
   def $A()$ # рекурсивная функция для нетериминала A
   def $B()$ # рекурсивная функция для нетериминала B ...
   # тело алгоритма
2. $lookahead = w_1$ # вначале lookahead=первый символ слова w, где lookahead — текущий символ, который мы видим на ленте

Алгоритм рекурсивного спуска для LL(1) грамматики

Вход: КС-грамматика $G = (\Sigma, \Gamma, P, S)$, цепочка $w \in \Sigma$, $w = w_1 w_2 \dots w_n$
Выход: Сообщение о том, что $w \in L(G)$ и дерево вывода T цепочки $w \in \Sigma$, если $w \in L(G)$ и сообщение об ошибке, если $w \notin L(G)$

1. def $S()$ # рекурсивная функция для нетериминала S
   def $A()$ # рекурсивная функция для нетериминала A
   def $B()$ # рекурсивная функция для нетериминала B ...
   # тело алгоритма
2. $lookahead = w_1$ # вначале lookahead=первый символ слова w, где lookahead — текущий символ, который мы видим на ленте
3. $T = \emptyset$ # вначале дерево вывода пустое

Алгоритм рекурсивного спуска для LL(1) грамматики

Вход: КС-грамматика $G = (\Sigma, \Gamma, P, S)$, цепочка $w \in \Sigma$, $w = w_1 w_2 \dots w_n$
Выход: Сообщение о том, что $w \in L(G)$ и дерево вывода T цепочки $w \in \Sigma$, если $w \in L(G)$ и сообщение об ошибке, если $w \notin L(G)$

1. def $S()$ # рекурсивная функция для нетериминала S
   def $A()$ # рекурсивная функция для нетериминала A
   def $B()$ # рекурсивная функция для нетериминала B ...
   # тело алгоритма
2. $lookahead = w_1$ # вначале lookahead=первый символ слова w, где lookahead — текущий символ, который мы видим на ленте
3. $T = \emptyset$ # вначале дерево вывода пустое
4. $S()$ # запускается рекурсивная процедура для аксиомы

Алгоритм рекурсивного спуска для LL(1) грамматики

Вход: КС-грамматика $G = (\Sigma, \Gamma, P, S)$, цепочка $w \in \Sigma$, $w = w_1 w_2 \dots w_n$
Выход: Сообщение о том, что $w \in L(G)$ и дерево вывода T цепочки $w \in \Sigma$, если $w \in L(G)$ и сообщение об ошибке, если $w \notin L(G)$

1. def $S()$ # рекурсивная функция для нетериминала S
   def $A()$ # рекурсивная функция для нетериминала A
   def $B()$ # рекурсивная функция для нетериминала B ...
   # тело алгоритма
2. $lookahead = w_1$ # вначале lookahead=первый символ слова w, где lookahead — текущий символ, который мы видим на ленте
3. $T = \emptyset$ # вначале дерево вывода пустое
4. $S()$ # запускается рекурсивная процедура для аксиомы
5. print('$w \in L(G)$') # не произошло преждевременного выхода и программы, цепочка успешно принялась

Алгоритм рекурсивного спуска для LL(1) грамматики

Вход: КС-грамматика $G = (\Sigma, \Gamma, P, S)$, цепочка $w \in \Sigma$, $w = w_1 w_2 \dots w_n$
Выход: Сообщение о том, что $w \in L(G)$ и дерево вывода T цепочки $w \in \Sigma$, если $w \in L(G)$ и сообщение об ошибке, если $w \notin L(G)$

1. def $S()$ # рекурсивная функция для нетериминала S
   def $A()$ # рекурсивная функция для нетериминала A
   def $B()$ # рекурсивная функция для нетериминала B ...
   # тело алгоритма
2. $lookahead = w_1$ # вначале lookahead=первый символ слова w, где lookahead — текущий символ, который мы видим на ленте
3. $T = \emptyset$ # вначале дерево вывода пустое
4. $S()$ # запускается рекурсивная процедура для аксиомы
5. print('$w \in L(G)$') # не произошло преждевременного выхода и программы, цепочка успешно принялась
6. print('T') # печать дерева T вывода цепочки w

Рекурсивная процедура $A()$ для нетерминала A для LL(1) грамматики

Обозначим для любых $A \in \Gamma$, $a \in \Sigma$
$\delta(A, a) = (\delta_1(A, a), \delta_2(A, a))$, где $\delta_1(A, a) \in (\Sigma \cup \Gamma)^*$, $\delta_2(A, a) \in \{\_, \rightarrow\}$.

Рекурсивная процедура $A()$ для нетерминала A для LL(1) грамматики

Обозначим для любых $A \in \Gamma$, $a \in \Sigma$
$\delta(A, a) = (\delta_1(A, a), \delta_2(A, a))$, где $\delta_1(A, a) \in (\Sigma \cup \Gamma)^*$, $\delta_2(A, a) \in \{\_, \rightarrow\}$.

(1) $str = \delta_1(A, lookahead)$ # строка $\alpha = \delta_1(A, lookahead)$ — правая часть правила $A \rightarrow \alpha$, которое выбирается по множеству SELECT$(A \rightarrow \alpha)$ и lookahead однозначно или пустая клетка (строка)

Рекурсивная процедура $A()$ для нетерминала A для LL(1) грамматики

Обозначим для любых $A \in \Gamma$, $a \in \Sigma$
$\delta(A, a) = (\delta_1(A, a), \delta_2(A, a))$, где $\delta_1(A, a) \in (\Sigma \cup \Gamma)^*$, $\delta_2(A, a) \in \{\_, \rightarrow\}$.

(1) $str = \delta_1(A, lookahead)$ # строка $\alpha = \delta_1(A, lookahead)$ — правая часть правила $A \rightarrow \alpha$, которое выбирается по множеству SELECT$(A \rightarrow \alpha)$ и lookahead однозначно или пустая клетка (строка)

(2) if $str \neq$ пустая строка: # $str = \delta_1(A, lookahead)$, где $\alpha = \delta_1(A, lookahead)$ — правая часть правила $A \rightarrow \alpha$, т.е. клетка в таблице переходов МП автомата $\mathcal{M}$ непустая

Рекурсивная процедура $A()$ для нетерминала A для LL(1) грамматики (продолжение)

for i in range(lenght(str)): # двигаемся по строке str, пока не дойдем до конца или не получим ошибку

Рекурсивная процедура $A()$ для нетерминала A для LL(1) грамматики (продолжение)

for i in range(lenght(str)): # двигаемся по строке str, пока не дойдем до конца или не получим ошибку

(2.1) $X = str[i]$ # $X =$ — очередной символ строки α

Рекурсивная процедура $A()$ для нетерминала A для LL(1) грамматики (продолжение)

for i in range(lenght(str)): # двигаемся по строке str, пока не дойдем до конца или не получим ошибку

(2.1) $X = str[i]$ # $X =$ — очередной символ строки α

(2.2) $T = T \cup \{(A, X)\}$ # наращиваем дерево вывода еще на одно ребро

Рекурсивная процедура $A()$ для нетерминала A для LL(1) грамматики (продолжение)

for i in range(lenght(str)): # двигаемся по строке str, пока не дойдем до конца или не получим ошибку

(2.1) $X = str[i]$ # $X =$ — очередной символ строки α

(2.2) $T = T \cup \{(A, X)\}$ # наращиваем дерево вывода еще на одно ребро

(2.3) case $X \in \Gamma$: # если X — нетерминал

$X()$ # запускаем рекурсивную процедуру $X()$

Рекурсивная процедура $A()$ для нетерминала A для LL(1) грамматики (продолжение)

for i in range(lenght(str)): # двигаемся по строке str, пока не дойдем до конца или не получим ошибку

(2.1) $X = str[i]$ # $X =$ — очередной символ строки α

(2.2) $T = T \cup \{(A, X)\}$ # наращиваем дерево вывода еще на одно ребро

(2.3) case $X \in \Gamma$: # если X — нетерминал

$X()$ # запускаем рекурсивную процедуру $X()$

(2.4) case $X \in \Sigma$: # если X — терминал

Рекурсивная процедура $A()$ для нетерминала A для LL(1) грамматики (продолжение)

for i in range(lenght(str)): # двигаемся по строке str, пока не дойдем до конца или не получим ошибку

(2.1) $X = str[i]$ # $X =$ — очередной символ строки α

(2.2) $T = T \cup \{(A, X)\}$ # наращиваем дерево вывода еще на одно ребро

(2.3) case $X \in \Gamma$: # если X — нетерминал

$X()$ # запускаем рекурсивную процедуру $X()$

(2.4) case $X \in \Sigma$: # если X — терминал

(2.4.1) if $X ==$ *lookahead* :

lookahead=next symbol

Рекурсивная процедура $A()$ для нетерминала A для LL(1) грамматики (продолжение)

for i in range(lenght(str)): # двигаемся по строке str, пока не дойдем до конца или не получим ошибку

(2.1) $X = str[i]$ # $X =$ — очередной символ строки α

(2.2) $T = T \cup \{(A, X)\}$ # наращиваем дерево вывода еще на одно ребро

(2.3) case $X \in \Gamma$: # если X — нетерминал

$X()$ # запускаем рекурсивную процедуру $X()$

(2.4) case $X \in \Sigma$: # если X — терминал

(2.4.1) if $X ==$ *lookahead* :

lookahead=next symbol

(2.4.2) else: # $X! =$ *lookahead*

ОШИБКА

print('$w \notin L(G)$')

exit из алгоритма

Рекурсивная процедура $A()$ для нетерминала A для LL(1) грамматики (продолжение)

for i in range(lenght(str)): # двигаемся по строке str, пока не дойдем до конца или не получим ошибку

(2.1) $X = str[i]$ # $X =$ — очередной символ строки α

(2.2) $T = T \cup \{(A, X)\}$ # наращиваем дерево вывода еще на одно ребро

(2.3) case $X \in \Gamma$: # если X — нетерминал

$X()$ # запускаем рекурсивную процедуру $X()$

(2.4) case $X \in \Sigma$: # если X — терминал

(2.4.1) if $X ==$ *lookahead* :

lookahead=next symbol

(2.4.2) else: # $X! =$ *lookahead*

ОШИБКА

print('$w \notin L(G)$')

exit из алгоритма

(2.5) case $X = \varepsilon$: # если X — пустое слово

$T = T \cup \{(A, X)\}$ # наращиваем дерево вывода еще на одно ребро

Рекурсивная процедура $A()$ для нетерминала A для LL(1) грамматики

(3) else: # если $\delta(A, lookahead)$ — пустая клетка

Рекурсивная процедура $A()$ для нетерминала A для LL(1) грамматики

(3) else: # если $\delta(A, lookahead)$ — пустая клетка
    # ОШИБКА
    print('$w \notin L(G)$')
    exit из программы

Протокол работы алгоритма рекурсивного спуска для цепочки $w = x + x * x$

- Рассмотрим протокол работы алгоритма рекурсивного спуска для цепочки $w = x + x * x$ в виде строящегося дерева T вывода $w = x + x * x$.

Протокол работы алгоритма рекурсивного спуска для цепочки $w = x + x * x$

- Рассмотрим протокол работы алгоритма рекурсивного спуска для цепочки $w = x + x * x$ в виде строящегося дерева T вывода $w = x + x * x$.
- str_S # экранирование переменной str в запуске рекурсивной функции $S()$

Протокол работы алгоритма рекурсивного спуска для цепочки $w = x + x * x$

- Рассмотрим протокол работы алгоритма рекурсивного спуска для цепочки $w = x + x * x$ в виде строящегося дерева T вывода $w = x + x * x$.
- str_S # экранирование переменной *str* в запуске рекурсивной функции $S()$
- str_{A1} # экранирование переменной *str* в первом запуске рекурсивной функции $A()$

Протокол работы алгоритма рекурсивного спуска для цепочки $w = x + x * x$

- Рассмотрим протокол работы алгоритма рекурсивного спуска для цепочки $w = x + x * x$ в виде строящегося дерева T вывода $w = x + x * x$.
- str_S # экранирование переменной *str* в запуске рекурсивной функции $S()$
- str_{A1} # экранирование переменной *str* в первом запуске рекурсивной функции $A()$
- str_{A2} # экранирование переменной *str* во втором запуске рекурсивной функции $A()$

Протокол работы алгоритма рекурсивного спуска для цепочки $w = x + x * x$

- Рассмотрим протокол работы алгоритма рекурсивного спуска для цепочки $w = x + x * x$ в виде строящегося дерева T вывода $w = x + x * x$.
- str_S # экранирование переменной *str* в запуске рекурсивной функции $S()$
- str_{A1} # экранирование переменной *str* в первом запуске рекурсивной функции $A()$
- str_{A2} # экранирование переменной *str* во втором запуске рекурсивной функции $A()$
- И т.д.
- X # очередной символ строки α — правой части текущего правила

Протокол работы алгоритма рекурсивного спуска для цепочки $w = x + x * x$

- Рассмотрим протокол работы алгоритма рекурсивного спуска для цепочки $w = x + x * x$ в виде строящегося дерева T вывода $w = x + x * x$.
- str_S # экранирование переменной str в запуске рекурсивной функции $S()$
- str_{A1} # экранирование переменной str в первом запуске рекурсивной функции $A()$
- str_{A2} # экранирование переменной str во втором запуске рекурсивной функции $A()$
- И т.д.
- X # очередной символ строки α — правой части текущего правила
- При этом если запускается очередная рекурсивная процедура $A()$, то указатель на ленте показывает на один из элементов множества $FIRST(A)$ в случае, когда нетерминал A — не аннулирующий, и на один из элементов множества $FOLLOW(A)$ в противном случае.

Протокол работы алгоритма рекурсивного спуска для цепочки $w = x + x * x$. Иллюстрация

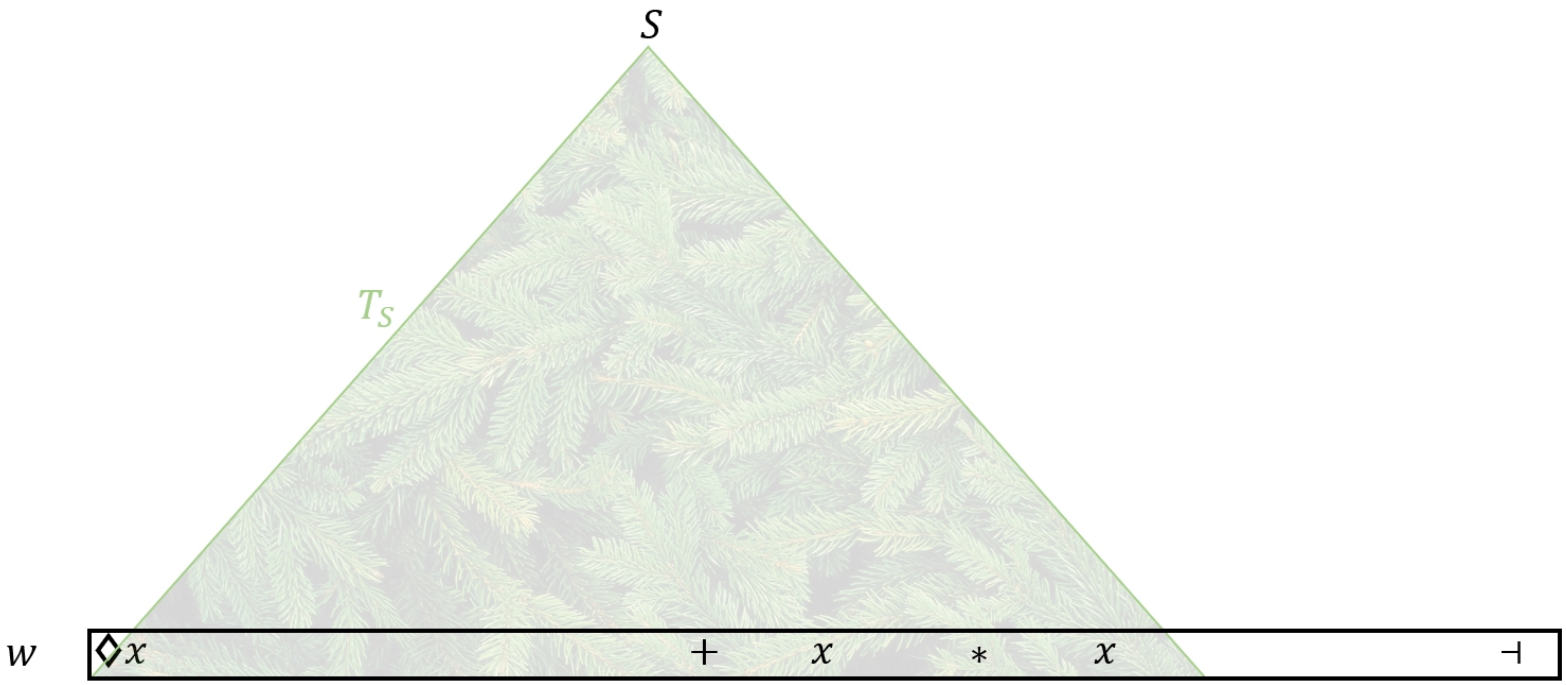


Рис. 1

Протокол работы алгоритма рекурсивного спуска для цепочки $w = x + x * x$. Иллюстрация

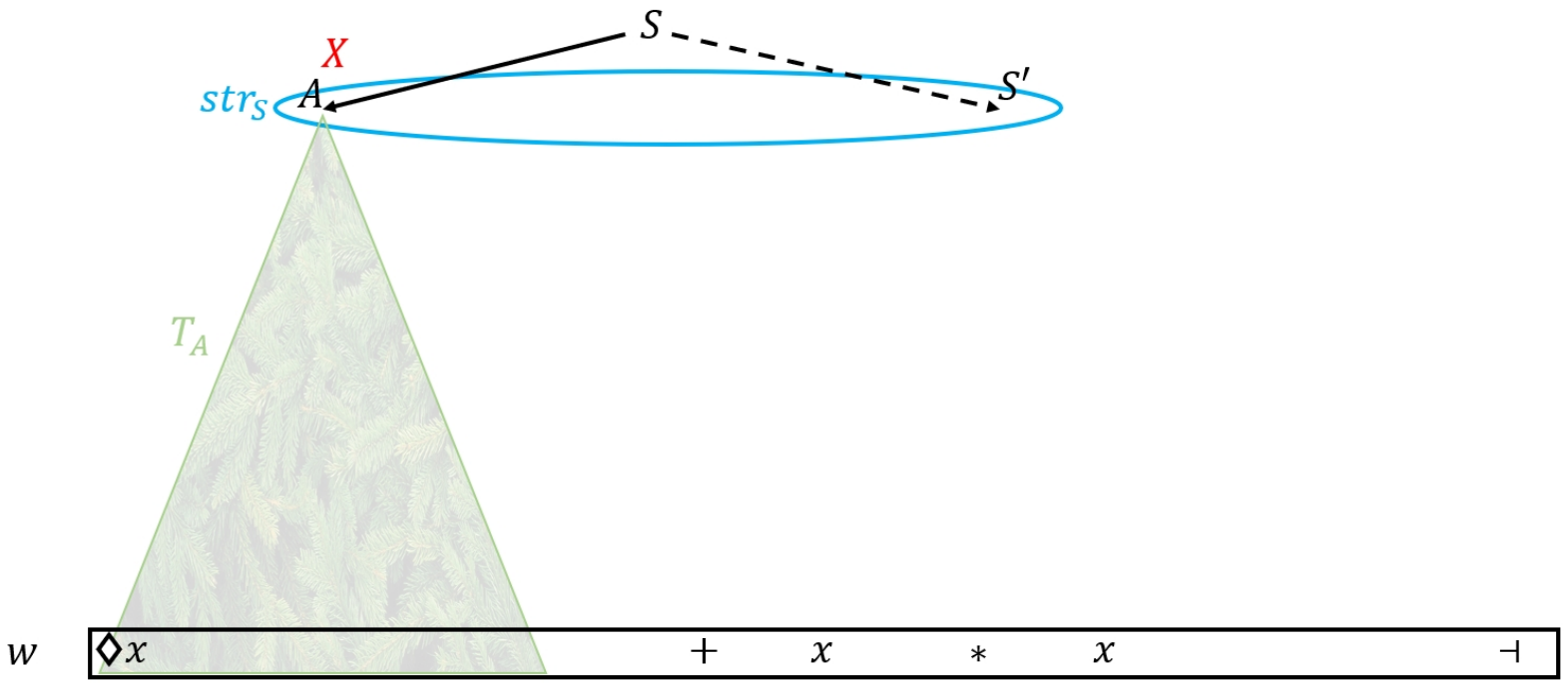


Рис. 2

Протокол работы алгоритма рекурсивного спуска для цепочки $w = x + x * x$. Иллюстрация

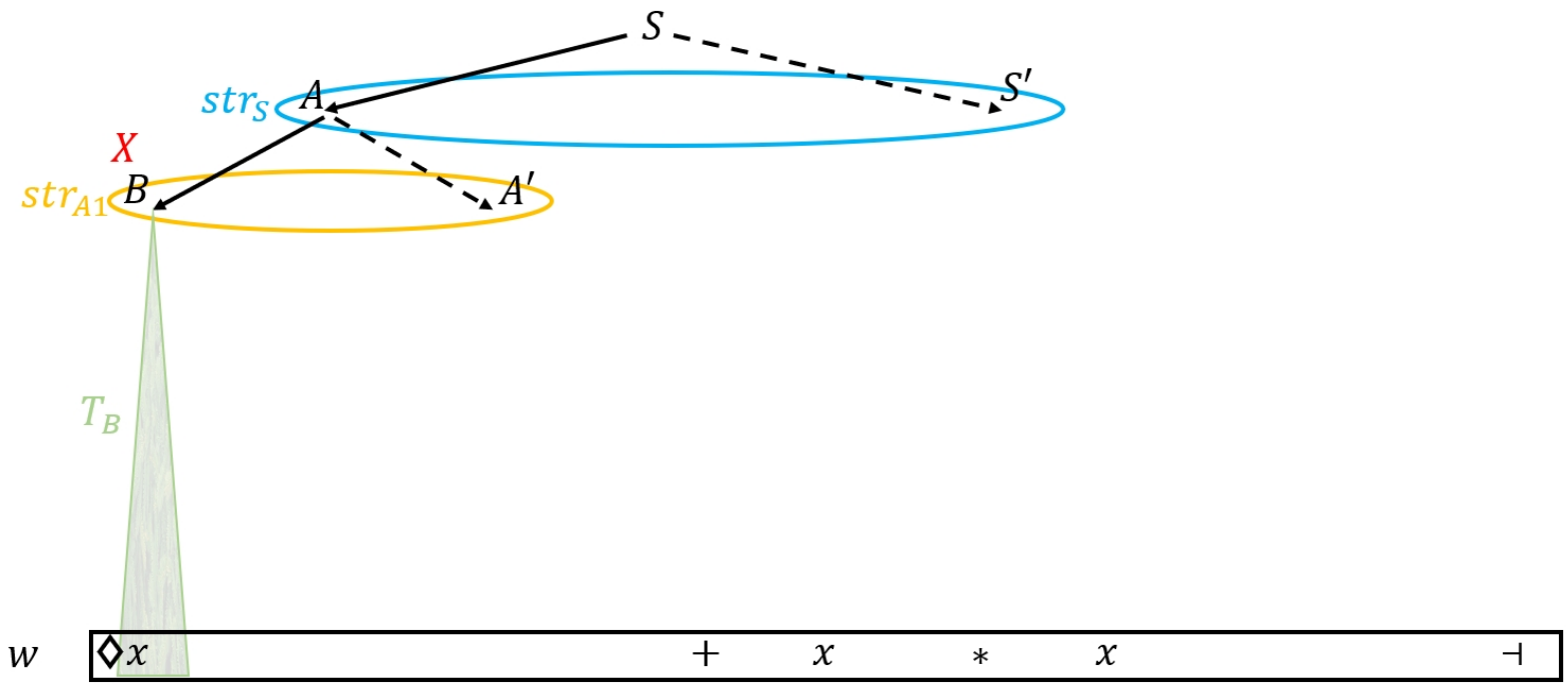
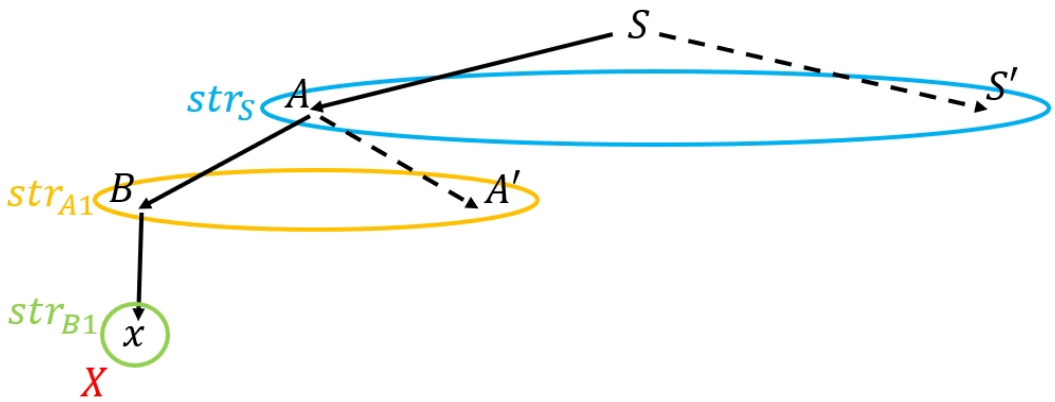


Рис. 3

Протокол работы алгоритма рекурсивного спуска для цепочки $w = x + x * x$. Иллюстрация



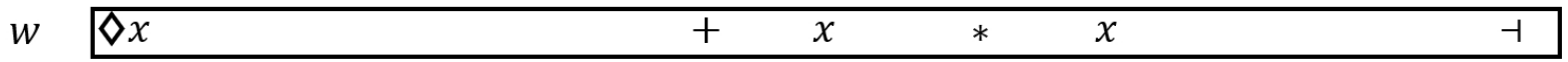


Рис. 4

Протокол работы алгоритма рекурсивного спуска для цепочки $w = x + x * x$. Иллюстрация

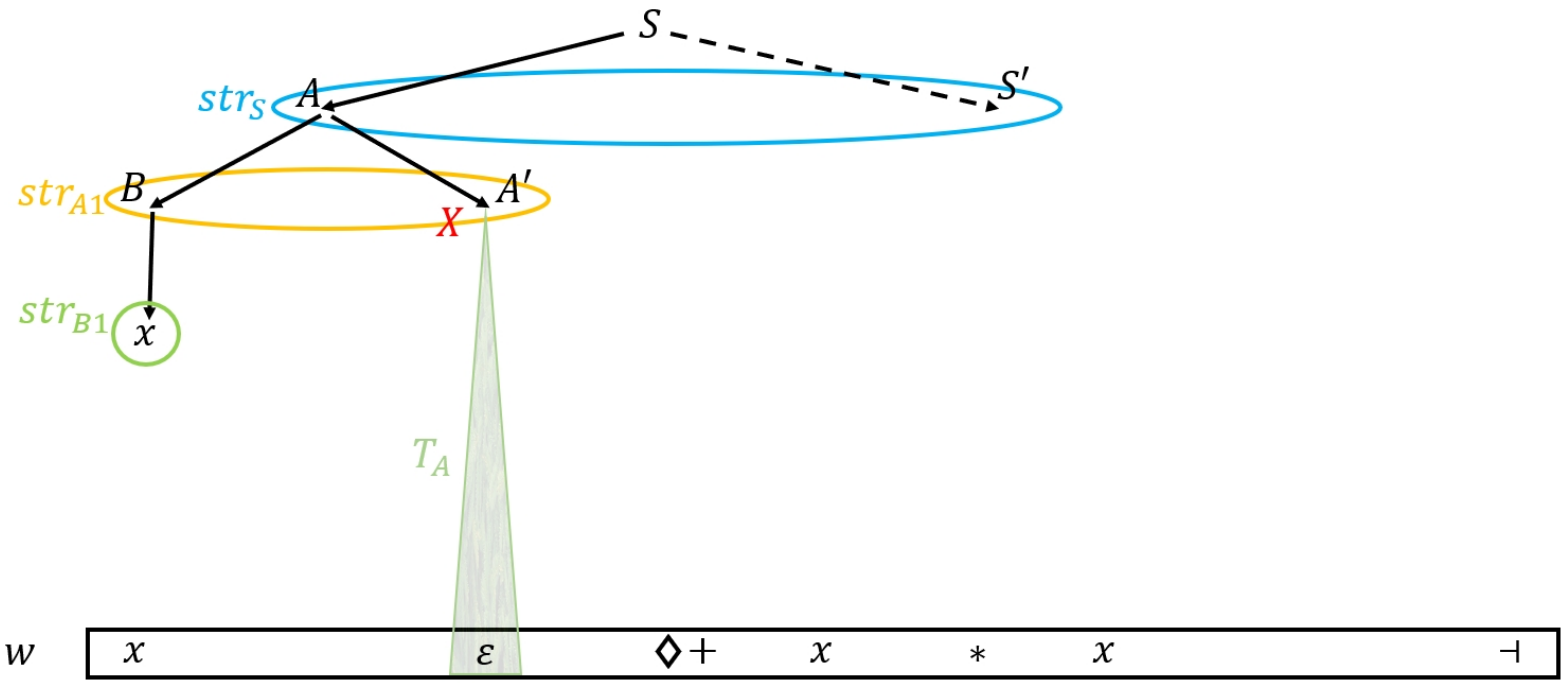
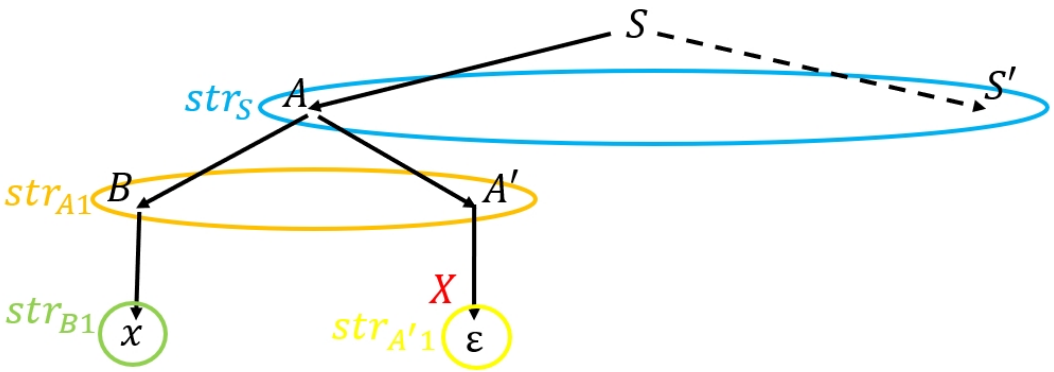


Рис. 5

Протокол работы алгоритма рекурсивного спуска для цепочки $w = x + x * x$. Иллюстрация



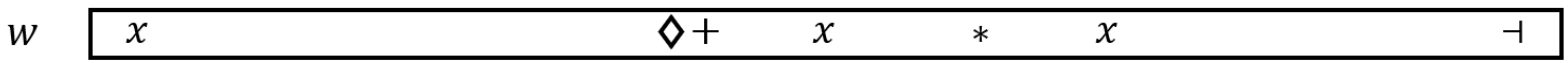


Рис. 6

Протокол работы алгоритма рекурсивного спуска для цепочки $w = x + x * x$. Иллюстрация

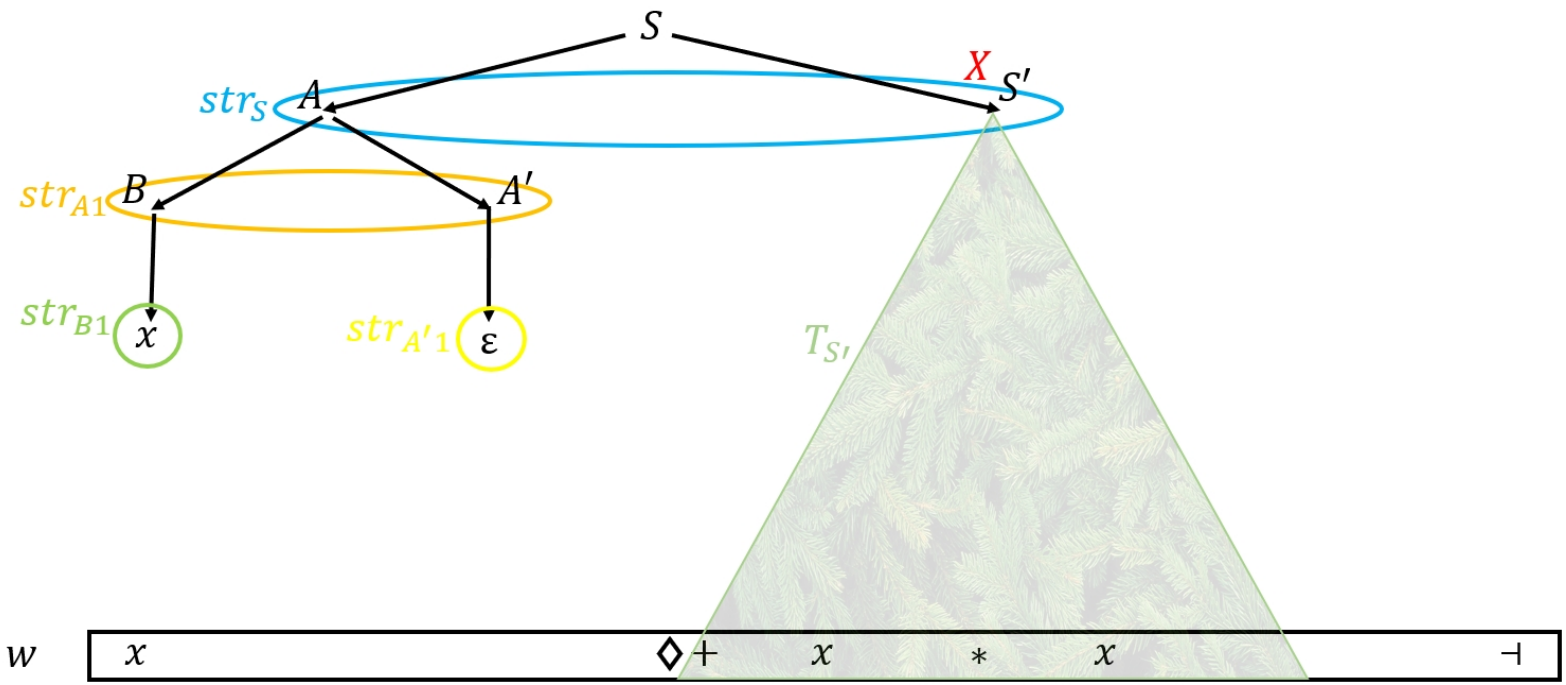


Рис. 7

Протокол работы алгоритма рекурсивного спуска для цепочки $w = x + x * x$. Иллюстрация

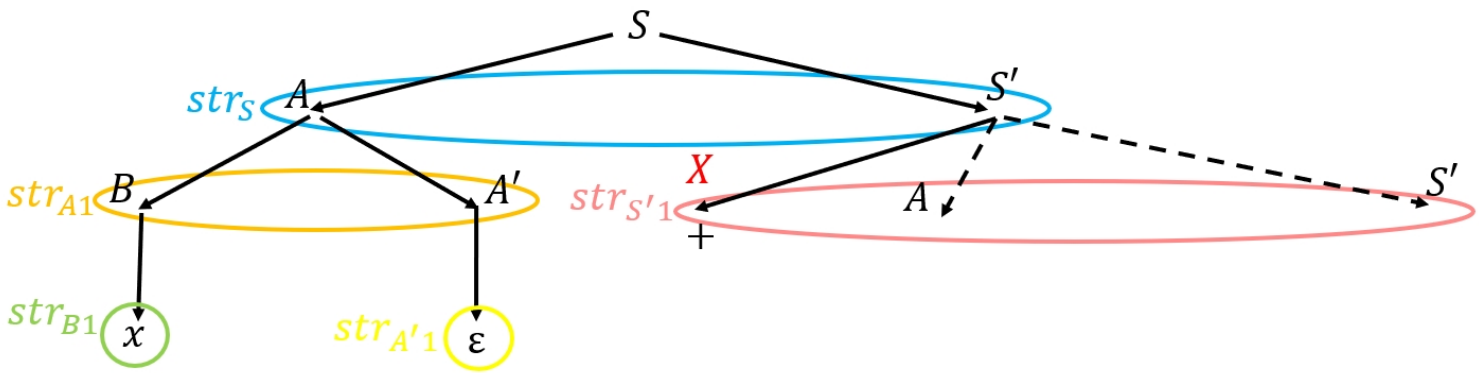


Рис. 8

Протокол работы алгоритма рекурсивного спуска для цепочки $w = x + x * x$. Иллюстрация

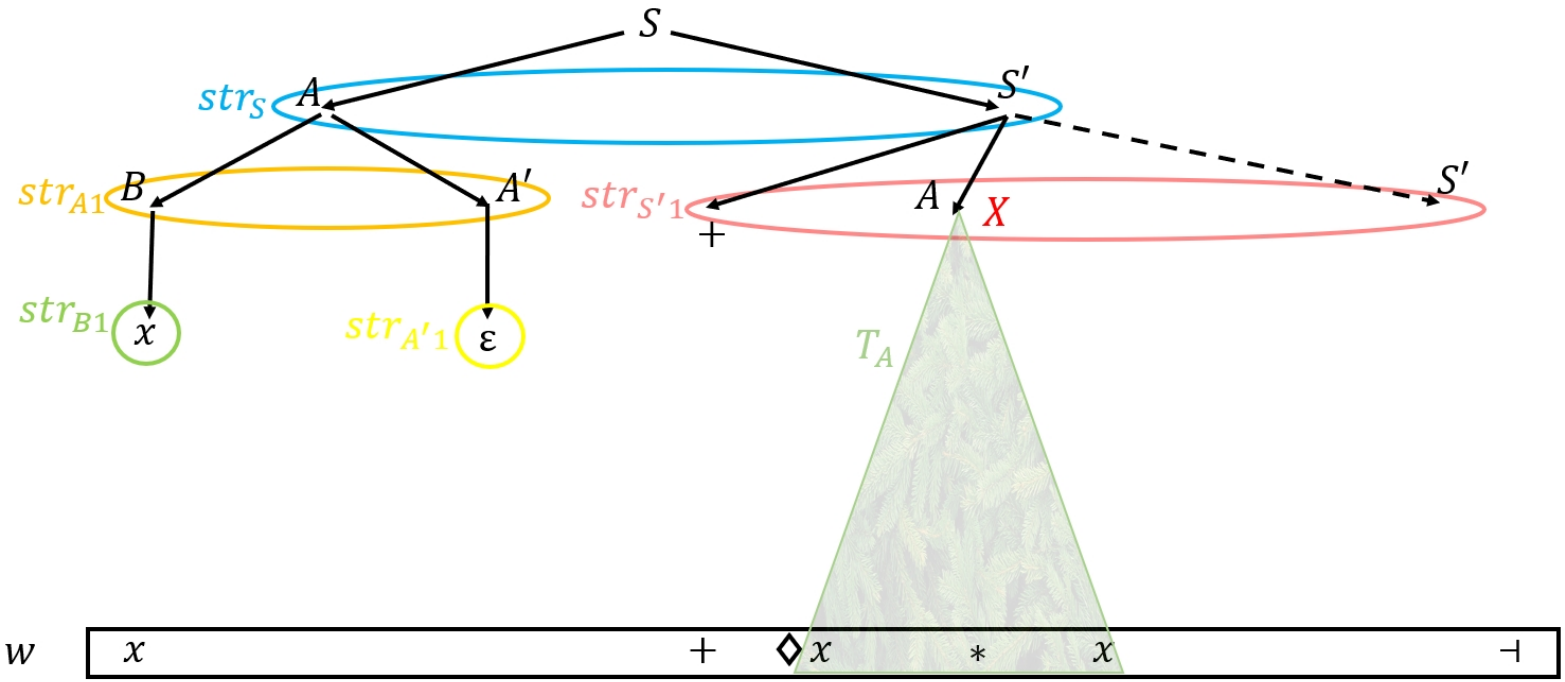


Рис. 9

Протокол работы алгоритма рекурсивного спуска для цепочки $w = x + x * x$. Иллюстрация

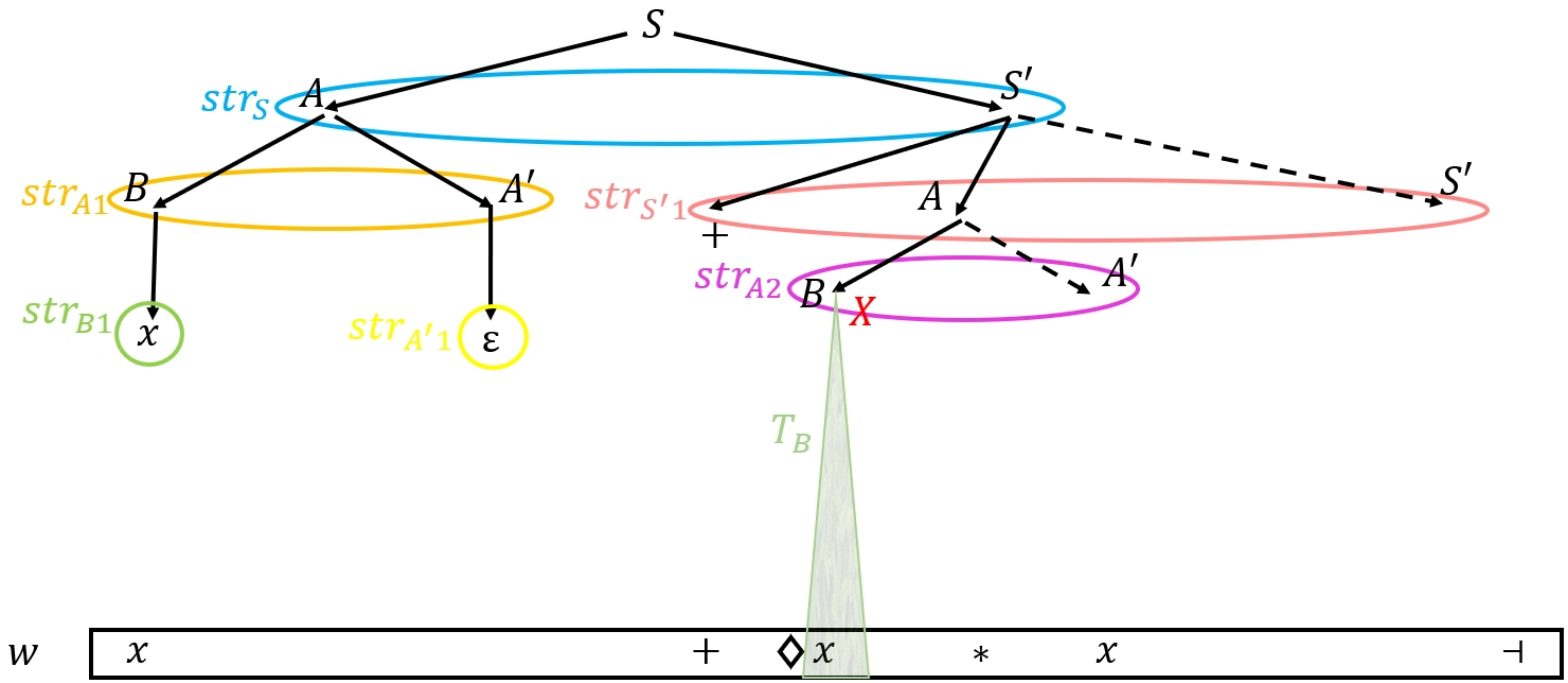


Рис. 10

Протокол работы алгоритма рекурсивного спуска для цепочки $w = x + x * x$. Иллюстрация

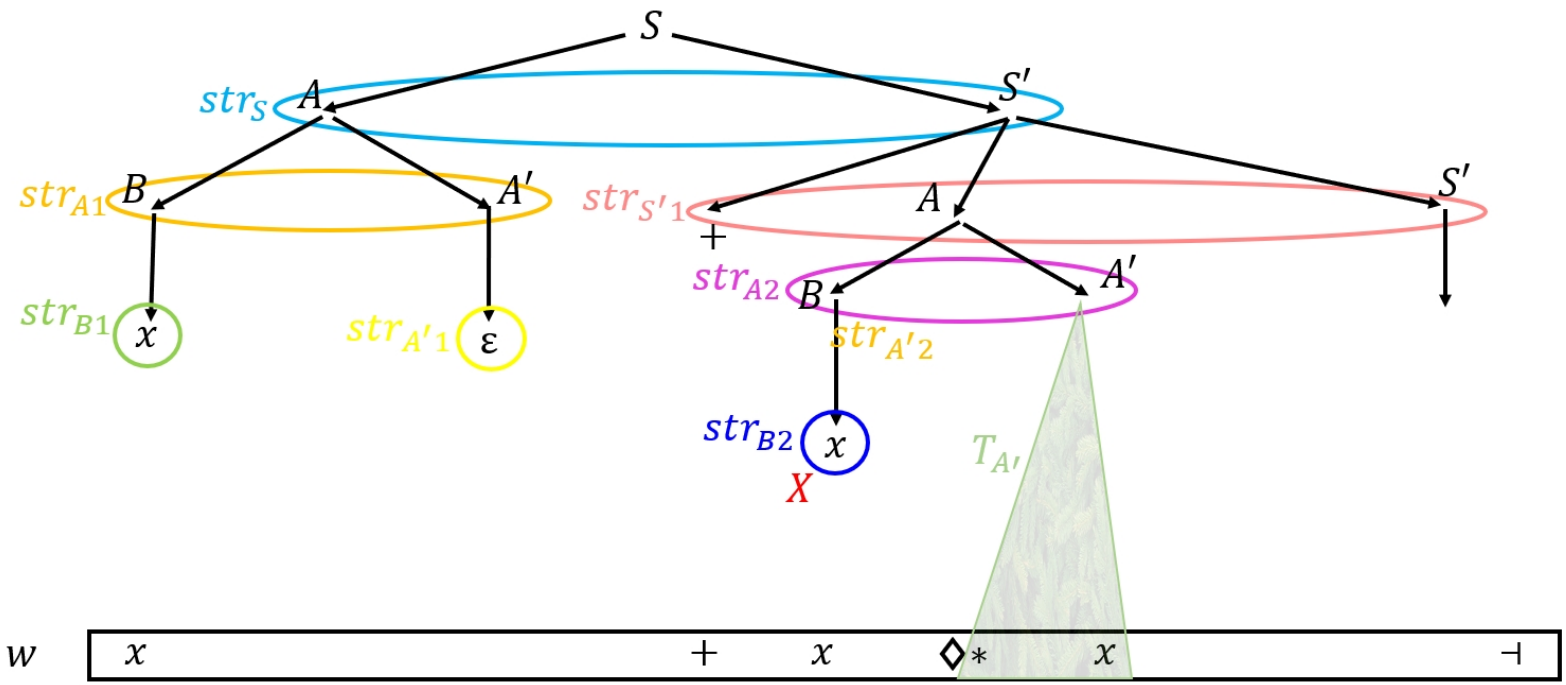
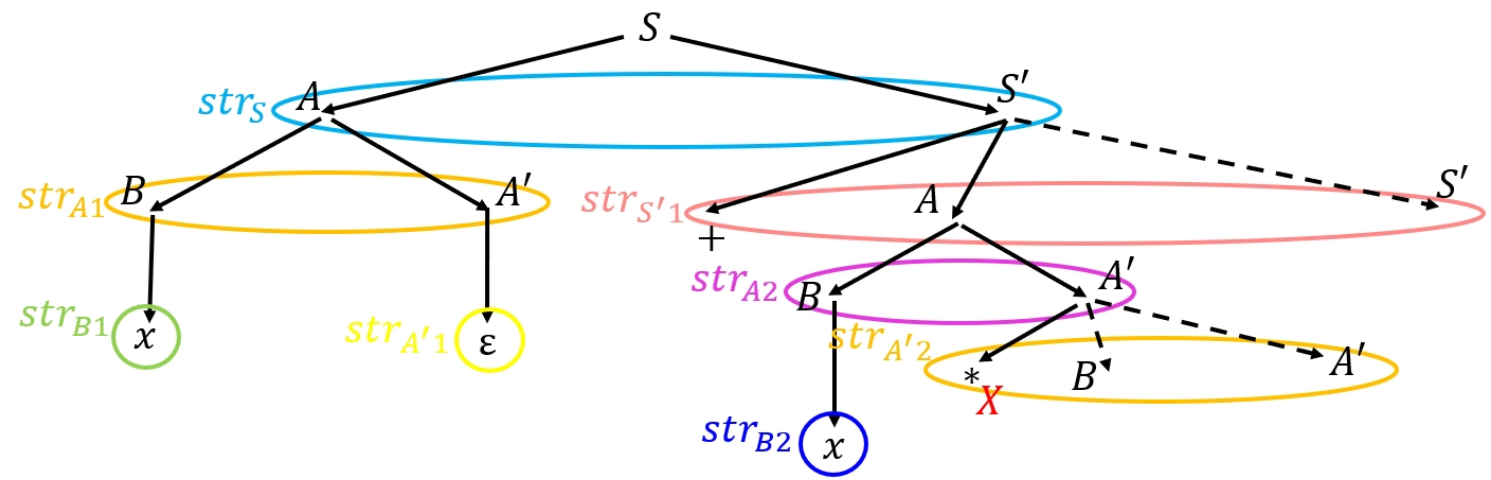


Рис. 11

Протокол работы алгоритма рекурсивного спуска для цепочки $w = x + x * x$. Иллюстрация



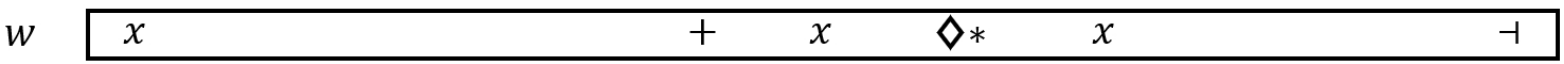


Рис. 12

Протокол работы алгоритма рекурсивного спуска для цепочки $w = x + x * x$. Иллюстрация

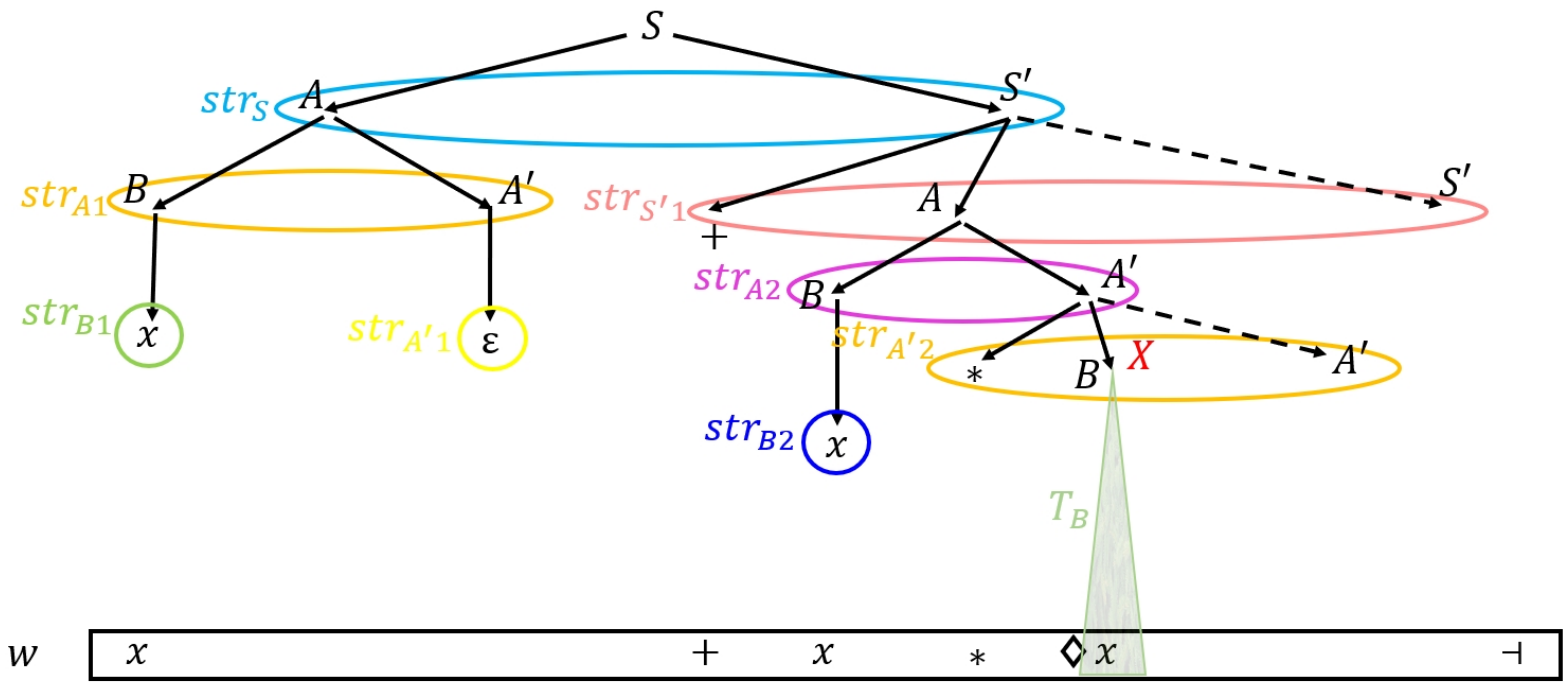


Рис. 13

Протокол работы алгоритма рекурсивного спуска для цепочки $w = x + x * x$. Иллюстрация

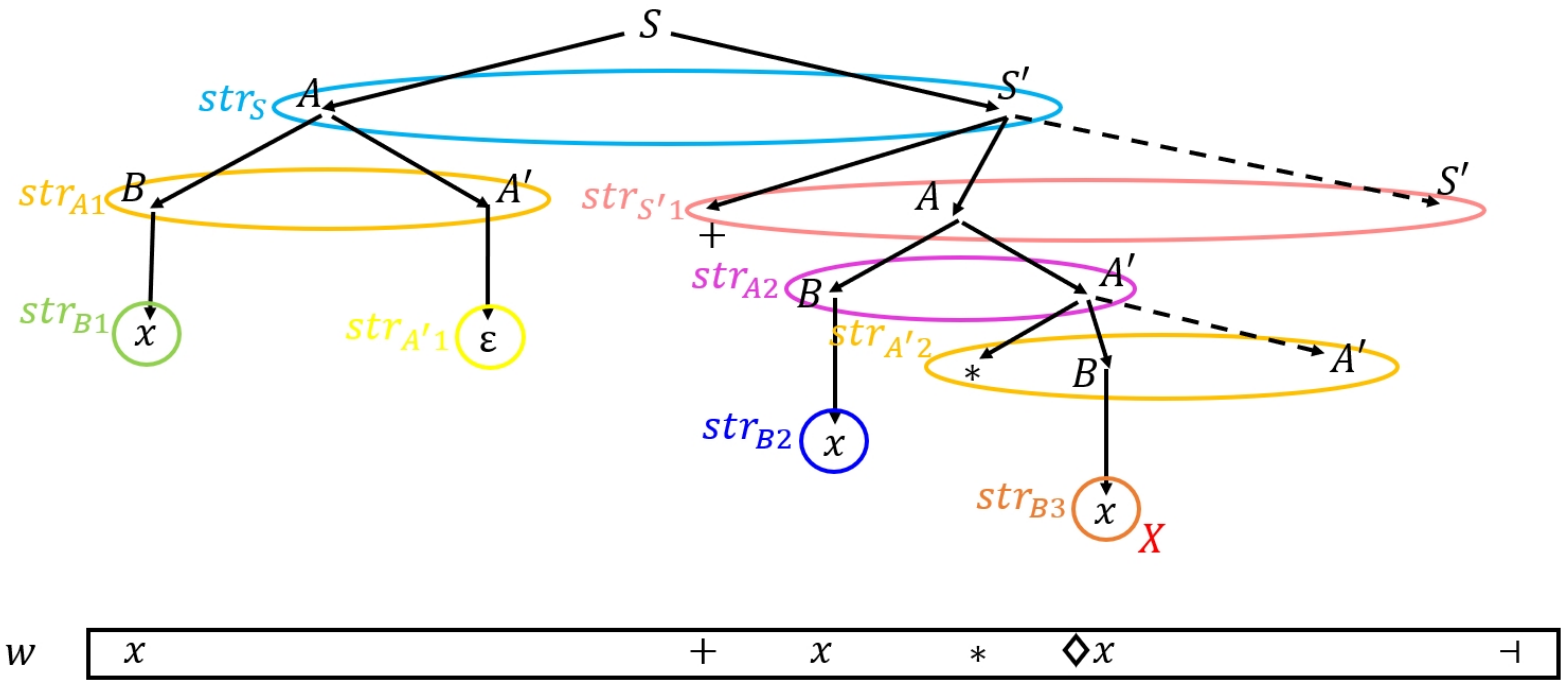


Рис. 14

Протокол работы алгоритма рекурсивного спуска для цепочки $w = x + x * x$. Иллюстрация

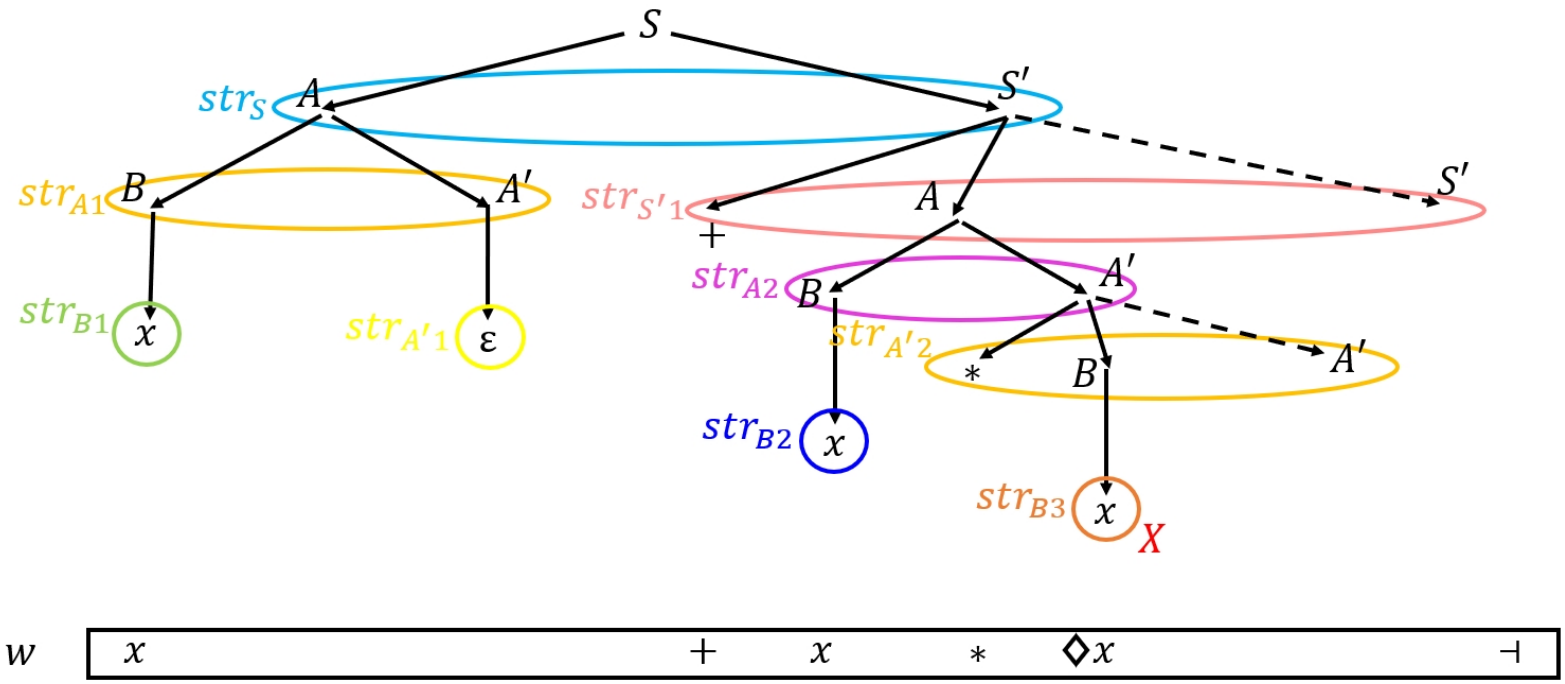


Рис. 15

Протокол работы алгоритма рекурсивного спуска для цепочки $w = x + x * x$. Иллюстрация

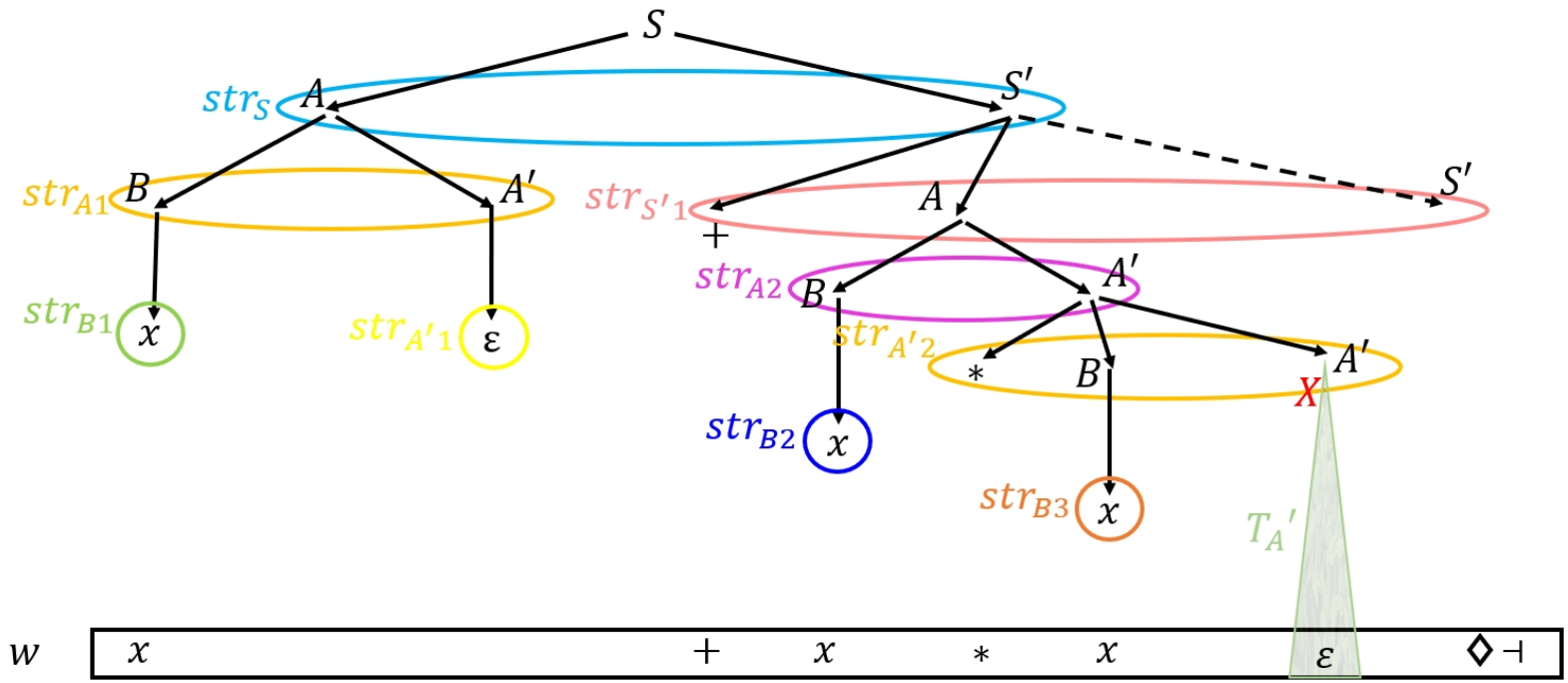


Рис. 16

Протокол работы алгоритма рекурсивного спуска для цепочки $w = x + x * x$. Иллюстрация

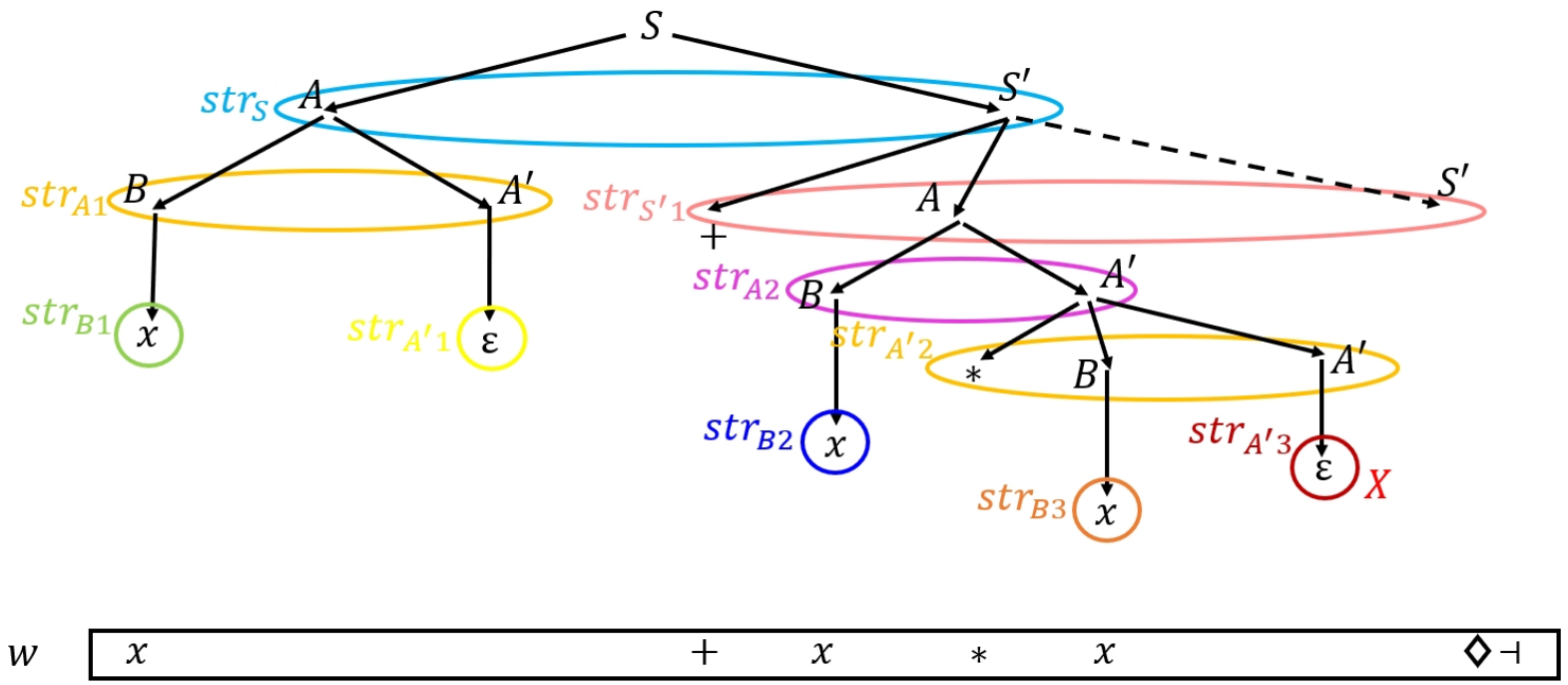


Рис. 17

Протокол работы алгоритма рекурсивного спуска для цепочки $w = x + x * x$. Иллюстрация

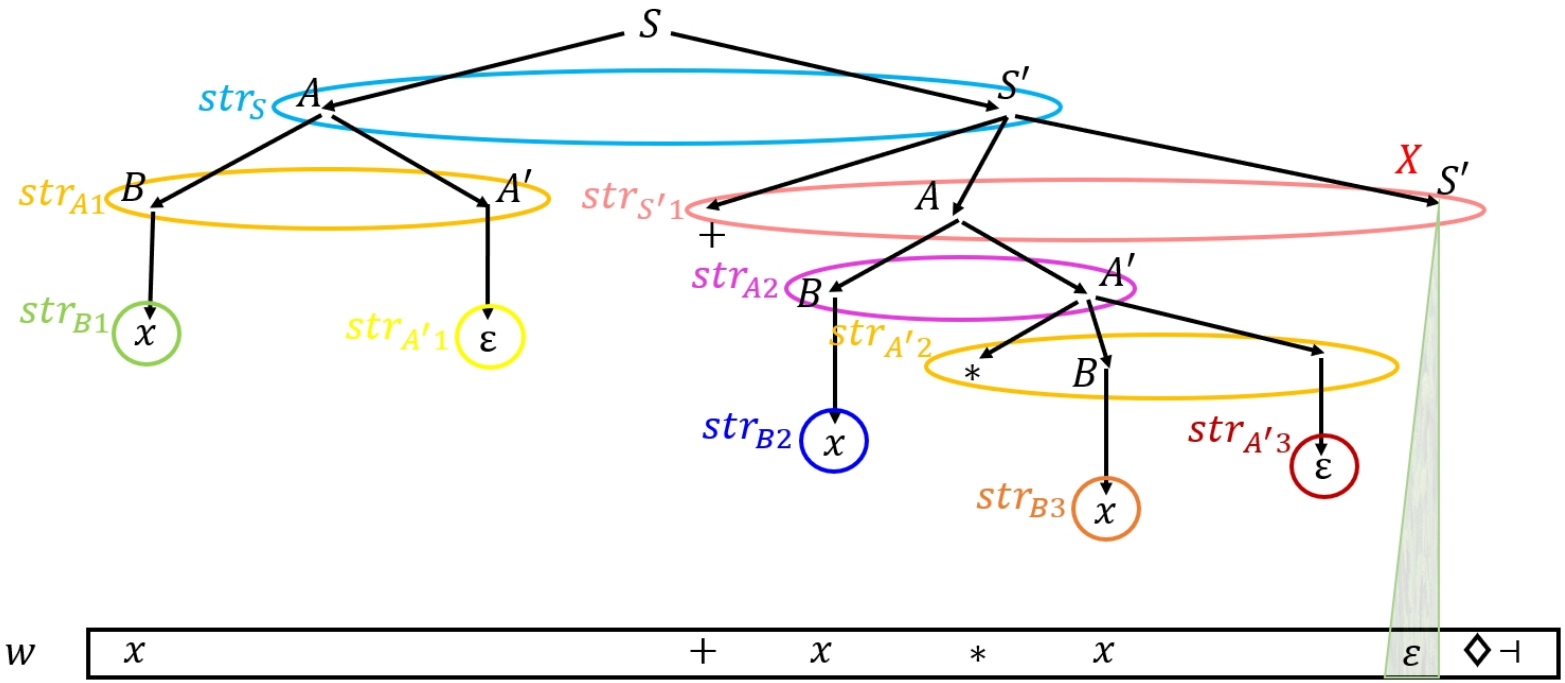


Рис. 18

Протокол работы алгоритма рекурсивного спуска для цепочки $w = x + x * x$. Иллюстрация

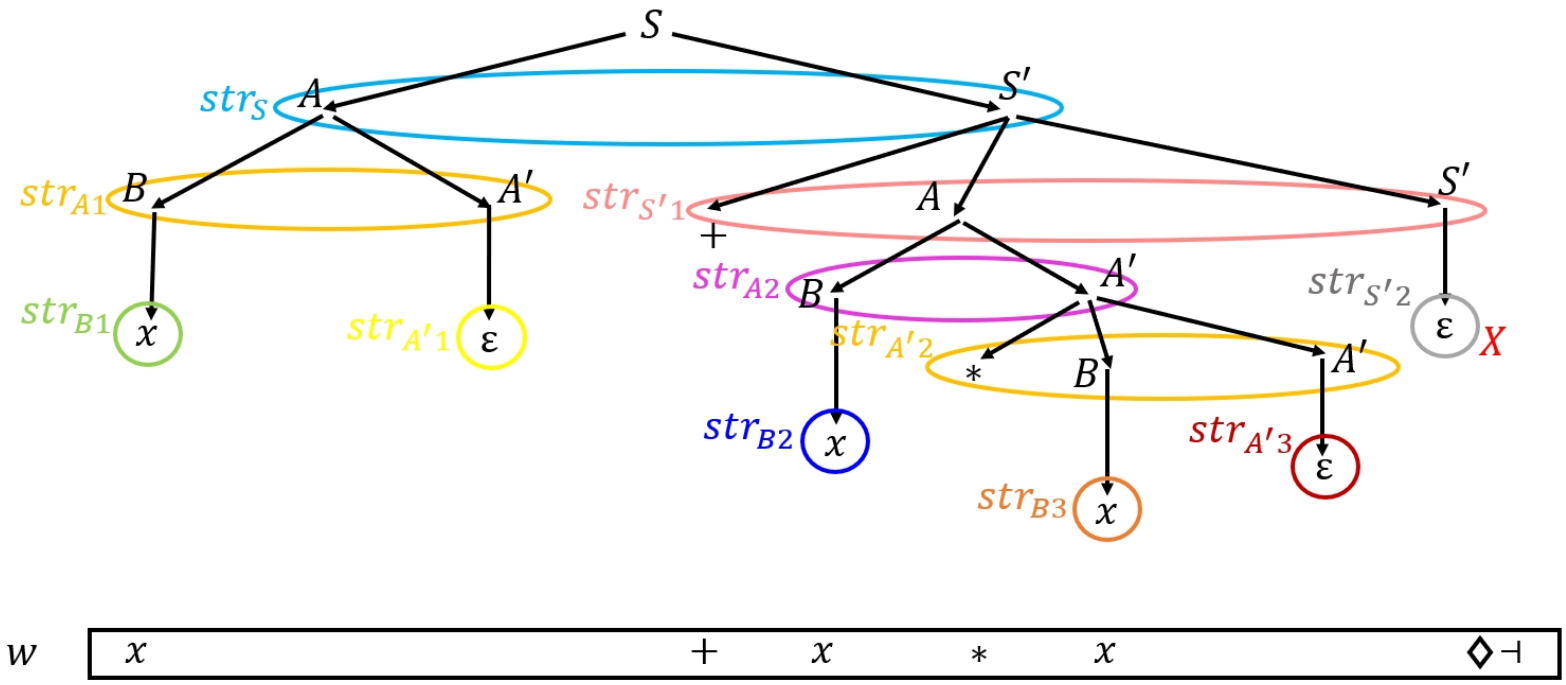


Рис. 19

Модификация алгоритма рекурсивного спуска для более общего случая

- Если мы хотим использовать таблицу переходов δ' модифицированного по лемме 3 ДМПА $\mathcal{M}''$ (см., например, слайд 8), то после (2)-го пункта перед подпунктами (2.1), (2.2), (2.3) для рекурсивной процедуры $A()$ алгоритма рекурсивного спуска надо добавить
- if $\delta'_2(A, lookahead) = \rightarrow$:
  lookahead=next symbol

Модификация алгоритма рекурсивного спуска для более общего случая

- Если мы хотим использовать таблицу переходов δ' модифицированного по лемме 3 ДМПА $\mathcal{M}''$ (см., например, слайд 8), то после (2)-го пункта перед подпунктами (2.1), (2.2), (2.3) для рекурсивной процедуры $A()$ алгоритма рекурсивного спуска надо добавить
- if $\delta'_2(A, lookahead) = \rightarrow$:
  lookahead=next symbol
- Подумать! Как исправить алгоритм рекурсивного спуска для ДМПА, чтобы он работал для НМПА.