Определение: Пусть T_1 и T_2 — алфавиты. *Формальный перевод* τ — это подмножество множества всевозможных пар цепочек в алфавитах T_1 и T_2: $\tau \subseteq (T_1^* \times T_2^*)$.

Назовем *входным* языком перевода τ язык $L_{вх}(\tau) = \{\alpha \mid \exists\ \beta : (\alpha, \beta) \in \tau\}$.

Назовем *целевым* (или *выходным*) языком перевода τ язык $L_{ц}(\tau) = \{\beta \mid \exists\ \alpha : (\alpha, \beta) \in \tau\}$.

Перевод τ *неоднозначен*, если для некоторых $\alpha \in T_1^*$, β, $\gamma \in T_2^*$, $\beta \neq \gamma$ справедливы соотношения: $(\alpha, \beta) \in \tau$ и $(\alpha, \gamma) \in \tau$.

Рассматриваемая далее грамматика с действиями по печати $G_{\texttt{expr\_polish}}$ задает однозначный перевод: каждому выражению ставится в соответствие единственная польская запись. Неоднозначные переводы могут быть интересны при изучении моделей естественных языков; для трансляции языков программирования используются однозначные переводы.

Синтаксически управляемый перевод

В основе синтаксически управляемого перевода лежит уже известная нам грамматика с действиями.

$G_{\texttt{expr}}$ – грамматика, описывающая простейшее арифметическое выражение:

$$E \to T \{+T\}$$
$$T \to F \{*F\}$$
$$F \to a \mid b \mid (E)$$

Рассмотрим цепочку $a+b*a$

Синтаксически управляемый перевод

В основе синтаксически управляемого перевода лежит уже известная нам грамматика с действиями.

$G_{\texttt{expr}}$ – грамматика, описывающая простейшее арифметическое выражение:

$$E \to T \{+T\}$$
$$T \to F \{*F\}$$
$$F \to a \mid b \mid (E)$$

$G_{\texttt{expr\_polish}}$ – грамматика с действиями по переводу выражения в ПОЛИЗ :

$$E \to T \{+T \langle\, cout << '+';\rangle\}$$
$$T \to F \{*F \langle\, cout << '*';\rangle\}$$
$$F \to a \langle\, cout << 'a';\rangle \mid b \langle\, cout << 'b';\rangle \mid (E)$$

В процессе анализа методом рекурсивного спуска входной цепочки $a + b * a$ по грамматике $G_{\texttt{expr\_polish}}$ в выходной поток будет выведена цепочка $a\ \ b\ \ a * +$

Непосредственное вычисление значения выражения методом рекурсивного спуска

$G_{\texttt{expr}}$ – грамматика, описывающая простейшее арифметическое выражение:

$$E \to T\ \{+T\}$$
$$T \to F\ \{*F\}$$
$$F \to 0 \mid 1 \mid 2 \mid \ldots \mid 9 \mid (E)$$

Каждому нетерминалу соответствует функция, вычисляющая значение подвыражения

БНФ:

$$\langle expr \rangle ::= \langle add \rangle\ \{+\ \langle add \rangle\}$$
$$\langle add \rangle ::= \langle mult \rangle\ \{*\langle mult \rangle\}$$
$$\langle mult \rangle ::= 0 \mid 1 \mid 2 \mid \ldots \mid 9 \mid (\langle expr \rangle)$$

```
program Calculator (input, output); {на языке Паскаль}
{ Вычисление значения выражения, содержащего цифры '0'-'9', знаки
  операций '+', '*' и скобки '(', ')'. Операндами могут быть
  только  числа от 0 до 9. Выражение записано без ошибок }

var  curlex:char;  { текущая лексема (лексема в данном примере -
                     это знак  операции, скобка  или цифра) }

procedure getlex;{ выделяет из входного потока очередную лексему}
var c:char;
begin
  read(c); while (c=' ') do read(c);
  curlex:=c
end;

function  expr: integer;forward;
     { распознает выражение и вычисляет его   значение }

function add: integer; forward;
     { распознает слагаемое и вычисляет его значение }

function mult: integer;forward;
     { распознает множитель и вычисляет его значение }
```

```
function expr: integer;
var e:integer;
begin
  e:=add;
  while curlex = '+' do
    begin getlex; e:=e+add end;
  expr:=e
end;

function add: integer;
var a:integer;
begin
  a:=mult;
  while curlex = '*' do
    begin getlex; a:=a*mult end;
  add:=a
end;
```

```
function mult: integer;
var   m:integer;
begin  case curlex of
     '0'..'9': m:=ord(curlex)-ord('0');
     '(': begin getlex; m:=expr end
       end;
getlex;{считали следующую после ')' или цифры}
mult:=m;
end;

begin
    write('==>');
    getlex;
    writeln(expr);
end.
```

Регулярные выражения

Кроме регулярных грамматик и конечных автоматов, существует еще один широко используемый в математических теориях и приложениях формализм, задающий регулярные языки. Это регулярные выражения. Они позволяют описать любой регулярный язык над заданным алфавитом, используя три вида операций: + (объединение), · (конкатенация), * (итерация).

Определение. Пусть L, L_1, L_2 — языки над алфавитом Σ. Тогда будем называть язык $L_1 \cup L_2$ *объединением* языков L_1 и L_2;

язык $L_1 \cdot L_2 = \{\varphi \cdot \psi | \ \varphi \in L_1, \psi \in L_2\}$ — *конкатенацией* (*сцеплением*) языков L_1 и L_2 (содержит всевозможные цепочки, полученные сцеплением цепочек из L_1 с цепочками из L_2);

i-ой степенью языка L назовем язык $L^i = L^{i-1} \cdot L$ для $i > 0$, $L^0 = \{\varepsilon\}$.

Язык $L^* = \bigcup_{n=0}^{\infty} L^n$ назовем *итерацией* языка L.

Определение. Пусть Σ — алфавит, не содержащий символов $*$, $+$, ε, $\varnothing$, (,). Определим рекурсивно *регулярное выражение* γ над алфавитом Σ и регулярный язык $L(\gamma)$, задаваемый этим выражением:

1) $a \in \Sigma \cup \{\varepsilon, \varnothing\}$ есть регулярное выражение; $L(a) = \{a\}$ для $a \in \Sigma \cup \{\varepsilon\}$; $L(\varnothing) = \varnothing$;

2) если α и β — регулярные выражения, то:
   а) $(\alpha) + (\beta)$ — регулярное выражение; $L\,((\alpha) + (\beta)) = L\,(\alpha) \cup L\,(\beta)$;
   б) $(\alpha)(\beta)$ — регулярное выражение; $L\,((\alpha)(\beta)) = L\,(\alpha)\,L\,(\beta)$;
   в) $(\beta)^*$ — регулярное выражение; $L\,((\beta)^*) = (L\,(\beta))^*$;

3) все регулярные выражения конструируются только с помощью пунктов 1 и 2.

Если считать, что операция «+» имеет самый низкий приоритет, а операция «*» — наивысший, то в регулярных выражениях можно опускать лишние скобки.

Примеры регулярных выражений над алфавитом $\{a, b\}$:

$$a + b, \qquad (a + b)^*, \qquad (aa\,(ab)^*bb)^*.$$

Соответствующие языки:

$$L(\,a + b\,) = \{a\} \cup \{b\} = \{a, b\},$$
$$L(\,(a + b)^*\,) = \{a, b\}^*,$$

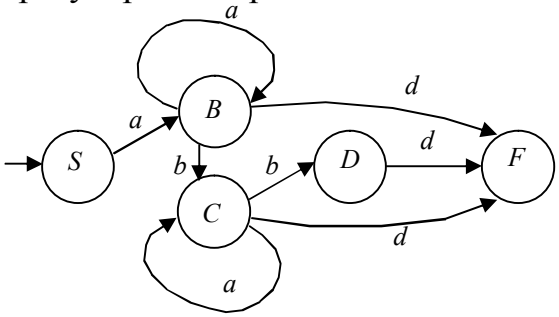
$L(\,(aa\,(ab)^*bb)^*) = \{\varepsilon\} \cup \{\,aa\,x_1bb\;aa\,x_2bb \ldots x_kbb \mid k \geq 1, x_i \in \{(ab)^n \mid n \geq 0\}$ для $i = 1, \ldots, k\,\}$.

Существуют алгоритмы построения регулярного выражения по регулярной грамматике или конечному автомату и обратные алгоритмы, позволяющие преобразовать выражение в эквивалентную грамматику или автомат. (На эту тему см. книгу: Д. Хопкрофт, Р. Мотвани, Д. Ульман. Введение в теорию автоматов, языков и вычислений).

Регулярные выражения используются для описания лексем языков программирования, в задачах редактирования и обработки текстов; подходят для многих задач сравнения изображений и автоматического преобразования. Расширенные их спецификации (эквивалентные по описательной силе, но более удобные для практики) входят в промышленный стандарт открытых операционных систем *POSIX* и в состав базовых средств языков программирования *Perl, Python* и др.

Вопросы и задачи

1. Постройте КС-грамматику для языка $\{a^n b^k \mid n, k \geq 0\}$. Добавьте в грамматику действия по переводу цепочек данного языка в цепочки языка $\{c^{2k} d^n \mid n, k \geq 0\}$.
2. Дайте определение регулярного выражения.
3. Постройте конечный автомат по регулярному выражению $(a+(aa+bb)^*b)^*+ba$.
4. Постройте регулярное выражение по конечному автомату



5. Эквивалентны ли выражения $(a + (ba)^* + b)^*$ и $(a^*b^* + \varepsilon)^*$?
6. Какова звездная высота выражений из задачи 5 ?
7. Найдите пересечение языка $\{a^n b^k \mid n, k \geq 0\}$ и языка, заданного выражением $a((a)^*b)^*b$