proc(□)\*(start(cos(y\*)+sin(x+)))\* □+ end

**Пусть:**

proc – a

□ – b

start – c

cos – d

sin – e

end – f

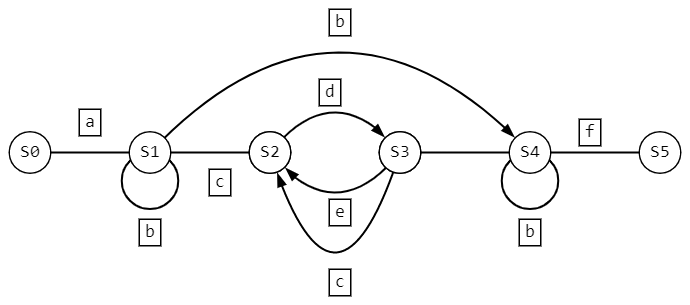
**Тогда выражение примет вид:**

a(b)\* (c(d + e)\* b+ f

**Примеры цепочек:**

1. abbf
2. abcdbf
3. abbbcebbf
4. abcecebbbbf
5. abbcdcdbbbf
6. abbbbcdcdcdcdbbbbbbf
7. abcebf

Граф конечного автомата:



M = ({S0, S1, S2, S3, S4, S5}, {a, b, c, d, e, f}, δ, S0, S5)

Таблица переходов состояний:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| s | a | b | c | d | e | f | λ |
| S0 | S1 | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ |
| S1 | ∅ | S1, S4 | S2 | ∅ | ∅ | ∅ | ∅ |
| S2 | ∅ | ∅ | ∅ | S3 | S3 | ∅ | ∅ |
| S3 | ∅ | S4 | S2 | ∅ | ∅ | ∅ | ∅ |
| S4 | ∅ | S4 | ∅ | ∅ | ∅ | S5 | ∅ |
| S5 | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ |

Диаграмма мгновенных состояний для цепочки abcdbf:

abcdbf –> bcdbf –>cdbf–>dbf–>bf–>f–> λ

**Алфавит 𝐼**

Алфавит *I* — это конечное множество символов, используемых для построения строк (слов). Символы из алфавита составляют основу для формальных языков и грамматик.

**Обозначения 𝜆, 𝐼∗, 𝐼+**

* 𝜆: Пустая строка, то есть строка, не содержащая ни одного символа.
* 𝐼∗: Множество всех возможных строк (включая пустую строку 𝜆*λ*), которые можно составить из символов алфавита 𝐼*I*.
* 𝐼+: Множество всех возможных непустых строк, которые можно составить из символов алфавита.

**Язык 𝐿(𝐼) над алфавитом 𝐼**

Язык 𝐿(𝐼) над алфавитом 𝐼 — это любое подмножество множества *I*∗, то есть множество строк, составленных из символов алфавита 𝐼.

**Определение формальной грамматики 𝐺**

Формальная грамматика 𝐺 — это математическая система, определяемая как четверка 𝐺=(𝑁,Σ,𝑃,𝑆), где:

* 𝑁 — конечное множество нетерминальных символов.
* Σ — конечное множество терминальных символов.
* 𝑃 — конечное множество правил подстановки (продукций) вида 𝛼→𝛽*α*→*β*.
* 𝑆 — начальный символ (*S*∈*N*).

**Обозначения 𝛼⇒𝛽*α*⇒*β* и 𝛼⇒∗𝛽*α*⇒∗*β***

* 𝛼⇒𝛽*α*⇒*β*: Строка 𝛼непосредственно выводится в строку 𝛽 по одному из правил подстановки из множества 𝑃.
* 𝛼⇒∗𝛽*α*⇒∗*β*: Строка 𝛼 выводится в строку 𝛽 за конечное число шагов, применяя одно или более правил подстановки из множества *P*.

**Язык *L*(*G*), порождаемый грамматикой *G***

Язык *L*(*G*), порождаемый грамматикой *G*, — это множество всех строк, состоящих только из терминальных символов, которые могут быть выведены из начального символа 𝑆

**Форма Бэкуса-Наура (BNF)**

Форма Бэкуса-Наура (BNF) — это нотация для описания синтаксиса языков программирования и формальных языков. Она использует правила продукции вида <𝑛𝑜𝑛−𝑡𝑒𝑟𝑚𝑖𝑛𝑎𝑙>::=<𝑒𝑥𝑝𝑟𝑒𝑠𝑠𝑖𝑜𝑛><*non*−*terminal*>::=<*expression*>, где выражение может состоять из терминальных и нетерминальных символов, а также операторов для объединения и последовательности.

**Регулярная грамматика**

Регулярная грамматика — это грамматика, в которой каждое правило продукции имеет один из следующих видов:

* *A*→*aB* или 𝐴→𝑎, где 𝐴 и 𝐵— нетерминалы, *a* — терминал (леворегулярная грамматика).
* *A*→*Ba* или 𝐴→𝑎*A*→*a*, где *A* и *B* — нетерминалы, 𝑎 — терминал (праворегулярная грамматика).

**Регулярное множество**

Регулярное множество — это множество строк, которое может быть описано регулярным выражением, соответствующее регулярной грамматике или распознаваемое конечным автоматом.

**Регулярный язык**

Регулярный язык — это язык, который может быть описан регулярным выражением, порожден регулярной грамматикой или распознаваем конечным автоматом.

**Лексический анализ**

Лексический анализ — это процесс преобразования входной последовательности символов (текста) в последовательность токенов (лексем), где каждый токен представляет собой значимую единицу языка.

**Лексический анализатор**

Лексический анализатор (лексер) — это программа или часть компилятора, выполняющая лексический анализ. Он читает входной текст, распознает токены и передает их синтаксическому анализатору.

**Входная и выходная информация для лексического анализатора**

* Входная информация: Исходный текст программы или данных.
* Выходная информация: Последовательность токенов (лексем).

**Последовательные и параллельные лексические анализаторы**

* Последовательный лексический анализатор: Анализирует текст по одному символу за раз, последовательно распознавая токены.
* Параллельный лексический анализатор: Использует многопоточность или другие методы для одновременного анализа различных частей текста, ускоряя процесс распознавания токенов.

**Регулярное выражение над алфавитом *I***

Регулярное выражение над алфавитом 𝐼— это выражение, определяющее язык над 𝐼 с использованием операций объединения (∣), конкатенации (⋅), и замыкания Клини (∗). Регулярные выражения строятся из символов алфавита 𝐼, пустой строки 𝜆, и операторов.

**Конечный автомат 𝑀=(𝑆,𝐼,𝛿,𝑠0,𝐹)**

Конечный автомат 𝑀*M* определяется как пятерка:

* 𝑆 — конечное множество состояний.
* *I* — алфавит.
* *δ* — функция переходов.
* *s*0​ — начальное состояние.
* *F* — множество заключительных (допустимых) состояний.

**Отличие детерминированного и недетерминированного автомата**

* Детерминированный конечный автомат (DFA): В каждом состоянии для каждого символа алфавита имеется не более одного перехода в следующее состояние.
* Недетерминированный конечный автомат (NFA): В каждом состоянии для каждого символа алфавита может быть несколько переходов в следующее состояние, или даже отсутствие переходов.

**Мгновенное состояние конечного автомата**

Мгновенное состояние конечного автомата — это пара (𝑠,𝑤)(*s*,*w*), где 𝑠*s* — текущее состояние автомата, а 𝑤*w* — оставшаяся часть входного слова.

**Соотношение регулярной грамматики, регулярного языка, регулярного выражения, конечного автомата и графа состояний конечного автомата**

Все эти понятия эквивалентны в смысле выразительной мощности:

* Регулярная грамматика порождает регулярный язык.
* Регулярный язык описывается регулярным выражением.
* Регулярный язык распознается конечным автоматом (DFA или NFA).
* Граф состояний конечного автомата визуально представляет работу конечного автомата, показывая состояния и переходы между ними.