# Лабораторная работа №1

# Тема: «Регулярные языки»

## Цель работы

Ознакомиться с классификацией Хомского, понятием регулярной грамматики и действиями над грамматикой; научиться составлять конечный автомат согласно заданной грамматике; ознакомиться с функционалом программы JFLAP, позволяющей автоматизировать, упрощать и проверять поставленные задачи.

### Теоретическая справка

Формальным языком с алфавитом A (в данном случае – $V_T$) называется множество слов L, составленное на базе конечного алфавита. В свою очередь алфавит – это конечное множество неделимых элементов (в данном случае – символов).

Пример алфавита:

$V_T$ = {0, 1} (I)

$V_T$ = {a, b, c} (II)

Всякая последовательность символов алфавита называется цепочкой или словом. Например, слово x=11101 будет словом, построенным на базе алфавита (I), с длиной, равной |x| = 5.

Любой формальный язык определяется грамматикой: G = $(V_N, V_T, P, S)$, где:

$V_N$ – множество нетерминальных символов;

$V_T$ – множество терминальных символов;

P – множество правил;

S – аксиома, с которой начинается вывод цепочки;

Как было сказано ранее, формальный язык — это множество слов. А таких слов может большое, а иногда – и бесконечное количество. Потому задавать такое множество методом перечисления неудобно, а в некоторых случаях – невозможно. Для этого существует способ записи **общего вида языка**. Это задание множества с помощью терминальных символов, и некоторых операций над ними. Список операций:

$\underline{a^+}$ – «повторение», то есть символ ‘a’ должен быть использован хотя бы 1 раз (может повторяться n раз, где n>0);

$\underline{a^*}$ – «повторение», то есть символ ‘a’ может и не быть использованным (пустой символ), а может и повторяться (может повториться n раз, где $n \geq 0$); $a^i$ – «повторение», то есть символ ‘a’ повторится i раз;

$\underline{a, b}$ – «или», то есть выбор одного символа или последовательности из всех доступных Вариантов;

$\underline{ab}$ – «конкатенация», то есть объединение символов ‘a’ и ‘b’ в цепочку ‘ab’;

Порядок выполнения операций – слева направо, а приоритет определяется круглыми скобками, как в математике.

Пример нескольких цепочек, составленных по общему виду языка:

Общий вид: L(G) = {a (a, b) (d)*}. Цепочки: “abd”, “aa”, “aaddd”, “abdd”.

Общий вид: L(G) = {0, (0, 1)*, 1}.

Цепочки: “0011”, “01”, “01101”, “000001”, “010101”

Теперь пошагово. Общий вид: $\mathrm{L(G)} = \{(\mathrm{a, b, c})^+ \mathrm{d}\}$

Слева направо встречаются скобки с пометкой «+». Как было описано выше, элементы из скобок могут повторяться до бесконечности, но минимум – один раз.

То есть можно сформировать любую последовательность из символов «a, b, c», длина которой будет больше нуля. Это может быть “a”, “c”, “ba”, “ccbc”, и так далее. После скобок идет символ d, который является последним в цепочке. Таким образом, по общему виду могут быть составлены слова:

“abcbcad”, “ccbd”, “bcd”, “bacd”, “cd”, “ad”, и другие, до бесконечности.

Формальные языки делятся на четыре типа:

Тип 0 ⊃Тип 1⊃Тип 2⊃Тип 3

Данная запись означает, что Тип 3 содержится в Типе 2, который содержится в Типе 1, который в свою очередь, содержится в Типе 0. Таким образом, Тип 0 является общим для всех языков, и содержит в себе все остальные типы.

В данной лабораторной работе, будет рассмотрен **третий тип**, к которому относятся **регулярные языки**. Это самый строгий тип.

Общая характеристика третьего типа:

- В левой части правила может быть только один нетерминальный символ
- В правой части должен сперва быть один терминальный символ и за ним нетерминальный символ или `ε`, который, обычно, просто не пишется
- В правой части не может быть пустого символа `ε`

То есть, обобщая вышеуказанные характеристики, у типа 3 могут быть правила:

1. $A \to aB$ $\quad A, B \in V_N, a \in V_T$
2. $A \to a$ $\quad A \in V_N, a \in V_T$

**Второй тип** языков (к которым относятся **контекстно-независимые языки)** отличается от третьего типа тем, что его правила могут содержать в правой части сколько угодно терминальных и нетерминальных символов, как и пустой символ.

**Первый тип** (**контекстно-зависимые языки**) не имеет ограничений в количестве символов в левой части правила. Правая часть – как и в типе 2.

Любое правило для языка типа 1 будет выглядеть так: «$\alpha A \beta \to \alpha \omega \beta$», где: A – нетерминальный символ ($A \in V_N$);

ω – непустая цепочка, состоящая из терминальных и нетерминальных символов ( $\omega \in (V_T \cup V_N)^+$ );

α и β – любая цепочка, состоящая из терминальных и нетерминальных символов ($\alpha, \beta \in (V_T \cup V_N)^*$);

Примеры нескольких правил для языка первого типа:

S → aSBC

aBc → abSc

**Тип 0 (независимые языки)** почти не накладывают никаких условий на формирование слов.

Общий вид правила: «α → β»,

где:

α – любая цепочка, содержащая хоть один нетерминальный символ

$$\alpha \in (V_T \cup V_N)^* V_N (V_T \cup V_N)^*;$$

β – любая цепочка, содержащая терминальные и нетерминальные символы

$$\beta \in (V_T \cup V_N)^*;$$

Примеры слов языка типа 0:

A → bSC

aDSt → KBs

TRrS → OmTP

prS → ε

**Составление цепочки** по грамматике $G = (V_N, V_T, P, S)$, где:

$V_N = \{S, A, B, D\}$;

$V_T = \{a, b, c, d\}$;

P = {1. S → aA

2. A → bB

3. B → cD

4. B → c

5. D → dB}

Любое слово языка типа 3 всегда начинается с аксиомы (S). Далее, для имеющегося нетерминального символа (из $V_N$), выбирается соответствующее правило из множества правил (P). Правила рассматриваются в следующем порядке:

1. находится нужный нетерминальный символ в левой части правила. Если их несколько, то выбирается один из нескольких;
2. выбранный символ заменяется на набор символов из правой части правила.

Таким образом, при каждом переходе по правилу, в слове будет появляться новый терминальный символ. Слово считается оконченным, когда не содержит в себе ни одного нетерминального символа.

Пример составления слова:

В скобках пишется номер правила, по которому осуществляется переход.

$S \Rightarrow^{1} aA \Rightarrow^{2} abB \Rightarrow^{3} abcD \Rightarrow^{6} abcdS \Rightarrow^{1} abcdaA \Rightarrow^{2} abcdabB \Rightarrow^{4} abcdabc.$

Итоговое слово: «abcdabc».

Другой способ представления процесса формирования слова – это **дерево вывода.** Правила составления дерева:

Для правила ($Z \rightarrow x_1 x_2 \dots x_n$, где: $x_1 x_2 \dots x_n \in V_N \cup V_T$ и $Z \in V_N$) отображаем символ в правой части правила вершиной (узлом) дерева. Символы в правой части правила записываются как потомки узла Z в порядке слева – направо. Таким образом, первый символ будет самым левым потомком узла Z и так далее (рисунок 1). Алгоритм повторяется, пока все вершины дерева не будут являться терминальными символами.

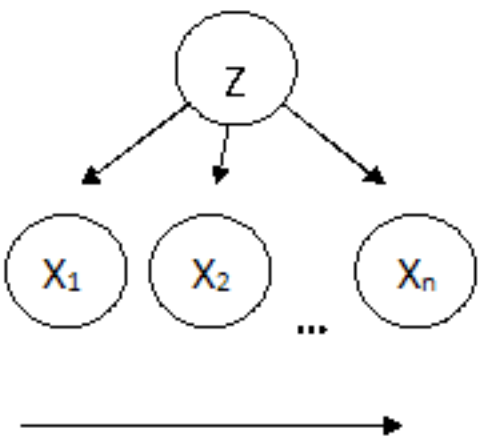
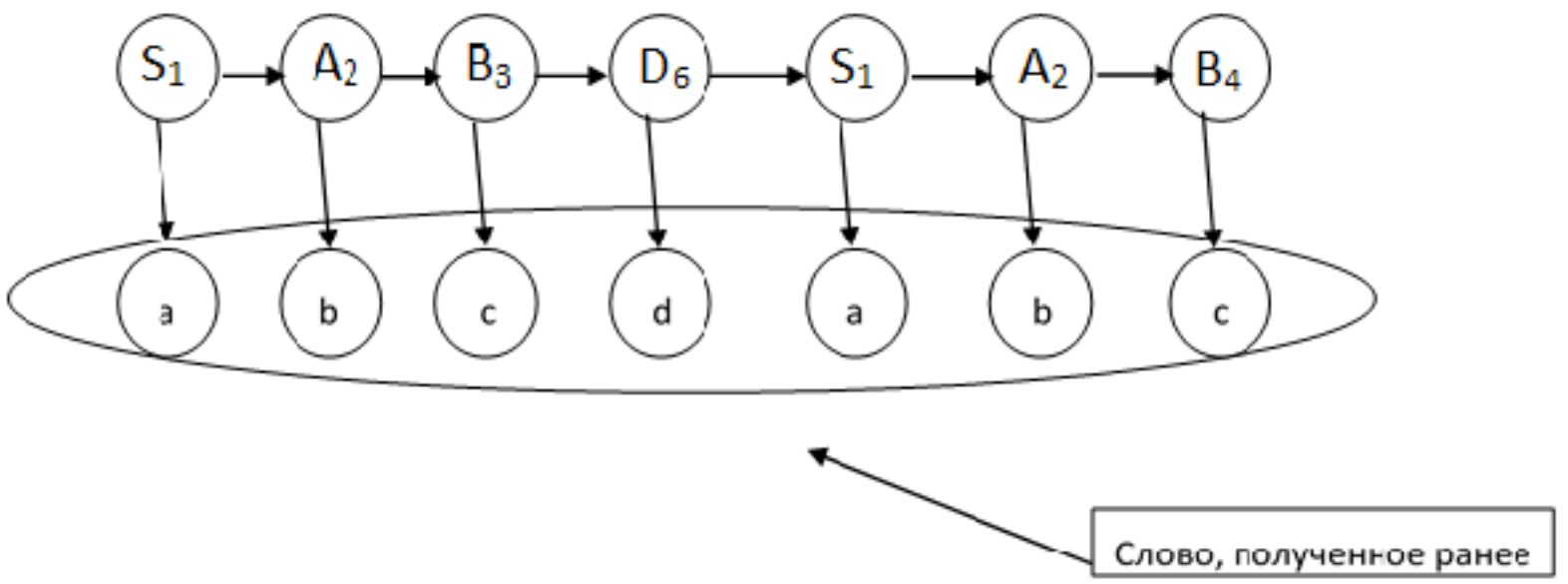



Рисунок 1

Пример составления дерева вывода для слова, описанного выше.




**Конечный автомат** можно представить, как устройство с блоком чтения, принимающее ленту с символами. Если лента будет пропущена конечным автоматом до конца – значит слово принадлежит языку конечного автомата. Если же конечный автомат остановится на одном символе, и не пропустит ленту дальше – значит была найдена комбинация символов, недопустимая для данного языка.

Для любой регулярной грамматики можно составить эквивалентный ему конечный автомат.

Конечный автомат, составленный на основе регулярного языка, описывается несколькими параметрами:

Q – множество состояний КА.

∑ – алфавит входных (терминальных) символов, допустимых КА.

$q_0$ – начальное состояние КА, из него будет осуществляться переход в другие состояния. Аналог аксиомы (S) из регулярной грамматики.

F – множество конечных состояний. Если КА находится в одном из этих состояний, и в очереди на прочтение символа входной ленты – пустой символ `ε`, то цепочка считается завершённой, и допускается конечным автоматом. В регулярной грамматике это значит, в цепочке не осталось нетерминальных символов.

δ – множество функций перехода из одного состояния КА в другое. Одна функция описывает Варианты перехода из текущего состояния по определённому символу, в одно из нескольких доступных.

**Составление конечного автомата, эквивалентного грамматике** происходит следующим образом:

Есть грамматика $G = (V_N, V_T, P, S)$

$V_N = \{S, A, B, D\}$; $V_T = \{a, b, c, d\}$;

P = {1. $S \rightarrow aA$

2. $A \rightarrow bB$

3. $B \rightarrow cD$

4. $B \rightarrow c$

5. $D \rightarrow dB$

6. $D \rightarrow dS$}

Необходимо составить эквивалентный конечный автомат КА = (Q, ∑, δ, $q_0$, F), то есть – определить все его параметры.

Первый параметр конечного автомата – множество `Q`. Это множество состояний КА, и оно составляется на основе множества нетерминальных символов `$V_N$` грамматики G. Но, в то же время, в него добавляется новый элемент ‘X’ - конечное состояние автомата.

$Q := V_N \cup \{X\} = \{S, A, B, D, X\}$

Далее следует параметр `∑`, алфавит КА. Элементы добавляются в соответствии с множеством терминальных символов `$V_T$` грамматики G.

∑ := $V_T$ = {a, b, c, d}

$q_0$ – начальное состояние КА. В грамматике G построение цепочки всегда начинается с аксиомы `S`, следовательно – аксиома и будет являться начальным состоянием для КА.

$q_0$ := S

Конечное множество состояний КА – `F`. Одним из них будет являться состояние `X`, добавленное в множество состояний Q при его формировании.

F := {X}

Множество функций перехода из одного состояния в другое - `δ`. Изначально все функции перехода являются пустыми множествами (δ(S, a):= ∅ ). Данное множество формируется на основе множества правил `P` грамматики G. Эти множества формируются по мере преобразования правил в функции перехода. В них заносятся нетерминальные символы, стоящие в правой части правила.

Например, δ (S, a) = δ (S, a) U {A} = {Ø} U {A} = {A}, так как в правой части правила «1. S → aA» нетерминальный символ - `A`.

Если существует два, и более правил перехода для одного нетерминального символа, то множество будет содержать столько же элементов, сколько и существует таких правил для одного символа.

Вернёмся к примеру, рассмотренному выше:

сначала δ (D, d) = {Ø},

потом δ (D, d) = δ (D, d) U {B} = {Ø} U {B} = {B}, и позже,

δ(D, d) = δ(D, d) U {S} = {B} U {S} = {B, S}.

В результате: δ(D, d) = {B, S}.

Символ `X` ставится в случае, когда в правиле, после терминального символа идёт символ `ε`, то есть – пустой символ.

Например, следуя из правила «4. B → c», получится, что δ (B, c) = {X}.

Обобщая вышеописанное, последовательность действий будет выглядеть так:

$\delta(S, a) = \delta(S, a) \cup \{A\} = \{\emptyset\} \cup \{A\} = \{A\}$

$\delta(A, b) = \delta(A, b) \cup \{B\} = \{\emptyset\} \cup \{B\} = \{B\}$

$\delta(B, c) = \delta(B, c) \cup \{D\} = \{\emptyset\} \cup \{D\} = \{D\}$

$\delta(B, c) = \delta(B, c) \cup \{X\} = \{D\} \cup \{X\} = \{D, X\}$

$\delta(D, d) = \delta(D, d) \cup \{B\} = \{\emptyset\} \cup \{B\} = \{B\}$

$\delta(D, d) = \delta(D, d) \cup \{S\} = \{B\} \cup \{S\} = \{B, S\}$

В итоге получилось:

$Q = \{S, A, B, D, X\}$

$\sum = \{a, b, c, d\}$

$q_0 = S$

$F = \{X\}$

$\delta (S, a) = \{A\}$

$\delta (A, b) = \{B\}$

$\delta (B, c) = \{D, X\}$

$\delta (D, d) = \{B, S\}$.

Чтобы доказать эквивалентность грамматики и автомата, согласно правилам грамматики составляется цепочка, которая проверяется на допуск автоматом. Если автомат допускает цепочку, то грамматика и автомат эквивалентны.

**Алгоритм проверки слова конечным автоматом**

Свою работу конечный автомат начинает из начального состояния $q_0$. На его вход поступает слово, которое рассматривается, начиная с первого символа. Если конечный автомат находится в состоянии $q_i$, существует функция перехода $\delta(q_i, a) = \{q_j, q_k\}$ и рассматривается символ «a», то для всех состояний перехода

рассматриваются все возможные переходы конечного автомата в каждое из них с удалением из ленты символа a.

Записывается это таким образом:

$$(q_i, ax)\vdash(q_j, x)$$
$$\top$$
$$(q_k, x)$$

Как видно на примере, конечный автомат разбился на две ветви. Если хоть одна ветвь автомата проверит всю цепочку и остановится в одном из конечных состояний, то цепочка принадлежит языку конечного автомата. Если же ни одна ветвь автомата не проходит до конца цепочки (нет подходящих функций перехода) или, пройдя всю цепочку, не останавливается в одном из конечных состояний, то цепочка не принадлежит языку конечного автомата.

**Пример доказательства эквивалентности грамматики и конечного автомата на основе составленного выше автомата:**

1. Составляется цепочка согласно правилам грамматики (правила были записаны выше):

   S=>[1]aA=>[2]abB=>[3]abcD=>[5]abcdB=>[4]abcdc

2. Цепочка проверяется на допуск автоматом:

Начальное состояние автомата: $(S, abcdc)$.

Существует функция перехода «δ(S, a) = {A}», следовательно – автомат изменяет свою конфигурацию:

$$(S, abcdc)\vdash(A, bcdc)$$

Аналогично расписывается оставшееся слово:

$$(S, abcdc)\vdash(A, bcdc)\vdash(B, cdc)\vdash(D, dc)\vdash(B, c)\vdash(\mathrm{X}, \varepsilon)-\text{Допуск}$$
$$\vdash \quad \vdash \quad \vdash$$
$$(\mathrm{X}, dc) \quad (S, c) \quad (D, \varepsilon)$$
$$\vdash \quad \vdash \quad \vdash$$
$$\text{Тупик} \quad \text{Тупик} \quad \text{Тупик}$$

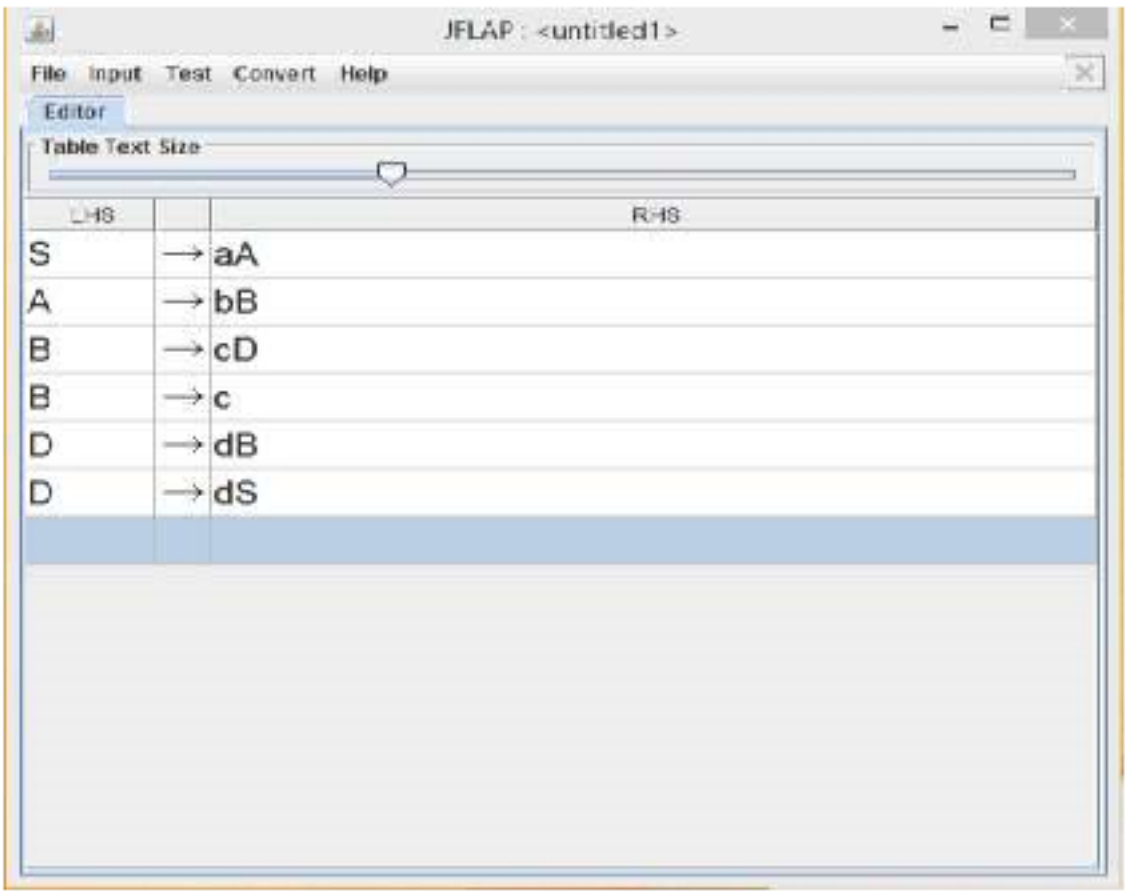
КА пропустил слово, принадлежащее грамматике, а значит – грамматика и КА эквивалентны.

## Работа в программе JFLAP

### Ввод грамматики в JFLAP

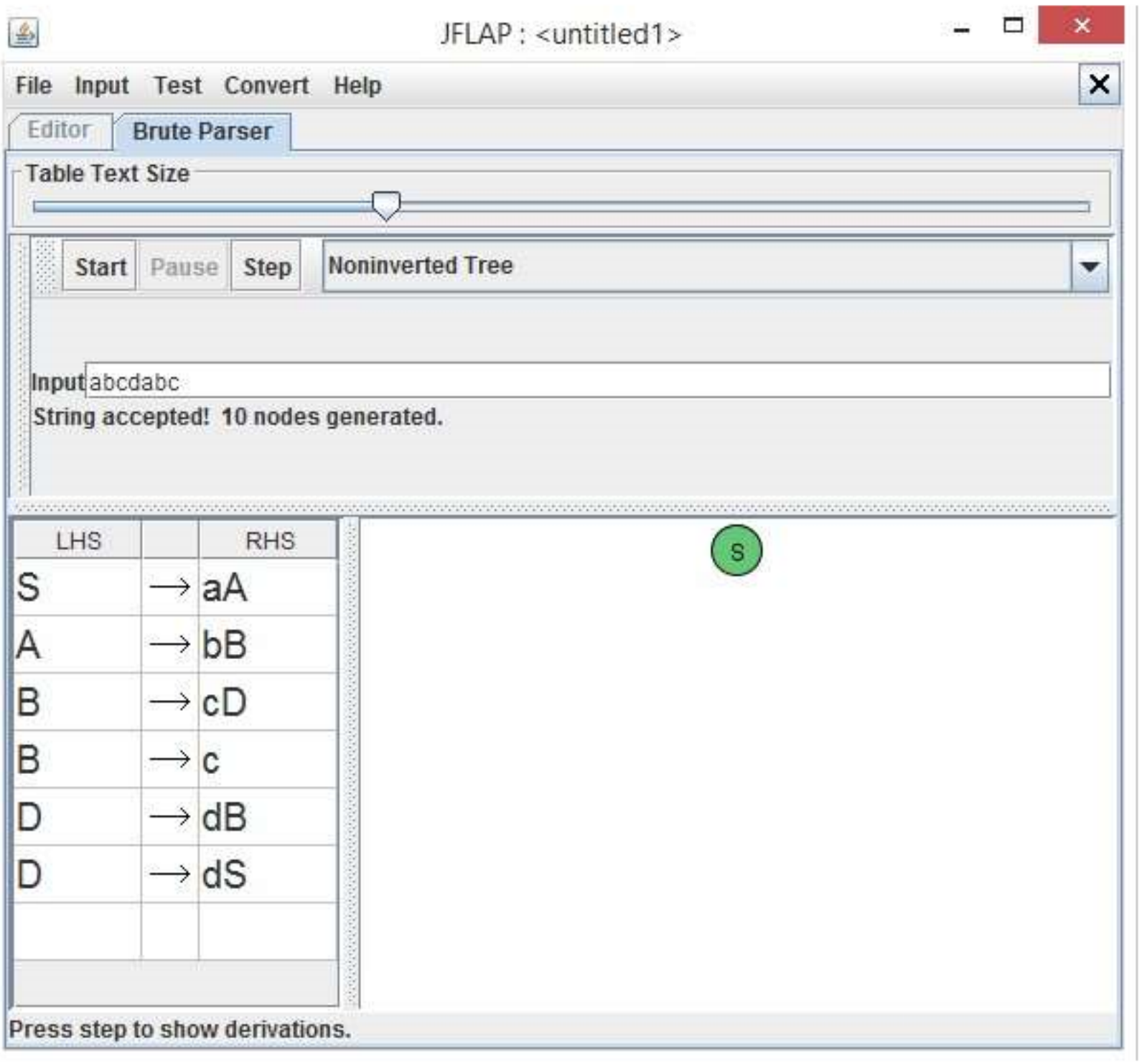
В главном меню – Grammar (Грамматика).


Откроется окно для ввода множества правил. В левом столбце (LHS) нужно ввести нетерминальный символ, из которого будет осуществляться переход, и нажать на клавиатуре два раза стрелку вправо, чтобы выделить правый столбец (RHS) для ввода. Средний столбец автоматически заполнится стрелкой перехода. В правый столбец (RHS) вводится выражение, в которое будет переходить нетерминальный символ слева.



**Проверка слова на принадлежность языку грамматики**

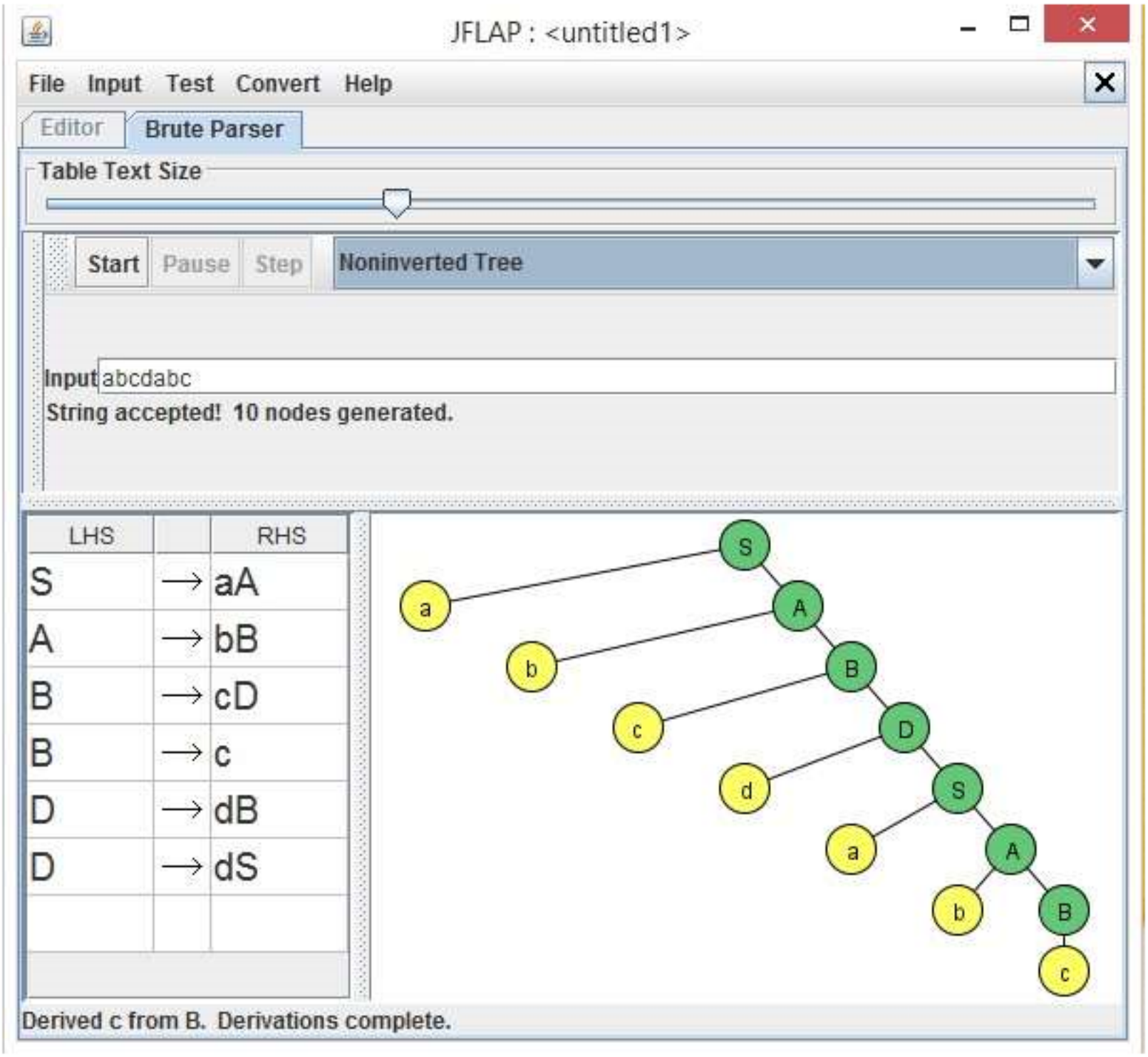
После окончания ввода грамматики, в меню “Input” нужно выбрать “Brute Force Parse”. Откроется новая вкладка, с полем “Input”, правилами языка (слева), и, пока что, пустым деревом вывода (справа). Нужно ввести слово, подлежащее проверке, в поле “Input”, и нажать Enter.



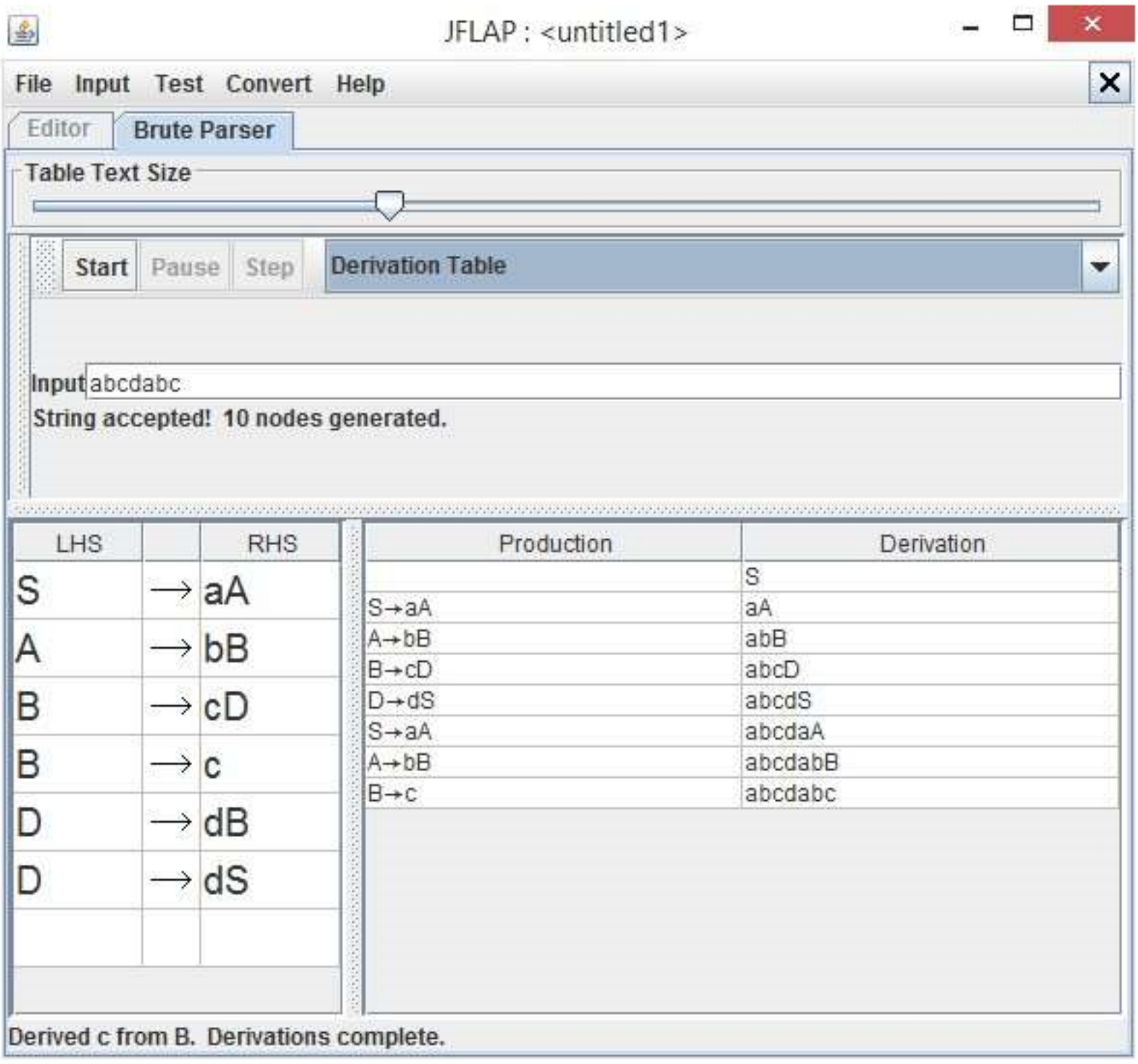

Ниже поля ввода появится результат: «String accepted! ‘X’ nodes generated», если слово принадлежит языку, или «String rejected. ‘X’ nodes generated», если слово не принадлежит языку. ‘X’ - количество вершин в дереве вывода для слова.

**Построение дерева вывода и таблицы вывода**

После проверки слова на принадлежность языку, можно построить для него дерево вывода. Для этого, в окне, где слово проверялось на принадлежность языку, нужно нажать несколько раз кнопку “Step” (Шаг), выше поля ввода слова. Если нужно получить дерево вывода, то правее кнопки “Step”, в выпадающем меню должен быть выбран Вариант “Noninverted Tree” (Неинвертированное дерево).



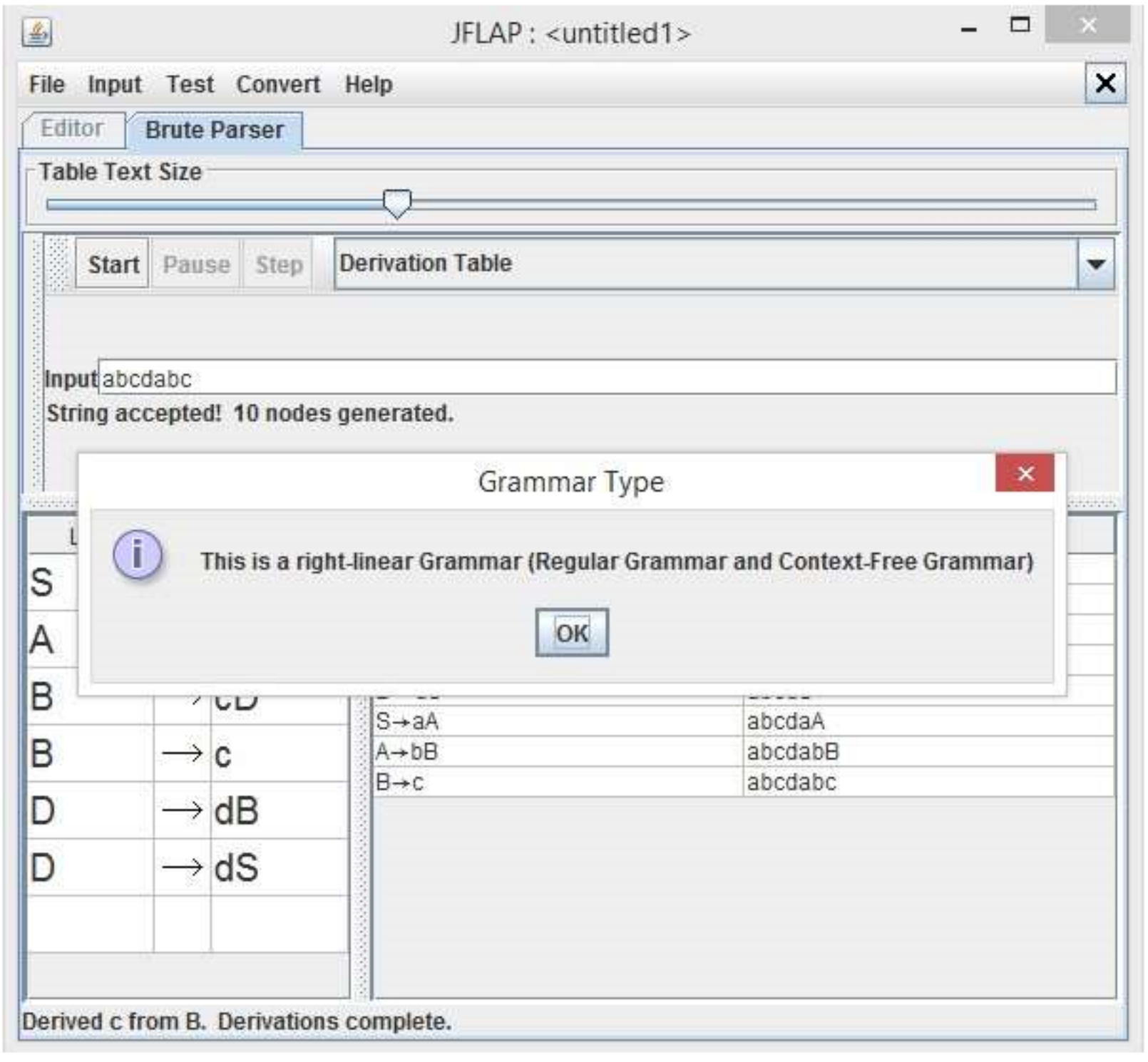
Если же нужна таблица вывода, то там нужно выбрать “Derivation Table” (Таблица Вывода).




Чтобы получить полное дерево/таблицу вывода, кнопку “Step” нужно нажимать столько раз, пока новые символы в дереве/таблице не закончат появляться. Кнопка станет неактивной.

**Проверка типа введённого языка**

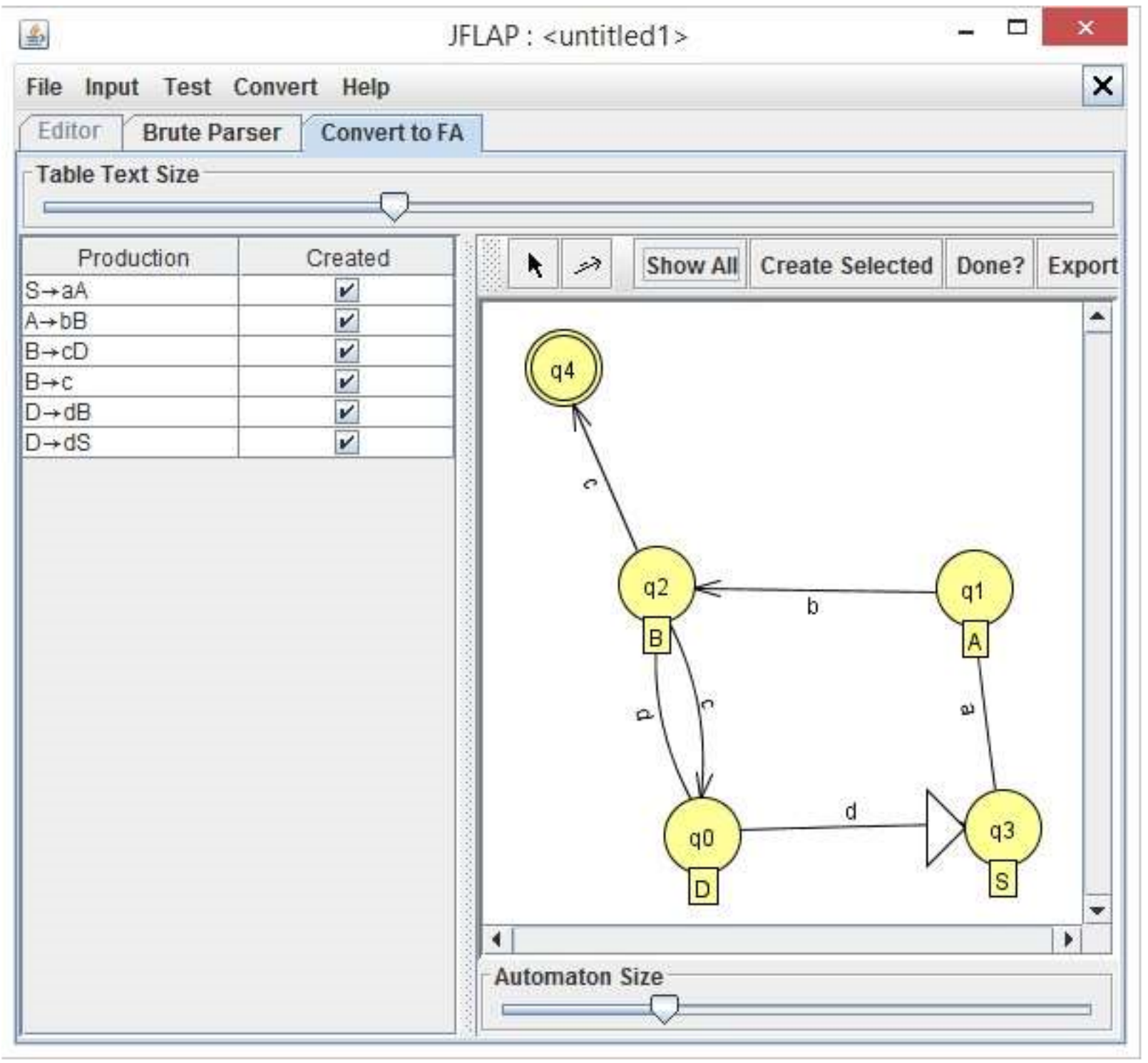
В этом же окне можно проверить тип введённого языка. Для этого нужно открыть меню “Test”, и выбрать единственный пункт “Test for Grammar Type” (Тестировать на принадлежность типу). Откроется диалоговое окно с информацией о введённом языке. В данном случае, там должно быть написано: “This is a right-linear Grammar (Regular Grammar and Context-Free Grammar)”, что переводится как: “Это язык с правым порождением (регулярный и контекстно-свободный язык)”.



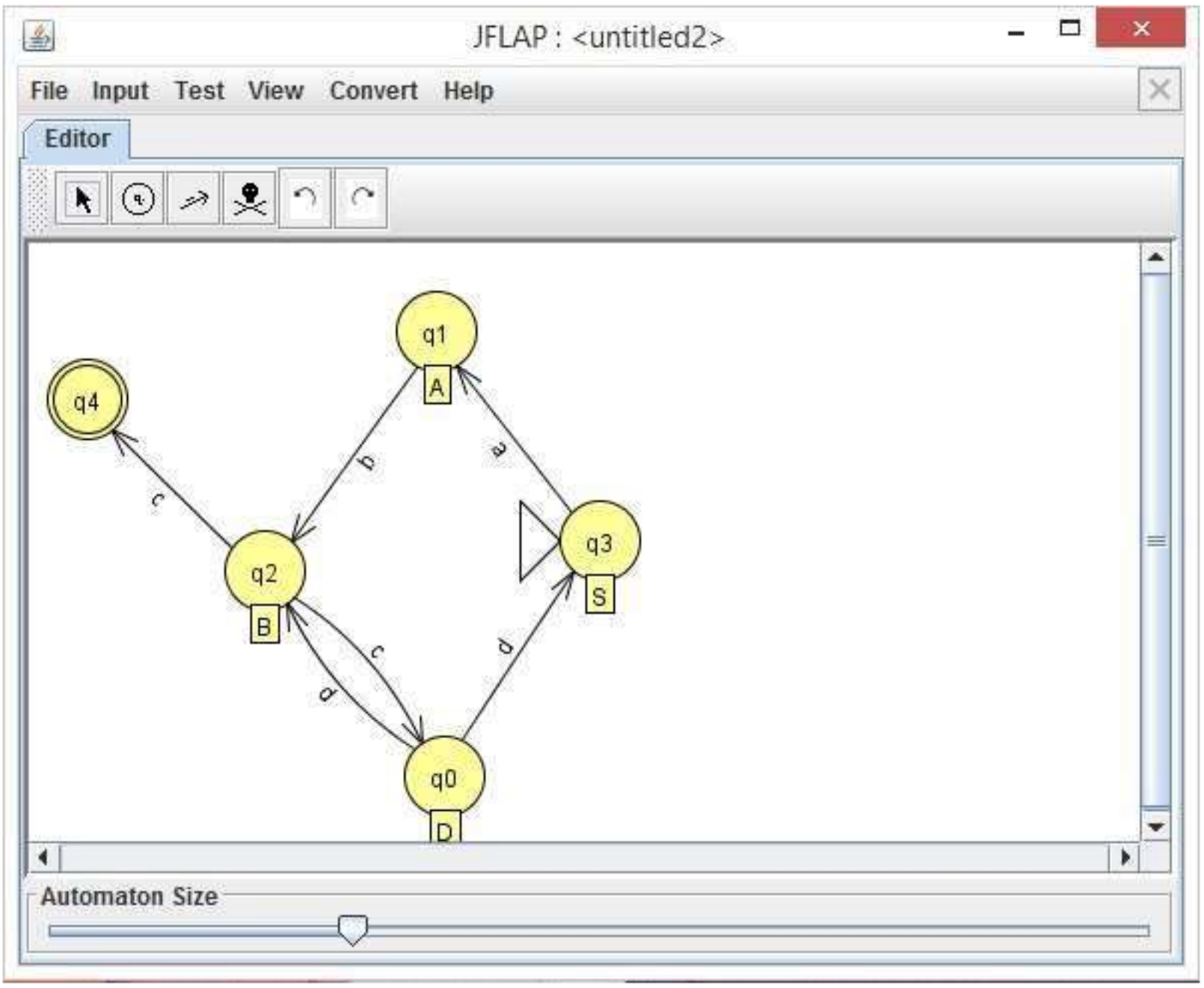

По классификации Хомского, это язык типа 3. Нажатие кнопки «ОК» закроет диалоговое окно.

**Построение конечного автомата, эквивалентного введённому языку**

Снова, в этом же окне, нужно открыть меню “Convert” (Преобразовать), и выбрать пункт “Convert Right-Linear Grammar to FA” (FA – Finite Automata = КА – Конечный Автомат), что значит “Конвертировать язык с правым порождением в КА”. Откроется новая вкладка для конечного автомата. Слева – множество правил, справа – граф конечного автомата. Чтобы показать все рёбра графа, нужно нажать кнопку “Show All”, чуть выше области с графом.




Далее, чтобы рассматривать конечный автомат, как новый элемент, отдельно от грамматики, правее “Show All” нужно нажать “Export”. Конечный автомат откроется в новом окне.

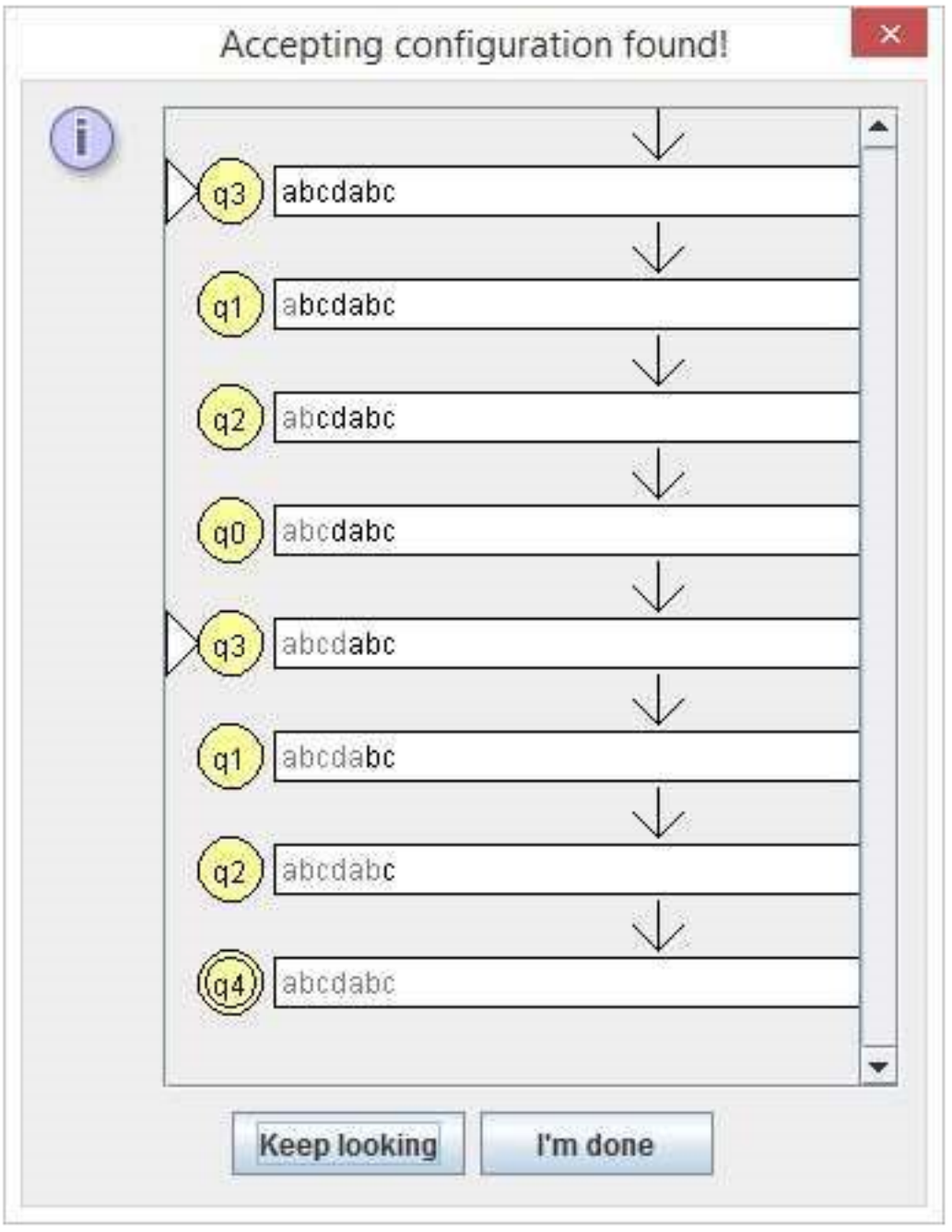



Ниже этой области есть ползунок для выбора масштаба картинки. Программа сама выбирает оптимальный масштаб, но при необходимости есть возможность изменения. Конечный автомат готов.

**Проверка слова на принадлежность языку конечного автомата**

В том же окне – выпадающее меню “Input” -> “Fast Run”. Откроется окно для ввода слова, подлежащего проверке. Проверится слово “abcdabc”, принадлежащее языку грамматики. Чтобы подтвердить – кнопка «ОК».

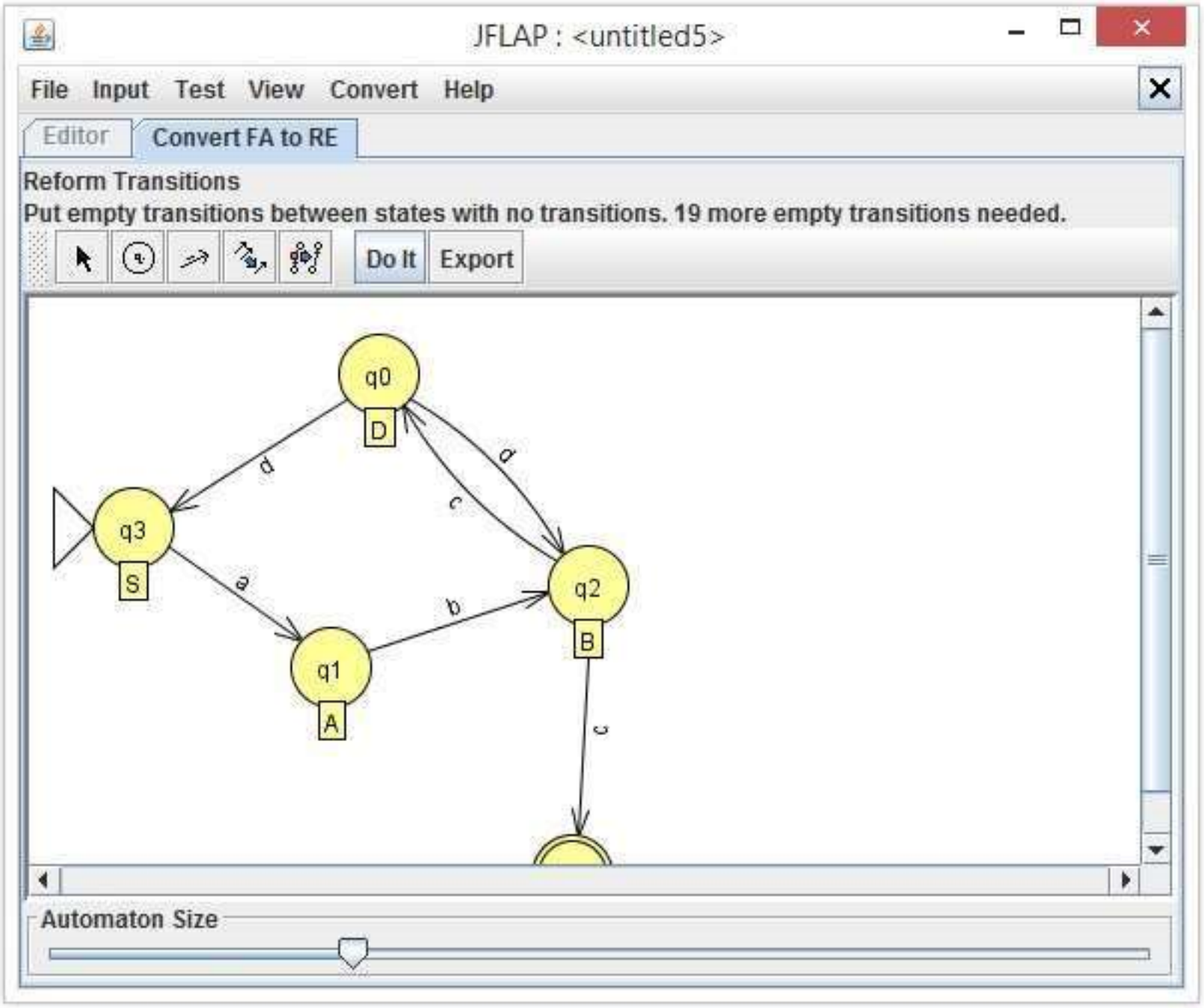

Появится окно, в котором можно будет увидеть последовательность перехода по состояниям КА.



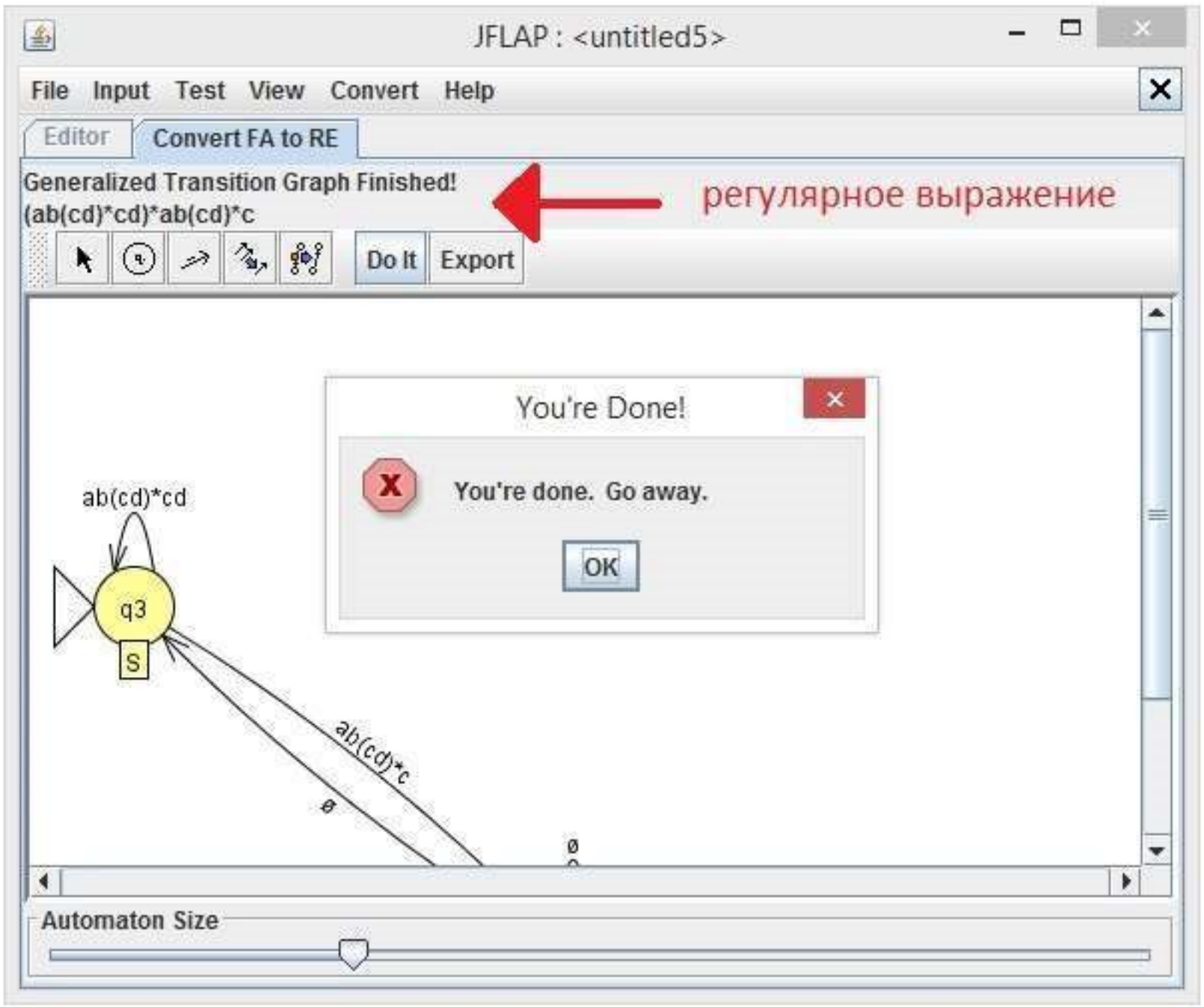

Прочитанные символы слова помечаются серым, а непрочитанные остаются чёрными. Если слово полностью прочтётся конечным автоматом (то есть все символы станут серыми), то слово допускается, а значит, и принадлежит языку КА. В заголовке текущего окна написано «Accepting configuration found!», что означает: «Найдена допускающая конфигурация», значит – слово принадлежит языку КА.

**Получение регулярного выражения для КА**

Для этого есть функция в меню “Convert” -> “Convert FA to RE” (RE – Regular Expression, что значит – регулярное выражение).




Теперь нужно нажимать кнопку “Do It”, пока не появится всплывающее окно с заголовком “You’re Done!”, что обозначит успешность операции. Всплывающее окно закроется по нажатию на “OK”, а регулярное выражение будет выведено выше кнопок управления графом.




Полученное регулярное выражение безусловно верно, но понять суть по нему трудно, а потому, зачастую регулярные выражения, составленные вручную отражают суть языка заметно проще. Например, регулярное выражение для этого языка, составленное вручную, будет выглядеть так: «(abcd)* ab(cd)*c», вместо автоматического «(ab(cd)*cd)*ab(cd)*c»

**Задание:**

1. Для заданной грамматики G = ($V_N$, $V_T$, P, S) построить 4 цепочки, принадлежащие порождаемому этой грамматикой языку L(G). Длина цепочки должна быть не меньше, чем количество символов в алфавите $V_N$ плюс 2.

2. Для каждой из этих цепочек построить дерево вывода. Два дерева вывода проверить в JFLAP.
3. Определить к какому типу грамматики по классификации Хомского относится заданная грамматика.
4. Построить конечный автомат, эквивалентный данной грамматике.
5. Написать общий вид цепочек, принадлежащих порождаемому этой грамматикой языку L(G).
6. Написать регулярное выражение **e**, эквивалентное заданной грамматике **G** и показать, что L(e) = L(G).

***Вариант 1***

$V_N$={S, D, Z}, $V_T$={a, b},

P= {1. S → aS

2. S → bS

3. S → aD

4. D → bZ

5. D→ b

6. Z→ aZ

7. Z → a}

***Вариант 2***

$V_N$={A, B, C, S}, $V_T$={a,d,n,f},

P= {1. S → aB

2. S → aC

3. B → aC

4. C → nC

5. C → fA

6. A → dA

7. C → f

8. A → d}

***Вариант 3***

$V_N$={S, B}, $V_T$={a, b,c,d},

P= {1. S → bS

2. S → aS

3. S → bB

4. B → cB

5. B → dB

6. B → c}

***Вариант 4***

$V_N$={S, D}, $V_T$={a, b,c},

P= {

1. S → aS

2. S → bS

3. S → cS

4. S → bD

5. S → b

6. D → bD}

***Вариант 5***

$V_N$={S, C,D}, $V_T$={a, b},

P= {1. S → aD

2. D → bS

3. D → bC

4. D → b

5. C → aC

6. C → a}

***Вариант 6***

$V_N$={S, D, A}, $V_T$={a, b, c}

P= {1. S → aA

2. A → bD

3. D → cA

4. D → cD

5. D → c

6. A → b}

***Вариант 7***

$V_N$={S, D}, $V_T$={a, b, e, f}

P= {1. S → aD

2. D → bS

3. S → a

4. D → eD

5. D → e

6. D → fD

7. D → f}

***Вариант 8***

$V_N$={S, A, B}, $V_T$={a, b, c},

P= {1. S → aA

2. A → bB

3. A → b

4. B → cB

5. B → c

6. B → cS}

***Вариант 9***

$V_N$={S, A, B}, $V_T$={a,b},

P= {1. S → aA

2. A → bB

3. B → bB

4. B → a

5. S → a}

***Вариант 10***

$V_N$={S, A, D, B}, $V_T$={a, b, c},

P= {1. S → aA

2. S → aB

3. A → bS

4. A → bB

5. B → cD

6. D → aD

7. D → a}

***Вариант 11***

$V_N$={S, A}, $V_T$={a, b},

P= {1. S → aS

2. S → aA

3. A → bA

4. A → b

5. A → bS}

***Вариант 12***

$V_N$={S, B, A}, $V_T$={a, b, c},

P= {1. S → aB

2. B → bB

3. B → b

4. S → a

5. B → bA

6. A → cS}

***Вариант 13***

$V_N$={S, B,D, A}, $V_T$={a, b, c, e},

P= {1. S → aA

2. S → aB

3. A → b

4. A → bD

5. B → c

6. B → cD}

***Вариант 14***

$V_N$={S, A, D, B}, $V_T$={a, b, c, d},

P= {1. S → aA

2. A → bS

3. S → aB

4. B → cD

5. B → c

6. D → cD

7. D → c

8. D → dB}

***Вариант 15***

$V_N$={S, B, A}, $V_T$={a, b},

P= {1. S → aA

2. A → bS

3. A → bB

4. B → aB

5. B → a

6. A → b}

***Вариант 16***

$V_N$={S, A, B }, $V_T$={a, b, c},

P= {1. S → aB

2. B → bA

3. S → c

4. A → cA

5. A → cS

6. B → bB

7. B → b}

***Вариант 17***

$V_N$={S, A, B }, $V_T$={a, b, c},

P= {1. S → aS

2. S → aA

3. A → bA

4. A → bB

5. B → cB

6. B → c

7. A → b}

***Вариант 18***

$V_N$={S, A, B}, $V_T$={a, b, c},

P= {1. S → aA

2. A → bA

3. A → bB

4. B → cS

5. A → b

6. B → bB

7. B → b}

***Вариант 19***

$V_N$={S, A, B}, $V_T$={a, b,c},

P= {1. S → aA

2. A → bA

3. A → cA

4. A → bB

5. A → b

6. B → cB

7. B → c}

***Вариант 20***

$V_N$={S, A, B}, $V_T$={a, b},

P= {1. S → aA

2. A → aB

3. A → a

4. B → bA

5. B → bB

6. B → b}

***Вариант 21***

$V_N$={S, A, B, D}, $V_T$={a, b, c}

P= {1. S → aA

2. A → bB

3. B → c

4. B → cD

5. B → cA

6. D → aD

7. D → a}

***Вариант 22***

$V_N$={S, A, B}, $V_T$={a, b}

P= {1. S → aA

2. A → a

3. A → aB

4. B → bB

5. B → b

6. B → bA}

***Вариант 23***

$V_N$={S, A, B, C}, $V_T$={a, b},

P= {1. S → aA

2. S → aC

3. A → bB

4. B → bC

5. B → b

6. C → aB}

***Вариант 24***

$V_N$={S, A, B}, $V_T$={a, c, d},

P= {1. S → aA

2. A → aA

3. A → a

4. A → aB

5. B → cA

6. A → dS}

***Вариант 25***

$V_N$={S, B, D, A}, $V_T$={a, b, c},

P= {1. S → aA

2. S → aB

3. A → aB

4. B → b

5. B → bD

6. D → cB}

***Вариант 26***

$V_N$={S, B, A}, $V_T$={a, b, c},

P= {1. S → aA

2. A → bS

3. A → bB

4. A → b

5. B → cS}

***Вариант 27***

$V_N$={S, B, D, A}, $V_T$={a, b, c},

P= {1. S → aB

2. B → bA

3. A → cB

4. B → bD

5. D → aD

6. D → a

7. B → cB}

***Вариант 28***

$V_N$={S, B, A}, $V_T$={a, b, c, d},

P= {1. S → aA

2. A → bS

3. A → bB

4. A → b

5. B → dB

6. B → d

7. B → cB

8. B → c}

***Вариант 29***

$V_N$={S, A, B}, $V_T$={a, b, c},

P= {1. S → aB

2. B → b

3. B → bB

4. B → bA

5. A → cB

6. S → a}

***Вариант 30***

$V_N$={S, B, D, A}, $V_T$={a, b, c, d},

P= {1. S → aS

2. S → aA

3. A → bB

4. B → cD

5. B → c

6. D → dA}