Artigo sobre Linguagens Formais em JFLAP

Nome: Luiz Henrique Botega Beraldi

Professora: Cinthyan Renata Sachs Camerlengo de Barbosa

O intuito do artigo é comparar as resoluções e ajudas do JFLAP com os slides e aulas ministradas pela professora Cinthyan.

DEFINIÇÕES

Alfabeto: O JFLAP reconhece o alfabeto por meio de letras minúsculas e dígitos para Terminais, e letras maiúsculas para Não-Terminais.

Palavra: O JFLAP reconhece uma sequência finita de símbolos do alfabeto justapostos como uma palavra.

Concatenação de frases/cadeias: O JFLAP reconhece as propriedades de concatenação como ministradas nos slides.

Concatenação sucessiva: O JFLAP consegue fazer uma concatenação sucessiva, tendo apenas que o usuário colocar a palavra e quantas n-vezes ela tem que se repetir.

Linguagem formal: O JFLAP entende que uma linguagem formal L é um conjunto de palavras sobre um alfabeto.

Sintaxe: O JFLAP permite inserirmos a sintaxe da linguagem, assim, delimitar o subconjunto V* com o conjunto de regras.

Leis de formação: O JFLAP permite inserirmos as leis de formação de uma gramática para definir a linguagem, colocando um Não-Terminal à esquerda e oque ele gera à sua direita, conseguimos criar inúmeras leis de formação, que chamamos de produção da gramática.

Derivação: O JFLAP permite derivarmos a nossa gramática da forma que desejarmos, no automático ou escolhendo por onde iremos derivar.

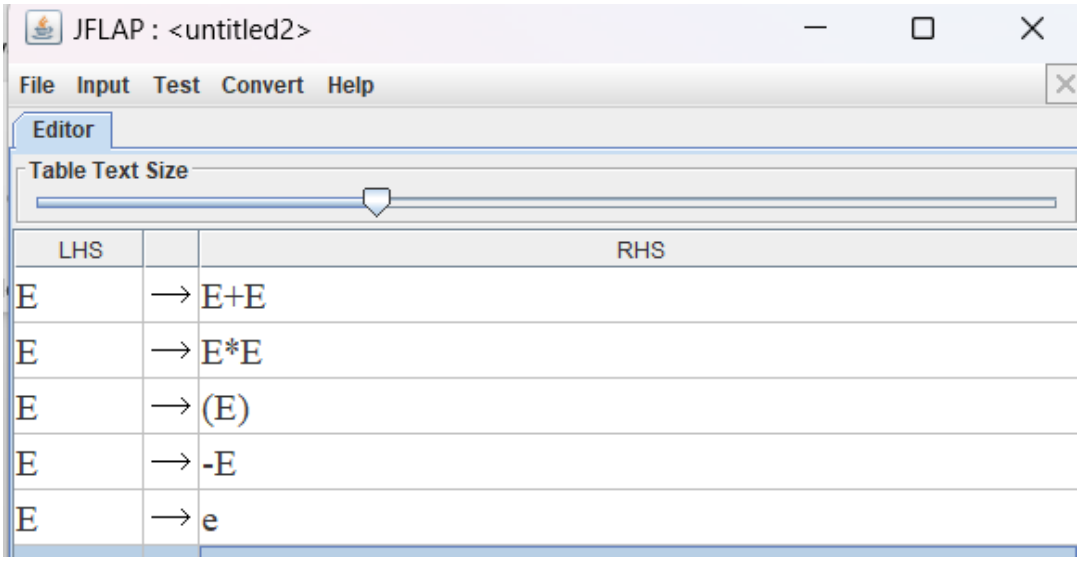
Linguagem gerada por uma gramática: O JFLAP permite que nós insiramos uma palavra e conferimos se ela faz parte ou não daquela

gramática, porém, mesmo que nós colocamos as leis de formação, ele não nos mostra a linguagem gerada e nem todas as palavras que aquela gramática pode gerar.

ENTER GRAMMAR

O JFLAP permite analisar e inserir gramáticas irrestritas e livres de contexto.

Se montarmos uma gramática do slide 3 (exemplo 1.15):



O programa mostra que a variável inicial está na primeira linha do programa, ele reconhece letras maiúsculas como Não-Terminais e letras minúsculas e números como Terminais. E λ é a cadeia vazia.

TYPE GRAMMAR

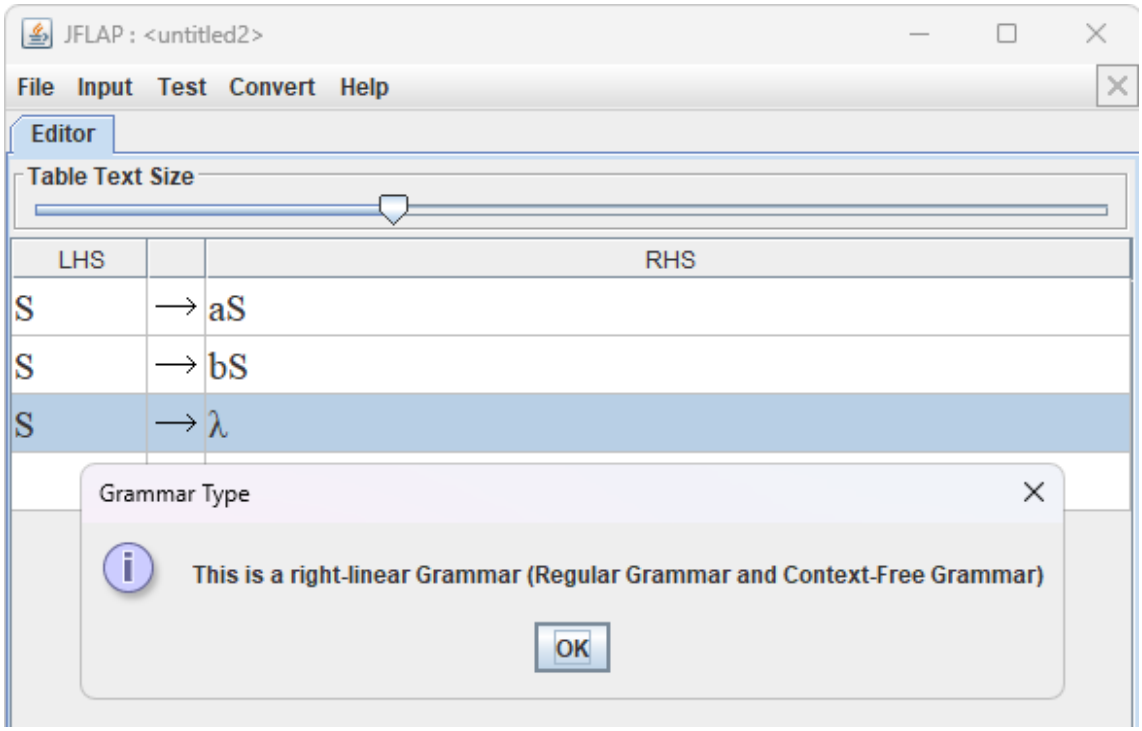
Ele nos permite colocar uma gramática e verificar de qual tipo ela é.

Ao tentar usar a gramática anterior ele não compila, simplesmente não aparecendo nada na tela, o que nos mostra que talvez ele não compreenda símbolos como “+”, “*”, etc. Achei que o problema poderia ser na última linha, já que talvez ele não reconhecesse “num” como um terminal, mas mesmo alterando para “e”, ainda não funcionou. Trata-se de uma gramática ambígua, Livre de Contexto, ou seja, existe uma palavra que possui duas ou mais árvores de derivação.

Se colocarmos a gramática do exemplo do slide 2 (exemplo 1.13), onde:

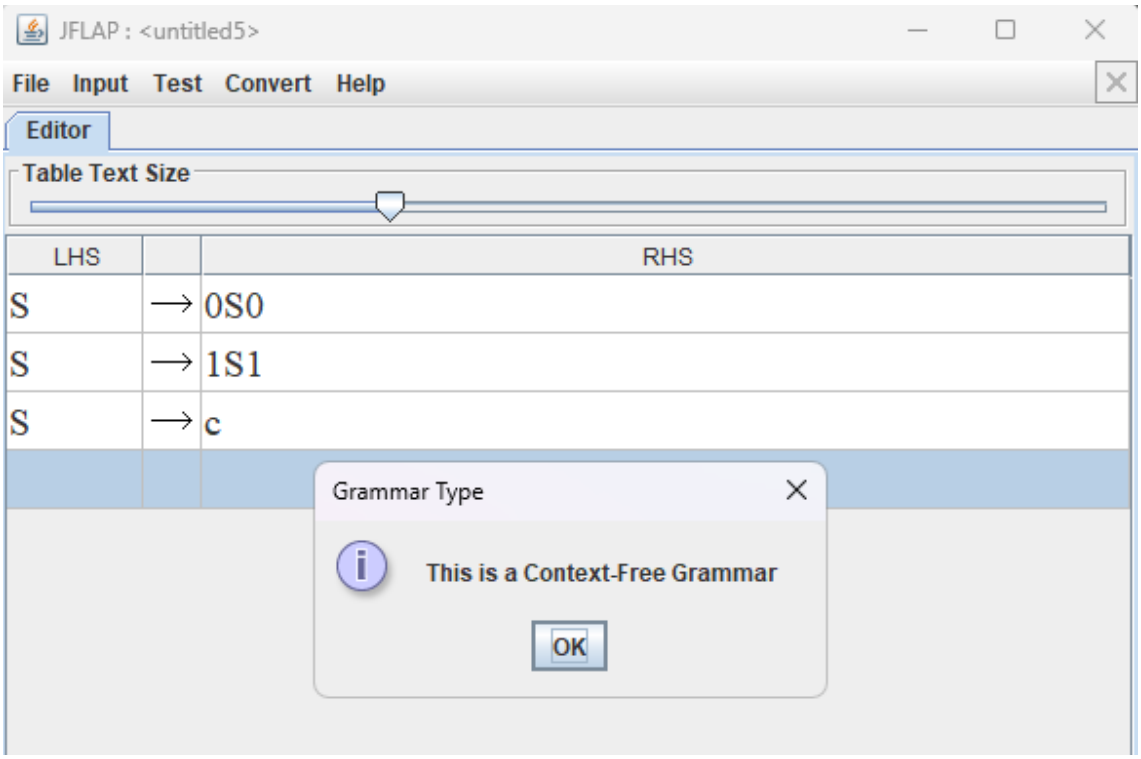
S → aS | bS | λ

Esta gramática representa a linguagem $a^n b^m$, onde n ≥ 0 e m ≥ 0.



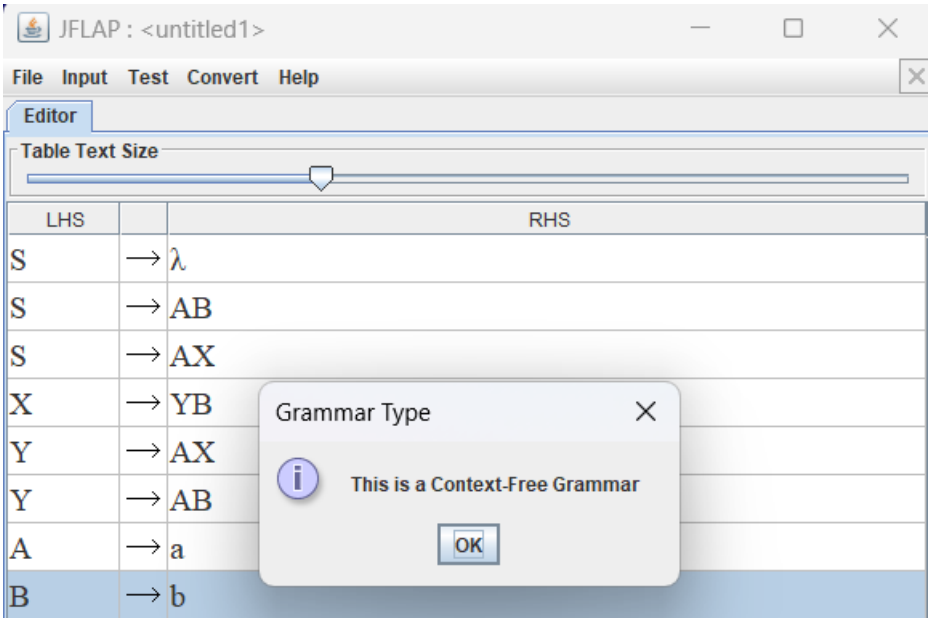
O programa nos mostra que é uma Gramática Regular e Livre de Contexto, além de que ela é uma Gramática linear à direita.

Agora para mostrar que o programa reconhece números como Não-Terminais, irei utilizar o slide 4, onde temos o exercício c), onde a gramática é L(G3) = {wcw^t/ w está em {0,1} *}.



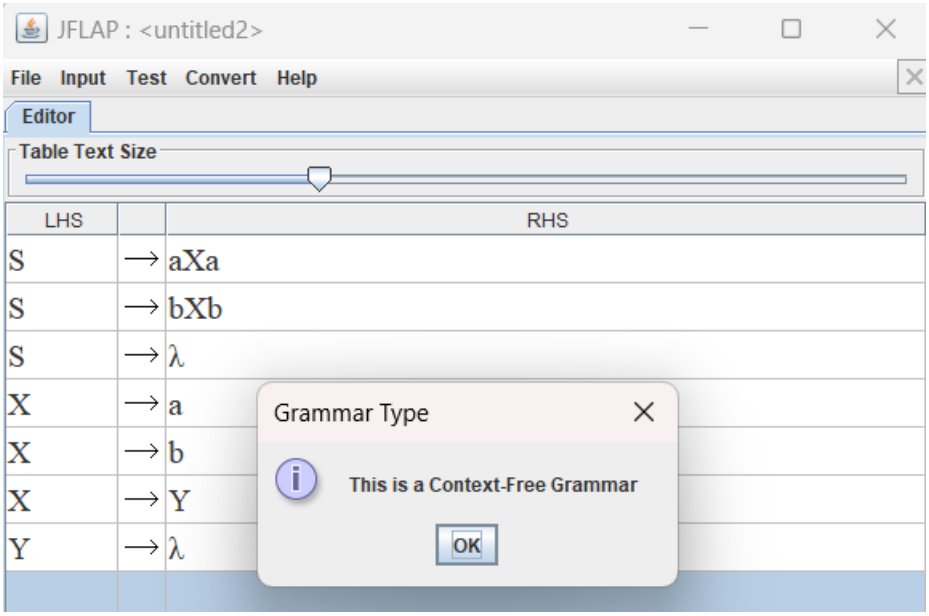
O programa nos indica que a gramática acima é uma Livre de Contexto e nos mostra que ele compreende números como Não-Terminais.

Pegando mais uma gramática como exemplo, agora o exemplo 1.19 do slide 5, onde S → λ | AB | AX, X → YB, Y → AX | AB, A → a, B → b.



O JFLAP nos mostra que é uma Gramática Livre de Contexto assim como diz o slide.

Outro exemplo que podemos usar é o exemplo 1.21 do slide 5, onde S → aXa | bXb | λ, X → a | b | Y, Y → λ.

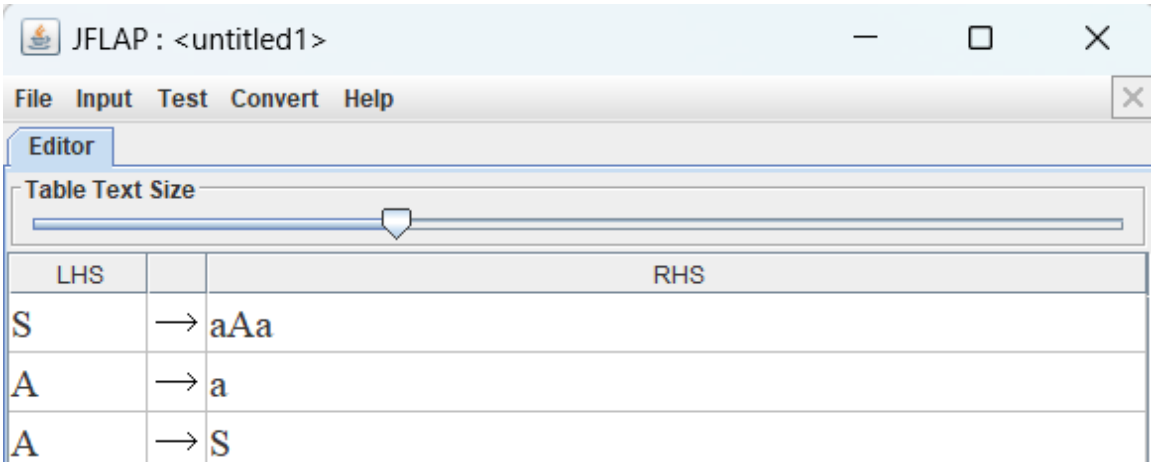


Novamente, o JFLAP nos mostra que a gramática inserida é uma Gramática Livre de Contexto.

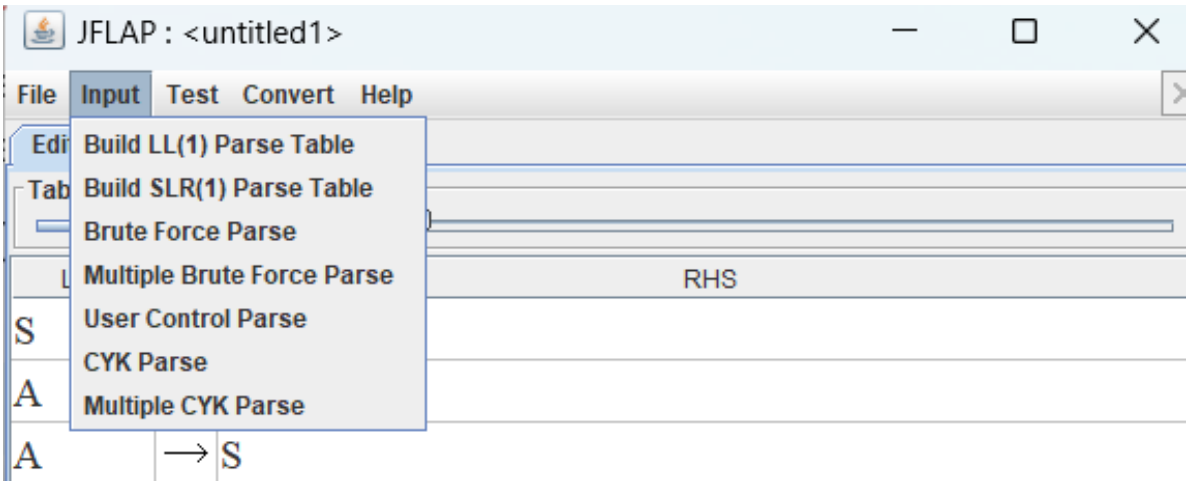
BRUTE FORCE PARSER – REGULAR OR CFG(context-free grammars)

O programa nos permite também colocar uma sequência de caracteres e conferir se faz parte ou não da linguagem da gramática.

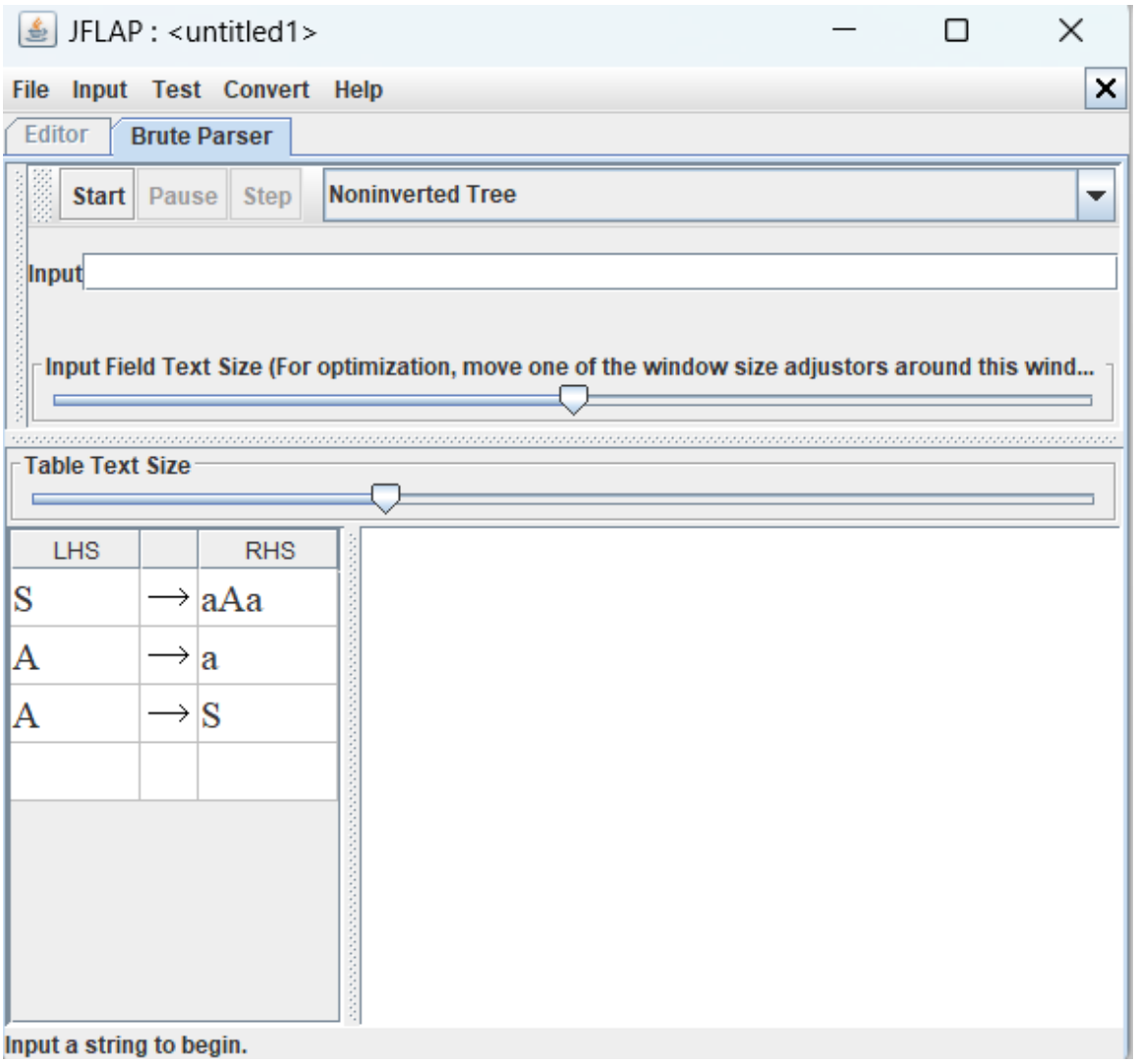
Vou pegar como exemplo a gramática G = ({S,A},{a},P,S), onde P = {S → aAa, A → a / S} que é o exemplo 1.20 do slide 5.



Agora clicamos em Input

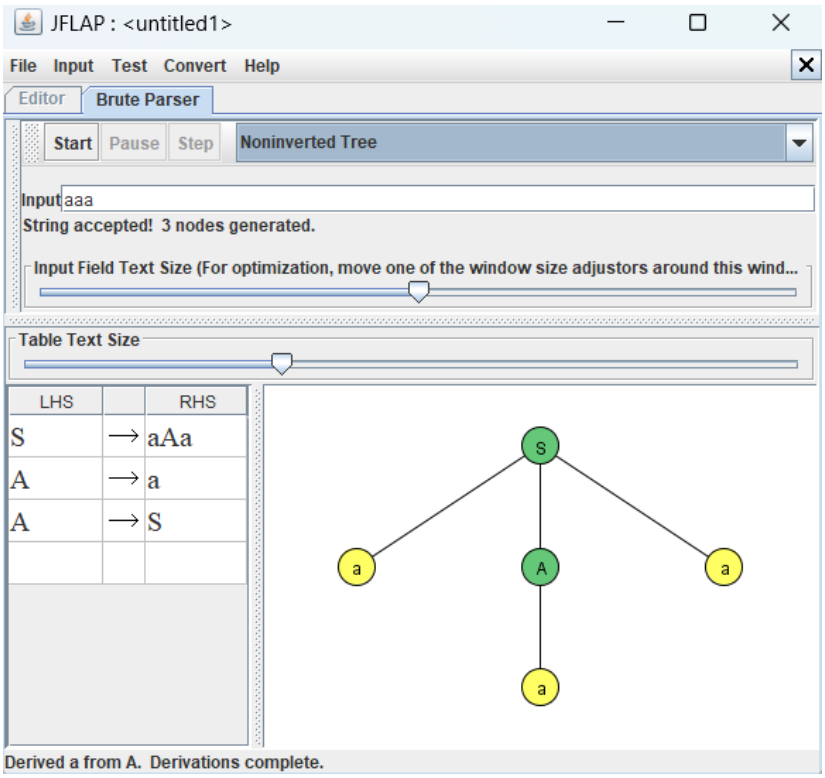


E selecionamos Brute Force Parse, o programa irá ficar assim



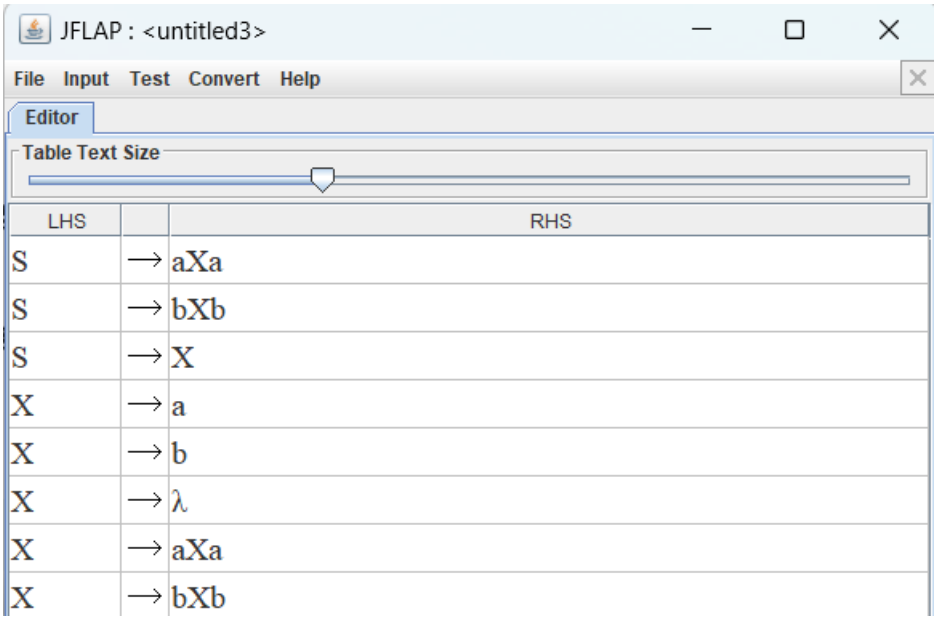
Em Input, podemos digitar os caracteres e dando enter, assim ele nos mostra se o que digitamos está ou não na gramática.

Como exemplo, digitei “aaa” e é isso que ele nos mostra

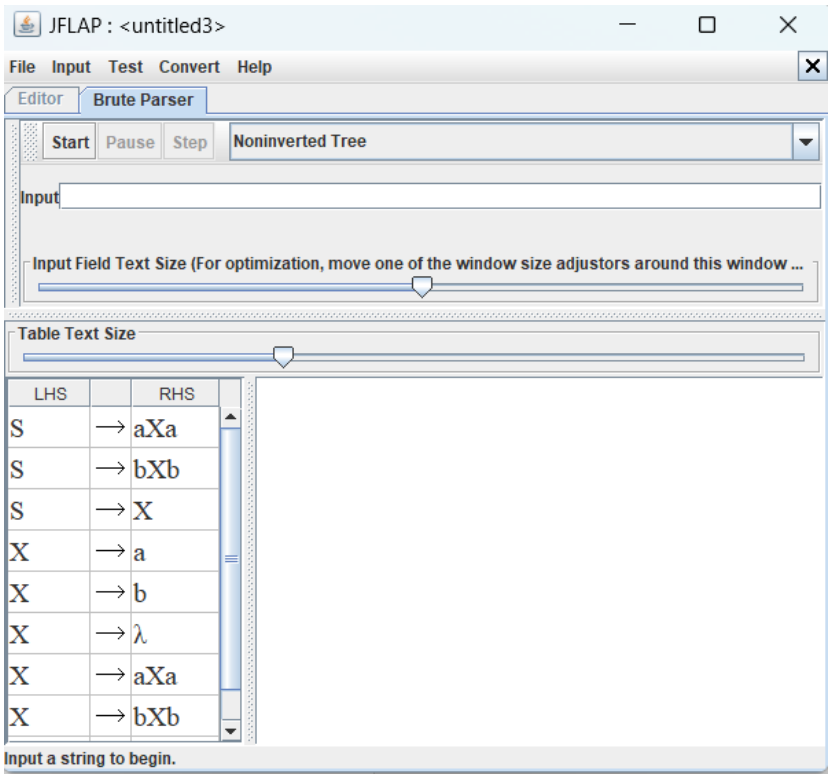


O programa nos diz que “aaa” está na gramática e ainda nos mostra a árvore de derivação para chegar até “aaa”.

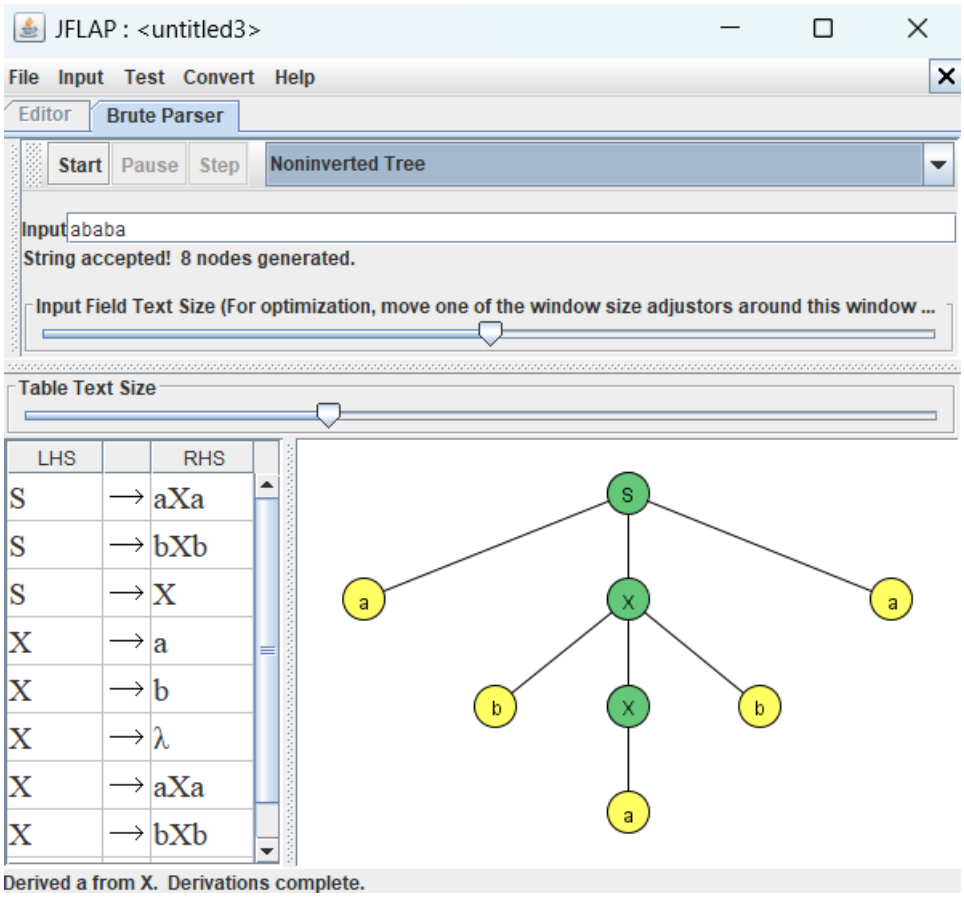
Pegando mais um exemplo, agora o exemplo 1.22 do slide 6, onde G = ({S,X},{a,b},P,S) e P = {S → aXa | bXB | X, X → a | b | λ | aXa | bXb.



Fazendo os mesmos passos demonstrados antes, clicando em Input e depois em Brute Force Parse, aparece essa tela:



Agora colocando a palavra “ababa” em Input e clicando em step, iremos obter a árvore de derivação e ele nos diz se a palavra está ou não na gramática que inserimos anteriormente.

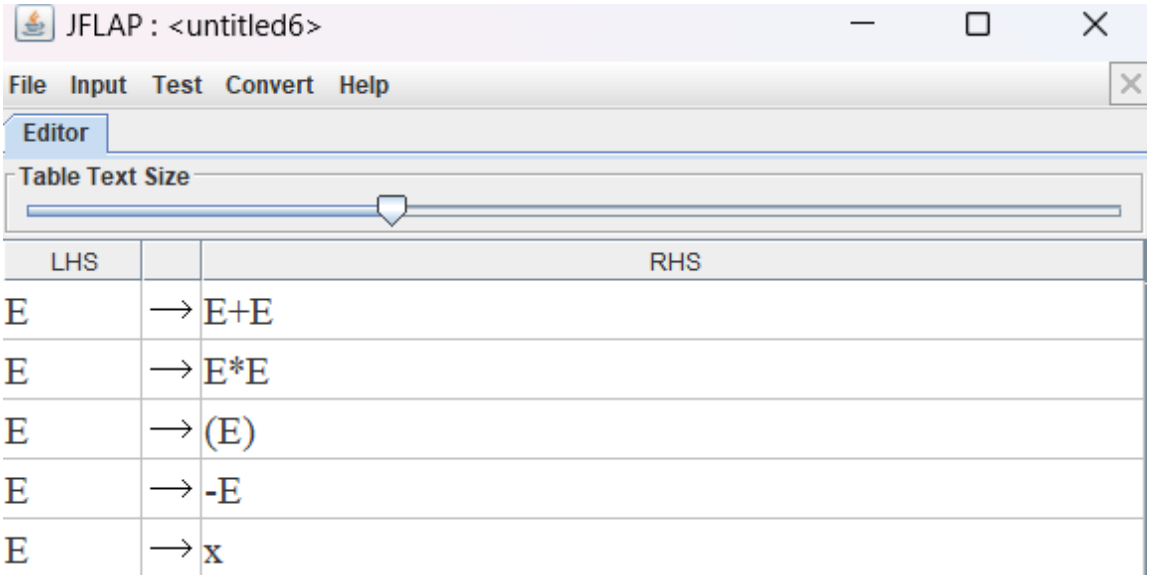


Ele nos mostra a árvore de derivação e diz que a palavra “ababa” está na gramática que havíamos inserido.

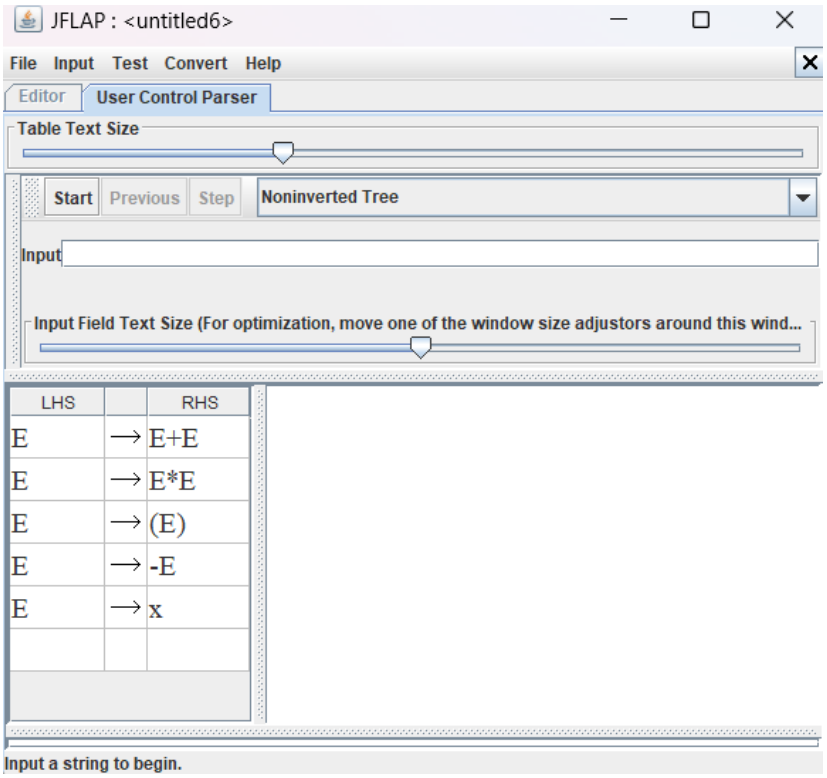
USER CONTROL PARSE

O programa também nos permite escolher o que vamos derivar, nessa opção nos podemos colocar nossa gramática, escrever a sequência de caracteres que nós queremos e verificarmos se derivando, essa sequência de caracteres pertence ou não à gramática, e ainda escolhemos o que vamos derivar.

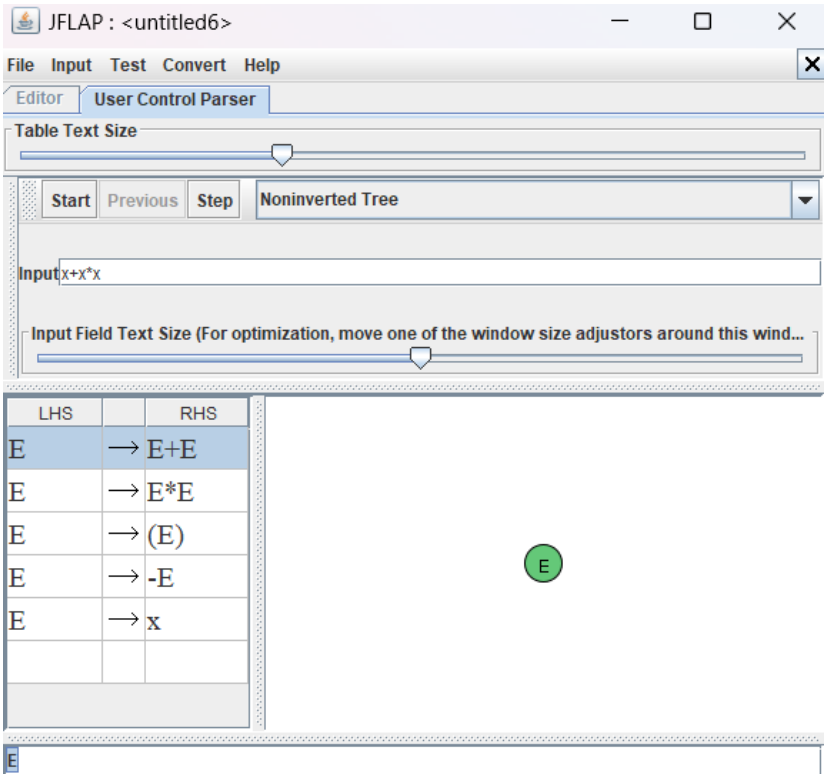
Para esse exemplo escolhi a gramática E → E+E / E*E/ (E) / -E / x, que é o exemplo 1.15 do slide 3.



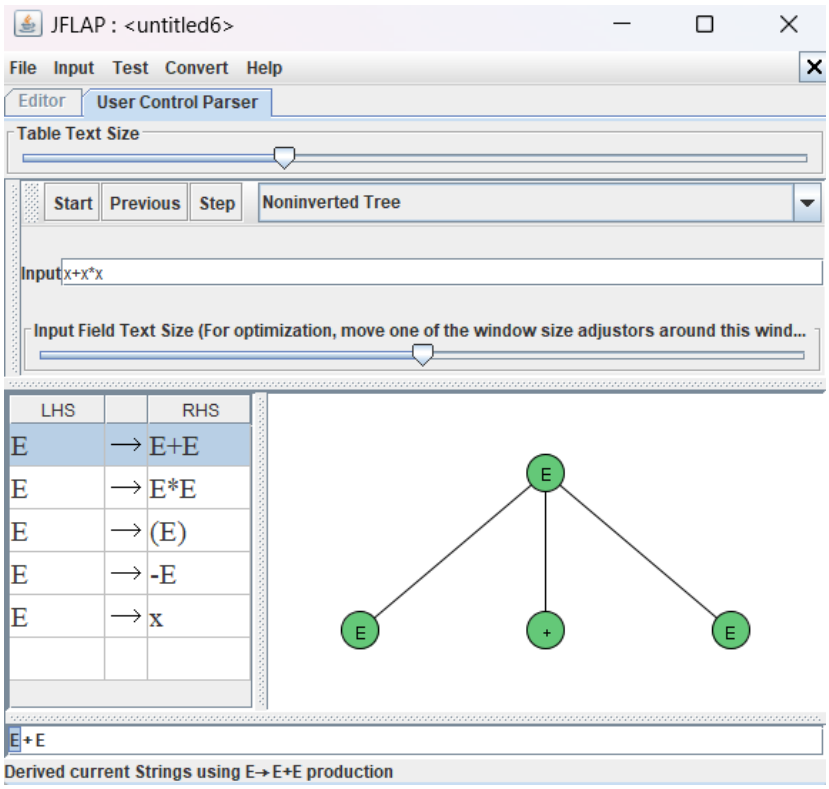
Agora clicando em Input e em User Control Parse, irá abrir o seguinte menu



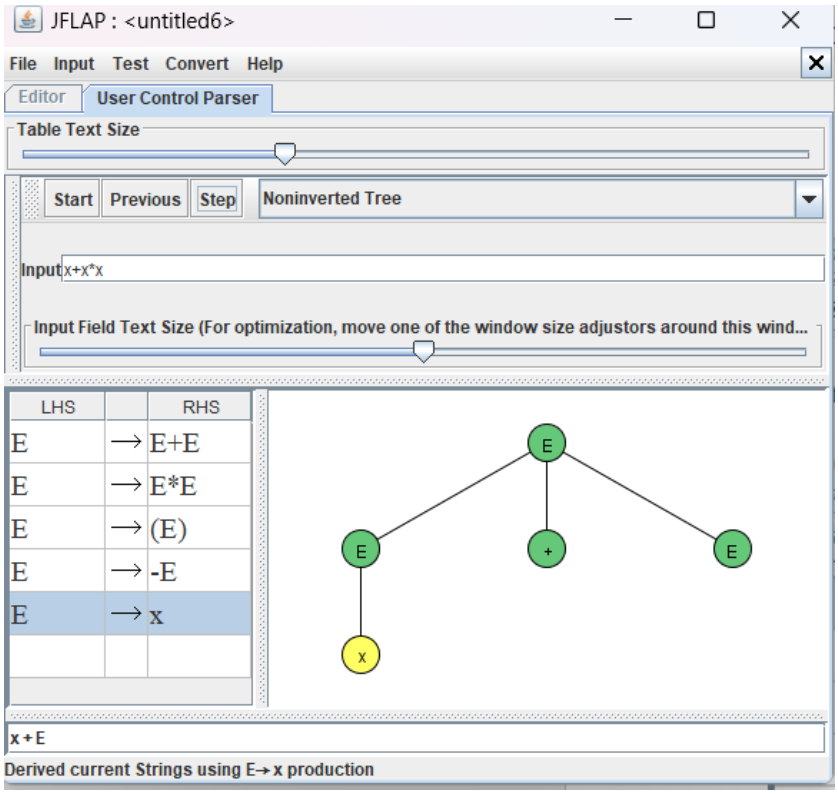
Aqui, em Input colocamos a palavra que queremos verificar se está na nossa gramática e apertamos em Start, irei colocar “x+x*x”.



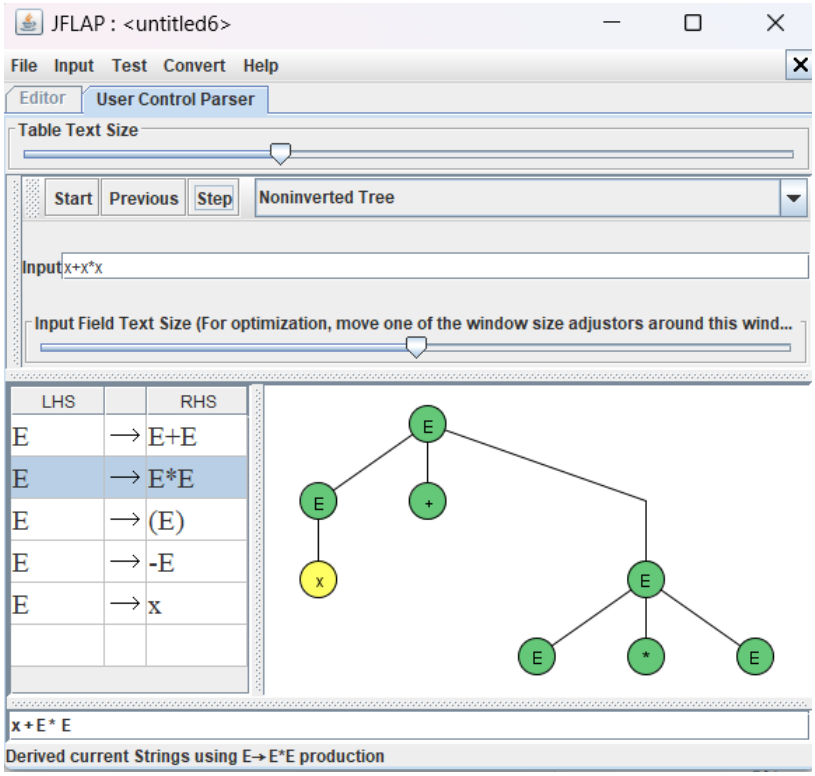
Agora iremos selecionar o Não-Terminal E que ali embaixo e selecionaremos a derivação que iremos fazer nele, irei selecionar E → E+E.



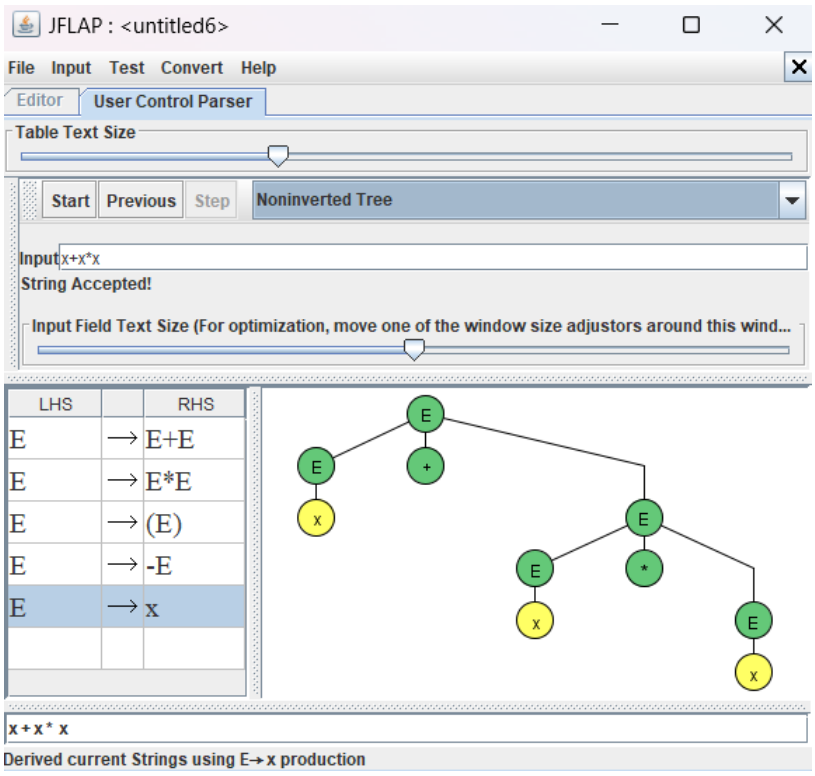
Agora podemos selecionar por qual Não-Terminal iremos começar, o mais à direita ou o mais à esquerda, nesse caso não muda nada já que essa gramática em específico é ambígua e podemos seguir vários caminhos para chegar no mesmo lugar. Irei selecionar o E mais à esquerda e derivá-lo em E → x.



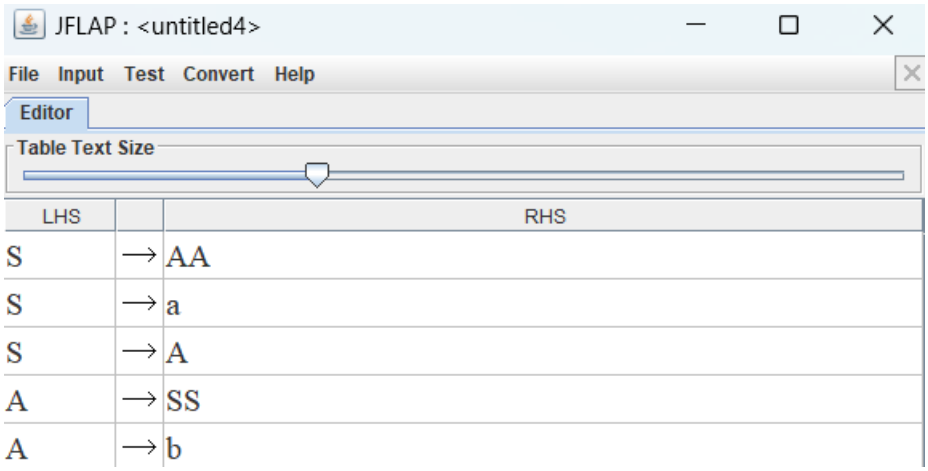
Agora o E mais à direita, e irei derivar ele em E → E*E.



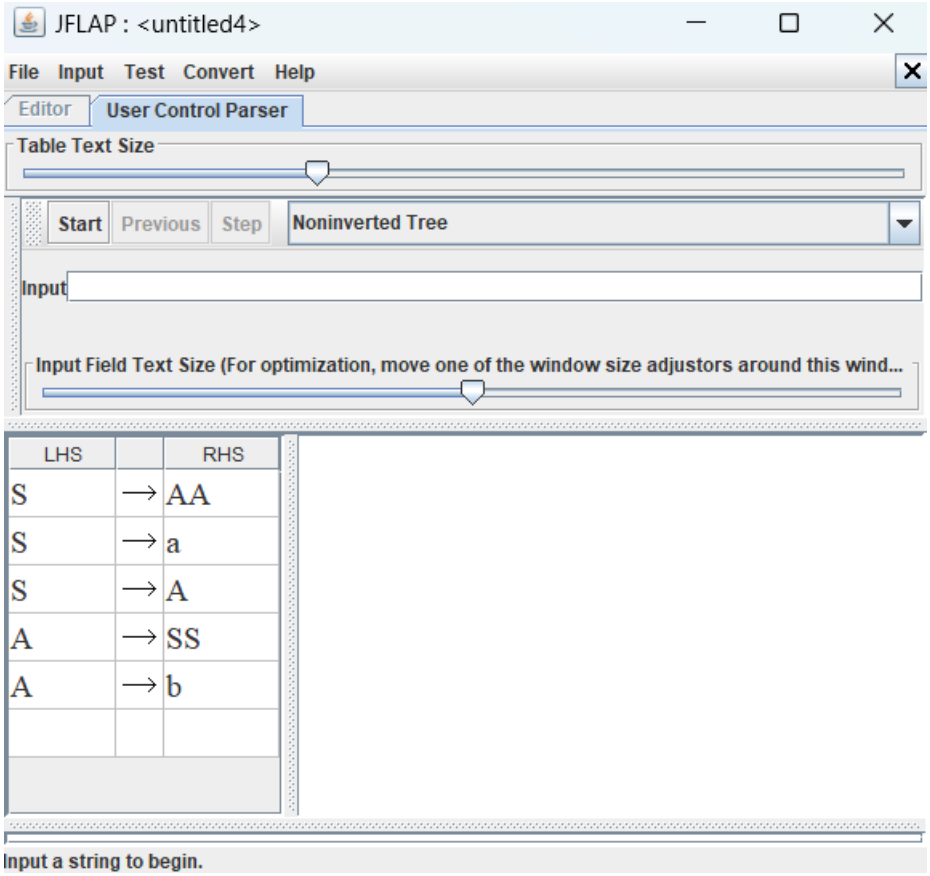
Irei derivar os dois E Não-Terminais em x e assim iremos obter “x+x*x”.



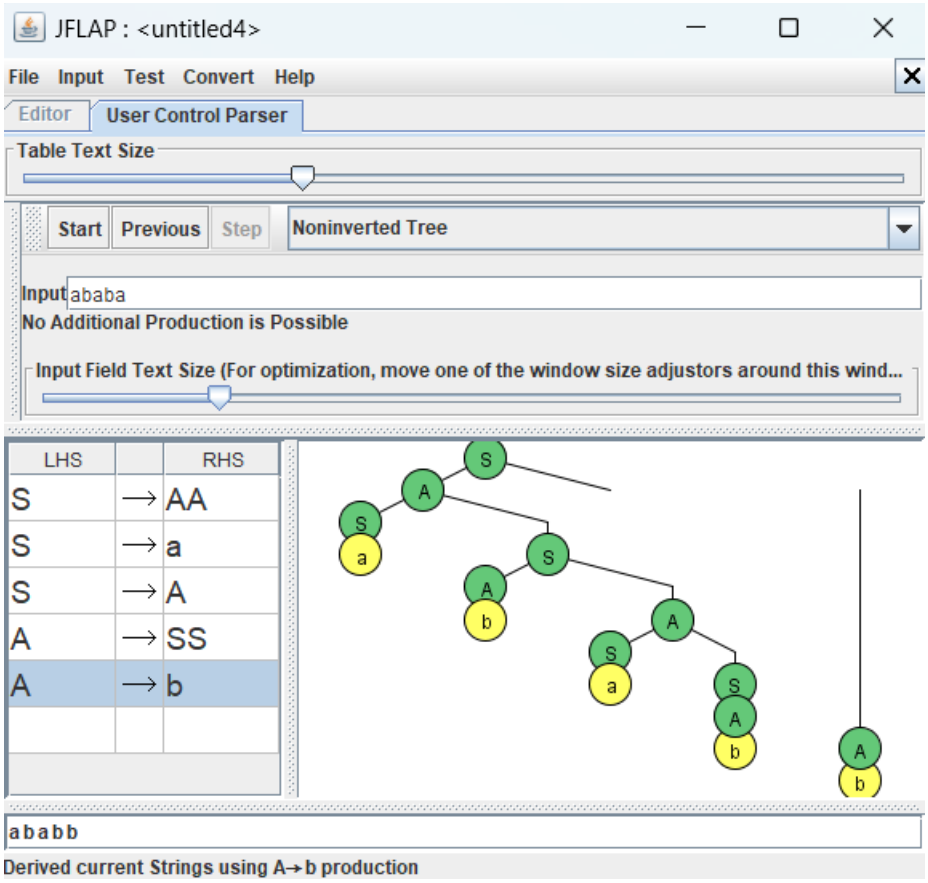
Pegando mais um exemplo, agora o exemplo 1.23 do slide 8, onde G = ({S,A},{a,a},P,S) e P = {S → AA | a, A → SS | b}, colocando a gramática no JFLAP fica:



Agora clicando em Input e em User Control Parse, irá abrir o seguinte menu



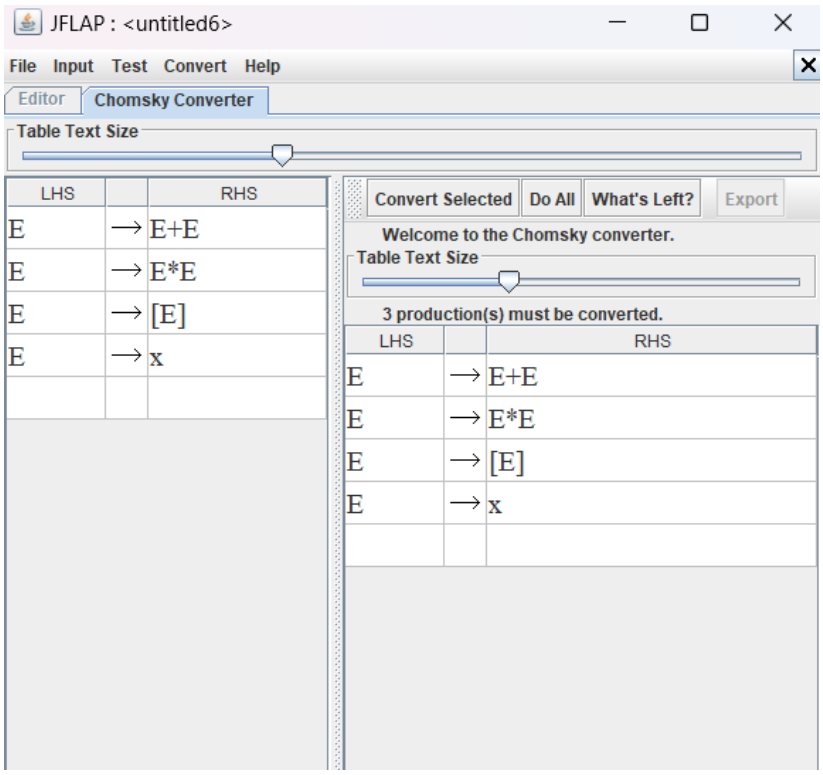
Agora, colocando a palavra “ababb” em Input, podemos verificar que ela está na gramática, e agora podemos escolher como derivá-la, que eu já fiz e irei colocar diretamente o resultado final aqui:



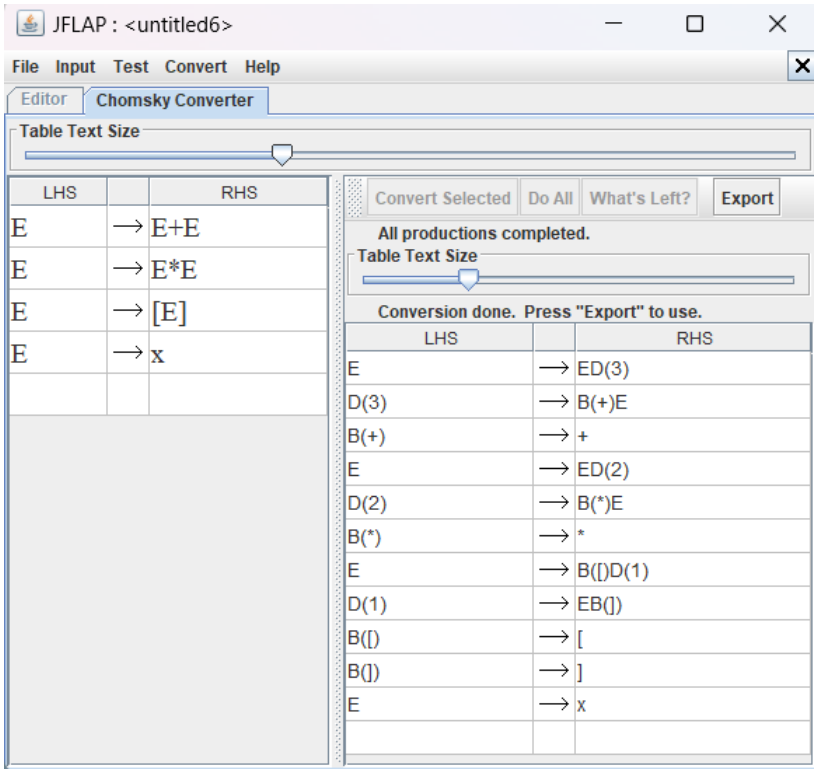
TRANSFORM GRAMMAR

Nessa parte do JFLAP é onde transformamos gramáticas na Forma Nominal de Chomsky (FNC). O JFLAP primeiramente remove as produções de lambda e depois remove as produções inúteis.

Irei utilizar como exemplo o exercício do slide 7, onde G = ({E},{+,*,[,],x},P,E) e E → E+E / E*E / [E] / x, agora clicamos em Convert e em Transform Grammar.



Podemos selecionar as instruções que iremos fazer isso ou fazer todas de uma vez, irei fazer todas de uma vez clicando em “Do All”, com o JFLAP ficando assim:

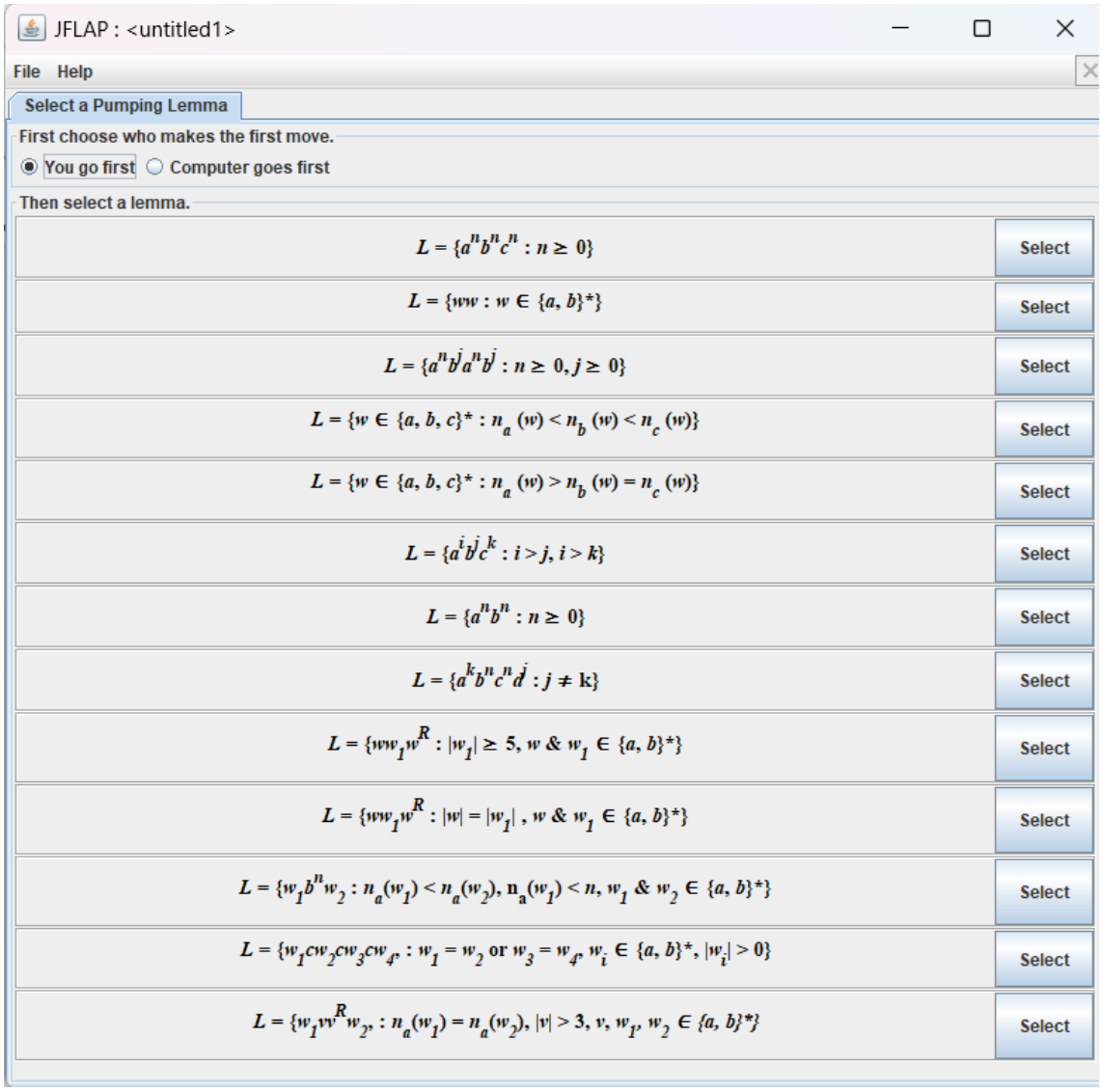


Ele mostra cada transformação que foi feita e onde ela ocorre, o resultado do JFLAP

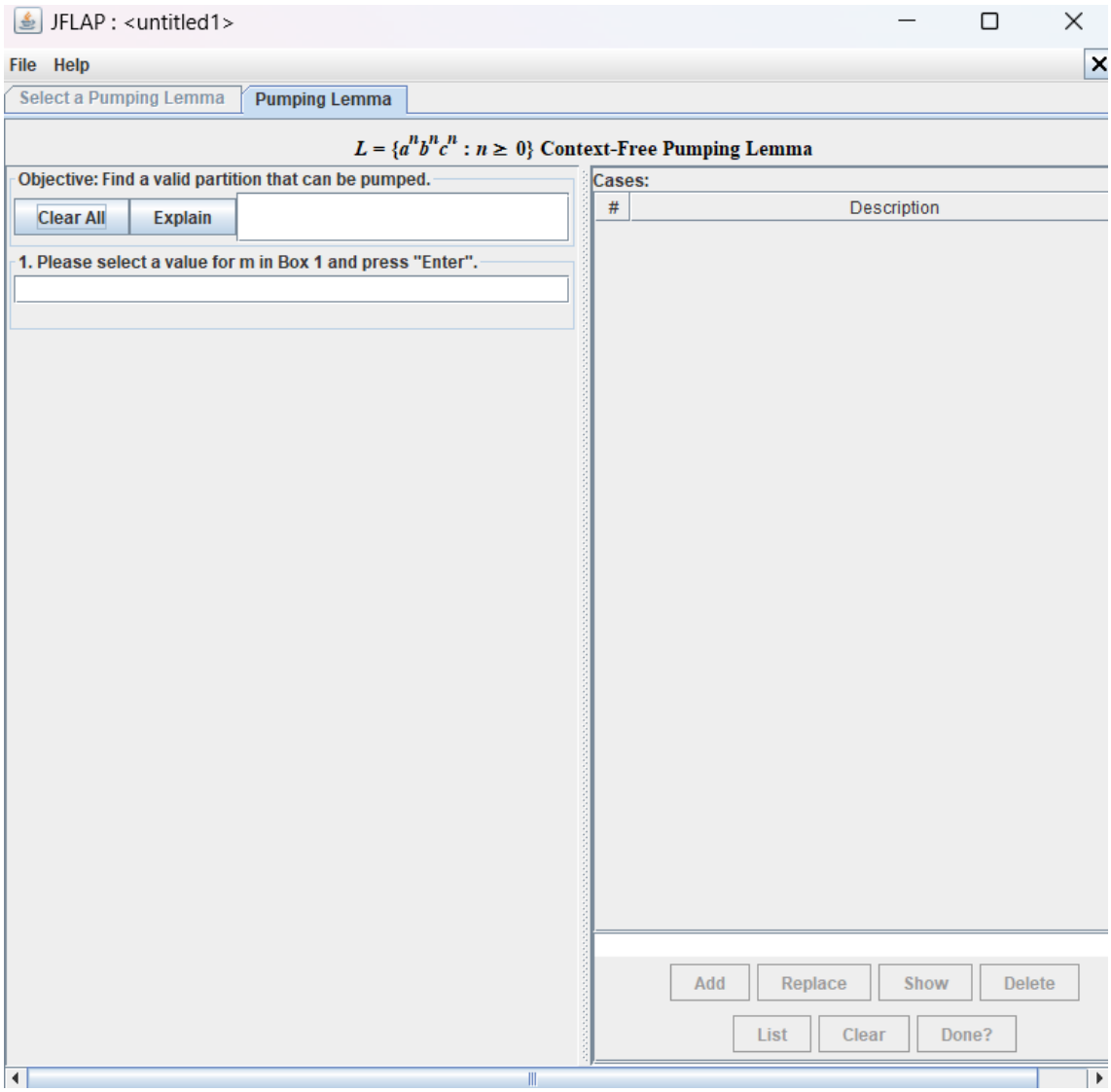
é o mesmo do slide, mostrando que está correto e é uma maneira confiável de testar a Forma Nominal de Chomsky.

CONTEXT-FREE PUMPING LEMMA

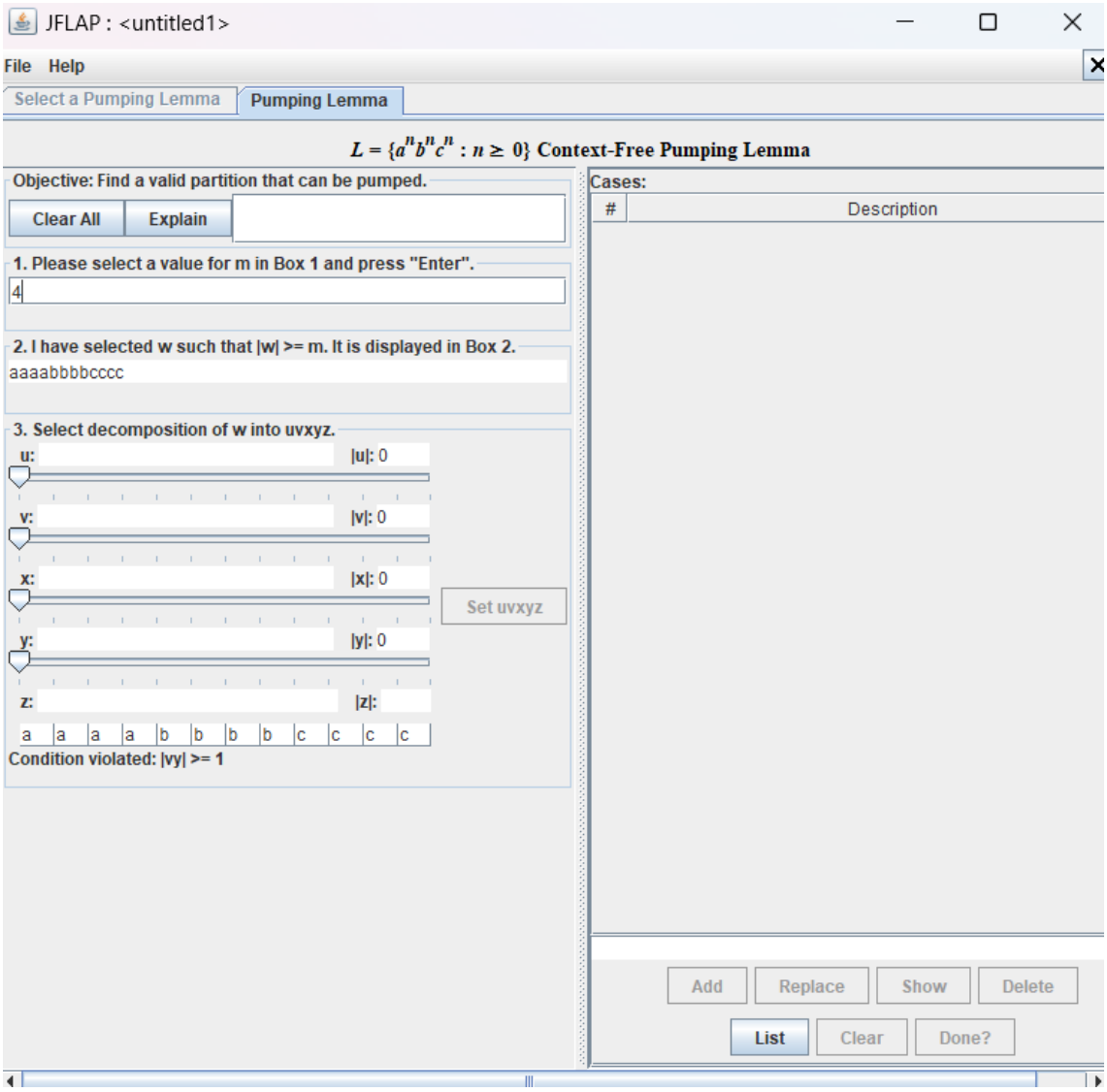
O JFLAP é capaz de fazer bombeamentos tanto para o teste da gramática livre de contexto quanto para gramáticas regulares, porém só é possível testar as linguagens fornecidas pelo software.



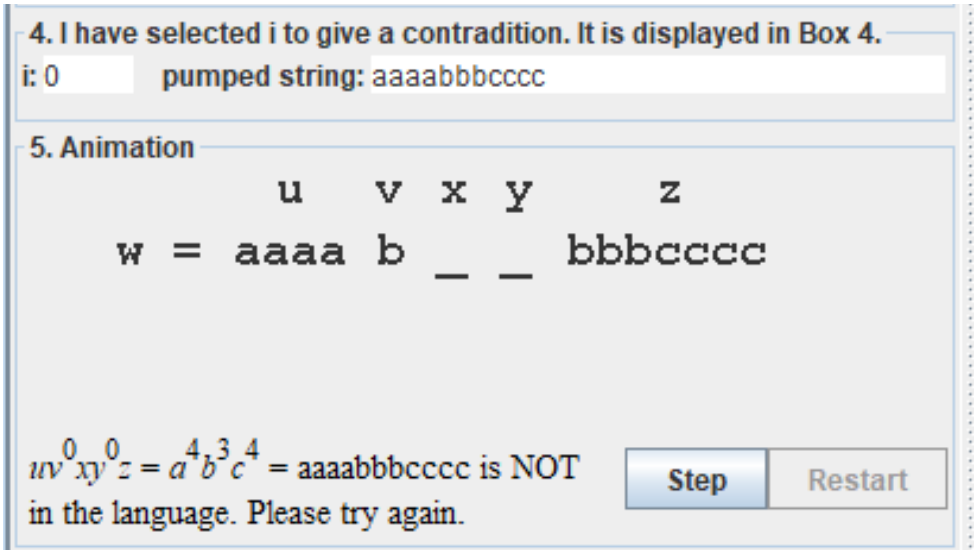
Essas são as linguagens fornecidas para os testes de bombeamento para gramáticas livres de contexto, o software também nos fornece a opção de irmos primeiro, ou seja, nós entramos com o tamanho mínimo da palavra w a ser testada e fazemos a divisão das variáveis UVXYZ.



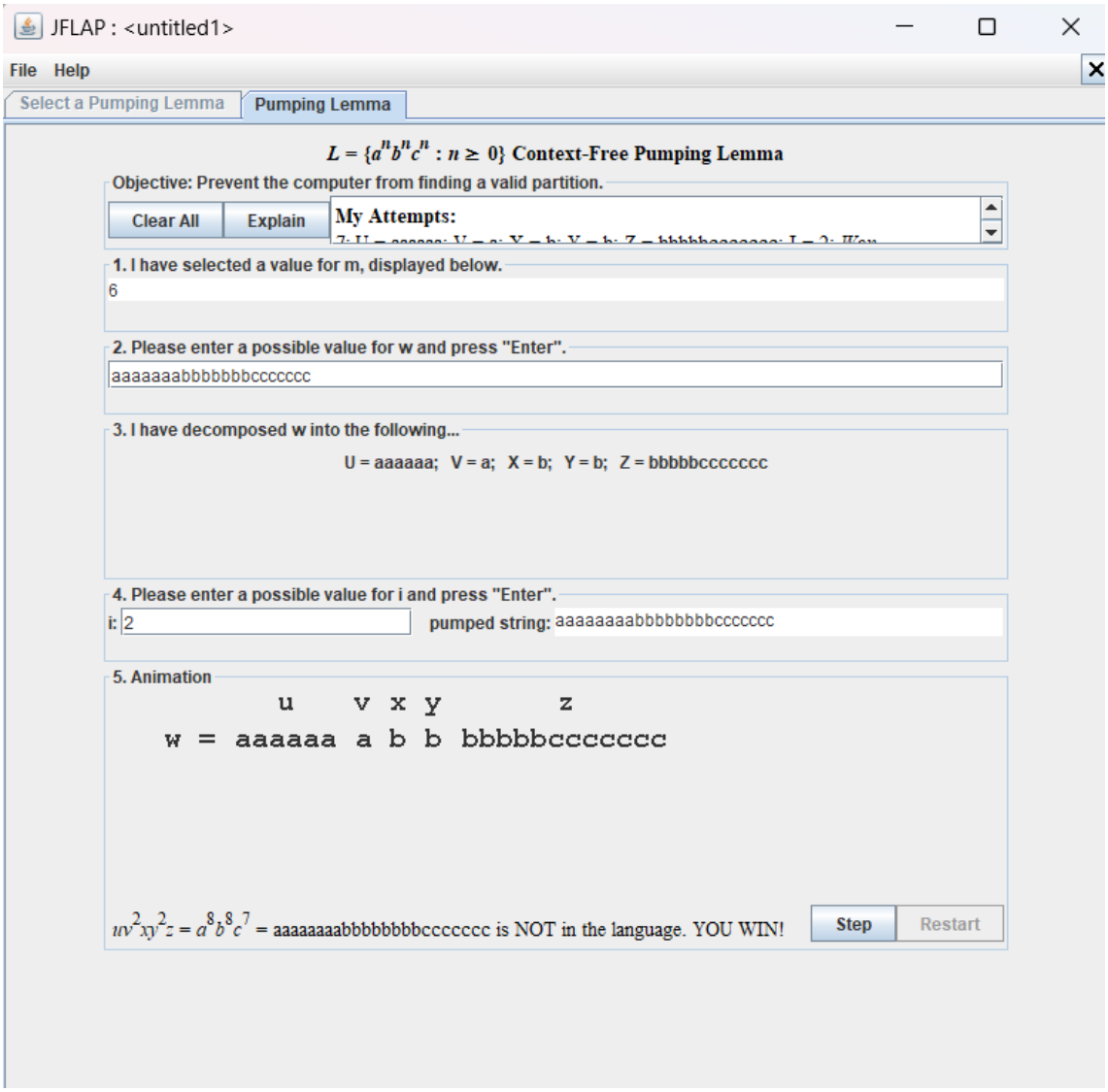
Com a opção de o usuário agir primeiro habilitada e a linguagem L = $\{a^n b^n c^n$: n ≥ 0} selecionada, devemos agora selecionar um m tal que |w| ≥ m.



Após selecionar m, o computador nos dá uma palavra formada, então devemos escolher nossas variáveis UVXYZ tal que |VXY| ≤ m, e após setadas as variáveis temos a seguinte tela:

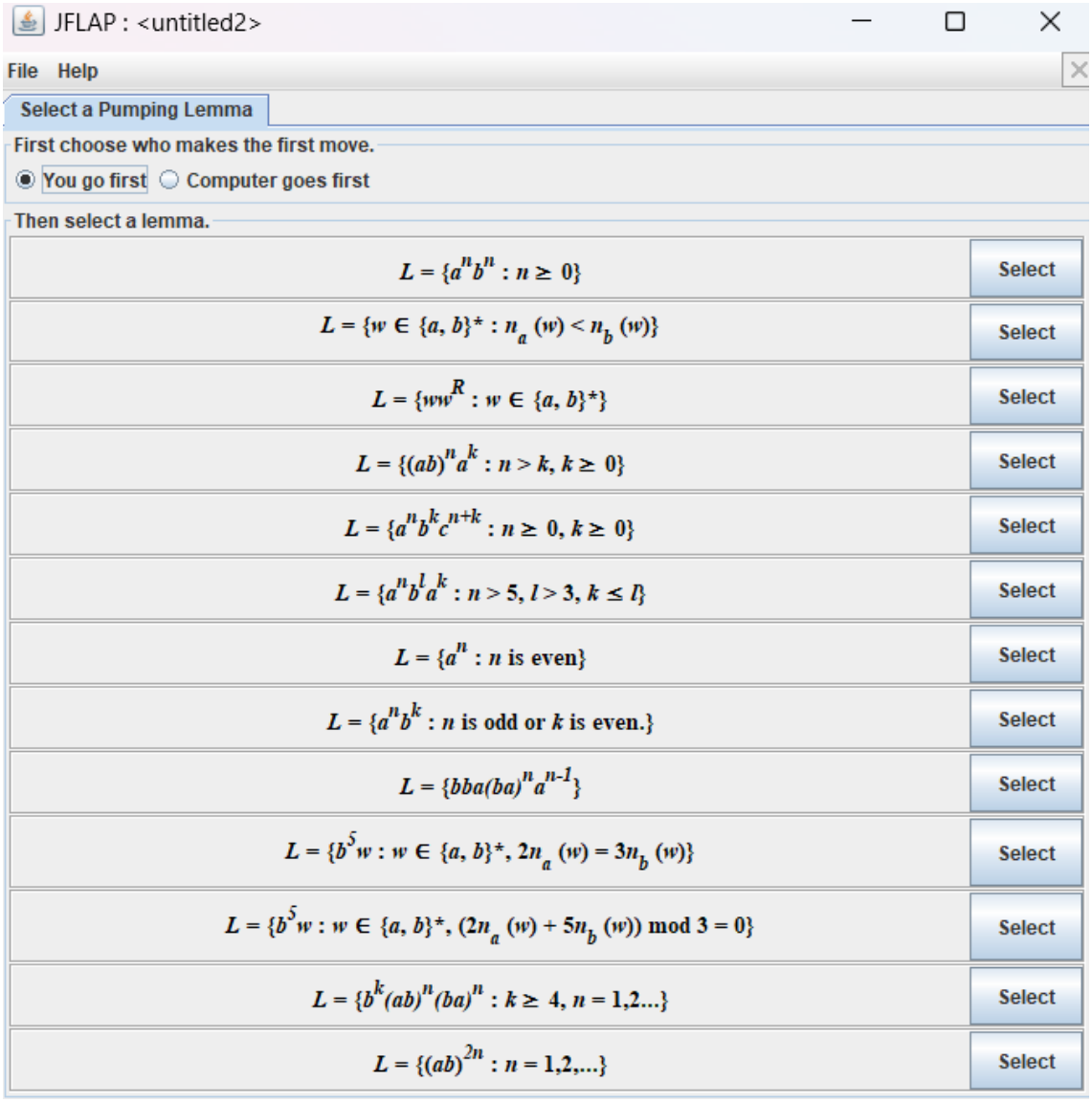


Temos que a palavra formada quando o iterado i é igual a zero, não existe na linguagem, logo a linguagem não pode ser livre de contexto.

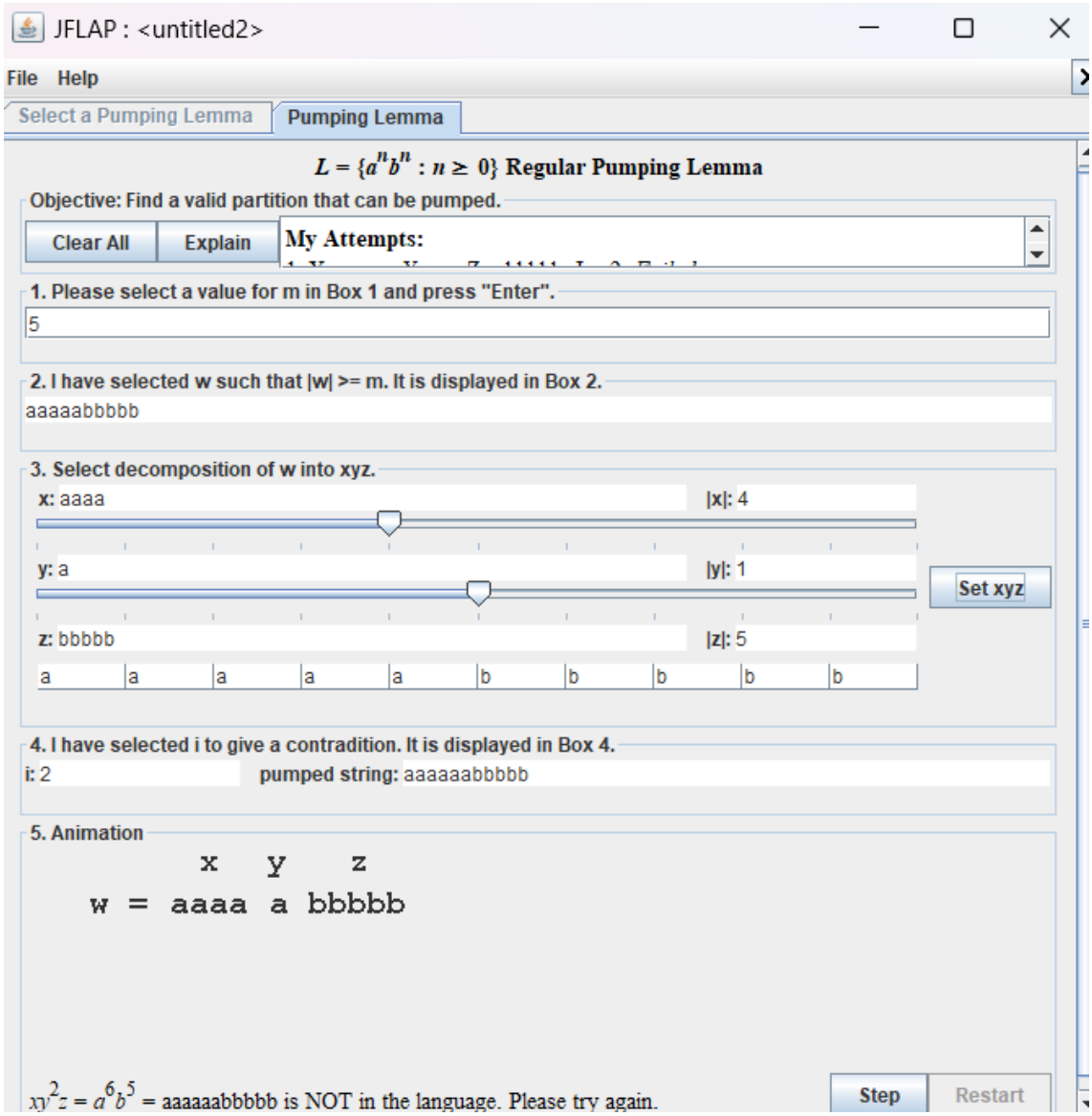


No teste em que o JFLAP age primeiro, o computador nos dá o valor de m e nós devemos entrar com uma palavra que esteja na linguagem selecionada, após isso o computador separa as variáveis UVXYZ e nós selecionamos o valor do iterado.

REGULAR PUMPING LEMMA



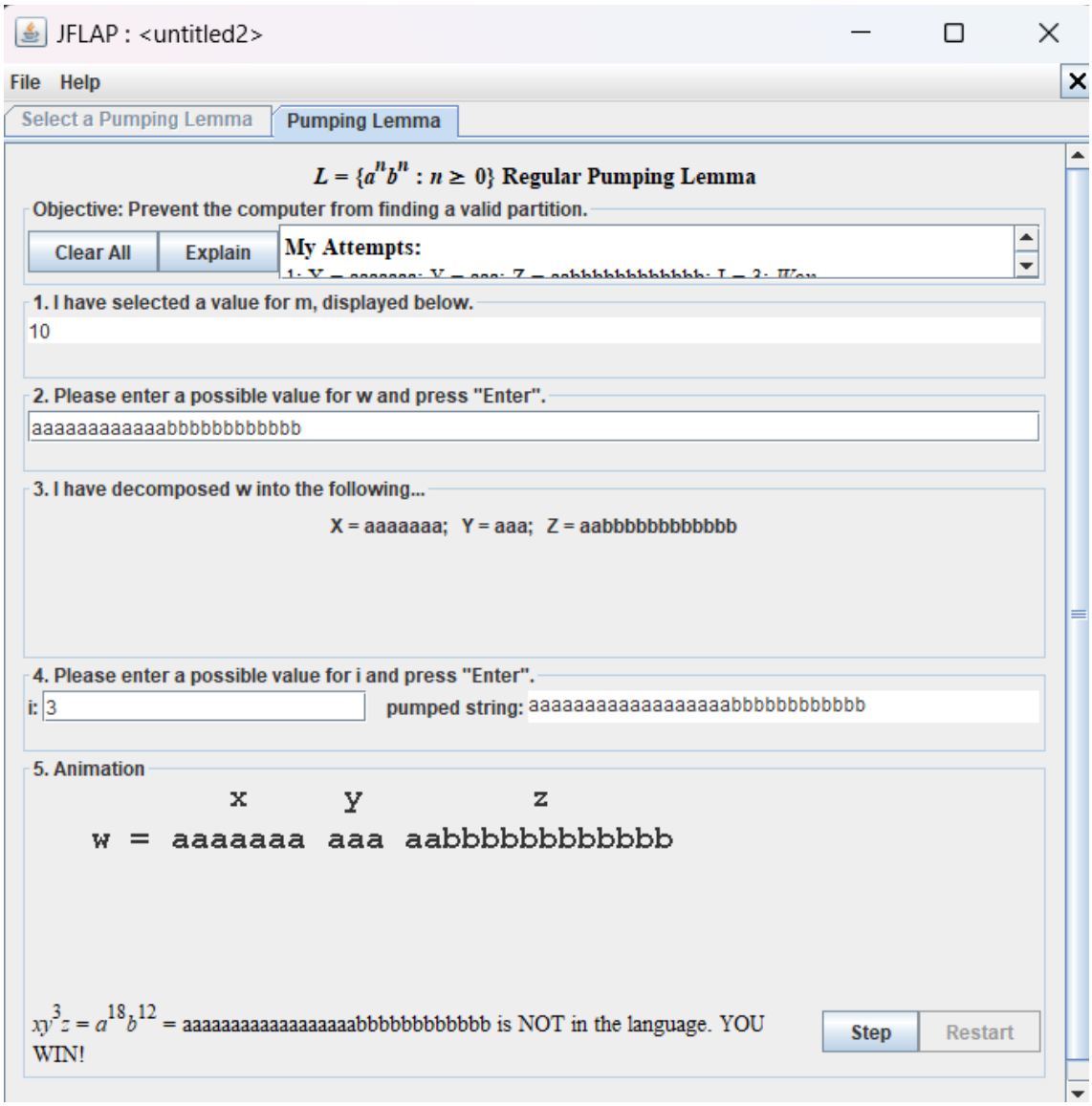
No caso do Bombeamento para a linguagem regular o caso é parecido. O sistema nos cede algumas linguagens para testarmos e podemos selecionar quem faz o primeiro movimento, nós ou o JFLAP.



A linguagem selecionada foi L = $\{a^n b^n$: n ≥ 0} e a opção selecionada foi que o usuário fosse primeiro.

O JFLAP pediu para que eu entrasse com um valor para m, então forneceu uma palavra que está na linguagem, após isso eu tive que selecionar as variáveis XYZ tal que |Y| > 0 e |XY| ≤ m.

Setadas as variáveis o sistema escolhe um i e checa se a nova palavra obtida está ou não na linguagem, caso não esteja ela não é uma linguagem regular.



E este é teste da mesma linguagem feita agora com o primeiro movimento sendo executado pelo computador.

Neste caso, o JFLAP escolhe o valor de m e nós temos que entrar com uma palavra válida, maior que m, que esteja na linguagem. Após isso escolhemos o valor de i e o sistema separa as variáveis e define se a nova palavra está ou não na linguagem.

CONSIDERAÇÕES FINAIS

O JFLAP é um bom ajudante na hora de conferir e testar exercícios para Linguagens Formais, porém ele não é uma plataforma de ensino, ele não contém as definições, tendo assim, que o usuário já saber previamente ou pesquisar sobre no processo. A página WEB

de ajuda do JFLAP é de grande ajuda para entender o que cada função do JFLAP faz, mas um empecilho é que está todo em inglês.

O programa também não tem nada sobre a Forma Normal de Greibach, não citando e nem tendo alguma parte que resolva ou ajude, nem para o Teorema (Fundamental) das Gramáticas Sensíveis ao Contexto e nem sobre o Teorema de Operações Fechadas sobre LLC.

Em resumo, o JFLAP é muito bom para testar as gramáticas e saber quais são seus tipos, além de gerar a Forma Nominal de Chomsky, porém, falta algumas coisas importantes como Greibach.