Context-Free Languages

Regular languages:

- keywords in a programming language
- names of identifiers
- integers
- all misc symbols: = ;

Not Regular languages:

- $\{a^n c b^n | n > 0\}$
- expressions - $((a + b) - c)$ $(^n)^n$
- block structures ({} in Java/C++ and begin ... end in pascal)

Definition: A grammar G=(V,T,S,P) is context-free if all productions are of the form

$$\mathbf{A \rightarrow x}$$

Where A∈V and x∈(V∪T)*.

Definition: L is a context-free language (CFL) iff ∃ context-free grammar (CFG) G s.t. L=L(G).

Example: G=({S},{a,b},S,P)

$$\mathbf{S \rightarrow aSb \mid ab}$$

CFG

Derivation of aaabbb:

$$\mathbf{S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaabbb}$$

L(G) = $\{a^n b^n \mid n > 0\}$

Also linear grammar - at most one variable on r.h.s.

Example: G=({S},{a,b},S,P)

$$\mathbf{S} \rightarrow \mathbf{aSa} \mid \mathbf{bSb} \mid \mathbf{a} \mid \mathbf{b} \mid \lambda$$

Derivation of ababa:

$$\mathbf{S} \Rightarrow \mathbf{aSa} \Rightarrow \mathbf{abSba} \Rightarrow \mathbf{ababa}$$

$\Sigma = \{a, b\}$, $\mathbf{L(G)} =$ $\{w \in \Sigma^* \mid w = w^R\}$

Example: G=({S,A,B},{a,b,c},S,P)

$$\mathbf{S} \rightarrow \mathbf{AcB}$$
$$\mathbf{A} \rightarrow \mathbf{aAa} \mid \lambda$$
$$\mathbf{B} \rightarrow \mathbf{Bbb} \mid \lambda$$

CFG
not Reg Gram
is Reg L

L(G) = $\{a^{2n} c b^{2m} \mid n,m \geq 0\}$

not linear grammar

Derivations of aacbb:

1. $\mathbf{S \Rightarrow \underline{A}cB \Rightarrow a\underline{A}acB \Rightarrow aac\underline{B} \Rightarrow aac\underline{B}bb \Rightarrow aacbb}$

leftmost der

2. $\mathbf{S \Rightarrow Ac\underline{B} \Rightarrow Ac\underline{B}bb \Rightarrow \underline{A}cbb \Rightarrow a\underline{A}acbb \Rightarrow aacbb}$

rightmost der

Note: Next variable to be replaced is underlined.

Definition: Leftmost derivation - in each step of a derivation, replace the leftmost variable. (see derivation 1 above).

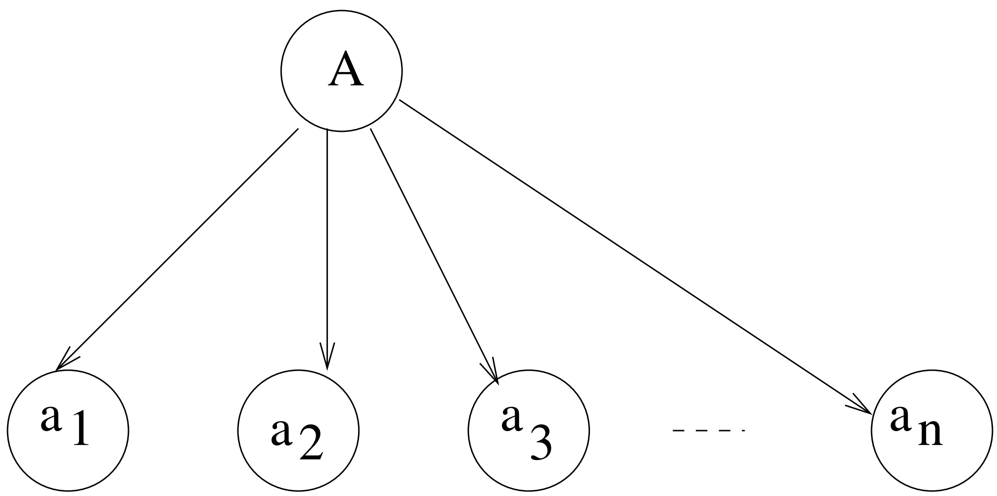
Definition: Rightmost derivation - in each step of a derivation, replace the rightmost variable. (see derivation 2 above).

Derivation Trees (also known as "parse trees")

A derivation tree represents a derivation but does not show the order productions were applied.

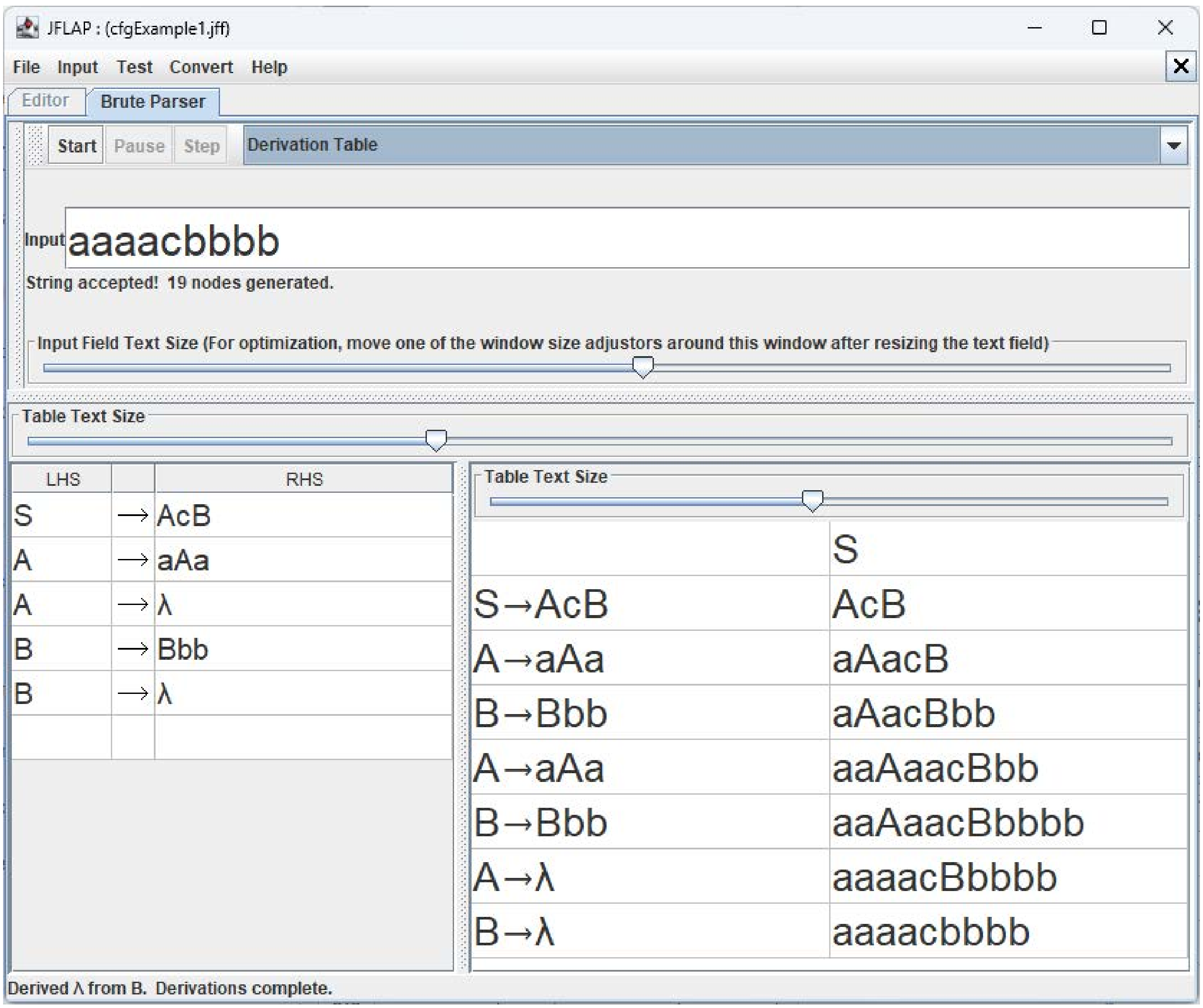
A derivation tree for G=(V,T,S,P):

- **root is labeled S**
- **leaves labeled x, where x**$\in$**T**$\cup\{\lambda\}$
- **nonleaf vertices labeled A, A**$\in$**V**
- **For rule A**$\rightarrow a_1 a_2 a_3 \ldots a_n$**, where A**$\in$**V,** $a_i \in$**(T**$\cup$**V**$\cup\{\lambda\}$**),**

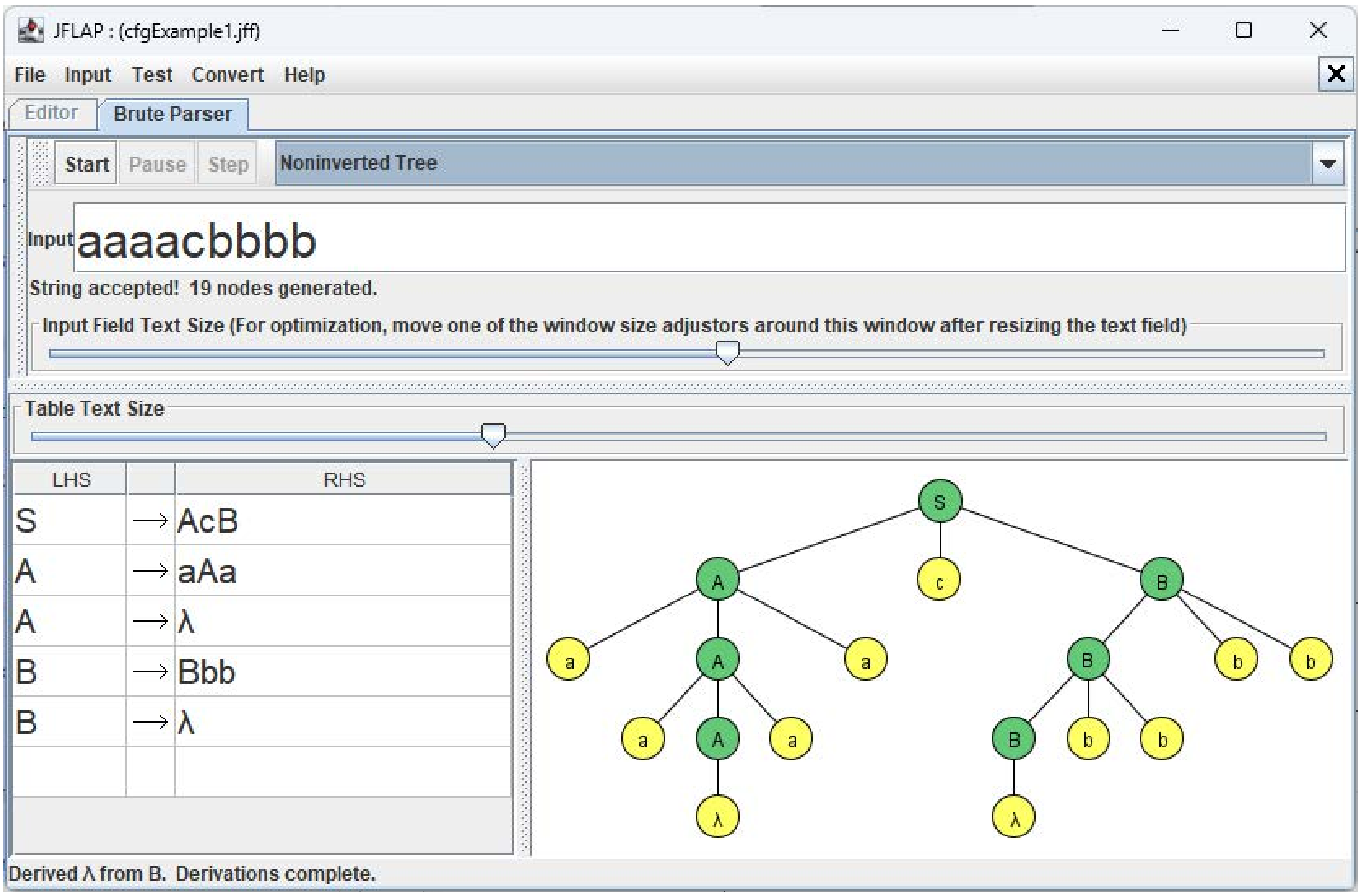


Example: G=({S,A,B},{a,b,c},S,P)

$$
\begin{aligned}
\mathbf{S} &\rightarrow \mathbf{AcB}\\
\mathbf{A} &\rightarrow \mathbf{aAa} \mid \lambda\\
\mathbf{B} &\rightarrow \mathbf{Bbb} \mid \lambda
\end{aligned}
$$



With parse tree



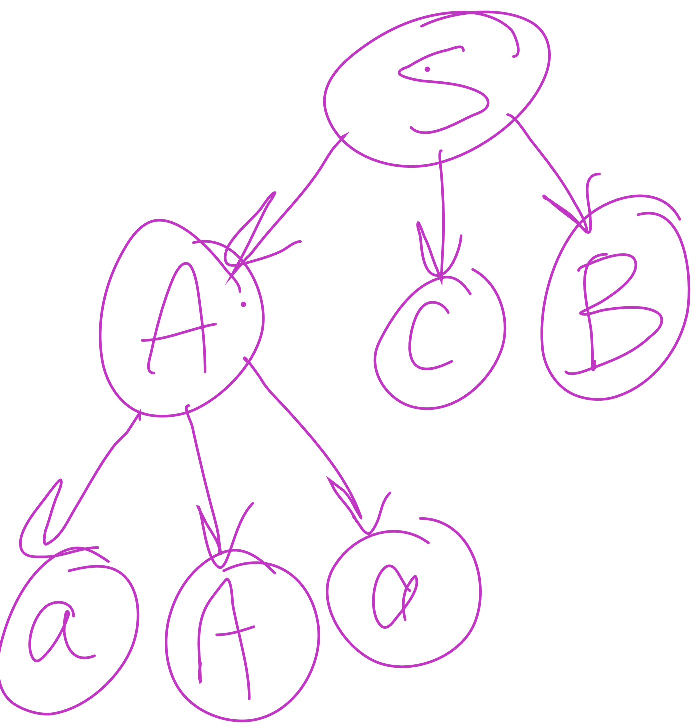
Definitions Partial derivation tree - subtree of derivation tree.

If partial derivation tree has root S then it represents a sentential form.

Leaves from left to right in a derivation tree form the *yield* of the tree.

Yield (w) of derivation tree is such that w∈L(G).

The yield for the example ~~above~~ below is



aAacB

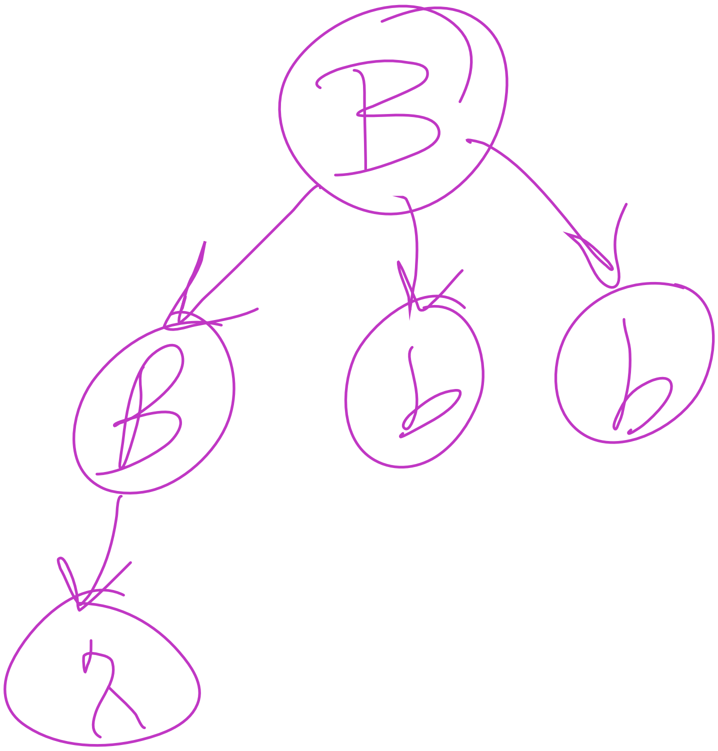
Sential form because partial tree starts w/ S

Example of partial derivation tree that has root S:

See above

The yield of this example is ________________ which is a sentential form.

Example of partial derivation tree that does not have root S:



yield is bb

S ⇒ AcB ⇒ aAacB ⇒ aacB ⇒ aacBbb ⇒ aacbb

all sentential forms

Membership Given CFG G and string w$\in \Sigma^*$, is w$\in$L(G)?

If we can find a derivation of w, then we would know that w is in L(G).

Motivation

G is grammar for Java
w is Java program.
Is w syntactically correct?

Example

G=({S}, {a,b}, S, P), P=

$$\mathbf{S} \rightarrow \mathbf{SS} \mid \mathbf{aSa} \mid \mathbf{b} \mid \lambda$$

L_1=**L(G)** = $\{w \in \Sigma^* \mid$ w has an even number of a's $\}$

Is abbab ∈ L(G)?

Exhaustive Search Algorithm

For all i=1,2,3,...
Examine all sentential forms yielded by i substitutions

Example: Is abbab $\in$ L(G)?

$i=1$

$S \Rightarrow SS$

$S \Rightarrow aSa$

$S \Rightarrow b$

$S \Rightarrow \lambda$

$i=2$ $S \Rightarrow SS \Rightarrow SSS$

$S \Rightarrow SS \Rightarrow aSaS$

$S \Rightarrow SS \Rightarrow bS$

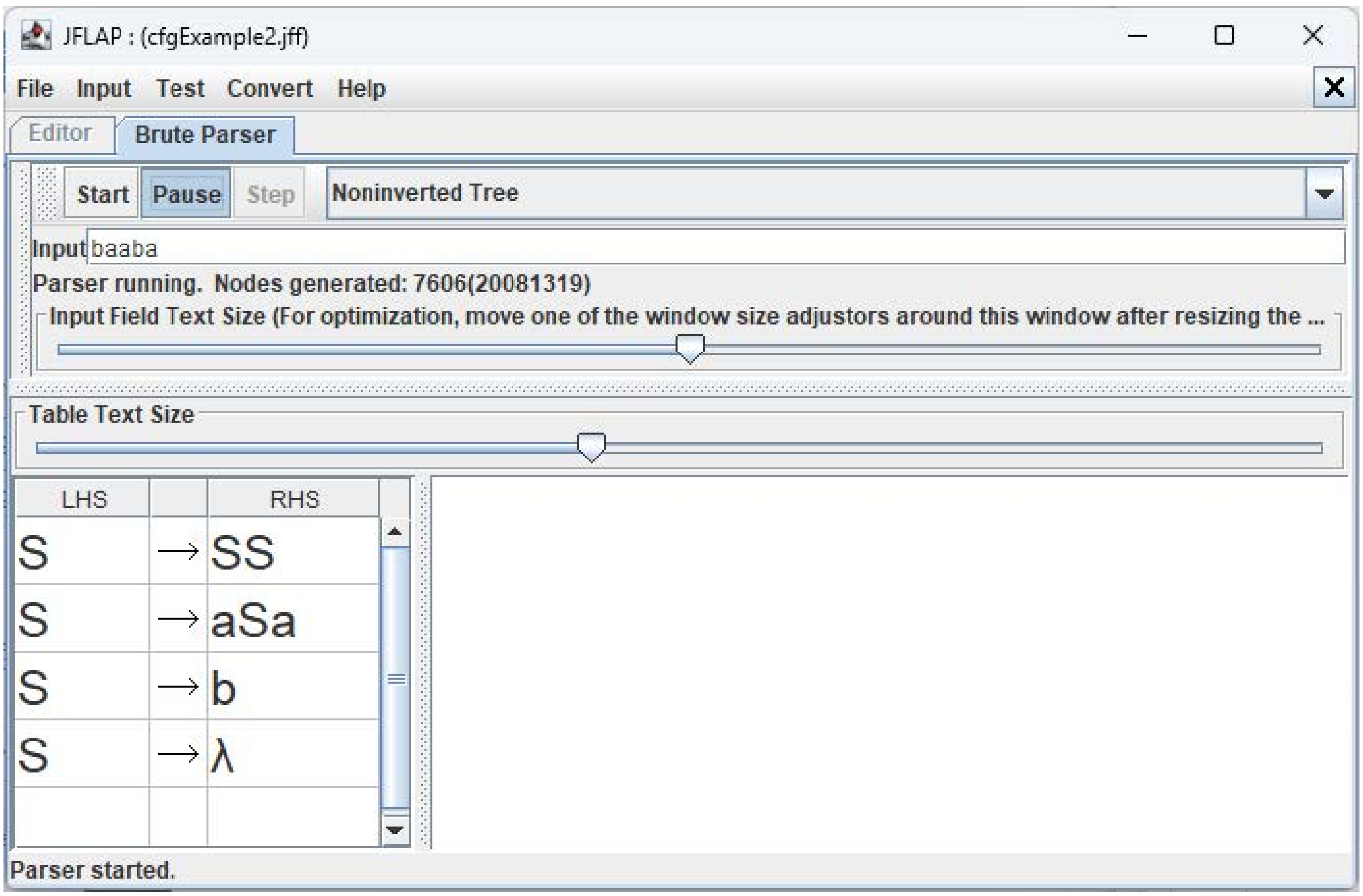
$S \Rightarrow SS \Rightarrow S$

$S \Rightarrow SS \Rightarrow Sb$

$\vdots$

$S \Rightarrow aSa \Rightarrow aSSa$

The brute force parser is trying to derive baaba but it is taking too long for it to realize that string should be rejected.



Theorem If CFG G does not contain rules of the form

$$\mathbf{A} \rightarrow \lambda$$
$$\mathbf{A} \rightarrow \mathbf{B}$$

Unit prod

shrinks
same size

where A,B∈V, then we can determine if w∈ L(G) or if w∉ L(G).

- **Proof: Consider**
  1. **length of sentential forms**
  2. **number of terminal symbols in a sentential form**

Either 1 or 2 increases with each derivation

Derivation of string w in L(G) takes ≤ 2|w|

Example: Let $L_2 = L_1 - \{\lambda\}$. L_2=L(G) where G is:

$$\mathbf{S \rightarrow SS \mid aa \mid aSa \mid b}$$

Show baaba $\notin$ L(G).

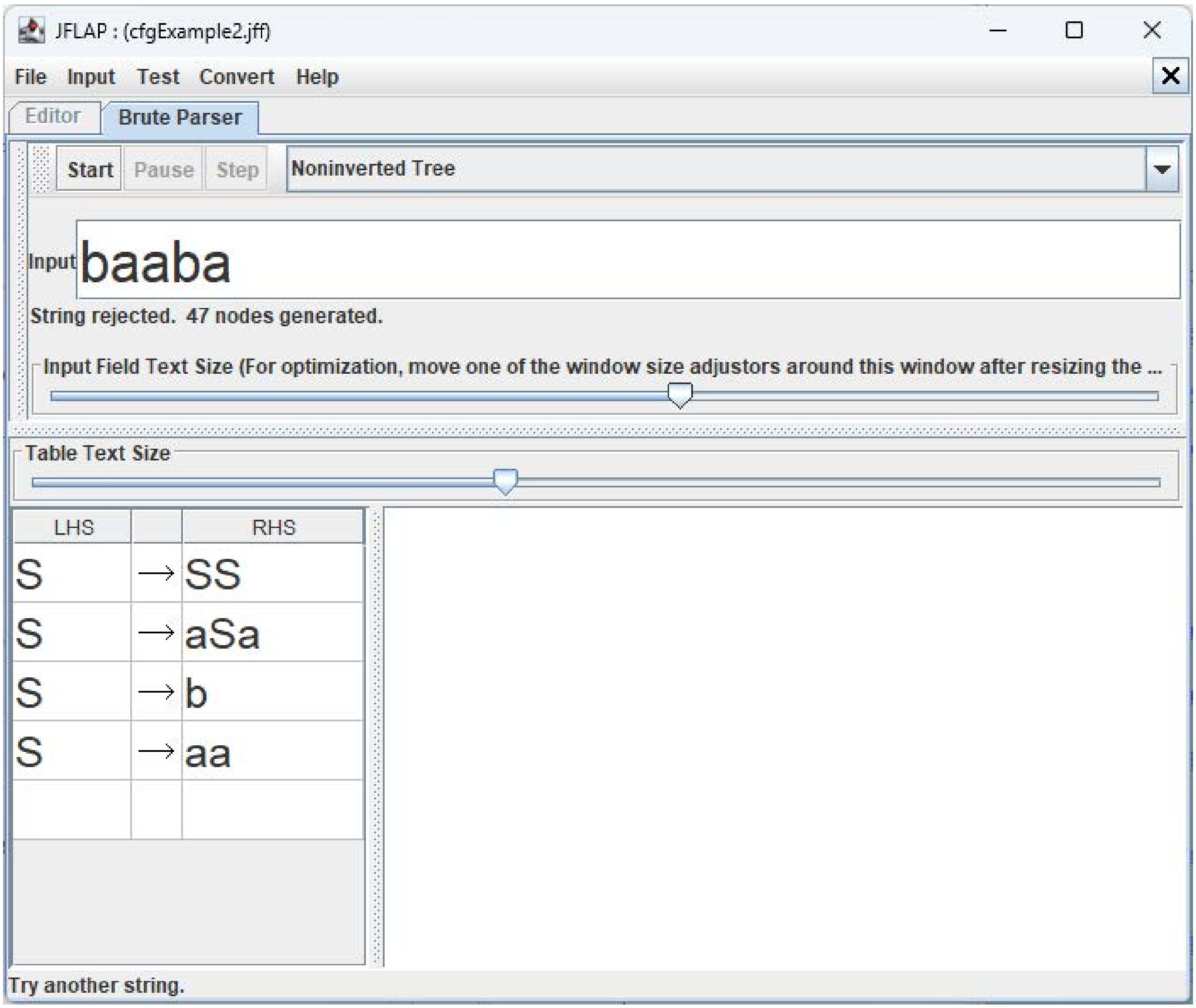
i=1
1. **S ⇒ SS**
2. **S ⇒ aSa**
3. **S ⇒ aa**
4. **S ⇒ b**

i=2
1. **S ⇒ SS ⇒ SSS**
2. **S ⇒ SS ⇒ aSaS**
3. **S ⇒ SS ⇒ aaS**
4. **S ⇒ SS ⇒ bS**
5. **S ⇒ aSa ⇒ aSSa**
6. **S ⇒ aSa ⇒ aaSaa**
7. **S ⇒ aSa ⇒ aaaa**
8. **S ⇒ aSa ⇒ aba**

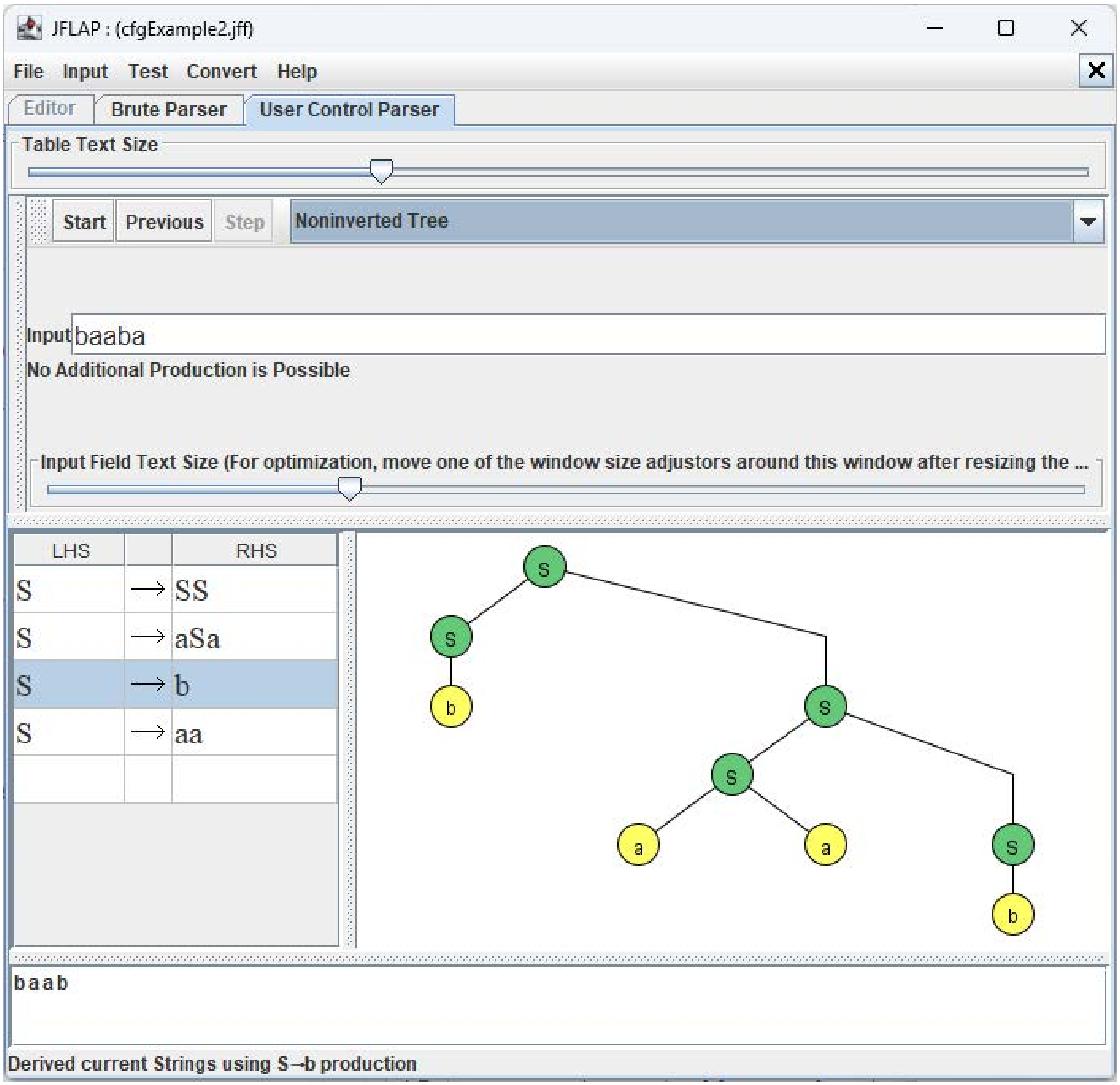
⋮

stop at 10 steps
if not found it

With the modified grammar, the brute force parser rejects the string quickly



You can use the User Control Parse to see why a string cannot be derived. If there is more than one variable in the sentential form, you will need to click on which variable you want to expand in the very bottom window (not in the parse tree)



Definition Simple grammar (or s-grammar) has all productions of the form:

$$\mathbf{A \rightarrow ax}$$

where $\mathbf{A \in V}$, $\mathbf{a \in T}$, and $\mathbf{x \in V^*}$ AND any pair (A,a) can occur in at most one rule.

Ambiguity

Definition: A CFG G is ambiguous if $\exists$ some w$\in$ L(G) which has two distinct derivation trees.

ambiguous grammar not good

Example Expression grammar

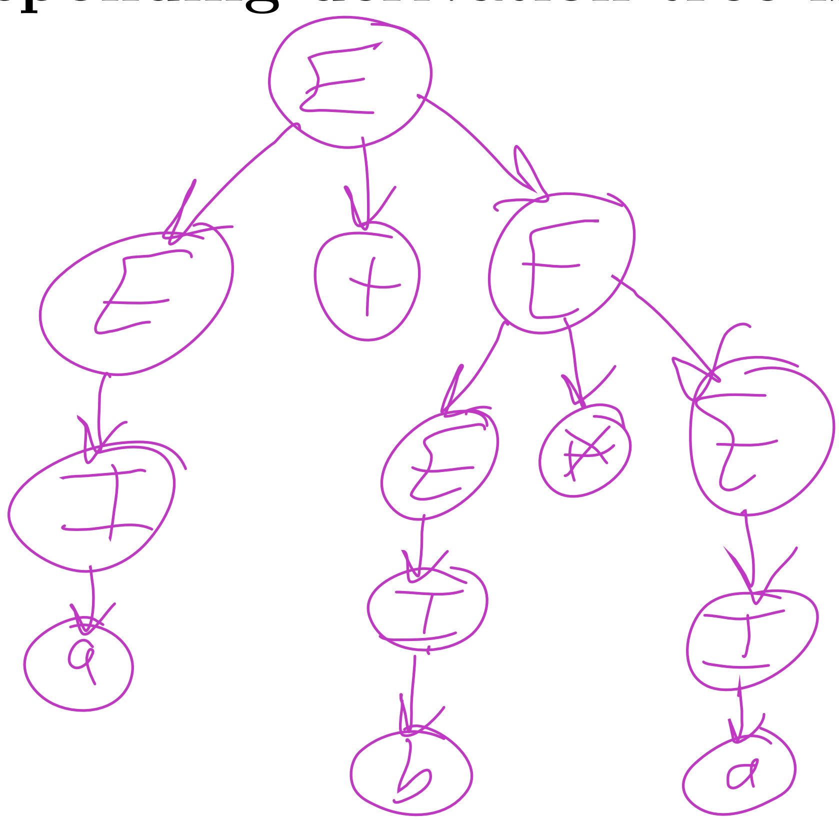
$$\mathbf{G=(\{E,I\}, \{a,b,+,*,(,)\}, E, P), P=}$$

$$\mathbf{E \rightarrow E{+}E \mid E{*}E \mid (E) \mid I}$$
$$\mathbf{I \rightarrow a \mid b}$$

Derivation of a+b∗a is:

$$\mathbf{E \Rightarrow \underline{E}{+}E \Rightarrow \underline{I}{+}E \Rightarrow a{+}\underline{E} \Rightarrow a{+}\underline{E}{*}E \Rightarrow a{+}\underline{I}{*}E \Rightarrow a{+}b{*}\underline{E} \Rightarrow a{+}b{*}\underline{I} \Rightarrow a{+}b{*}a}$$

Corresponding derivation tree is:



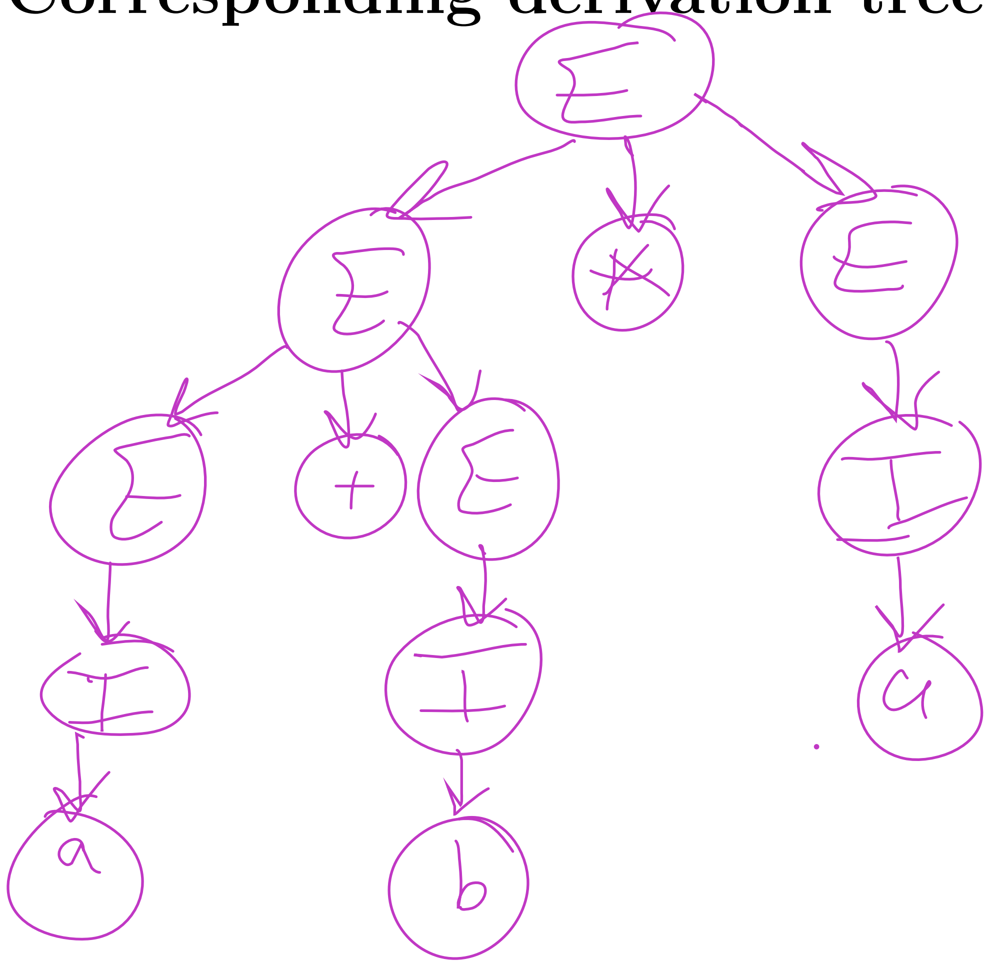
probably the tree you want

mult higher prec

Another derivation of a+b∗a is:

$$\mathbf{E} \Rightarrow \underline{\mathbf{E}}*\mathbf{E} \Rightarrow \underline{\mathbf{E}}+\mathbf{E}*\mathbf{E} \Rightarrow \underline{\mathbf{I}}+\mathbf{E}*\mathbf{E} \Rightarrow \mathbf{a}+\underline{\mathbf{E}}*\mathbf{E} \Rightarrow \mathbf{a}+\underline{\mathbf{I}}*\mathbf{E} \Rightarrow \mathbf{a}+\mathbf{b}*\underline{\mathbf{E}} \Rightarrow \mathbf{a}+\mathbf{b}*\underline{\mathbf{I}} \Rightarrow \mathbf{a}+\mathbf{b}*\mathbf{a}$$

Corresponding derivation tree is:



Stopped here

Rewrite the grammar as an unambiguous grammar. (with meaning that multiplication has higher precedence than addition)

$$\begin{aligned}
&\mathbf{E} \rightarrow \mathbf{E+T} \mid \mathbf{T}\\
&\mathbf{T} \rightarrow \mathbf{T*F} \mid \mathbf{F}\\
&\mathbf{F} \rightarrow \mathbf{I} \mid \mathbf{(E)}\\
&\mathbf{I} \rightarrow \mathbf{a} \mid \mathbf{b}
\end{aligned}$$

There is only one derivation tree for a+b∗a:

Definition If L is CFL and G is an unambiguous CFG s.t. L=L(G), then L is unambiguous.

Backus-Naur Form of a grammar:

- Nonterminals are enclosed in brackets <>
- For "→" use instead "::="

Sample C++ Program:

```
main ()
{
   int a;      int b;   int sum;
   a = 40;     b = 6;   sum = a + b;
   cout << "sum is "<< sum << endl;
}
```

“Attempt” to write a CFG for C++ in BNF (Note: <program> is start symbol of grammar.)

```
<program>    ::= main () <block>
<block>      ::= { <stmt-list> }
<stmt-list>  ::= <stmt> | <stmt><stmt-list>
                 <decl> | <decl><stmt-list>
<decl>       ::= int <id> ; | double <id> ;
<stmt>       ::= <asgn-stmt> | <cout-stmt>
<asgn-stmt>  ::= <id> = <expr> ;
<expr>       ::= <expr> + <expr>
                 | <expr> * <expr>
                 | ( <expr> ) | <id>
<cout-stmt>  ::= cout <out-list> ;
```

etc., Must expand all nonterminals!

So a derivation of the program test would look like:

<program> $\Rightarrow$ **main () <block>**
$\Rightarrow$ **main () { <stmt-list> }**
$\Rightarrow$ **main () { <decl> <stmt-list> }**
$\Rightarrow$ **main () { int <id>; <stmt-list>**
$\Rightarrow$ **main () { int a ; <stmt-list> }**
$\overset{*}{\Rightarrow}$ *complete C++ program*

More on CFG for C++

We can write a CFG G s.t. L(G)={syntactically correct C++ programs}.

But note that {semantically correct C++ programs} $\subset$ L(G).

Can't recognize redeclared variables:

```
int x;
double x;
```

Can't recognize if formal parameters match actual parameters in number and types:

```
declar: int Sum(int a, int b, int c) ...
call:   newsum = Sum(x,y);
```