E::= E + E

A ::= ab | a

A::= a(b|λ)

* 1. Toda produção é da forma A::= yα, onde y é um terminal.

A::= bB

B::= cC

* 1. Se A::= y1α1| y2α2 |....| ynαn são todas as alternativas para o não-terminal A, então os terminais yi são todos diferentes entre si.

A::= bB | cD |defC

E ::= a | b | +EE | \*EE

\*+abb

E=>\*EE=>\*+EEE=>\*+aEE=>\*+abE=>\*+abb

Considere a gramática

A::=aB | cC

Qual é o FIRST(A) = ?

...axxx ou cxxx

....=> aB ou cC

S ::= AS | BA

A ::= aB | C

B ::= bA | d

C ::= c

Símbolo inicial da gramática S

considere a palavra “abaddc”

S => AS => aBS =>abAS => abaBS =>abadS

FIRST(a) = {a}

FIRST(A) = {a,FIRST(C)}= {a,c}

**regra 2:** Se α::=λ for uma produção, adiciona λ ao FIRST(α).

E::= eB | λ

FIRST(E) = {e , λ}

**regra 3:** Se α for um não-terminal e α::= y1 y2 ...yk uma produção, colocar em FIRST(α) tudo o que estiver em FIRST(y1). Se y1 derivar λ, então é adicionado FIRST(α) o FIRST(y2), e assim por diante, só é adicionado ao FIRST(α) o λ se forem adicionados sucessivamente os FIRST(y1), FIRST(y2) até FIRST(yk-1) e yk derivar λ.

α::= y1y2y3

E::= FDG

FIRST(E) = {FIRST(F), FIRST(D), FIRTS(G),... }={f,d,g}

F::= f | λ

FIRST(F) = {f, λ}

D ::= d | λ

FIRST(D)={ d, λ}

G::= g

Calculando FIRST para gramática abaixo considere que S é produção inicial da gramática.

S ::= AS | BA

A ::= aB | C

B ::= bA | d

C ::= c

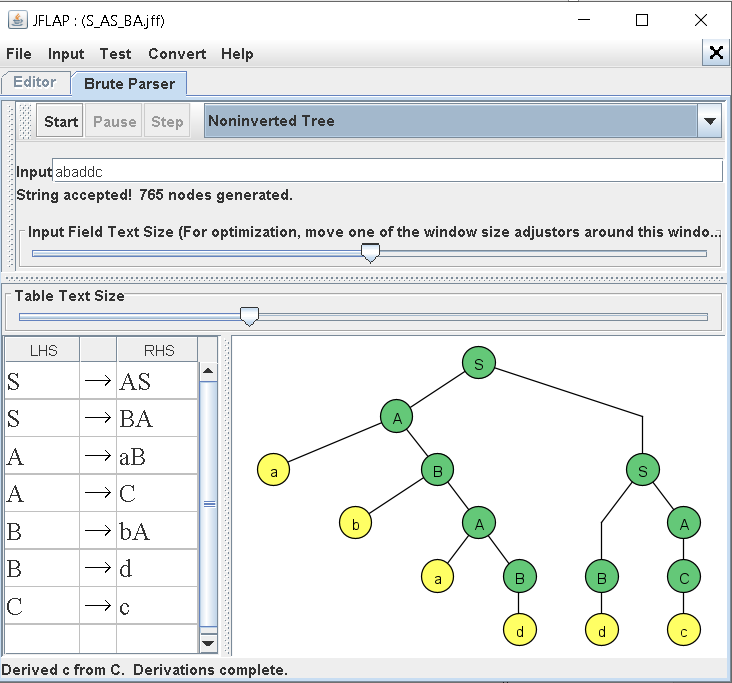
FIRST( C ) = { c }

FIRST( B ) = { b, d }

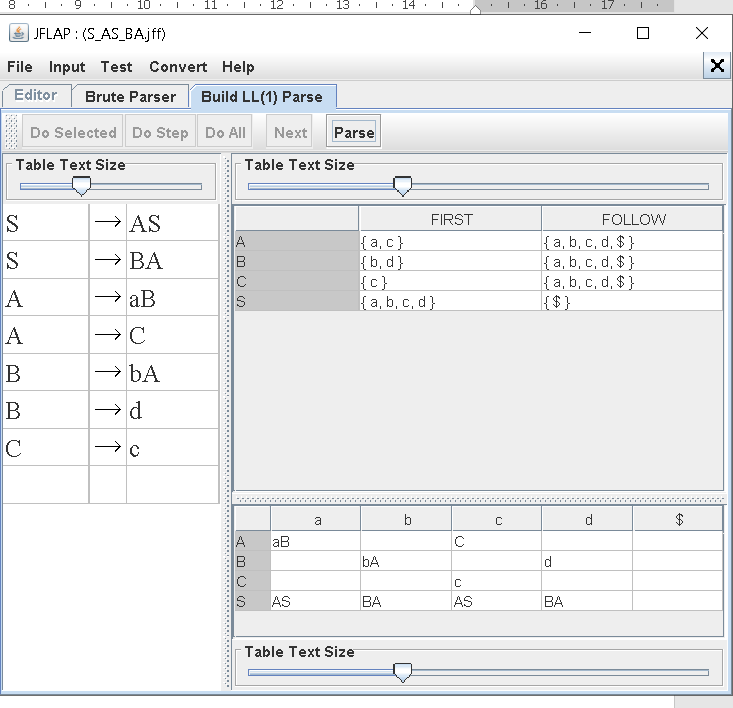
FIRST( A ) = { a, FIRST( C ) } = { a, c }

FIRST( S ) = { FIRST(A), FIRST(B)} = {a,c,b,d}

1. Construa a árvore de derivação para a sentença “abaddc”.



1. Calcule o **FIRST** para os não-terminais da gramática.



**Eliminação de recursividade à esquerda**

Gramática ambigua

E ::= E + E | num

onde num = [0-9], ou seja, 0 | 1 | 2 | 3 | .. | 9

1+2+3

Gramática com eliminação de ambiguidade

E ::= E + F | F

F ::= num

Imagine agora que iremos implementar o ASDR para gramática

void E(){

E();

consome(‘+’);

F();

}

void F(){

consome(num);

}

1

E ⇒ F ⇒num

+F repetiu 0 vezes

1+2

E ⇒ E+F ⇒ F+F ⇒num+ num

+F repetiu 1 vezes

1+2+3

E ⇒ E+F ⇒ E+F+F ⇒ F+F+F ⇒num + num + num

+F repetiu 2 vezes

1+2+3+4+...+9

E ⇒ E+F ⇒ E+F+F ..+F ⇒ F+F+..+F ⇒num+num+...+num

+F repetiu várias vezes

Regra geral

A::= y1 | y2 | ... | yn | Aα ou A::= Aα | y1 | y2 | ... | yn

A ⇒ A α ⇒ A α α ⇒ A α α..... α ⇒(y1 | y2 | ... | yn) α α..... α

Eliminando a recursividade à esquerda

A::= ⇒(y1 | y2 | ... | yn){ α}

Eliminando a recursividade à esquerda da gramática

Gramática com eliminação de ambiguidade

E ::= E + F | F

A::= A α | y1

E ::= F { + F }

Considere a gramática

E ::= E + F | E – F | F

E=> E (+F |-F) => E(+F |-F) (+F |-F)=> F(+F |-F) ..... (+F |-F)

Gramática fatorada à esquerda

E ::= E (+|-) F | F

Eliminando a recursividade.

E ::= E (+|-) F | F

E::= F { (+|-)F }