Analisadores Descendentes e Ascendentes

Analisadores descendentes e ascendentes

Os métodos de análise sintática podem ser classificados segundo a maneira pela qual a árvore de derivação da cadeia analisada X é construída:

Ascendentes - a árvore é construída de baixo para cima, ou seja, das folhas (tokens) para o símbolo inicial S.

Descendentes - a árvore é construída de cima para baixo, ou seja, do símbolo inicial S para as folhas (tokens).

Analisadores descendentes e ascendentes

Métodos **descendentes e ascendentes** constroem a árvore da **esquerda para a direita**. A razão para isso é que a escolha das regras deve se basear na cadeia a ser gerada, que é lida da esquerda para a direita.

Analisadores Descendentes

Analisadores descendentes

Os métodos **descendentes** (***top-down***) fazem a expansão da árvore de derivação a partir da raiz S, expandindo sempre o não-terminal mais à esquerda. Exemplo:

Tokens: | a | c | $ |

R1) S → Tc

R2) T → a

R3) T → b

Analisadores descendentes

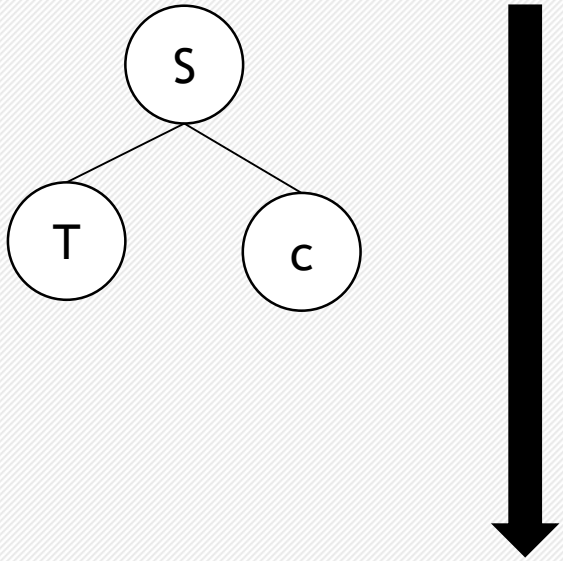
Os métodos **descendentes** (***top-down***) fazem a expansão da árvore de derivação a partir da raiz S, expandindo sempre o não-terminal mais à esquerda. Exemplo:

Tokens: | a | c | $ |

R1) S → Tc

R2) T → a

R3) T → b



Inicia-se a derivação pela raiz S

Analisadores descendentes

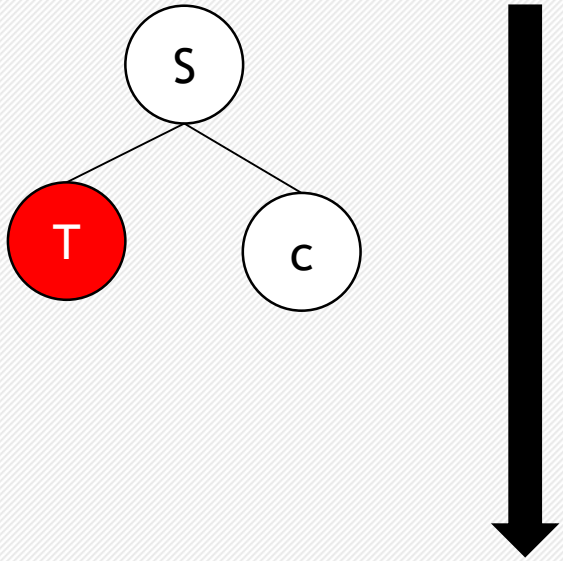
Os métodos **descendentes** (***top-down***) fazem a expansão da árvore de derivação a partir da raiz S, expandindo sempre o não-terminal mais à esquerda. Exemplo:

Tokens: | a | c | $ |

R1) S → Tc

R2) T → a

R3) T → b



O próximo nó (na pré-ordem) é não terminal

Analisadores descendentes

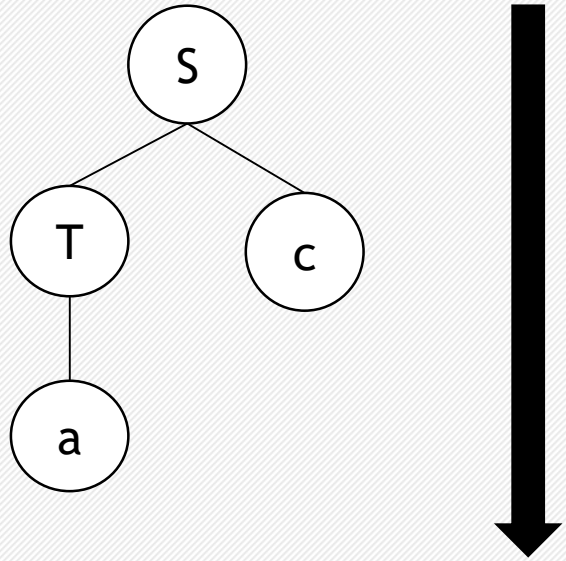
Os métodos **descendentes** (***top-down***) fazem a expansão da árvore de derivação a partir da raiz S, expandindo sempre o não-terminal mais à esquerda. Exemplo:

Tokens: | a | c | $ |

R1) S → Tc

R2) T → a

R3) T → b



Próxima produção para o nó não terminal T

Analisadores descendentes

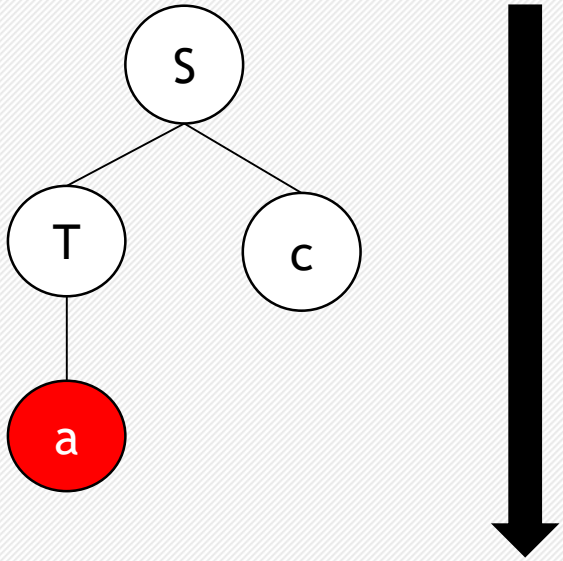
Os métodos **descendentes** (***top-down***) fazem a expansão da árvore de derivação a partir da raiz S, expandindo sempre o não-terminal mais à esquerda. Exemplo:

Tokens: | a | c | $ |

R1) S → Tc

R2) T → a

R3) T → b



O próximo nó (na pré-ordem) a é terminal!!!

Analisadores descendentes

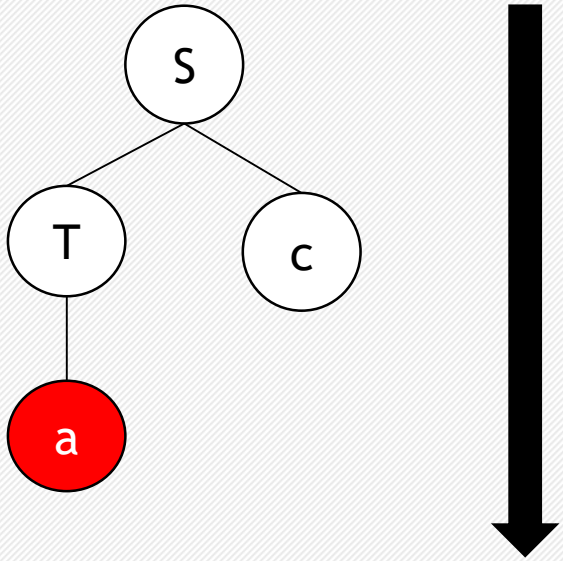
Os métodos **descendentes** (***top-down***) fazem a expansão da árvore de derivação a partir da raiz S, expandindo sempre o não-terminal mais à esquerda. Exemplo:

Tokens: | a | c | $ |

R1) S → Tc

R2) T → a

R3) T → b



E casa (match) com a entrada!

Analisadores descendentes

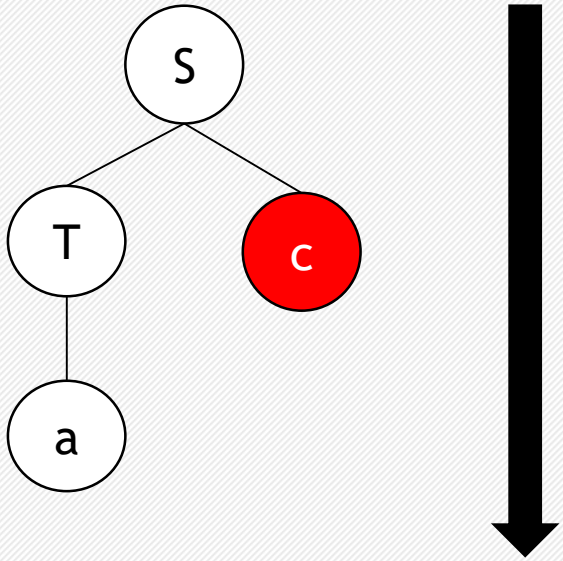
Os métodos **descendentes** (***top-down***) fazem a expansão da árvore de derivação a partir da raiz S, expandindo sempre o não-terminal mais à esquerda. Exemplo:

Tokens: | c | $ |

R1) S → Tc

R2) T → a

R3) T → b



O próximo nó (na pré-ordem) é terminal

Analisadores descendentes

Os métodos **descendentes** (***top-down***) fazem a expansão da árvore de derivação a partir da raiz S, expandindo sempre o não-terminal mais à esquerda. Exemplo:

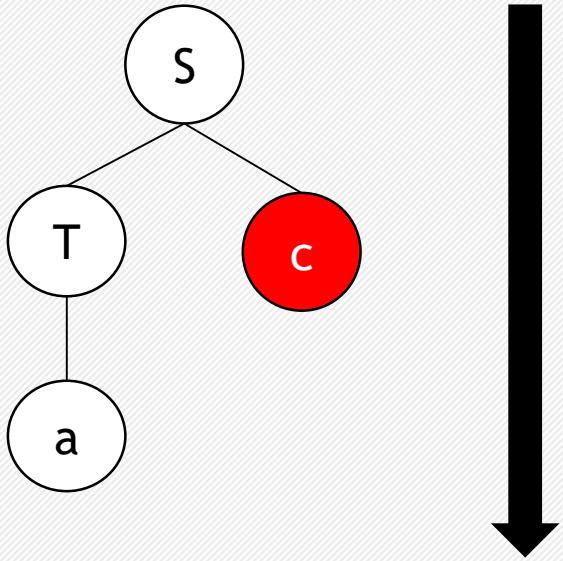
Tokens: c $

R1) S → Tc

R2) T → a

R3) T → b

E casa (match) com a entrada



Analisadores descendentes

Os métodos **descendentes** (***top-down***) fazem a expansão da árvore de derivação a partir da raiz S, expandindo sempre o não-terminal mais à esquerda. Exemplo:

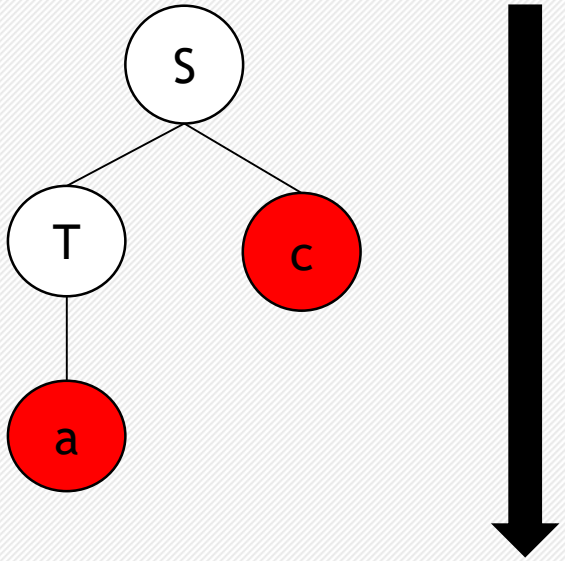
Tokens: $

R1) S → Tc

R2) T → a

R3) T → b

Não há mais não terminais...



Analisadores descendentes

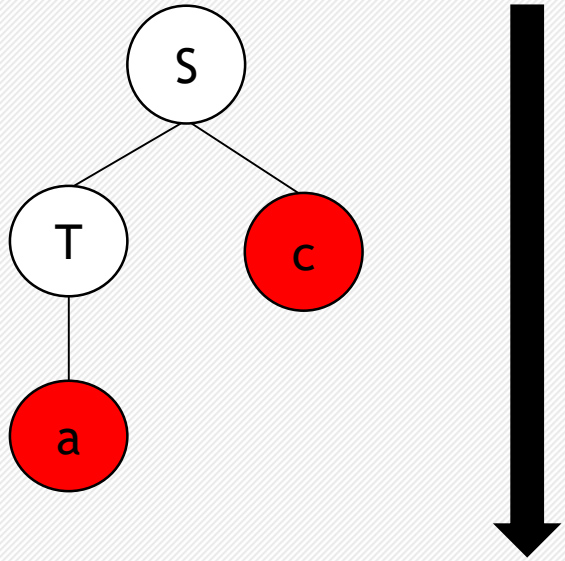
Os métodos **descendentes** (***top-down***) fazem a expansão da árvore de derivação a partir da raiz S, expandindo sempre o não-terminal mais à esquerda. Exemplo:

Tokens: $

R1) S → Tc

R2) T → a

R3) T → b



...e a próxima entrada é $. Cadeia aceita!

Analisadores descendentes
O problema da recursão à esquerda!

Analisadores **descendentes** (***top-down***) não lidam com recursão à esquerda, pois isto os leva a uma não terminação. Exemplo:

Tokens: | c | c | $ |

R1) S → Sc

R2) S → d

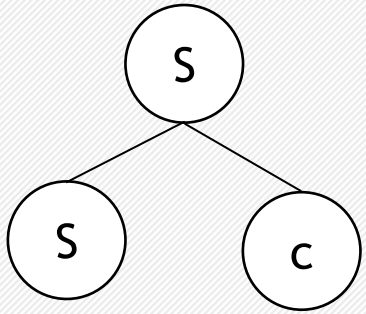
Analisadores descendentes
O problema da recursão à esquerda!

Analisadores **descendentes** (***top-down***) não lidam com recursão à esquerda, pois isto os leva a uma não terminação. Exemplo:

Tokens: | c | c | $ |

R1) S → Sc

R2) S → c



Inicia-se a derivação pela raiz S

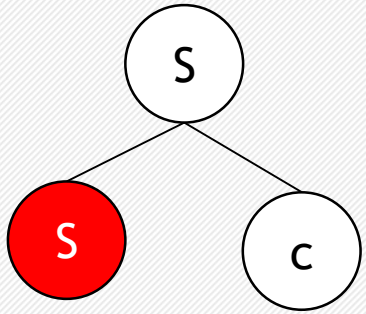
Analisadores descendentes
O problema da recursão à esquerda!

Analisadores **descendentes** (***top-down***) não lidam com recursão à esquerda, pois isto os leva a uma não terminação. Exemplo:

Tokens: | c | c | $ |

R1) S → Sc

R2) S → c



O próximo nó (na pré-ordem) é não terminal

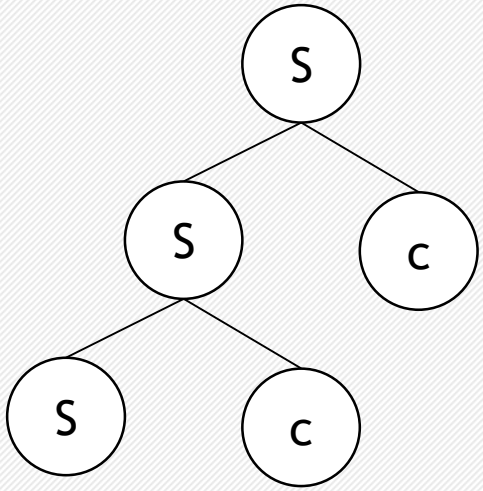
Analisadores descendentes
O problema da recursão à esquerda!

Analisadores **descendentes** (***top-down***) não lidam com recursão à esquerda, pois isto os leva a uma não terminação. Exemplo:

Tokens: | c | c | $ |

R1) S → Sc

R2) S → c



Próxima produção para o nó não terminal S

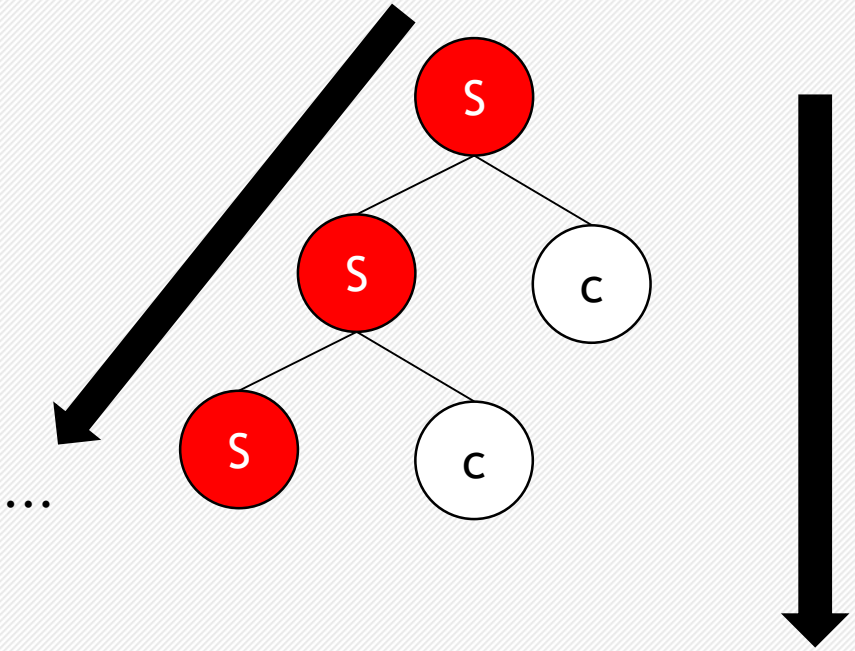
Analisadores descendentes
O problema da recursão à esquerda!

Analisadores **descendentes** (***top-down***) não lidam com recursão à esquerda, pois isto os leva a uma não terminação. Exemplo:

Tokens: | c | c | $ |

R1) S → Sc

R2) S → c



Entrou em loop!!!

Analisadores descendentes
O problema da recursão à esquerda!

Eliminando a recursão:

R1) S → Sc

R2) S → c

Analisadores descendentes
O problema da recursão à esquerda!

Eliminando a recursão:

~~R1) S → Sc~~ Eliminamos a produção recursiva

R2) S → c

Analisadores descendentes
O problema da recursão à esquerda!

Eliminando a recursão:

~~R1) S → Sc~~ Eliminamos a produção recursiva

R2) S → cT Adicionamos um novo não terminal em R2

Analisadores descendentes
O problema da recursão à esquerda!

Eliminando a recursão:

~~R1) S → Sc~~	Eliminamos a produção recursiva
R2) S → cT	Adicionamos um novo não terminal em R2
R3) T → cT	que deve produzir a parte “não recursiva” da regra eliminada (R1) com recursão à direita

Analisadores descendentes
O problema da recursão à esquerda!

Eliminando a recursão:

~~R1) S → Sc~~	Eliminamos a produção recursiva
R2) S → cT	Adicionamos um novo não terminal em R2
R3) T → cT	que deve produzir a parte "não recursiva" da regra eliminada (R1) com recursão à direita
R4) T → ε	e a cadeia vazia

Analisadores descendentes
O problema do determinismo!

Analisadores descendentes (*top-down*) lidam apenas com gramáticas **fatoradas** (determinísticas).

Uma gramática fatorada é uma gramática que não apresenta produções do tipo $A \rightarrow ab^1|ab^2|...|ab^n$

R1) S → c S

R2) S → c

Não fatorada! Mesmo prefixo para as 2 alternativas de S

Analisadores descendentes
O problema do determinismo!

Fatorando a gramática:

R1) S → c S

R2) S → c

Analisadores descendentes
O problema do determinismo!

Fatorando a gramática:

R1) S → c S
R2) S → c

R1) S → c Cria-se um regra com a fonte do não determinismo

Analisadores descendentes
O problema do determinismo!

Fatorando a gramática:

R1) S → c S
R2) S → c

R1) S → cT Adiciona-se um novo não terminal

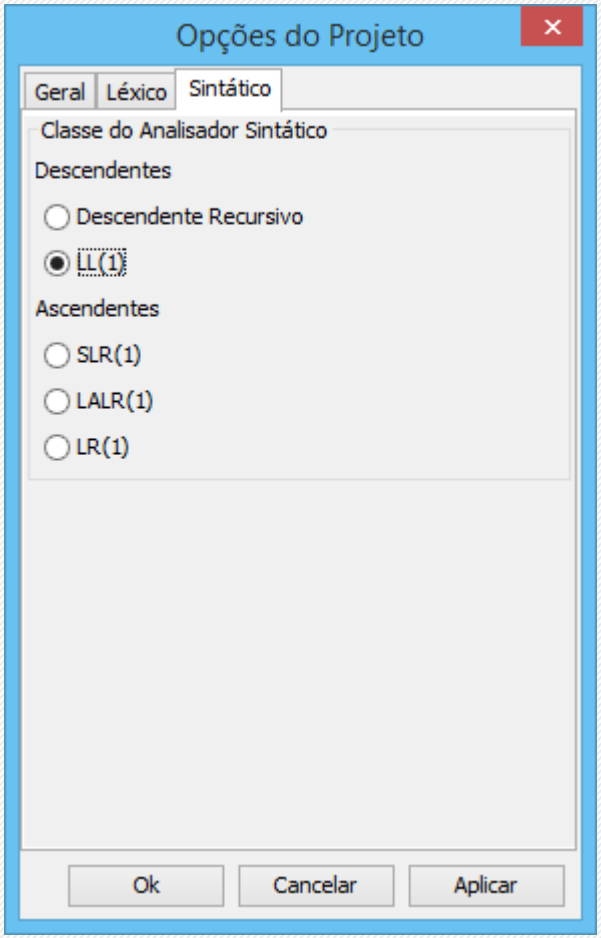
Analisadores descendentes
O problema do determinismo!

Fatorando a gramática:

R1) S → c S
R2) S → c

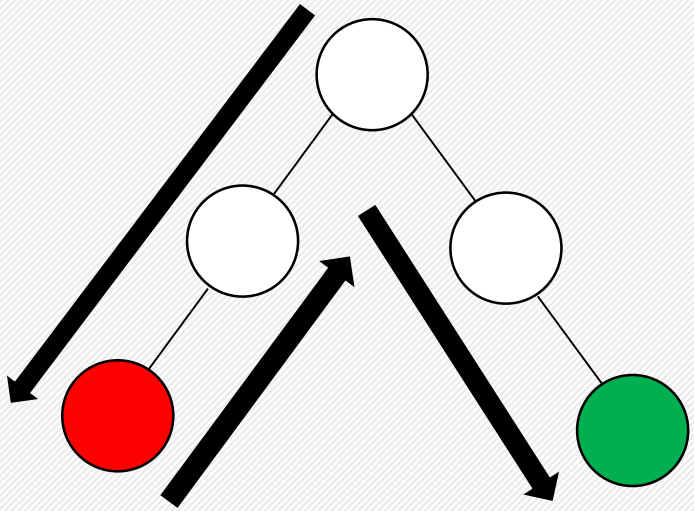
R1) S → cT Adiciona-se um novo não terminal
R2) T → S que deverá produzir as cadeias restantes
R3) T → ε

Tipos de analisadores descendentes



Analisadores descendentes recursivos

Testa diferentes possibilidades de análise sintática de entrada, retrocedendo se alguma possibilidade falhar.



Analisadores descendentes recursivos

Tokens: [a] $

R1) S → a

R2) S → [L]

R3) L → S ; L

R4) L → S

Analisadores descendentes recursivos

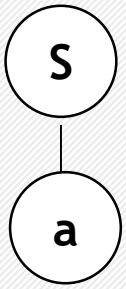
Tokens: | [| a |] | $ |

R1) S → a

R2) S → [L]

R3) L → S ; L

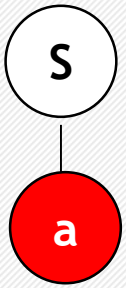
R4) L → S



Inicia-se a derivação pela raiz S

Analisadores descendentes recursivos

Tokens: [a] $



R1) S → a

R2) S → [L]

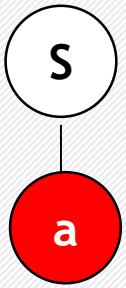
R3) L → S ; L

R4) L → S

O próximo nó (na pré-ordem) é terminal

Analisadores descendentes recursivos

Tokens: [| a |] | $



R1) S → a

R2) S → [L]

R3) L → S ; L

R4) L → S

Mas não casa (match) com a entrada

Analisadores descendentes recursivos

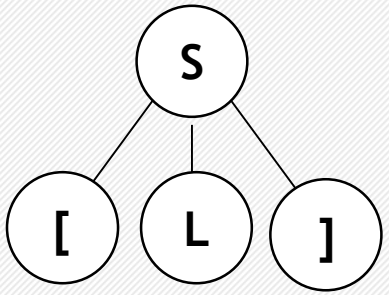
Tokens: [a] $

R1) S → a

R2) S → [L]

R3) L → S ; L

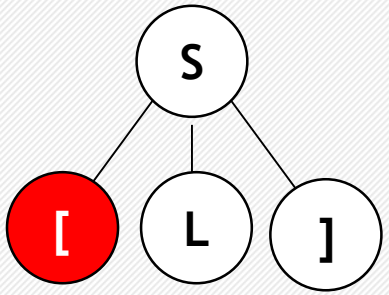
R4) L → S



Há outro caminho a partir de S...

Analisadores descendentes recursivos

Tokens: [a] $



R1) S → a

R2) S → [L]

R3) L → S ; L

R4) L → S

O primeiro nó (na pré-ordem) é terminal

Analisadores descendentes recursivos

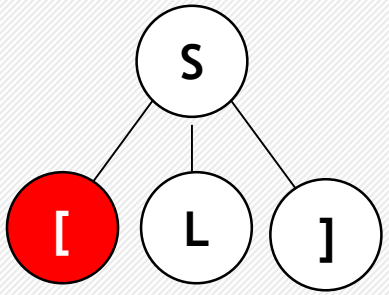
Tokens: [| a |] | $

R1) S → a

R2) S → [L]

R3) L → S ; L

R4) L → S



E casa (match) com a entrada!

Analisadores descendentes recursivos

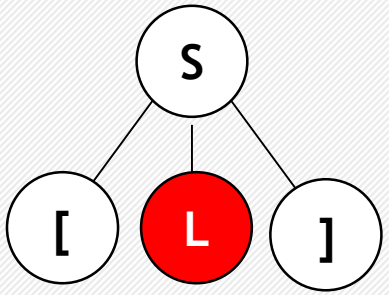
Tokens: | a |] | $ |

R1) S → a

R2) S → [L]

R3) L → S ; L

R4) L → S



O próximo nó (na pré-ordem) é não terminal

Analisadores descendentes recursivos

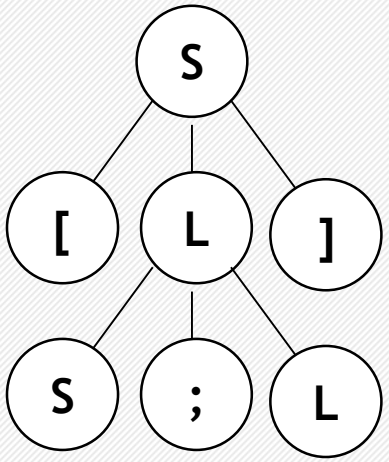
Tokens: | a |] | $ |

R1) S → a

R2) S → [L]

R3) L → S ; L

R4) L → S



Próxima produção para o nó não terminal L

Analisadores descendentes recursivos

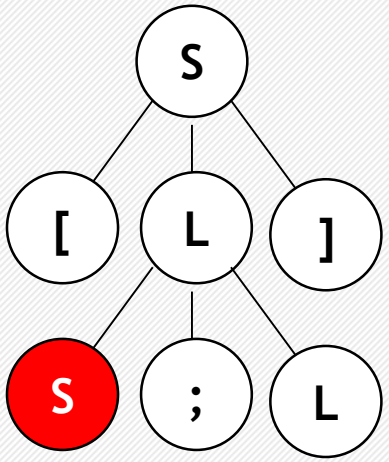
Tokens: | a |] | $ |

R1) S → a

R2) S → [L]

R3) L → S ; L

R4) L → S



O próximo nó (na pré-ordem) é não terminal

Analisadores descendentes recursivos

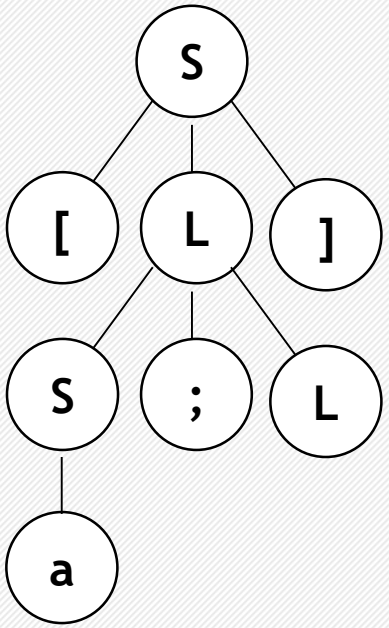
Tokens: | a |] | $ |

R1) S → a

R2) S → [L]

R3) L → S ; L

R4) L → S



Próxima produção para o nó não terminal S

Analisadores descendentes recursivos

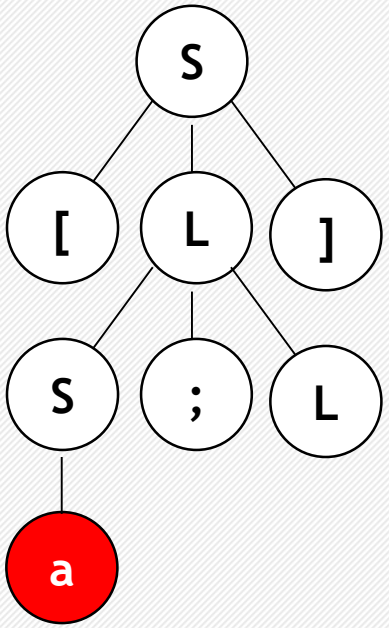
Tokens: | a |] | $ |

R1) S → a

R2) S → [L]

R3) L → S ; L

R4) L → S



O próximo nó (na pré-ordem) é terminal

Analisadores descendentes recursivos

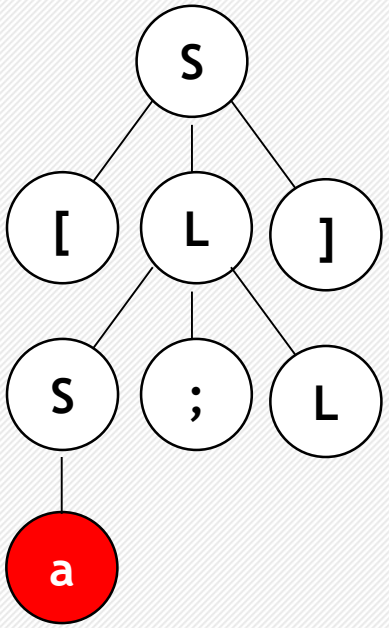
Tokens: | a |] | $ |

R1) S → a

R2) S → [L]

R3) L → S ; L

R4) L → S



E casa (match) com a entrada!

Analisadores descendentes recursivos

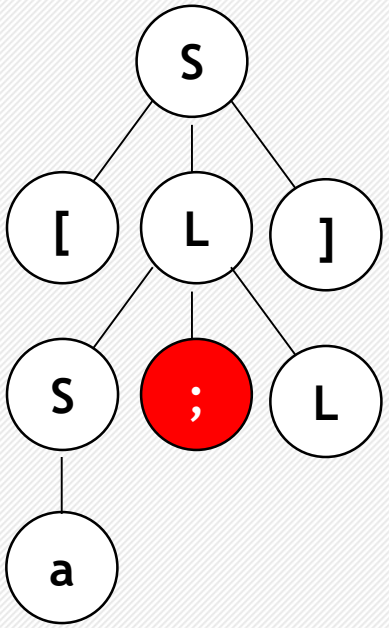
Tokens: |] | $ |

R1) S → a

R2) S → [L]

R3) L → S ; L

R4) L → S



O próximo nó (na pré-ordem) é terminal

Analisadores descendentes recursivos

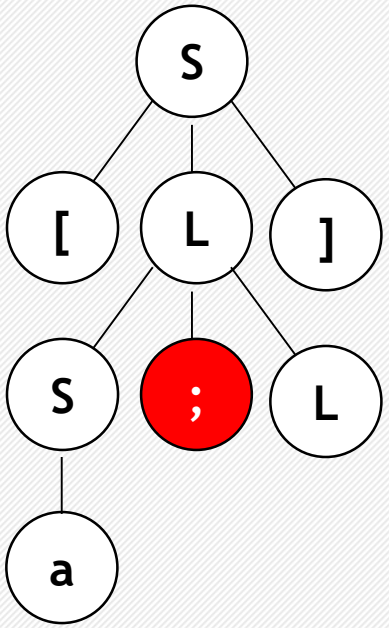
Tokens:] $

R1) S → a

R2) S → [L]

R3) L → S ; L

R4) L → S



Mas não casa (match) com a entrada

Analisadores descendentes recursivos

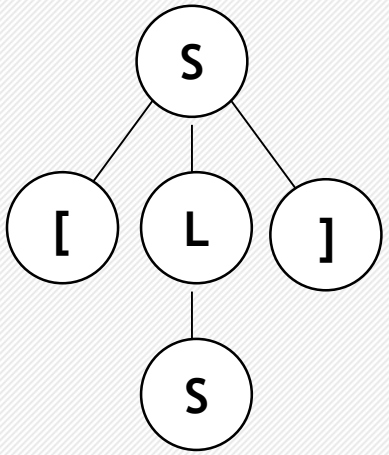
Tokens: | a |] | $ |

R1) S → a

R2) S → [L]

R3) L → S ; L

R4) L → S



Tenta-se a próxima opção de produção para L

Analisadores descendentes recursivos

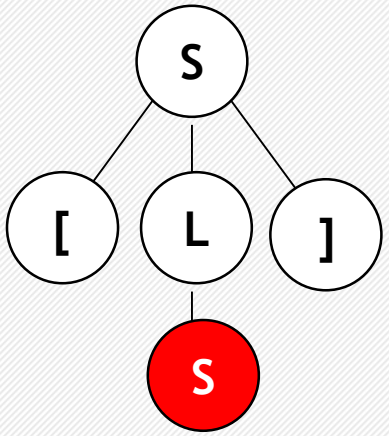
Tokens: | a |] | $ |

R1) S → a

R2) S → [L]

R3) L → S ; L

R4) L → S



O próximo nó (na pré-ordem) é não terminal

Analisadores descendentes recursivos

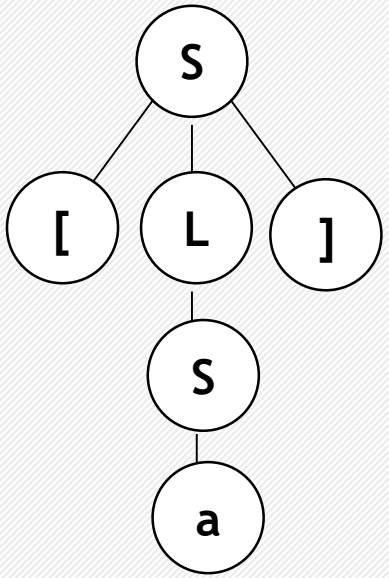
Tokens: | a |] | $ |

R1) S → a

R2) S → [L]

R3) L → S ; L

R4) L → S



Próxima produção para o nó não terminal S

Analisadores descendentes recursivos

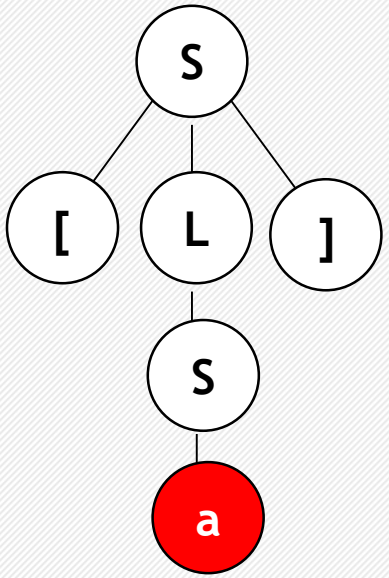
Tokens: | a |] | $ |

R1) S → a

R2) S → [L]

R3) L → S ; L

R4) L → S



O próximo nó (na pré-ordem) é terminal

Analisadores descendentes recursivos

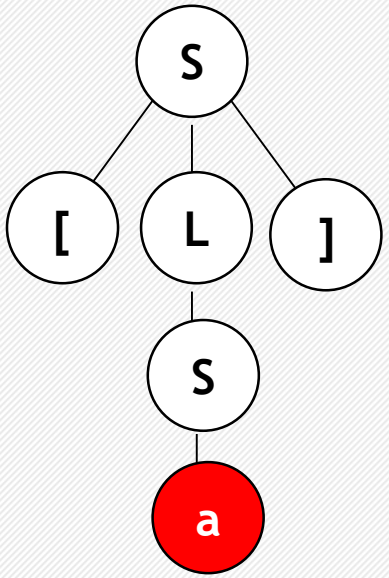
Tokens: | a |] | $ |

R1) S → a

R2) S → [L]

R3) L → S ; L

R4) L → S



E casa com a entrada

Analisadores descendentes recursivos

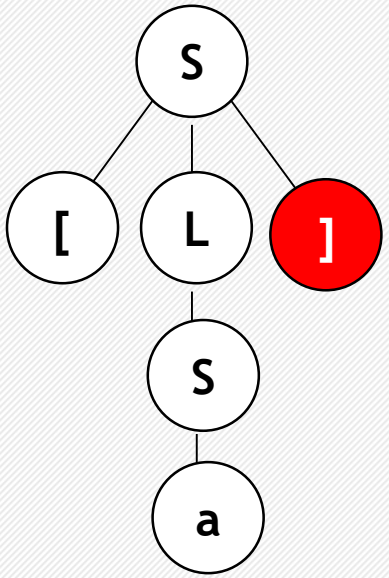
Tokens: |] | $ |

R1) S → a

R2) S → [L]

R3) L → S ; L

R4) L → S



O próximo nó (na pré-ordem) é terminal

Analisadores descendentes recursivos

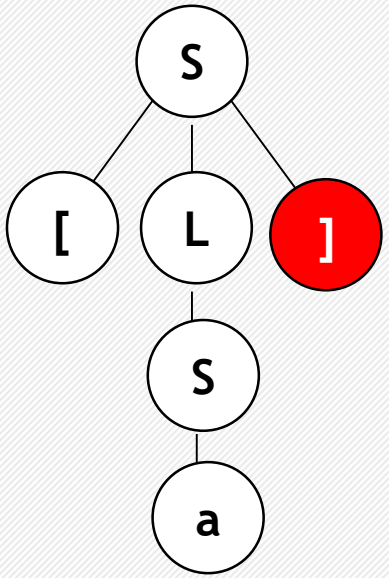
Tokens: |] | $ |

R1) S → a

R2) S → [L]

R3) L → S ; L

R4) L → S



E casa com a entrada

Analisadores descendentes recursivos

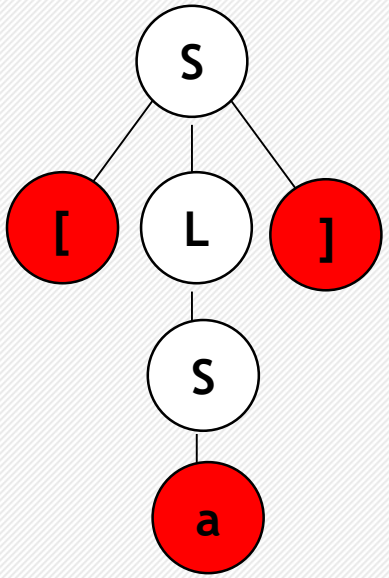
Tokens: $

R1) S → a

R2) S → [L]

R3) L → S ; L

R4) L → S



Não há mais não terminais...

Analisadores descendentes recursivos

ASDR – Analisador Sintático Descendente Recursivo

```
<prog> ::= programa inicio <corpo> fim ;
<corpo> ::= <cmd><corpo> | î ;
<cmd> ::= id atrib <exp> | escreva "(" <exp> ")"
<exp> ::= id | num
```

```
void main () {
  token *prox_tk;
  prox_tk = analise_lexica();
  prog ();
}
```

Analisadores descendentes recursivos

ASDR

```
<prog> ::= programa inicio <corpo> fim ;
<corpo> ::= <cmd><corpo> | î ;
<cmd> ::= id atrib <exp> | escreva "(" <exp> ")"
<exp> ::= id | num
```

```
void prog () {
  check (PROGRAMA);
  check (INICIO);
  corpo ();
  check (FIM);
}
```

Analisadores descendentes recursivos

ASDR

```
<prog> ::= programa inicio <corpo> fim ;
<corpo> ::= <cmd><corpo> | î ;
<cmd> ::= id atrib <exp> | escreva "(" <exp> ")"
<exp> ::= id | num
```

```
void corpo() {
  // first de CMD
  if (prox_tk == ID || prox_tk == ESCREVA)
{
    cmd ();
    corpo ();
  }
  // sem else pois aceita vazio
}
```

ASDR

```
<prog> ::= programa inicio <corpo> fim ;
<corpo> ::= <cmd><corpo> | î ;
<cmd> ::= id atrib <exp> | escreva "(" <exp> ")"
<exp> ::= id | num
```

```
void cmd () {
  if (prox_tk == ID) {
     prox_tk++;
     check (ATRIB);
     exp ();
  } else
  if (prox_tk == ESCREVA) {
     prox_tk++;
     check (AB_PAR);
     exp();
     check (FE_PAR);
  }
  else {
     printf ("erro: esperado ID ou ESCREVA")
     exit ();
  }
}
```

Analisadores descendentes recursivos

ASDR

```
<prog> ::= programa inicio <corpo> fim ;
<corpo> ::= <cmd><corpo> | î ;
<cmd> ::= id atrib <exp> | escreva "(" <exp> ")"
<exp> ::= id | num
```

```
void exp () {
  if (prox_tk == ID) {
     prox_tk++;
  } else
  if (prox_tk == num) {
     prox_tk++;
  }
  else {
     printf ("erro")
     exit ();
  }
}
```

Analisadores descendentes preditivos LL(1)

Tentam prever a construção seguinte na cadeia de entrada com base em uma ou mais marcas de verificação à frente.

L – indica que o processamento ocorre da esquerda (Left) para a direita.

L – indica que é realizada uma derivação mais à esquerda (Left) para a cadeia de entrada.

1 – indica que é verificado 1 símbolo à frente.

LL(1): Left-left 1

Analisadores descendentes preditivos LL(1)

Como a construção é prevista?

Através dos conjuntos

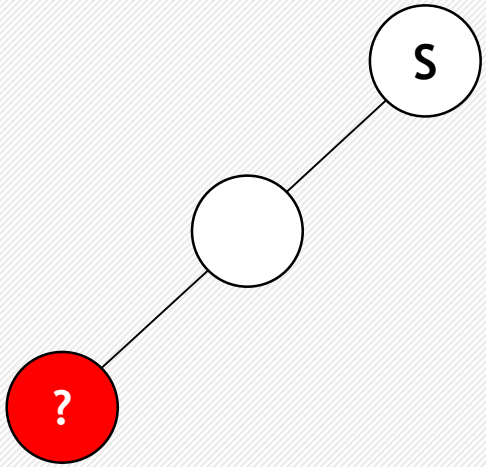
FIRST {}

FOLLOW {}

Analisadores descendentes preditivos LL(1)

FIRST {}

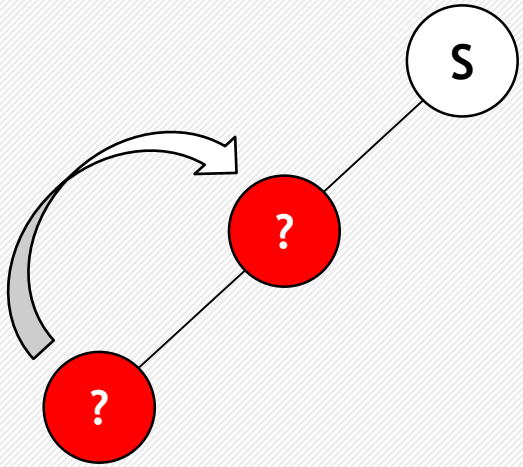
Conjunto de terminais que podem aparecer na posição mais à esquerda.



Analisadores descendentes preditivos LL(1)

FOLLOW {}

Conjunto de terminais que podem aparecer imediatamente após.



Analisadores descendentes preditivos LL(1)

S → XYZ

X → aXb | ε

Y → cYZcX | d

Z → eZYe | f

FIRST(X) = {a, ε}

FIRST(Y) = {c,d}

FIRST(Z) = {e,f}

FIRST(S) = {a,c,d}

FOLLOW(X) = {c,d,b,e,f}

FOLLOW(Y) = {e,f}

FOLLOW(Z) = {$,c,d}

FOLLOW(S) = {$}

Analisadores descendentes preditivos LL(1)

S → XYZ

X → aXb | ε

Y → cYZcX | d

Z → eZYe | f

FIRST(X) = {a, ε}

FIRST(Y) = {c,d}

FIRST(Z) = {e,f}

FIRST(S) = {a,c,d}

FOLLOW(X) = {c,d,b,e,f}

FOLLOW(Y) = {e,f}

FOLLOW(Z) = {$,c,d}

FOLLOW(S) = {$}

Analisadores descendentes preditivos LL(1)

S → XYZ

X → aXb | ε

Y → cYZcX | d

Z → eZYe | f

FIRST(X) = {a, ε}

FIRST(Y) = {c,d}

FIRST(Z) = {e,f}

FIRST(S) = {a,c,d}

FOLLOW(X) = {c,d,b,e,f}

FOLLOW(Y) = {e,f}

FOLLOW(Z) = {$,c,d}

FOLLOW(S) = {$}

Analisadores descendentes preditivos LL(1)

S → XYZ

X → aXb | ε

Y → cYZcX | d

Z → eZYe | f

FIRST(X) = {a, ε}

FIRST(Y) = {c,d}

FIRST(Z) = {e,f}

FIRST(S) = {a,c,d}

FOLLOW(X) = {c,d,b,e,f}

FOLLOW(Y) = {e,f}

FOLLOW(Z) = {$,c,d}

FOLLOW(S) = {$}

Analisadores descendentes preditivos LL(1)

S → XYZ

X → aXb | ε

Y → cYZcX | d

Z → eZYe | f

FIRST(X) = {a, ε}

FIRST(Y) = {c,d}

FIRST(Z) = {e,f}

FIRST(S) = {a,c,d}

FOLLOW(X) = {c,d,b,e,f}

FOLLOW(Y) = {e,f}

FOLLOW(Z) = {$,c,d}

FOLLOW(S) = {$}

Analisadores descendentes preditivos LL(1)

S → XYZ

X → aXb | ε

Y → cYZcX | d

Z → eZYe | f

FOLLOW(Y)

FIRST(X) = {a, ε}

FIRST(Y) = {c,d}

FIRST(Z) = {e,f}

FIRST(S) = {a,c,d}

FOLLOW(X) = {c,d,b,e,f}

FOLLOW(Y) = {e,f}

FOLLOW(Z) = {$,c,d}

FOLLOW(S) = {$}

Analisadores descendentes preditivos LL(1)

S → XYZ

X → aXb | ε

Y → cYZcX | d

Z → eZYe | f

FOLLOW(Y)

S->X Y Z
X(cYZcX)Z

FIRST(X) = {a, ε}

FIRST(Y) = {c,d}

FIRST(Z) = {e,f}

FIRST(S) = {a,c,d}

FOLLOW(X) = {c,d,b,e,f}

FOLLOW(Y) = {e,f}

FOLLOW(Z) = {$,c,d}

FOLLOW(S) = {$}

Analisadores descendentes preditivos LL(1)

S → XYZ

X → aXb | ε

Y → cYZcX | d

Z → eZYe | f

FIRST(X) = {a, ε}

FIRST(Y) = {c,d}

FIRST(Z) = {e,f}

FIRST(S) = {a,c,d}

FOLLOW(X) = {c,d,b,e,f}

FOLLOW(Y) = {e,f}

FOLLOW(Z) = {$,c,d}

FOLLOW(S) = {$}

Analisadores descendentes preditivos LL(1)

S → XYZ ← $

X → aXb | ε

Y → cYZcX | d

Z → eZYe | f

FIRST(X) = {a, ε}

FIRST(Y) = {c,d}

FIRST(Z) = {e,f}

FIRST(S) = {a,c,d}

FOLLOW(X) = {c,d,b,e,f}

FOLLOW(Y) = {e,f}

FOLLOW(Z) = {$,c,d}

FOLLOW(S) = {$}

Analisadores descendentes preditivos LL(1)

S → XYZ

X → aXb | ε

Y → cYZcX | d

Z → eZYe | f

FIRST(X) = {a, ε}

FIRST(Y) = {c,d}

FIRST(Z) = {e,f}

FIRST(S) = {a,c,d}

FOLLOW(X) = {c,d,b,e,f}

FOLLOW(Y) = {e,f}

FOLLOW(Z) = {$,c,d}

FOLLOW(S) = {$}

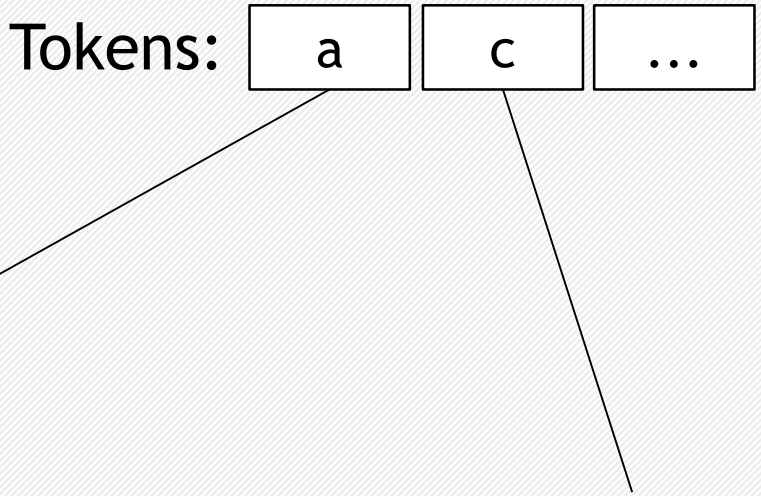
Analisadores descendentes preditivos LL(1)

S → XYZ

X → aXb | ε

Y → cYZcX | d

Z → eZYe | f



FIRST(X) = {a, ε}

FIRST(Y) = {c,d}

FIRST(Z) = {e,f}

FIRST(S) = {a,c,d}

FOLLOW(X) = {c,d,b,e,f}

FOLLOW(Y) = {e,f}

FOLLOW(Z) = {$,c,d}

FOLLOW(S) = {$}

Referências

- AHO, Alfred V.; VIEIRA, Daniel. Compiladores: princípios, técnicas e ferramentas. 2. ed. São Paulo, SP: Pearson/Addison Wesley, c2007, c2008. x, 634 p. ISBN 9788588639249.
- LOUDEN, Kenneth C. Compiladores: princípios e práticas. São Paulo, SP: Pioneira Thomson Learning, 2004. 569 p. ISBN 8522104220
- PRICE, Ana Maria de Alencar; TOSCANI, Simão Sirineo. Implementação de linguagens de programação: compiladores. 3. ed. Porto Alegre, RS: Sagra, 2005. 197 p. (Serie Livros Didaticos 9) ISBN 8524106395.