Análise Sintática

Parte II – Análise Sintática Top-Down

Análise Top-Down

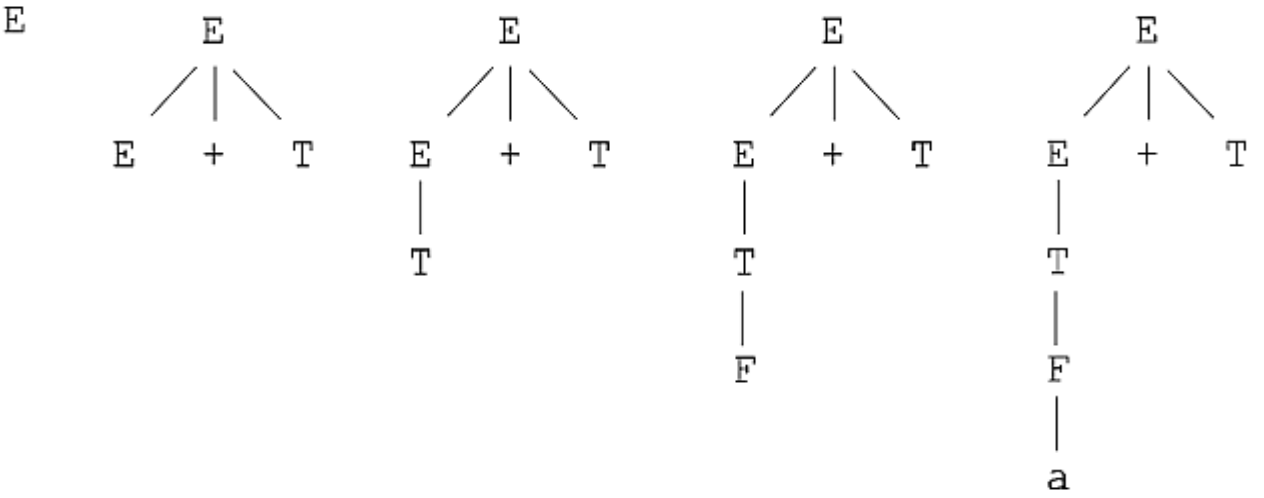
- A análise top-down (ou descendente) de uma sentença pode ser vista como uma tentativa de construir uma árvore de derivação em pré-ordem (da esquerda para a direita) para a sentença em questão:
  - 1) cria a raiz e, a seguir
  - (2) cria as subárvores filhas, da esquerda para a direita
- Esse processo produz uma *derivação mais à esquerda* da sentença em análise

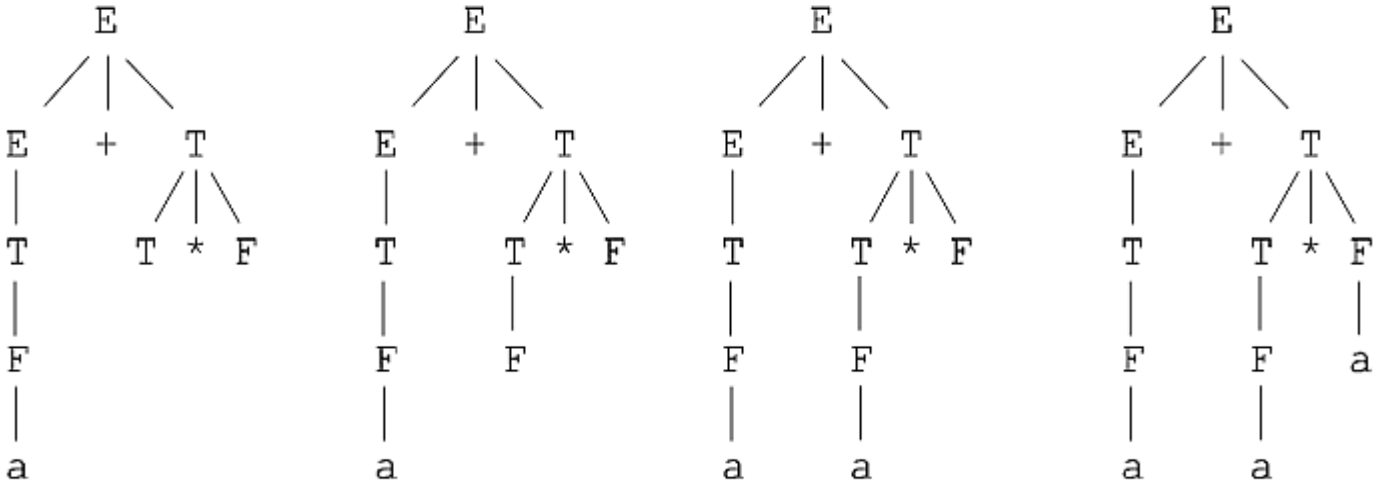
LEMBRANDO...

Gramática:

1. **E → E + T**
2. **E → T**
3. **T → T*F**
4. **T → F**
5. **F → (E)**
6. **F → a**

Sentença: a + a * a





E ⇒ E+T ⇒ T+T⇒ a+T ⇒ a+T*F ⇒ a+F*F ⇒ a+a*F ⇒ a+a*a

Análise Top-Down

- Há duas formas de analisadores top-down:
  - **Analisadores com retrocesso**: testa diferentes possibilidades de análise sintática da entrada, retrocedendo se alguma possibilidade falhar
  - **Analisadores preditivos**: tentam prever a construção seguinte na cadeia de entrada com base em uma ou mais marcas de verificação à frente

Análise Recursiva com Retrocesso

- A análise recursiva com retrocesso faz a expansão da árvore de derivação a partir da raiz, expandindo sempre o não-terminal mais à esquerda
- Quando existe mais de uma regra de produção para o não-terminal a ser expandido, a opção escolhida é função do símbolo corrente na cadeia de entrada
  - Se o token de entrada não define univocamente a produção a ser usada, então todas as alternativas serão tentadas
  - Até o sucesso ou até que todas falhem

Análise Recursiva com Retrocesso

Exemplo

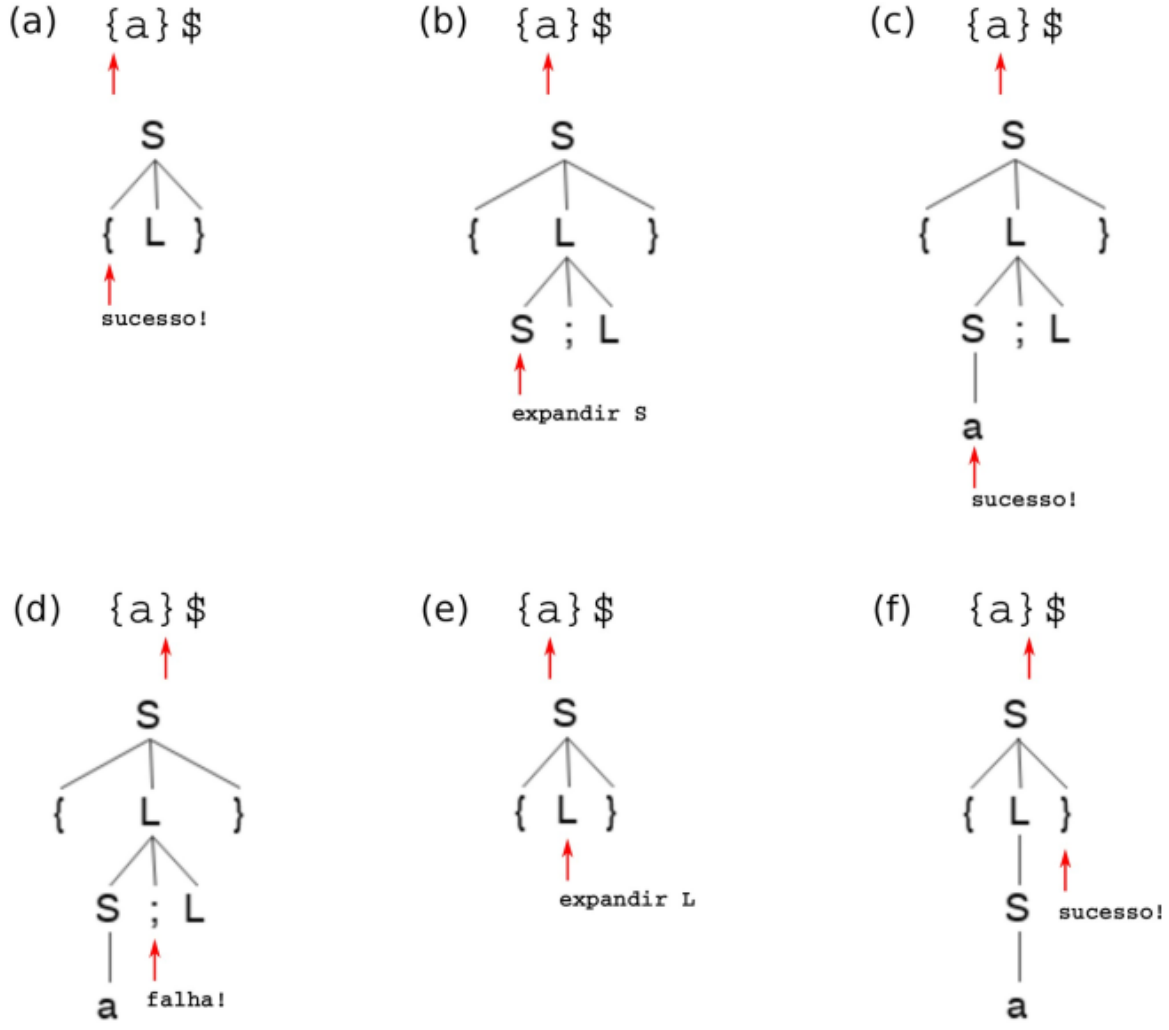
Gramática:

S → a | { L }

L → S;L | S

Sentença:

‘{a}’



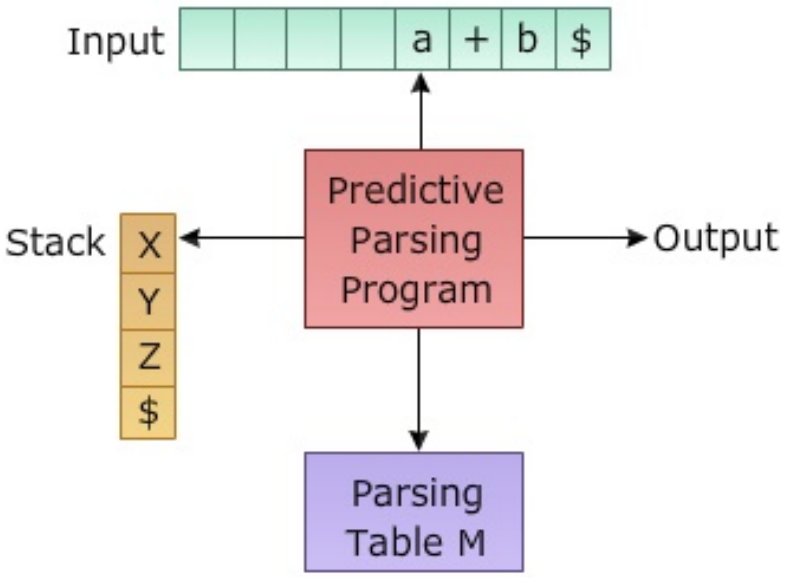
Análise Recursiva com Retrocesso

- Embora os analisadores com retrocesso mais poderosos que os preditivos, eles são lentos e requerem tempo exponencial para a execução
- Na prática, eles **não são adequados** para a implementação de compiladores

Análise Preditiva Tabular

- É possível construir um analisador preditivo não recursivo que utiliza uma pilha explícita em vez de chamadas recursivas
- Esse tipo de analisador implementa um autômato de pilha controlado por uma tabela de análise
  - Por isso o nome *tabular*

Análise Preditiva Tabular



INPUT: Contém string a ser analisada com '$' como marcador final

STACK (Pilha): Contém sequência de símbolos gramaticais com $ como marcador inferior. Inicialmente, a pilha contém apenas '$'

TABELA DE PARSING: Uma matriz bidimensional M[A, a], onde A é um não terminal e a é um terminal

O analisador preditivo usa um algoritmo diretor para analisar a string de entrada usando das entradas da tabela

Análise Preditiva Tabular

- O algoritmo diretor é dado na literatura
  - Fixo
  - Depende das entradas e da tabela
- A tabela deve ser gerada para cada gramática
  - Algumas considerações devem ser levadas em conta
  - Alguns passos são necessários para a sua construção

Gramáticas LL(1)

- As gramáticas que podem ser analisadas sintaticamente usando analisadores top-down são chamadas gramáticas LL(x)
  - x refere-se à quantidade de tokens posteriores ao símbolo atual que são usados para tomar decisões na análise
  - *Lookahead*
  - Gramáticas LL(1) são as mais populares, pela necessidade de verificar somente **um token** posterior para a análise

Gramáticas LL(1)

- Gramáticas LL(1) são as mais populares
  - verificam somente **um token** adiante
- O primeiro L de LL(1) significa varredura da entrada da esquerda para a direita (*left to right*)
- O segundo, a produção de uma derivação mais à esquerda (*left linear*);
- (1) um único símbolo de *lookahead* a cada passo

Gramáticas LL(1)

As gramáticas LL(1) possuem algumas propriedades importantes:

1. Nenhuma gramática **ambígua ou recursiva à esquerda** pode ser LL(1).

   Pela forma como a técnica de construção descendente opera, ela não pode ser aplicada a gramáticas com produções recursivas à esquerda

   - recursão infinita na análise pela tentativa de expandir sempre a mesma regra sem consumir símbolo algum da entrada

2. Em uma produção $A \rightarrow \alpha|\beta$, α e β não derivam, ao mesmo tempo, cadeias começando pelo mesmo terminal '**a**', qualquer seja '**a**'.

   - Primeiros terminais deriváveis devem ser capazes de identificar, univocamente, a produção a ser aplicada na análise

Gramáticas LL(1)

Exemplo 1:

A gramática:

E → E + T | T

T → T * F | F

F → (E) | id

É LL(1)??

Gramáticas LL(1)

Exemplo 1:

A gramática:

E → E + T | T

T → T * F | F

F → (E) | id

É LL(1)??

NÃO!

Não é uma gramática LL(1) devido ao fato de que existe uma recursão à esquerda na primeira produção.

Não atende à regra 1...

Gramáticas LL(1)

Exemplo 2:

E a gramática:

CMD → if EXPR then cmd else cmd |

if EXPR then cmd

??

Gramáticas LL(1)

Exemplo 2:

E a gramática:

CMD → if EXPR then cmd else cmd |

if EXPR then cmd

??

TAMBÉM NÃO!

CMD deriva para duas regras de produção que começam com o mesmo terminal **if**

Não atende à regra 2...

Gramáticas LL(1)

- Considerando os exemplos anteriores, o primeiro passo para a construção de um analisador preditivo tabular é reescrever a gramática para que atenda os requisitos
- Duas técnicas padrão:

  1) Eliminação de recursão à esquerda e

  2) Fatoração à esquerda

Eliminação de recursão à esquerda

- É possível transformar uma produção recursiva à esquerda em uma recursiva à direita que descreve as mesmas sentenças através da seguinte técnica:
  - Sejam β e δ duas sequências de símbolos que não sejam iniciadas pelo símbolo não-terminal A e sejam as produções para A:

    A → A β | δ

  - As produções acima podem ser descritas pelas produções recursivas à direita:

    A → δ A'

    A'→ β A'

    A'→ ε

Eliminação de recursão à esquerda

Generalização:

Agrupar as produções com recursão à esquerda como:

$A \rightarrow A\alpha_1 \mid A\alpha_2 \mid \ldots \mid A\alpha_m \mid \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$

onde nenhum β_i começa com um A.

Em seguida, substituímos as produções recursivas por:

$A \rightarrow \beta_1\ A' \mid \beta_2\ A' \mid \ldots \mid \beta_n\ A'$

$A' \rightarrow \alpha_1\ A' \mid \alpha_2\ A' \mid \ldots \mid \alpha_n\ A' \mid \varepsilon$

Eliminação de recursão à esquerda

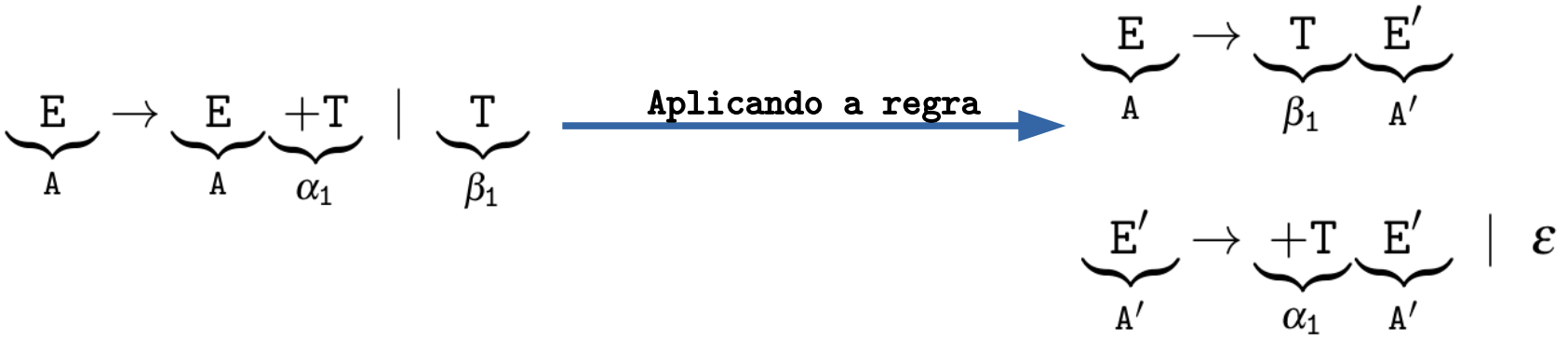
Exemplo

E → E + T | T

T → T * F | F

F → (E) | id

Remoção de recursão à esquerda para a produção **E → E + T | T**



Eliminação de recursão à esquerda

Fazemos o mesmo para **T → T * F | F** …

Assim, eliminando as recursões para E e T, obtemos uma **gramática equivalente** que não contém recursões à esquerda:

$$
\begin{aligned}
E &\rightarrow T\ E' \\
E' &\rightarrow +\ T\ E' \mid \varepsilon \\
T &\rightarrow F\ T' \\
T' &\rightarrow *\ F\ T' \mid \varepsilon \\
F &\rightarrow (\ E\ ) \mid \mathrm{id}
\end{aligned}
$$

Fatoração à esquerda

- A fatoração à esquerda é requerida quando duas ou mais escolhas de regras gramaticais compartilham uma cadeia de prefixo comum, como na regra:

  CMD → **if EXPR then cmd** else cmd |
  if EXPR then cmd

- Um analisador LL(1) não pode diferenciar entre as escolhas de produções em situações desse tipo

Fatoração à esquerda

- Dada uma produção:

$$A \rightarrow \alpha\beta \mid \alpha\gamma$$

com as duas opções iniciadas com α, ela pode ser reescrita como:

$$A \rightarrow \alpha A'$$
$$A' \rightarrow \beta \mid \gamma$$

OBS: Para a fatoração funcionar adequadamente, precisamos garantir que α seja de fato a cadeia mais longa compartilhada no lado direito.

Fatoração à esquerda

Exemplo 1:

Considere a gramática para declarações if:

```
CMD → if EXPR then cmd else cmd |
      if EXPR then cmd
```

- Neste caso o prefixo comum α é **if EXPR then cmd**, β corresponde à **else cmd** e γ igual à **ε**

- Assim, obtém-se:

```
CMD      → if EXPR then cmd ELSE_CMD
ELSE_CMD → else cmd | ε
```

Fatoração à esquerda

Exemplo 2:

```
decl       → atrib-decl | ativ-decl | outra
atrib-decl → id := exp
ativ-decl  → id(exp-list)
```

- Essa gramática não é LL(1) pois id é compartilhada como primeira marca por atribuicao-decl e ativacao-decl
- **Prefixo comum de forma indireta!**
- Infelizmente, a gramática não pode ser fatorada à esquerda

Fatoração à esquerda

1. Necessário reescrever a gramática:

```
decl → id := exp |
       id(exp-list) |
       outra
```

2. Fatoração à esquerda:

```
decl → id decl' |
       outra
decl' → := exp |
        (exp-lista)
```

Que agora é LL(1).

Ok!
Já vimos os dois passos necessários para conseguirmos uma gramática LL(1).

Depois de fazer isso, podemos avançar com a construção da tabela preditiva.

First e Follow

- Para a construção da tabela de análise é necessário a computação dos conjuntos First e Follow (Primeiro e Seguinte)

First

- First(X) é o conjunto de todos os **símbolos terminais** que podem iniciar qualquer string derivada de X
- Regras First:

1. Se X $\rightarrow$ aYZ, onde a é terminal, então First(a) está em First(X). O conjunto First de um terminal, é o próprio terminal.
2. Se X $\rightarrow \varepsilon$ é uma produção, então adicione ε a First(X).
3. Se X $\rightarrow$ YZW, onde Y, Z e W são não-terminais, First(X) contém First(Y)-$\{\varepsilon\}$. Se Y pode ser derivado para ε, ou seja, é *anulável*, então First(X) também contém First(Z)-$\{\varepsilon\}$. Se Y e Z são anuláveis, então First(W)-$\{\varepsilon\}$ também é acrescido em First(X). Se todos os símbolos à direita forem anuláveis, ε deve estar em First(X).

First

Exemplo

$$
\begin{aligned}
E &\rightarrow TE' \\
E' &\rightarrow \vee TE' \mid \varepsilon \\
T &\rightarrow FT' \\
T' &\rightarrow \wedge FT' \mid \varepsilon \\
F &\rightarrow \neg F \mid id
\end{aligned}
$$

Ver resolução na aula gravada ou na Apostila da disciplina.

Follow

- O conjunto Follow é definido para **símbolos não-terminais**
- Sendo X um não-terminal, Follow(X) é o conjunto de terminais que podem aparecer imediatamente à direita de X em alguma forma sentencial
- Regras:

1. Considerando S como o símbolo inicial da gramática, e $ como o marcador de fim de sentença, então $ está em Follow(S).
2. Se existe produção do tipo $A \rightarrow \alpha X \beta$, então todos os símbolos de First(β), exceto ε, fazem parte de Follow(X).
3. Se existe produção do tipo $A \rightarrow \alpha X$ ou $A \rightarrow \alpha X \beta$, com β podendo derivar em ε, então todos os símbolos que estiverem em Follow(A) fazem parte de Follow(X).

Observe que ε jamais fará parte de algum conjunto Follow.

Follow

Exemplo

$$
\begin{aligned}
E &\rightarrow TE' \\
E' &\rightarrow \vee TE' \mid \varepsilon \\
T &\rightarrow FT' \\
T' &\rightarrow \wedge FT' \mid \varepsilon \\
F &\rightarrow \neg F \mid id
\end{aligned}
$$

Ver resolução na aula gravada ou na Apostila da disciplina.

Construção da tabela

- Após garantir que a gramática foi reescrita para se tornar LL(1), e com os conjuntos First e Follow já calculados, pode-se então construir a tabela de análise sintática preditiva
  - Há um Algoritmo próprio para isso

- **Relembrando**: A tabela sintática é uma matriz bidimensional indexada por não-terminais (linhas) e terminais, incluindo o '$' (colunas)
  - Em cada célula há a escolha de produções que serão utilizados em cada caso

Construção da tabela: Algoritmo

Algorithm 1: Construção da tabela sintática preditiva

Input : Gramática G
Output : Tabela de análise M

```
1  begin
2      foreach produção A → α em G do
3          foreach terminal x de First(A) do
4              Adicione a produção A → α a M[A, x]. {Regra 1}
5          end
6          if ε ∈ First(A) then
7              Adicione A → α a M[A,b], para cada b em Follow(A). {Regra 2}
8          end
9      end
10 end
```

Construção da tabela

- Exemplo:

$E \rightarrow TE'$
$E' \rightarrow \vee TE' \mid \varepsilon$
$T \rightarrow FT'$
$T' \rightarrow \wedge FT' \mid \varepsilon$
$F \rightarrow \neg F \mid id$

First	Follow
E = {¬, id}	E = {$}
E’ = {∨, ε}	E’ = {$}
T = {¬, id}	T = {∨, $}
T’ = {∧, ε}	T’ = {∨, $}
F = {¬, id}	F = {∨, ∧, $}

Aplicando o Algoritmo do slide anterior, obtemos:

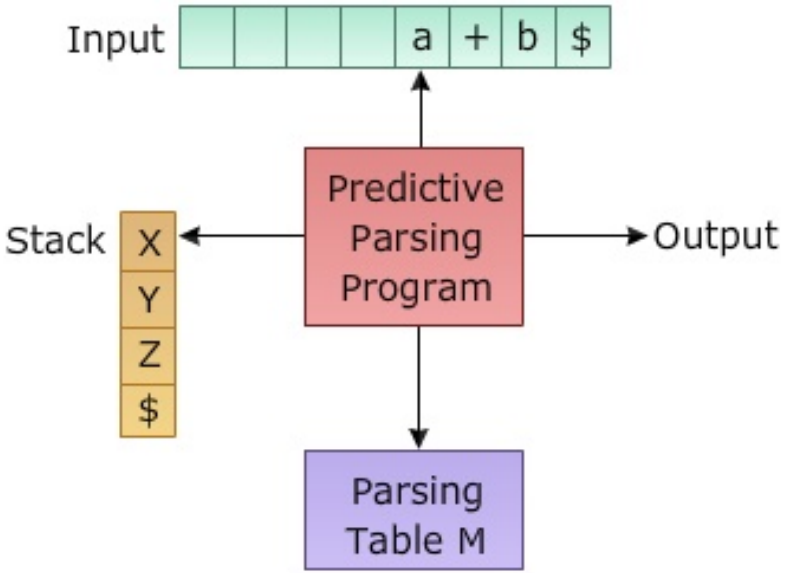
	id	$\vee$	$\wedge$	$\neg$	$
E	$E \rightarrow TE'$			$E \rightarrow TE'$	
E’		$E' \rightarrow \vee TE'$			$E' \rightarrow \varepsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$	
T’		$T' \rightarrow \varepsilon$	$T' \rightarrow \wedge FT'$		$T' \rightarrow \varepsilon$
F	$F \rightarrow id$			$F \rightarrow \neg F$	

Ver resolução na aula gravada ou na Apostila da disciplina.

Construção da tabela

- **Observação:**
  - Quando um símbolo a permite várias opções de produção para um não-terminal N, existe um conflito nesse símbolo para esse não-terminal
    - *Várias entradas para a mesma célula da tabela*
  - Os conflitos podem ser causados por gramáticas ambíguas, mas também existem gramáticas inequívocas que causam conflitos
  - Nestes casos é possível reescrever essa gramática para evitar conflitos, mas deve-se notar que isso nem sempre é possível

Análise Preditiva Tabular: relembrando...



Ok, após todos esses passos temos a matriz preditiva.

Falta vermos como aplicar o algoritmo usando a tabela sobre uma sentença de entrada.

Algoritmo de Análise Sintática Top-Down

O algoritmo que guia os movimentos de um analisador preditivo não recursivo é como segue:

Algorithm 2: Algoritmo de análise sintática Top-Down

Input : Sentença s (fita) e a tabela M para a gramática G
Output : Uma derivação mais à esquerda de s, se s está em $L(G)$, ou uma indicação de erro

```
1  Pilha contém $S
2  Fita de entrada contém s$
3  begin
4      X = topo da pilha;
5      a = primeiro símbolo da sentença (fita);
6      repeat
7          if X é terminal then
8              if X = a then
9                  desempilha X e avança a leitura da sentença;
10             else
11                 erro();
12             end
13         else
14             if M[X,a] = X → Y1Y2...Yk then
15                 desempilha X;
16                 empilha Y1Y2...Yk com Y1 no topo (produção ao contrário);
17                 imprime a produção X→ Y1Y2...Yk
18             else
19                 erro();
20             end
21         end
22     until X = $;
23 end
```

Algoritmo de Análise Sintática Top-Down

Exemplo:

Processamento de

‘id ∨ id ∧ id’

Gramática (LL(1))

$$
\begin{aligned}
E &\rightarrow TE' \\
E' &\rightarrow \vee TE' \mid \varepsilon \\
T &\rightarrow FT' \\
T' &\rightarrow \wedge FT' \mid \varepsilon \\
F &\rightarrow \neg F \mid id
\end{aligned}
$$

Algoritmo de Análise Sintática Top-Down

Exemplo:

Processamento de

‘id ∨ id ∧ id’

Gramática (LL(1))

$E \rightarrow TE'$

$E' \rightarrow \vee TE' \mid \varepsilon$

$T \rightarrow FT'$

$T' \rightarrow \wedge FT' \mid \varepsilon$

$F \rightarrow \neg F \mid id$

Tabela preditiva:

	id	$\vee$	$\wedge$	$\neg$	$
E	$E \rightarrow TE'$			$E \rightarrow TE'$	
E’		$E' \rightarrow \vee TE'$			$E' \rightarrow \varepsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$	
T’		$T' \rightarrow \varepsilon$	$T' \rightarrow \wedge FT'$		$T' \rightarrow \varepsilon$
F	$F \rightarrow id$			$F \rightarrow \neg F$	

	id	$\vee$	$\wedge$	$\neg$	$
E	$E \to TE'$			$E \to TE'$	
E'		$E' \to \vee TE'$			$E' \to \varepsilon$
T	$T \to FT'$			$T \to FT'$	
T'		$T' \to \varepsilon$	$T' \to \wedge FT'$		$T' \to \varepsilon$
F	$F \to id$			$F \to \neg F$	

Pilha	**Entrada**	**Ação**
$\underline{E}	$\underline{id}\vee id\wedge id$	desempilha E, empilha E'T, imprime $E \to TE'$
$E'\underline{T}	$\underline{id}\vee id\wedge id$	desempilha T, empilha T'F, imprime $T \to FT'$
$E'T'\underline{F}	$\underline{id}\vee id\wedge id$	desempilha F, empilha id, imprime $F \to id$
$E'T'\underline{id}	$\underline{id}\vee id\wedge id$	desempilha id, avança leitura sentença
$E'\underline{T'}	$\underline{\vee}id\wedge id$	desempilha T', empilha ε, imprime $T' \to \varepsilon$
$\underline{E'}	$\underline{\vee}id\wedge id$	desempilha E', empilha E'T$\vee$, imprime $E' \to \vee TE'$
E'T$\underline{\vee}$	$\underline{\vee}id\wedge id$	desempilha $\vee$, avança leitura sentença
E'$\underline{T}$	$\underline{id}\wedge id$	desempilha T, empilha T'F, imprime $T \to FT'$
$E'T'\underline{F}	$\underline{id}\wedge id$	desempilha F, empilha id, imprime $F \to id$
$E'T'\underline{id}	$\underline{id}\wedge id$	desempilha id, avança leitura sentença
$E'\underline{T'}	$\underline{\wedge}id$	desempilha T', empilha T'F$\wedge$, imprime $T' \to \wedge$ FT'
$E'T'F$\underline{\wedge}$	$\underline{\wedge}id$	desempilha $\wedge$, avança leitura sentença
$E'T'\underline{F}	$\underline{id}$	desempilha F, empilha id, imprime $F \to id$
$E'T'\underline{id}	$\underline{id}$	desempilha id, avança leitura sentença
$E'\underline{T'}	$	desempilha T', empilha ε, imprime $T' \to \varepsilon$
$	$	**sentença aceita!**

Recuperação de Erros

- Uma célula em branco na linha T, coluna x da tabela de análise LL(1) indica que a análise não espera ver o símbolo x, portanto, há um erro de sintaxe
- Como deve ser tratado o erro?
  - Pode-se apenas lançar uma exceção e parar a análise, mas isso não é muito amigável para o usuário
  - É melhor imprimir uma mensagem de erro e recuperar-se do erro, para que outros erros de sintaxe possam ser encontrados na mesma compilação
    - Por isso, geralmente um único erro acaba gerando diversos erros na compilação

Recuperação de Erros

- Uma forma padrão de recuperação de erros em analisadores top-down é denominada **modo pânico**
- O nome advém do fato que o manipulador de erros consome um número de símbolos de entrada até encontrar um token de sincronização
  - O conjunto de tokens de sincronização para um não-terminal A é formato pelos terminais em Follow(A)
  - Assim, quando o símbolo A estiver no topo da pilha e o símbolo de entrada for um token não esperado, mas pertence ao Follow(A), então a ação do analisador deve ser "desempilha A"

Recuperação de Erros

- A tabela de análise LL(1) pode ser reescrita com anotações dos símbolos de sincronização (*sinc*) nas posições vazias M[A,x] onde x são os símbolos em Follow(A)
- Deve-se também modificar o reconhecedor, incluindo o tratamento de erros
- Ao encontrar um token inesperado na sentença de análise, deve-se emitir uma mensagem de erro e executar uma das ações:
  - se a entrada na tabela estiver vazia, ler o próximo token (descarte do token lido);
  - se a entrada é sinc, desempilhar o não-terminal do topo da pilha;
    - se o token do topo não é igual ao símbolo de entrada, desempilha o token.

Recuperação de Erros

Exemplo:

Processamento de

‘id ∨∧ id’

Tabela estendida (inclui sinc’s):

	id	$\vee$	$\wedge$	$\neg$	\$
E	$E \to TE'$			$E \to TE'$	sinc
E’		$E' \to \vee TE'$			$E' \to \varepsilon$
T	$T \to FT'$	sinc		$T \to FT'$	sinc
T’		$T' \to \varepsilon$	$T' \to \wedge FT'$		$T' \to \varepsilon$
F	$F \to id$	sinc	sinc	$F \to \neg F$	sinc

	id	$\vee$	$\wedge$	$\neg$	$
E	$E \to TE'$			$E \to TE'$	sinc
E'		$E' \to \vee TE'$			$E' \to \varepsilon$
T	$T \to FT'$	sinc		$T \to FT'$	sinc
T'		$T' \to \varepsilon$	$T' \to \wedge FT'$		$T' \to \varepsilon$
F	$F \to id$	sinc	sinc	$F \to \neg F$	sinc

Pilha	**Entrada**	**Ação**
$\$\underline{E}$	id$\vee\wedge$id$	desempilha E, empilha E'T, imprime $E \to TE'$
$\$E'\underline{T}$	$\underline{id}\vee\wedge$id$	desempilha T, empilha T'F, imprime $T \to FT'$
$\$E'T'\underline{F}$	$\underline{id}\vee\wedge$id$	desempilha T, empilha T'F, imprime $T \to FT'$
$\$E'T'\underline{F}$	$\underline{id}\vee\wedge$id$	desempilha F, empilha id, imprime $F \to id$
$\$E'T'\underline{id}$	$\underline{id}\vee\wedge$id$	desempilha id, avança leitura sentença
$\$E'\underline{T'}$	$\underline{\vee}\wedge$id$	desempilha T', empilha ε, imprime $T' \to \varepsilon$
$\$\underline{E'}$	$\underline{\vee}\wedge$id$	desempilha E', empilha E'T$\vee$, imprime $E' \to \vee TE'$
$\$E'T\underline{\vee}$	$\underline{\vee}\wedge$id$	desempilha $\vee$, avança leitura sentença
$\$E'\underline{T}$	$\underline{\wedge}$id$	**M[T,$\wedge$] é vazia! Sinaliza erro e descarta $\wedge$ na entrada**
$\$E'\underline{T}$	$\underline{id}$$	desempilha T, empilha T'F, imprime $T \to FT'$
$\$E'T'\underline{F}$	$\underline{id}$$	desempilha F, empilha id, imprime $F \to id$
$\$E'T'\underline{id}$	$\underline{id}$$	desempilha id, avança leitura sentença
...	...	...
$	$	**Análise finalizada. 1 erro encontrado**.

O erro reportado deve ser algo do tipo "∧ encontrado. Espera-se id ou ¬".

Big Example:

Análise Top-Down

Gramática

$P \rightarrow$ begin D C end

$D \rightarrow$ int id I

$I \rightarrow$, id I | ε

$C \rightarrow C$; T = E | T = E

$E \rightarrow E + T \mid T$

$T \rightarrow$ id | id[E]

Tarefa 1:

- Construir a tabela

Big Example

Sentença:

```
begin
     int id, id
     id = id[id+id]
end
```

Tarefa 2:

- Processar a sentença acima usando a Tabela obtida na Tarefa 1.

Sugestões de Materiais/Vídeos

- **First/Follow**
  - https://www.youtube.com/watch?v=BSTBaPFxs3Q
  - https://www.youtube.com/watch?v=SBnjVW8dUqo
  - https://www.youtube.com/watch?v=_uSlP91jmTM
- **Construção da tabela**
  - https://www.youtube.com/watch?v=R1ZlWEZWMKk
  - https://www.youtube.com/watch?v=XpZZrQjOAJY
- **Análise usando a tabela**
  - https://www.youtube.com/watch?v=iLXOsKQ4MDs&feature=youtu.be

Análise Sintática Top-Down:
Exercícios

- **Apostila: Capítulo 4**
  - Todos

- **Exercícios com resposta:**
  - http://www.ybadoo.com.br/tutoriais/cmp/03/