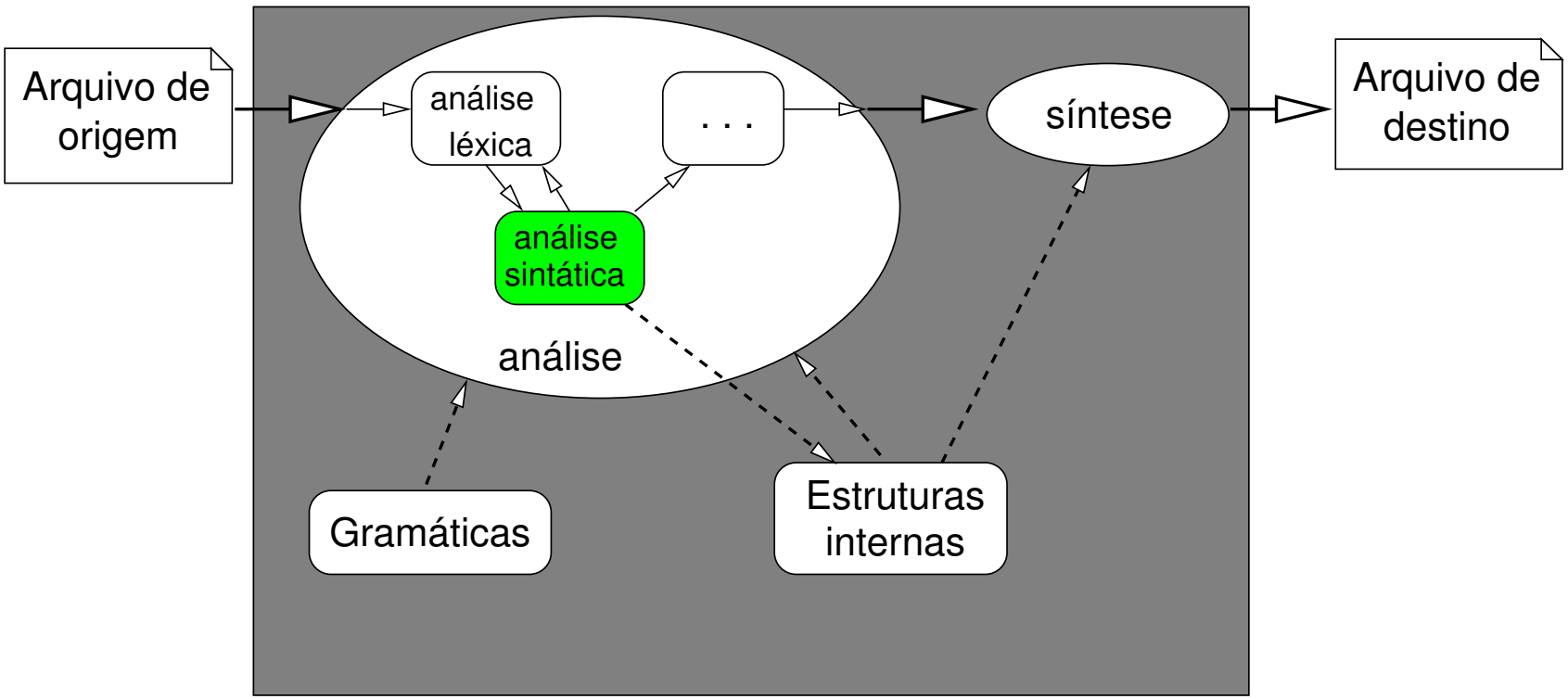
Análise sintática

Ivan Ricarte

2008

Sumário

Análise sintática



Reconhecimento de sentenças

- O reconhecimento de sentenças, ou *parsing*, é o procedimento que verifica se uma dada sentença pertence à linguagem gerada por uma gramática.
- Gramáticas tipo 2, ou gramáticas livres de contexto, são adequadas para representar boa parte das características de linguagens de programação.

Validação de uma sentença

Há duas formas de validar uma sentença:

Top-down ou descendente: a partir do símbolo sentencial, selecionar uma seqüência de aplicação de regras e aplicá-las, substituindo o lado esquerdo pelo lado direito, até que se obtenha a sentença sob análise; ou

Bottom-up ou ascendente: a partir da sentença, selecionar uma seqüência de aplicação de regras e aplicá-las, substituindo o lado direito pelo lado esquerdo, até que reste apenas o símbolo sentencial.

Reconhecimento de sentenças

Exemplo: uma gramática para expressões

Para reconhecer expressões (E):

$$
\begin{array}{lcll}
E & \rightarrow & E + E & 1 \\
E & \rightarrow & E \times E & 2 \\
E & \rightarrow & (E) & 3 \\
E & \rightarrow & v & 4
\end{array}
$$

- A expressão $(v + v) \times v$ faz parte dessa gramática?

Estratégias de validação

Há duas formas de responder a essa questão:

Validação descendente A partir do símbolo sentencial E, procura aplicar produções da gramática substituindo o lado esquerdo pelo lado direito até que a sentença $(v + v) \times v$ seja derivada

Validação ascendente A partir da sentença $(v + v) \times v$, procura aplicar produções da gramática substituindo o lado direito pelo lado esquerdo até que reste apenas o símbolo sentencial E

Validação descendente

Produções

$$
\begin{array}{lcll}
E & \rightarrow & E + E & 1 \\
E & \rightarrow & E \times E & 2 \\
E & \rightarrow & (E) & 3 \\
E & \rightarrow & v & 4
\end{array}
$$

Derivações

$$
\begin{aligned}
E &\overset{2}{\Longrightarrow} E \times E \\
&\overset{3}{\Longrightarrow} (E) \times E \\
&\overset{1}{\Longrightarrow} (E + E) \times E \\
&\overset{4}{\Longrightarrow} (v + E) \times E \\
&\overset{4}{\Longrightarrow} (v + v) \times E \\
&\overset{4}{\Longrightarrow} (v + v) \times v
\end{aligned}
$$

- Seqüência 2, 3, 1, 4, 4, 4

Validação descendente

Outras seqüências de reconhecimento podem ser obtidas para essa mesma estratégia:

- 2, 3, 4, 1, 4, 4

$$E \overset{2}{\Longrightarrow} E \times E \overset{3}{\Longrightarrow} (E) \times E \overset{4}{\Longrightarrow} (E) \times v$$
$$\overset{1}{\Longrightarrow} (E + E) \times v \overset{4}{\Longrightarrow} (E + v) \times v \overset{4}{\Longrightarrow} (v + v) \times v$$

- 2, 4, 3, 1, 4, 4

$$E \overset{2}{\Longrightarrow} E \times E \overset{4}{\Longrightarrow} E \times v \overset{3}{\Longrightarrow} (E) \times v$$
$$\overset{1}{\Longrightarrow} (E + E) \times v \overset{4}{\Longrightarrow} (E + v) \times v \overset{4}{\Longrightarrow} (v + v) \times v$$

Validação ascendente

Produções

E	$\rightarrow$	$E + E$	1
E	$\rightarrow$	$E \times E$	2
E	$\rightarrow$	(E)	3
E	$\rightarrow$	v	4

Derivações

$$
\begin{aligned}
(\underline{v} + v) \times v &\overset{4}{\Longleftarrow} (E + \underline{v}) \times v \\
&\overset{4}{\Longleftarrow} (E + E) \times \underline{v} \\
&\overset{4}{\Longleftarrow} (\underline{E + E}) \times E \\
&\overset{1}{\Longleftarrow} \underline{(E)} \times E \\
&\overset{3}{\Longleftarrow} \underline{E \times E} \\
&\overset{2}{\Longleftarrow} E
\end{aligned}
$$

- Seqüência de reconhecimento 4, 4, 4, 1, 3, 2
- Também há outras seqüências

Derivações canônicas

- Forma sistemática de selecionar qual símbolo da forma sentencial será substituído na próxima derivação
- Duas formas
  - Derivação canônica mais à esquerda (*leftmost derivation*)
  - Derivação canônica mais à direita (*rightmost derivation*)
- Para uma derivação canônica, a seqüência de reconhecimento é única

Derivações canônicas

Derivação canônica mais à esquerda

Derivação canônica mais à esquerda aplica uma produção da gramática para substituir o símbolo não-terminal mais à esquerda da forma sentencial

- A seqüência de produções aplicadas desde o símbolo sentencial até a sentença é denominada a seqüência de reconhecimento mais à esquerda, ou *leftmost parse*

Derivações canônicas

Derivação canônica mais à direita

Derivação canônica mais à direita símbolo não-terminal mais à direita da forma sentencial é selecionado para ser substituído pela aplicação de uma produção da gramática

- A seqüência de reconhecimento mais à direita, ou *rightmost parse*, é dada pelo reverso da seqüência de produções associada à derivação
- É a seqüência de aplicação de produções para partir da sentença e alcançar o símbolo sentencial

Derivações canônicas

Exemplo

Produções

E	$\rightarrow$	$E + E$	1
E	$\rightarrow$	$E \times E$	2
E	$\rightarrow$	(E)	3
E	$\rightarrow$	v	4

Derivações canônicas para $(v + v) \times v$

- Seqüência de reconhecimento mais à esquerda:
  2, 3, 1, 4, 4, 4
- Seqüência de reconhecimento mais à direita:
  4, 4, 1, 3, 4, 2

Árvores sintáticas

Representação das derivações utilizadas no reconhecimento de uma sentença através de uma estrutura de árvore

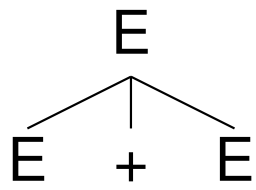
Em uma árvore sintática:

1. a raiz da árvore é um nó com o símbolo sentencial;
2. as folhas da árvore são símbolos terminais, na mesma seqüência em que estes ocorrem na sentença analisada; e
3. os nós intermediários da árvore correspondem a símbolos não terminais, onde um nó cujo rótulo é P com filhos $s_1, s_2, \ldots, s_n$ pode ocorrer apenas se houver uma regra $P \rightarrow s_1\ s_2 \ldots\ s_n$ na gramática.

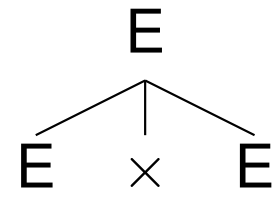
Árvores sintáticas

Subárvores para a gramática de expressões

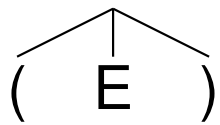
$E \rightarrow E + E$



$E \rightarrow E \times E$

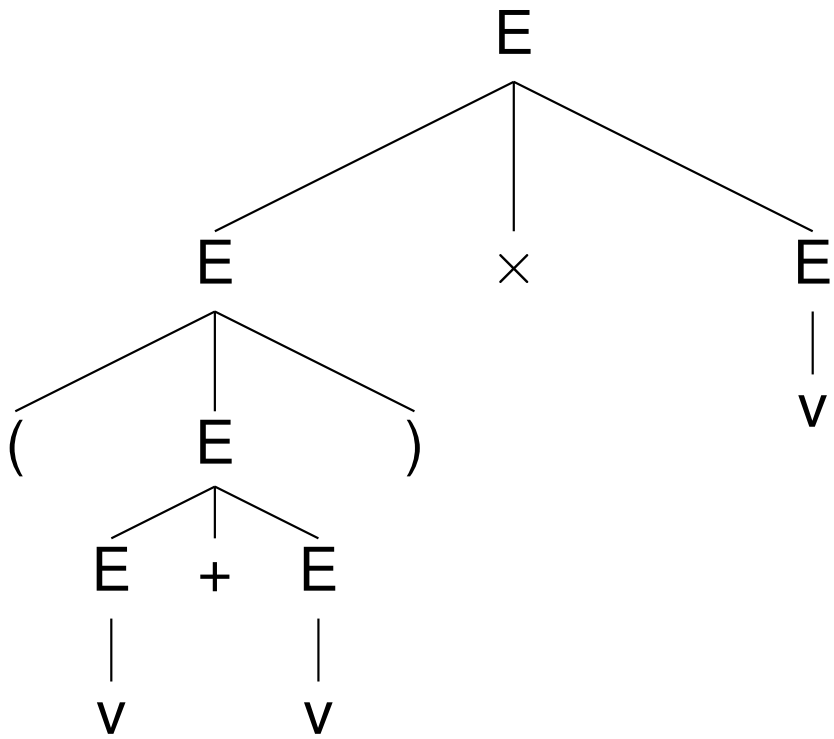


$E \rightarrow (E)$



$E \rightarrow v$

Árvore sintática para $(v + v) \times v$



Árvores sintáticas

Relação com derivações canônicas

- Estrutura da árvore sintática independe da seqüência de derivações utilizada para reconhecimento
- Uma derivação canônica está associada à estratégia utilizada para percorrer seqüencialmente os elementos da árvore

  pré-ordem raiz antes das folhas, folhas da esquerda para a direita

  - seqüência de reconhecimento mais à esquerda

  pós-ordem folhas da esquerda para a direita, raiz depois das folhas

  - seqüência de reconhecimento mais à direita

Gramáticas ambíguas

Gramática é ambígua se for possível obter, para uma sentença, mais de uma árvore sintática

- De modo equivalente, é ambígua se houver mais que uma seqüência de reconhecimento mais à direita
- Ou mais de uma seqüência de reconhecimento mais à esquerda
- Ou ainda, mais de uma derivação canônica mais à esquerda
- Ou mais de uma derivação canônica mais à direita

Gramáticas ambíguas

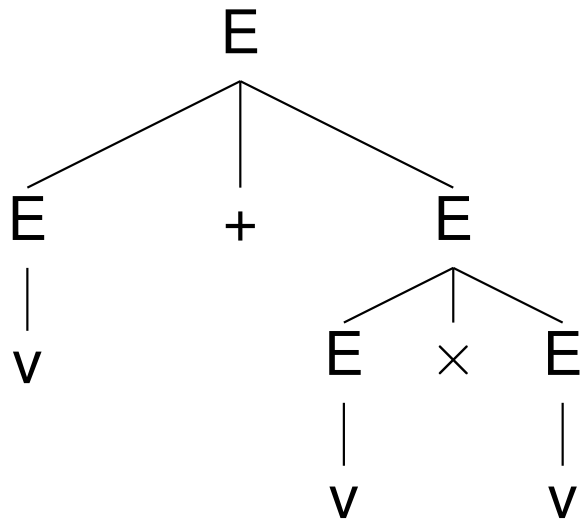
Exemplo

Considere a sentença $v + v \times v$ em

E	$\rightarrow$	$E + E$	1
E	$\rightarrow$	$E \times E$	2
E	$\rightarrow$	(E)	3
E	$\rightarrow$	v	4

Gramáticas ambíguas

Exemplo: árvore 1

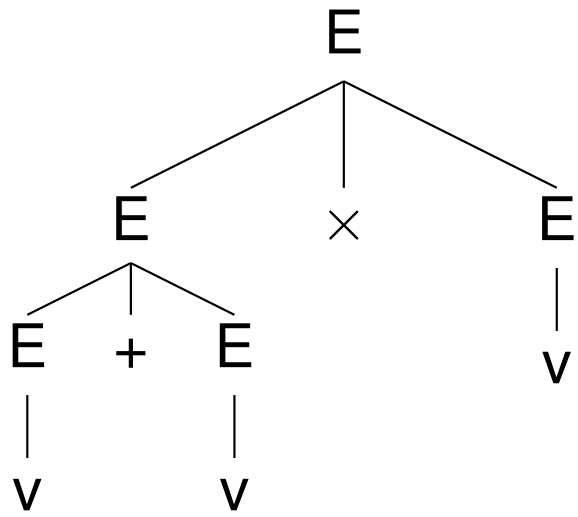


seqüência de reconhecimento mais à esquerda 1, 4, 2, 4, 4

seqüência de reconhecimento mais à direita 4, 4, 4, 2, 1

Gramáticas ambíguas

Exemplo: árvore 2



seqüência de reconhecimento mais à esquerda 2, 1, 4, 4, 4

seqüência de reconhecimento mais à direita 4, 4, 1, 4, 2

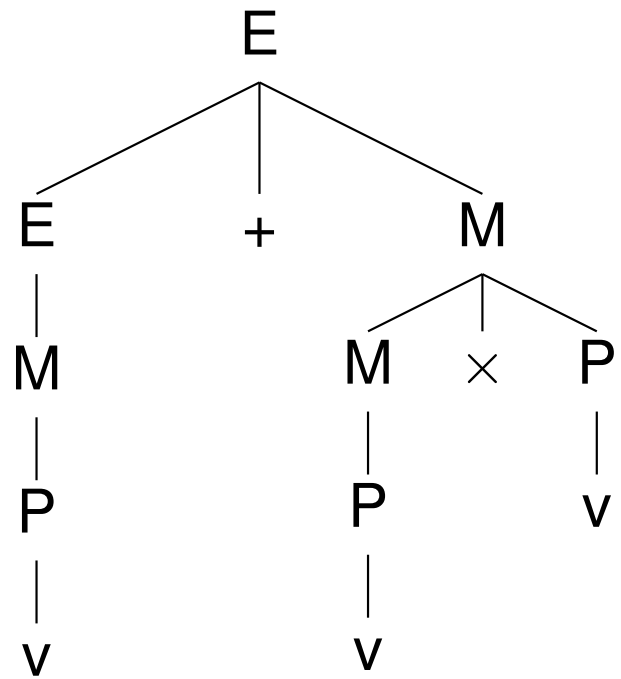
Versão não-ambígua da gramática de expressões

E	$\rightarrow$	$E + M$	1
E	$\rightarrow$	M	2
M	$\rightarrow$	$M \times P$	3
M	$\rightarrow$	P	4
P	$\rightarrow$	(E)	5
P	$\rightarrow$	v	6

Mesma sentença, na gramática revista

- Única interpretação possível para $v + v \times v$
- Por exemplo, para o reconhecimento descendente:

$$
\begin{aligned}
E &\overset{1}{\Rightarrow} E + M \overset{2}{\Rightarrow} M + M \\
&\overset{4}{\Rightarrow} P + M \overset{6}{\Rightarrow} v + M \\
&\overset{3}{\Rightarrow} v + M \times P \overset{4}{\Rightarrow} v + P \times P \\
&\overset{6}{\Rightarrow} v + v \times P \overset{6}{\Rightarrow} v + v \times v
\end{aligned}
$$



Analisadores sintáticos

- Programas que reconhecem automaticamente sentenças de uma dada gramática livre de contexto
  - Como para analisadores léxicos, um autômato é uma representação abstrata para o analisador
  - No entanto, para o analisador sintático o autômato precisa de uma memória auxiliar:

    Pilha estrutura linear com política de acesso LIFO *(Last In, First Out)*

Autômato de pilha

Definição formal

Sêxtupla $M = (K, \Sigma, \Gamma, \delta, s, F)$, na qual os elementos são:

1. o conjunto finito de estados, K;
2. o alfabeto de entrada finito, Σ;
3. o alfabeto de pilha finito, Γ;
4. a função de transição, $\delta : K \times \Sigma \times \Gamma \rightarrow K \times \Gamma$;
5. o estado inicial, s, $s \in K$; e
6. o conjunto de estados finais, F, com $F \subseteq K$.

Analisadores sintáticos

- Analisadores utilizam o autômato de pilha para reconhecer sentenças de uma linguagem livre de contexto
- Podem usar a técnica de construção descendente ou ascendente

  Descendente Pilha começa com símbolo sentencial
  Ascendente Pilha começa vazia
- Exemplos de analisadores

  Analisador sintático preditivo construção descendente
  Analisador de precedência fraca construção ascendente

Analisador sintático preditivo

Analisador apropriado para gramáticas LL(1)

- Varredura da sentença ocorre da esquerda para a direita *(Left-to-right)*
- Derivação canônica mais à esquerda *(Leftmost derivation)*
- Um único símbolo da sentença *(lookahead)* analisado a cada passo

Analisador sintático preditivo

Gramáticas LL(1)

Gramática com produções $A \rightarrow \alpha$ e $A \rightarrow \beta$ é LL(1) se:

1. α e β não iniciam com o mesmo símbolo terminal
   - Ou com símbolos não-terminais cujas expansões iniciem com o mesmo símbolo terminal
2. Só α ou só β tem expansões que levam à string vazia
3. Se α leva à string vazia, β não inicia expansão com símbolo terminal que possa ocorrer imediatamente à direita de A

Analisador sintático preditivo

Conversão de produções recursivas

- Técnica de construção descendente não pode ser aplicada a gramáticas com produções recursivas à esquerda
- Se houver produção dessa forma, é necessário convertê-la:

De

$$A \rightarrow A\ \beta$$
$$A \rightarrow \delta$$

Para

$$A \rightarrow \delta\ A'$$
$$A' \rightarrow \beta\ A'$$
$$A' \rightarrow \varepsilon$$

Construção do analisador sintático preditivo

- Uma tabela sintática representa as operações possíveis para o autômato associado ao analisador
- Construção dessa tabela é feita pela análise de cada produção
- Para cada par símbolo não-terminal, símbolo terminal, a tabela indica qual produção deve ser aplicada, se possível
  - Símbolo terminal é o próximo símbolo da sentença
  - Símbolo não-terminal é o símbolo no topo da pilha
- Além dos símbolos terminais, um delimitador de sentença (denotado aqui por $\boxed{\$}$) é uma entrada reconhecida pela tabela

Construção da tabela sintática

Para cada produção $A \rightarrow B$ na gramática,

- Insira a produção na tabela sintática na linha A, coluna t, para cada símbolo terminal t que pode iniciar a expansão de B
- Se $B = \varepsilon$, então insira a produção na tabela sintática na linha A, coluna t para cada símbolo terminal t que pode ocorrer à direita de A, sendo que:
  - O delimitador $\boxed{\$}$ pode estar à direita do símbolo sentencial;
  - t pode ser um símbolo terminal que inicia a expansão de um símbolo não-terminal que ocorre à direita de A em alguma produção
- Para cada produção na forma $A \rightarrow \ldots B$, os símbolos que podem ocorrer à direita de A também podem ocorrer à direita de B

Construção da tabela sintática

Exemplo

Considere a construção do analisador sintático preditivo para a seguinte gramática de expressões:

Gramática original

$$
\begin{array}{lcll}
E & \rightarrow & E + M & 1 \\
E & \rightarrow & M & 2 \\
M & \rightarrow & M \times P & 3 \\
M & \rightarrow & P & 4 \\
P & \rightarrow & (E) & 5 \\
P & \rightarrow & v & 6
\end{array}
$$

Construção da tabela sintática

Conversão da gramática

Inicialmente, é preciso eliminar das produções da gramática qualquer recursão à esquerda:

Gramática equivalente sem recursão à esquerda

$$E \rightarrow ME' \tag{1}$$
$$E' \rightarrow +ME' \tag{2}$$
$$E' \rightarrow \varepsilon \tag{3}$$
$$M \rightarrow PM' \tag{4}$$
$$M' \rightarrow \times PM' \tag{5}$$
$$M' \rightarrow \varepsilon \tag{6}$$
$$P \rightarrow (E) \tag{7}$$
$$P \rightarrow v \tag{8}$$

Construção da tabela sintática

Análise das produções (1)

A seguir, cada produção é analisada para obter as entradas na tabela:

Produção 1: $E \rightarrow ME'$

- Os símbolos terminais que podem iniciar a expansão de E são aqueles que iniciam a expansão de M
- Por sua vez, $M \rightarrow PM'$ — expansão de P
- Como $P \rightarrow (E)$ e $P \rightarrow v$, os símbolos terminais que podem iniciar essa expansão são (e v

	...	(	...	v	...
E		P1		P1	

Construção da tabela sintática

Análise das produções (2)

Produção 2: $E' \rightarrow +ME'$

- Um símbolo terminal pode iniciar a expansão de E' por essa produção: $+$

	...	+	...	...	...
E'		P2			

Construção da tabela sintática

Análise das produções (3)

Produção 3: $E' \longrightarrow \varepsilon$

- Não há expansão, a segunda regra deve ser usada
  - Se $A \longrightarrow \varepsilon$, então insira a produção na tabela sintática na linha A, coluna t para cada símbolo terminal t que pode ocorrer à direita de A
- Pelas produções, não há nenhum símbolo que ocorra diretamente à direita de E'
- Como $E \longrightarrow ME'$, o que pode ocorrer à direita de E também pode indiretamente ocorrer à direita de E'
  - $\boxed{\$}$ (E é o símbolo sentencial) e) (pela produção $P \longrightarrow (E)$)

	...	)	...	$	...
E'		P3		P3	

Construção da tabela sintática

Análise das produções (4)

Produção 4: $M \rightarrow PM'$

- Os símbolos terminais que podem iniciar a expansão de M são aqueles que iniciam a expansão de P
- Como $P \rightarrow (E)$ e $P \rightarrow v$, os símbolos terminais que podem iniciar essa expansão são (e v

	...	(	...	v	...
M		P4		P4	

Construção da tabela sintática

Análise das produções (5)

Produção 5: $M' \rightarrow \times PM'$

- Um símbolo terminal pode iniciar a expansão de M' por essa produção: $\times$

	...	$\times$	...	...	...
M'		P5			

Construção da tabela sintática

Análise das produções (6)

Produção 6: $M' \rightarrow \varepsilon$

- Não há expansão, a segunda regra deve ser usada
- Pelas produções, não há nenhum símbolo que ocorra diretamente à direita de M'
- Como $M \rightarrow PM'$, o que pode ocorrer à direita de M também pode indiretamente ocorrer à direita de M'
  - Pelas produções 1 e 2, E' aparece à direita de M; a expansão de E' pode iniciar com $+$
  - Como $E' \rightarrow \varepsilon$ pela produção 3, é preciso incluir também os símbolos que podem ocorrer à direita de E, pois $E \rightarrow ME'$:

    $\boxed{\$}$ e $)$

	...	)	+	$\boxed{\$}$	...
M'		P6	P6	P6	

Construção da tabela sintática

Análise das produções (7)

Produção 7: $P \rightarrow (E)$

- Um símbolo terminal pode iniciar a expansão de P por essa produção: (

	...	(	...	...	...
P		P7			

Construção da tabela sintática

Análise das produções (8)

Produção 8: $P \longrightarrow v$

- Um símbolo terminal pode iniciar a expansão de P por essa produção: v

	...	v	...	...	...
P		P8			

Construção da tabela sintática

Resultado

Tabela sintática derivada das produções

	$+$	$\times$	(	)	v	$
E			P1		P1	
E'	P2			P3		P3
M			P4		P4	
M'	P6	P5		P6		P6
P			P7		P8	

Reconhecimento de sentenças

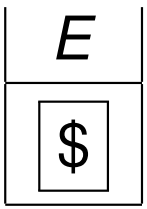
- O reconhecimento da sentença com o analisador preditivo utiliza duas estruturas auxiliares
  1. Uma pilha com a forma sentencial resultante da aplicação das produções, inicialmente com o símbolo sentencial (precedido do delimitador de sentença, $\boxed{\$}$)
  2. Uma lista com os símbolos terminais que compõem a sentença, com o delimitador de sentença $\boxed{\$}$ ao final
- Quando o topo da pilha e o início da sentença têm o mesmo símbolo, ambos são retirados
- Condição de reconhecimento ocorre quando a pilha de símbolos e a sentença tornam-se vazias

Reconhecimento de sentenças

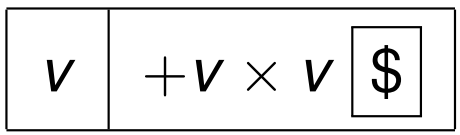
Exemplo: sentença $v + v \times v$ (1)

Estado inicial:

Pilha



Lista de tokens



Árvore sintática

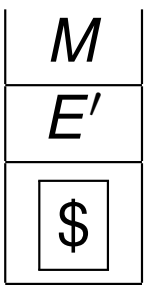
E

Reconhecimento de sentenças

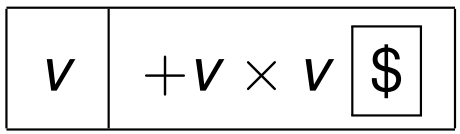
Exemplo: sentença $v + v \times v$ (2)

Para $[E, v]$, tabela indica aplicação da produção 1: $E \rightarrow ME'$

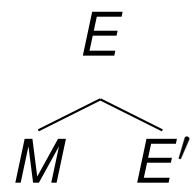
Pilha



Lista de tokens



Árvore sintática

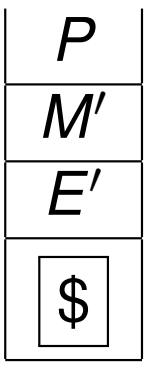


Reconhecimento de sentenças

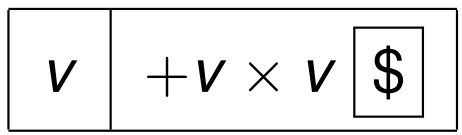
Exemplo: sentença $v + v \times v$ (3)

Para $[M, v]$, tabela indica aplicação da produção 4: $M \rightarrow PM'$

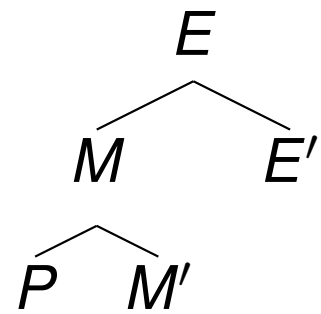
Pilha



Lista de tokens



Árvore sintática

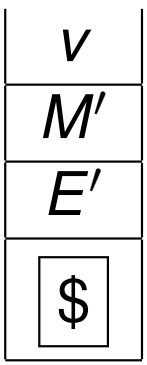


Reconhecimento de sentenças

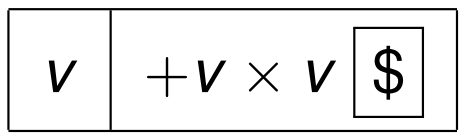
Exemplo: sentença $v + v \times v$ (4)

Para $[P, v]$, tabela indica aplicação da produção 8: $P \rightarrow v$

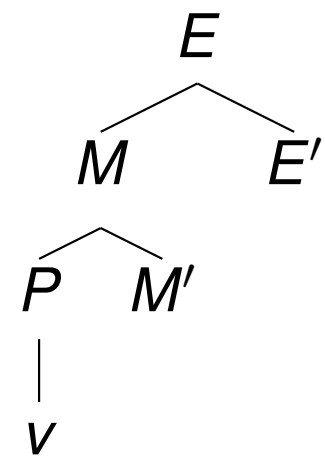
Pilha



Lista de tokens



Árvore sintática

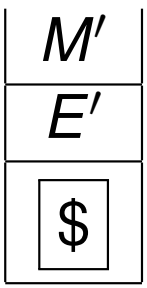


Reconhecimento de sentenças

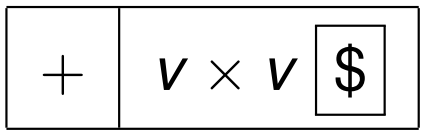
Exemplo: sentença $v + v \times v$ (5)

Mesmo símbolo no topo da pilha e no início da lista: consumi-los

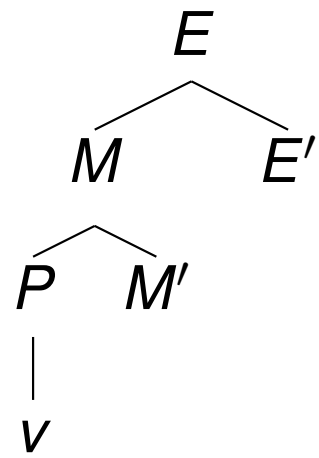
Pilha



Lista de tokens



Árvore sintática

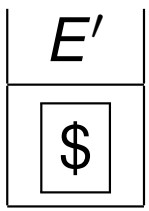


Reconhecimento de sentenças

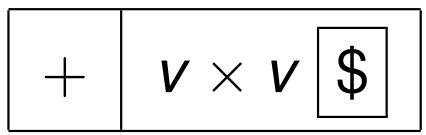
Exemplo: sentença $v + v \times v$ (6)

Para $[M', +]$, tabela indica aplicação da produção 6: $M' \longrightarrow \varepsilon$

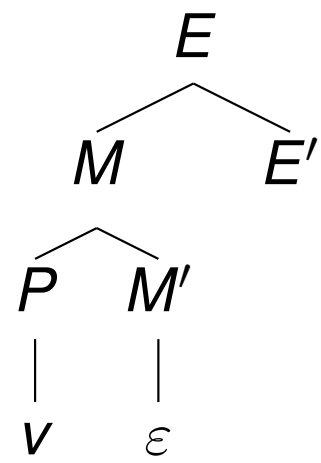
Pilha



Lista de tokens



Árvore sintática

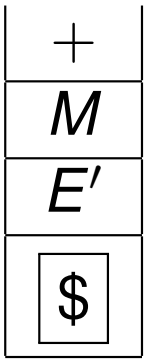


Reconhecimento de sentenças

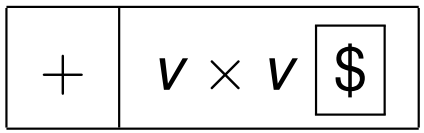
Exemplo: sentença $v + v \times v$ (7)

Para $[E', +]$, produção 2: $E' \rightarrow +ME'$

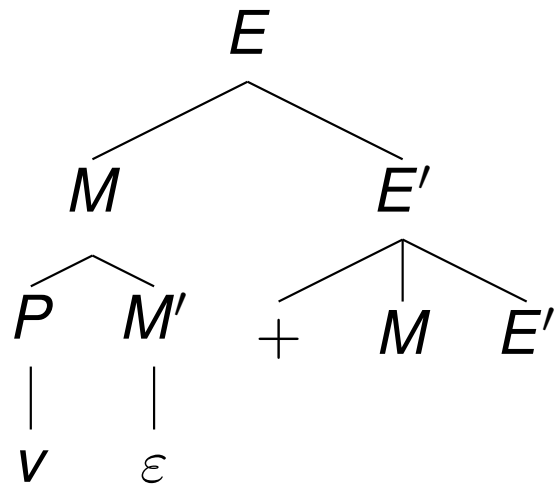
Pilha



Lista de tokens



Árvore sintática

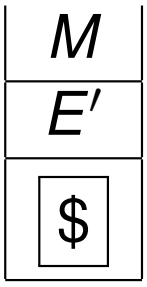


Reconhecimento de sentenças

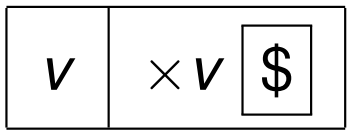
Exemplo: sentença $v + v \times v$ (8)

Mesmo símbolo no topo da pilha e início da lista: consumi-los

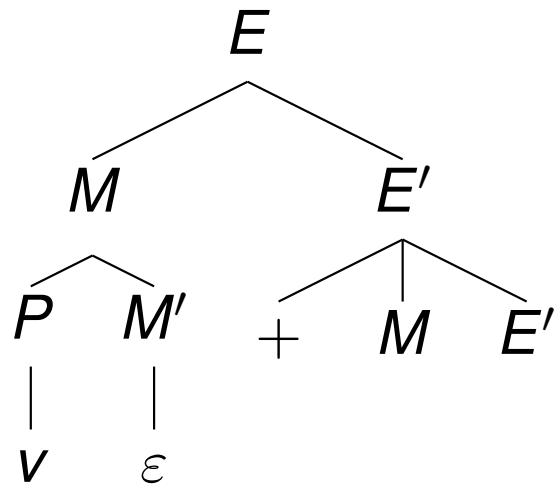
Pilha



Lista de tokens



Árvore sintática

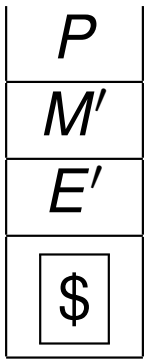


Reconhecimento de sentenças

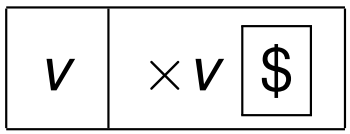
Exemplo: sentença $v + v \times v$ (9)

Para $[M, v]$, produção 4: $M \rightarrow PM'$

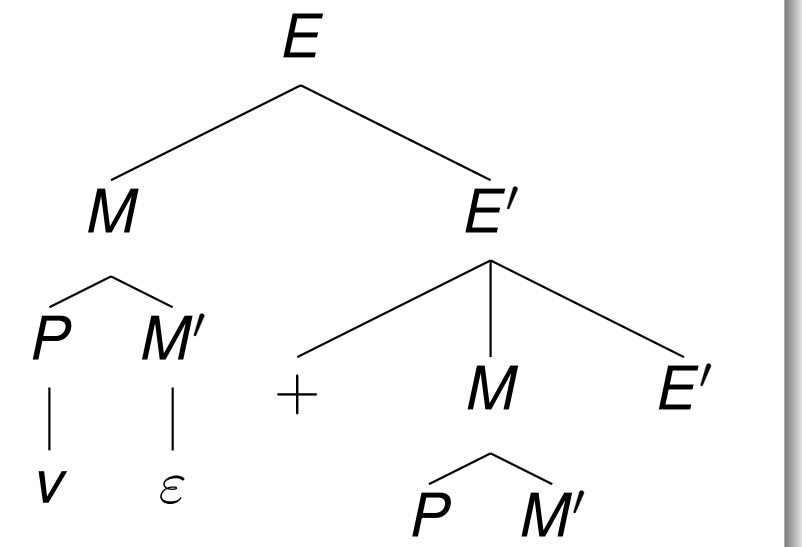
Pilha



Lista de tokens



Árvore sintática

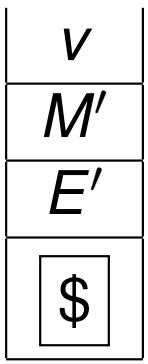


Reconhecimento de sentenças

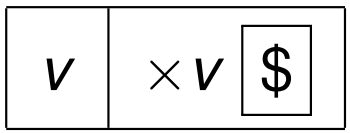
Exemplo: sentença $v + v \times v$ (10)

Para $[P, v]$, produção 8: $P \rightarrow v$

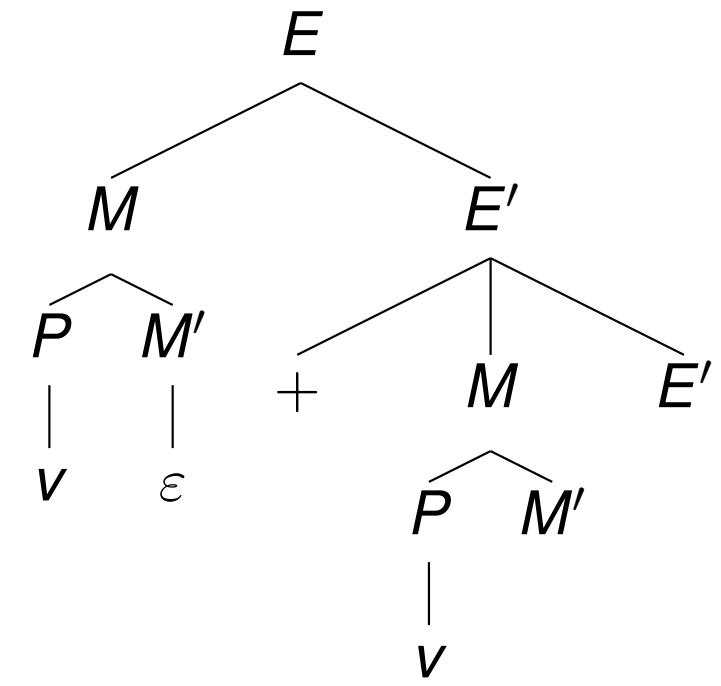
Pilha



Lista de tokens



Árvore sintática

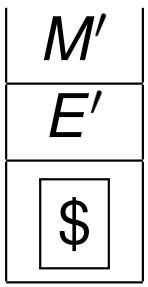


Reconhecimento de sentenças

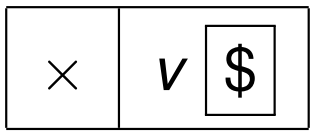
Exemplo: sentença $v + v \times v$ (11)

Mesmo símbolo no topo da pilha e início da lista: consumi-los

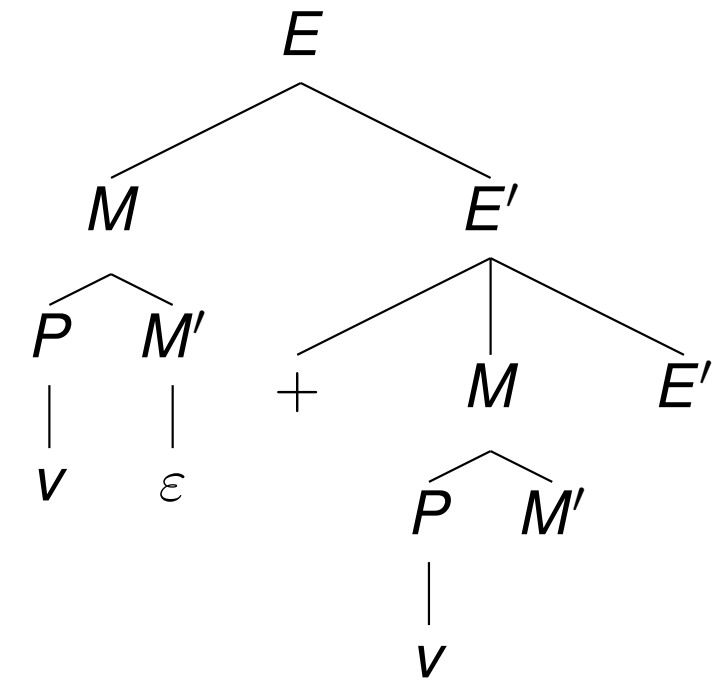
Pilha



Lista de tokens



Árvore sintática

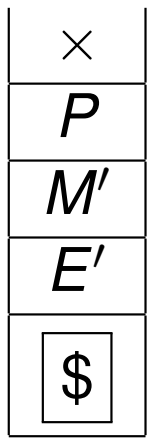


Reconhecimento de sentenças

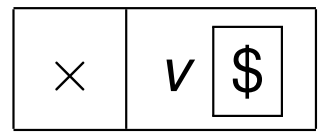
Exemplo: sentença $v + v \times v$ (12)

Para $[M', \times]$, produção 5: $M' \longrightarrow \times PM'$

Pilha



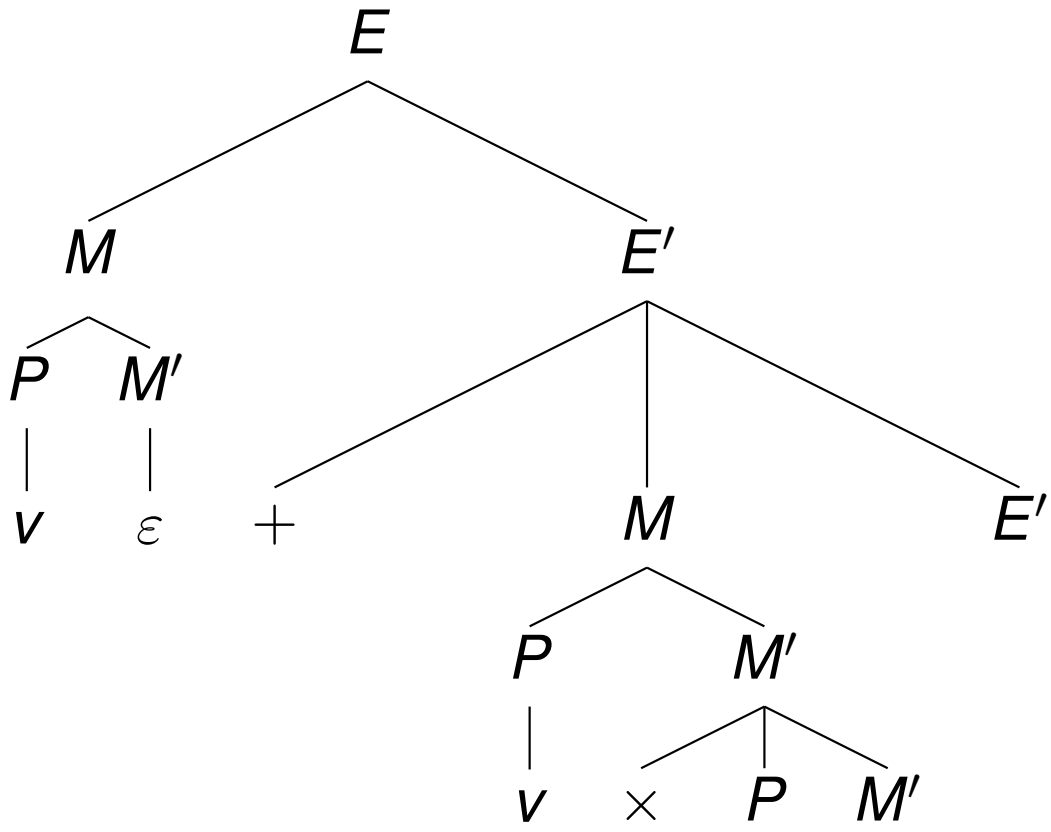
Lista de tokens



Reconhecimento de sentenças

Exemplo: sentença $v + v \times v$ (13)

Árvore sintática

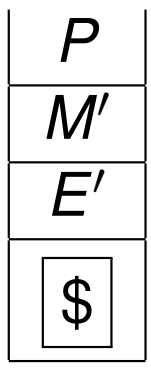


Reconhecimento de sentenças

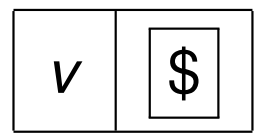
Exemplo: sentença $v + v \times v$ (14)

Mesmo símbolo no topo da pilha e início da lista: consumi-los

Pilha



Lista de tokens

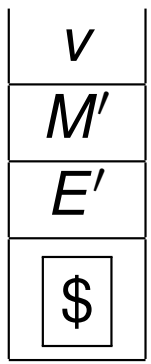


Reconhecimento de sentenças

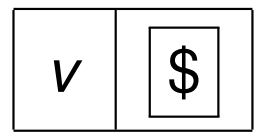
Exemplo: sentença $v + v \times v$ (15)

Para $[P, v]$, produção 8: $P \rightarrow v$

Pilha



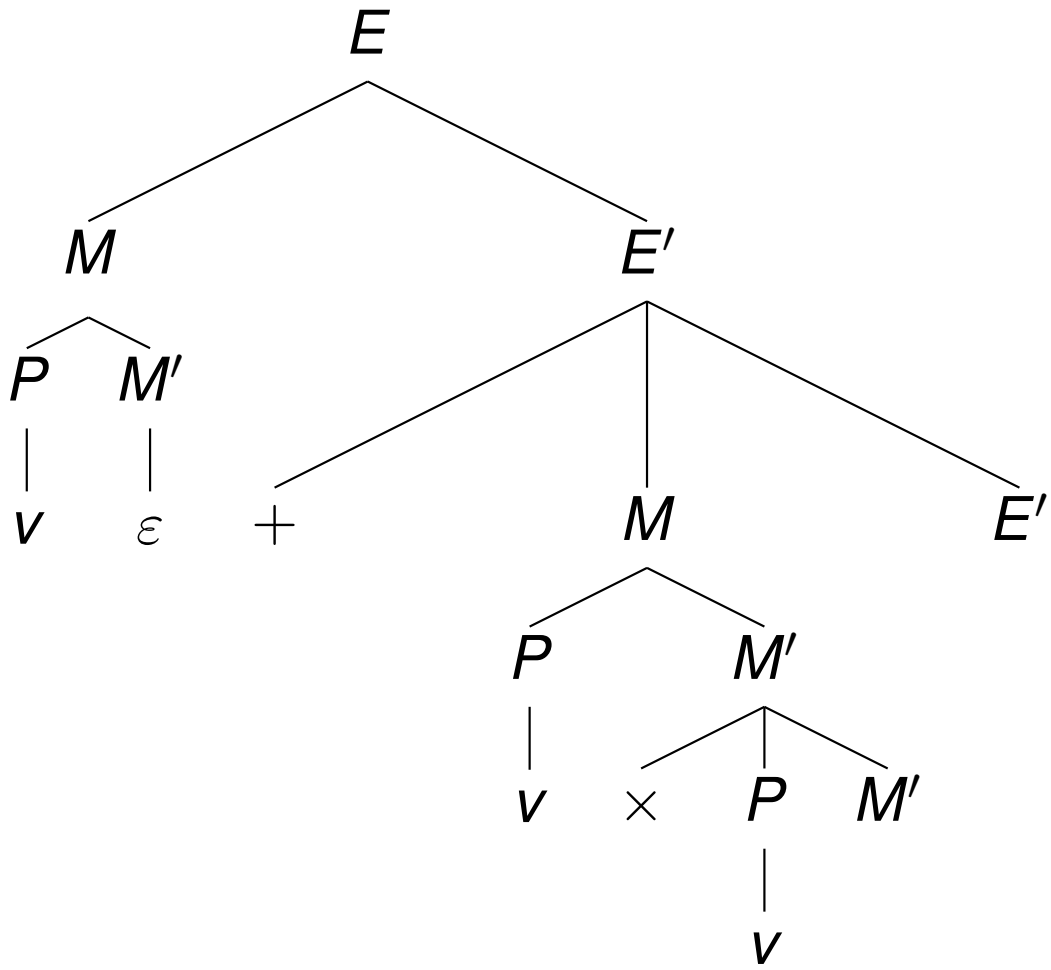
Lista de tokens



Reconhecimento de sentenças

Exemplo: sentença $v + v \times v$ (16)

Árvore sintática

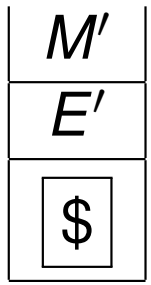


Reconhecimento de sentenças

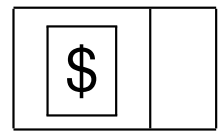
Exemplo: sentença $v + v \times v$ (17)

Mesmo símbolo no topo da pilha e início da lista: consumi-los

Pilha



Lista de tokens

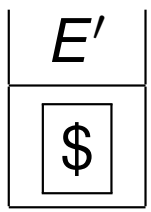


Reconhecimento de sentenças

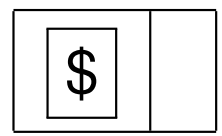
Exemplo: sentença $v + v \times v$ (18)

Para $[M', \boxed{\$}]$, produção 6: $M' \longrightarrow \varepsilon$

Pilha



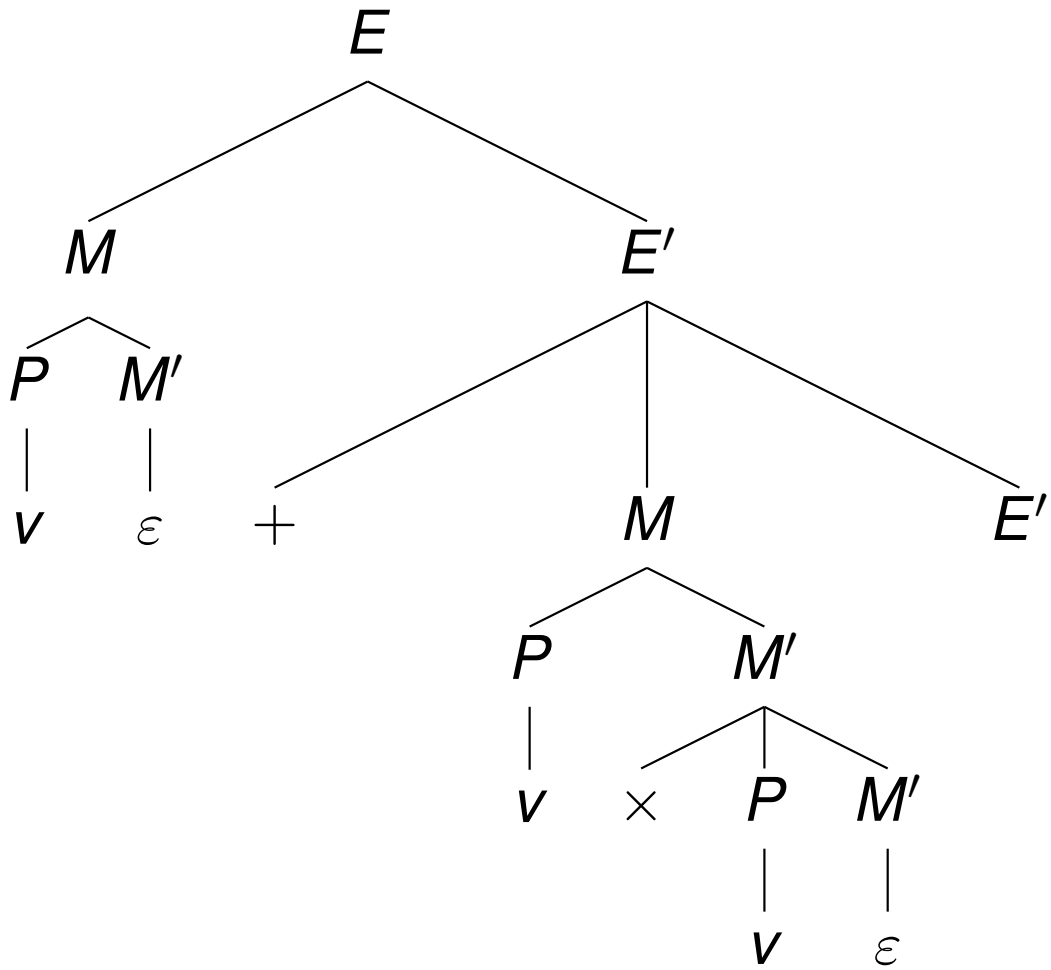
Lista de tokens



Reconhecimento de sentenças

Exemplo: sentença $v + v \times v$ (19)

Árvore sintática

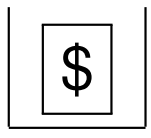


Reconhecimento de sentenças

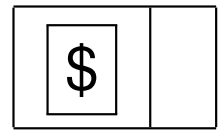
Exemplo: sentença $v + v \times v$ (20)

Para $[E', \boxed{\$}]$, produção 3: $E' \longrightarrow \varepsilon$

Pilha



Lista de tokens



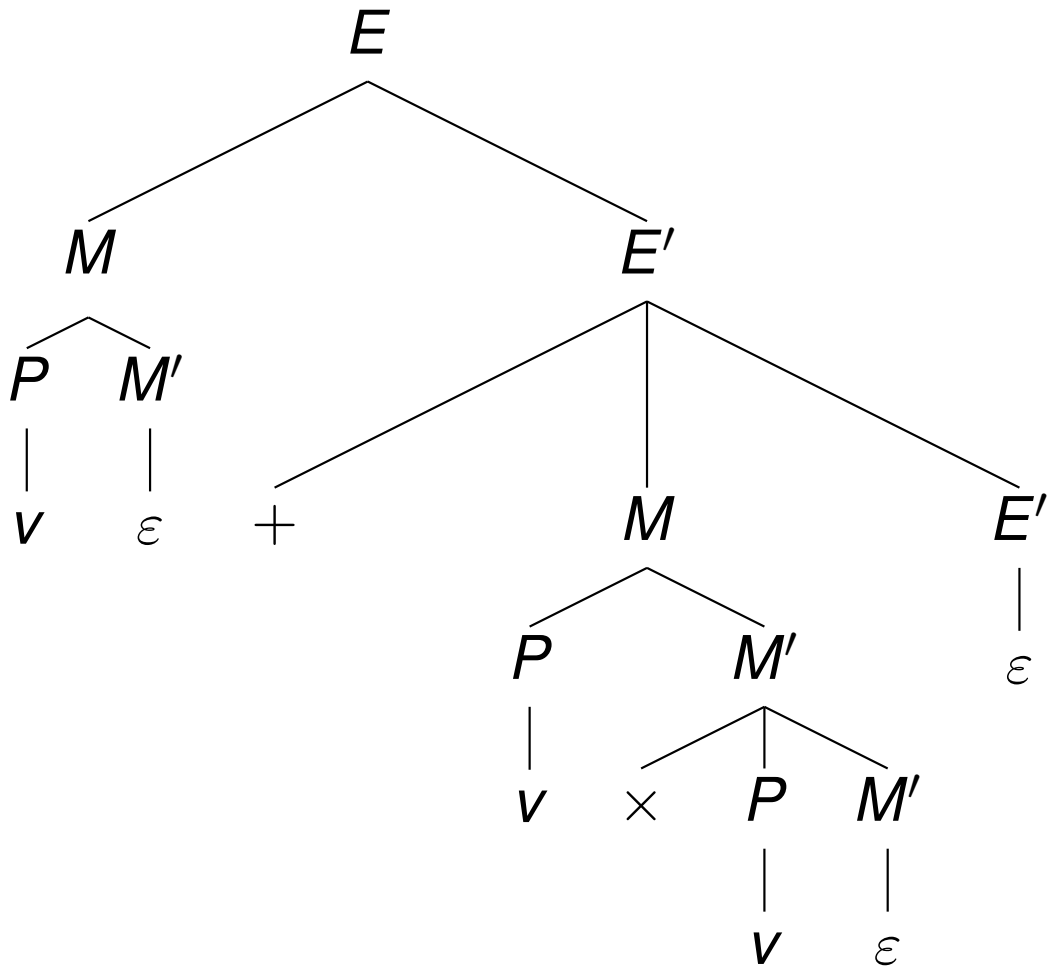
Condição de aceitação

Símbolo delimitador de sentença $\boxed{\$}$ no topo da pilha e no início da lista de tokens

Reconhecimento de sentenças

Exemplo: sentença $v + v \times v$ (21)

Árvore sintática final



Analisador sintático preditivo

- Construção da árvore sintática é descendente
  - Começa com o símbolo sentencial (conteúdo inicial da pilha)
  - Cada vez que uma produção é selecionada para aplicação, o nó não terminal mais à esquerda da árvore é expandido em uma subárvore com o lado direito da produção
  - Quando símbolo mais à esquerda da árvore é terminal, se for igual ao próximo símbolo da sentença há continuidade

Analisador de precedência fraca

- Analisador baseado na técnica de construção ascendente
  - Sentença é reconhecida quanto todos os símbolos da sentença são lidos e na forma sentencial resta apenas o símbolo sentencial
- Corresponde à criação da árvore sintática das folhas para a raiz
  - Sentença reconhecida quando todos os símbolos da sentença estão associados às folhas da árvore sintática e a raiz é o símbolo sentencial

Gramáticas de precedência fraca

- A gramática deve ser de precedência fraca para que o correspondente analisador de precedência fraca seja determinístico
  - Sem produções com string vazia no lado direito
  - Unicamente inversível (sem duas produções com mesmo lado direito)
  - Livre de ciclos
  - Se $X \prec Y$ ou $X \approx Y$, então $X \succ Y$ não ocorre
    - Que relações são estas?

Relações de Wirth-Weber

1. $X \approx Y$ se existe pelo menos uma produção em G cujo lado direito tenha X imediatamente antes de Y,

$$A \rightarrow \alpha XY \beta$$

sendo α e β quaisquer strings em G.

Relações de Wirth-Weber

2. $X \prec Y$ se existe um símbolo não terminal $Z \in V_N$ tal que

$$X \approx Z \quad \text{e} \quad Y \in ESQ(Z)$$

Definição auxiliar

$$ESQ(X) = \{Y \in V_T \cup V_N | X \stackrel{+}{\Rightarrow} Y\alpha, \text{ para } \alpha \in (V_T \cup V_N)*\}$$

Relações de Wirth-Weber

3. $X \succ a$, $a \in V_T$, se uma das duas condições é verdadeira:
   - Existe um símbolo não-terminal $Z \in V_N$ tal que

$$Z \approx a \quad \text{e} \quad X \in DIR(Z)$$

   ou
   - Existem símbolos não-terminais $Z_1, Z_2 \in V_N$ tal que

$$Z_1 \approx Z_2 \quad \text{e} \quad X \in DIR(Z_1) \quad \text{e} \quad a \in ESQ(Z_2)$$

Definição auxiliar

$$DIR(X) = \{Y \in V_T \cup V_N | X \stackrel{+}{\Rightarrow} \alpha Y, \text{ para } \alpha \in (V_T \cup V_N)*\}$$

Condições para construção do analisador

Além das já apresentadas:

- Se duas produções terminam com mesma seqüência de símbolos β

  $$A \rightarrow \alpha X \beta$$

  e

  $$B \rightarrow \beta$$

- O par de símbolos X, B não pode estar associado por nenhuma das relações de Wirth-Weber

Relações de Wirth-Weber

Gramática de expressões

E	$\rightarrow$	$E + M$	1
E	$\rightarrow$	M	2
M	$\rightarrow$	$M \times P$	3
M	$\rightarrow$	P	4
P	$\rightarrow$	(E)	5
P	$\rightarrow$	v	6

Conjuntos DIR e ESQ

Gramática de expressões

$$ESQ(E) = \{E, M, P, (, v\}$$
$$ESQ(M) = \{M, P, (, v\}$$
$$ESQ(P) = \{(, v\}$$

$$DIR(E) = \{M, P,), v\}$$
$$DIR(M) = \{P,), v\}$$
$$DIR(P) = \{), v\}$$

Tabela das relações de Wirth-Weber

Gramática de expressões

	E	M	P	$+$	$\times$	$($	$)$	v
E				$\approx$			$\approx$	
M				$\succ$	$\approx$		$\succ$	
P				$\succ$	$\succ$		$\succ$	
$+$		$\approx\prec$	$\prec$			$\prec$		$\prec$
$\times$			$\approx$			$\prec$		$\prec$
$($	$\approx\prec$	$\prec$	$\prec$			$\prec$		$\prec$
$)$				$\succ$	$\succ$		$\succ$	
v				$\succ$	$\succ$		$\succ$	

Construção da tabela de deslocamento e redução

- Analisador utiliza uma tabela para decidir que ação tomar durante o reconhecimento de uma sentença

  deslocar realizar a leitura do próximo símbolo da sentença

  reduzir substituir, na forma sentencial, símbolo(s) corrente(s) pelo lado esquerdo de uma produção

- Construção da Tabela DR pela análise de relações obtidas pela análise das produções da gramática

Tabela DR

- Duas regras adicionais são consideradas para obter relações que envolvem o símbolo delimitador de sentença, $\boxed{\$}$ e o símbolo sentencial S:
  - $\boxed{\$} \prec X$ para cada símbolo $X \in ESQ(S)$
  - $X \succ \boxed{\$}$ para cada símbolo $X \in DIR(S)$
- Para um símbolo qualquer X e um símbolo terminal t:
  - $X \prec t$ ou $X \approx t$: Deslocar (D)
  - $X \succ t$: Reduzir (R)

Tabela DR para a gramática de expressões

Relações adicionais

$$ESQ(E) = \{E, M, P, (, v\}$$

$$\boxed{\$} \prec E$$

$$\boxed{\$} \prec M$$

$$\boxed{\$} \prec P$$

$$\boxed{\$} \prec ($$

$$\boxed{\$} \prec v$$

$$DIR(E) = \{M, P,), v\}$$

$$M \succ \boxed{\$}$$

$$P \succ \boxed{\$}$$

$$) \succ \boxed{\$}$$

$$v \succ \boxed{\$}$$

Tabela DR para a gramática de expressões

	$+$	$\times$	(	)	v	$\boxed{\$}$
E	D			D		
M	R	D		R		R
P	R	R		R		R
$+$			D		D	
$\times$			D		D	
(			D		D	
)	R	R		R	R	R
v	R	R		R	R	R
$\boxed{\$}$			D		D	

Reconhecimento de sentença

Estruturas auxiliares

Como o analisador preditivo, trabalha com duas estruturas auxiliares:

- Pilha para a forma sentencial, inicialmente apenas com o delimitador de sentença $
- Lista com os símbolos da sentença, com o delimitador $ ao final

Reconhecimento de sentença

Procedimento

Analisador verifica topo da pilha (linha na tabela DR) e próximo símbolo da sentença (coluna na tabela DR) para decidir qual próxima ação

- D Empilha o próximo símbolo da sentença
- R Os símbolos no topo da pilha correspondem ao lado direito de alguma produção, substituir pelo correspondente lado esquerdo

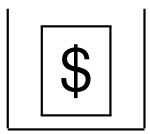
Se não houver ação D ou R na linha e coluna indicada, então há uma situação de erro.

Reconhecimento de sentença

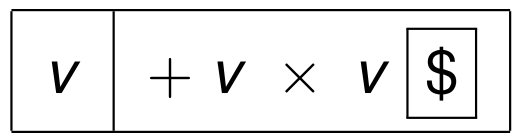
Exemplo (1): Estado inicial

Para reconhecer sentença $v + v \times v$ da gramática de expressões, o estado inicial é:

Pilha



Lista de tokens

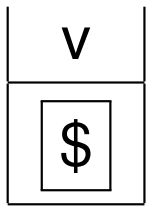


Reconhecimento de sentença

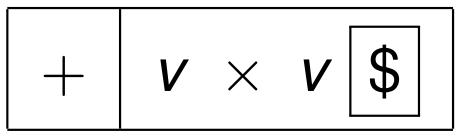
Exemplo (2)

Para [$, v], ação indicada é deslocar:

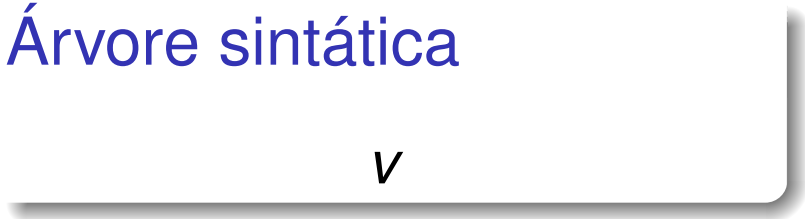
Pilha



Lista de tokens



Árvore sintática

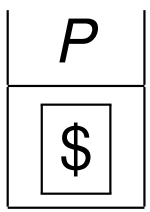


Reconhecimento de sentença

Exemplo (3)

Para $[v, +]$, ação indicada é reduzir $(P \rightarrow v)$:

Pilha



Lista de tokens

Árvore sintática

Reconhecimento de sentença

Exemplo (4)

Para $[P, +]$, reduzir $(M \rightarrow P)$:

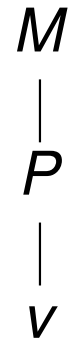
Pilha

M
$

Lista de tokens

$+$	$v \times v$ $

Árvore sintática

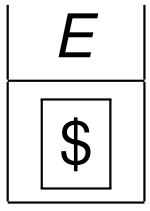


Reconhecimento de sentença

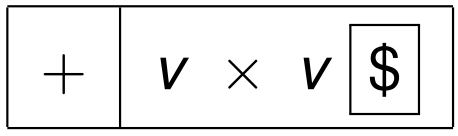
Exemplo (5)

Para $[M, +]$, reduzir $(E \rightarrow M)$:

Pilha



Lista de tokens



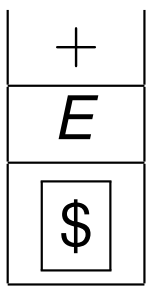
Árvore sintática

Reconhecimento de sentença

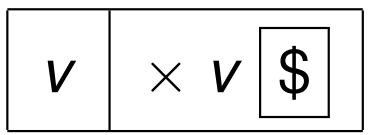
Exemplo (6)

Para $[E, +]$, deslocar:

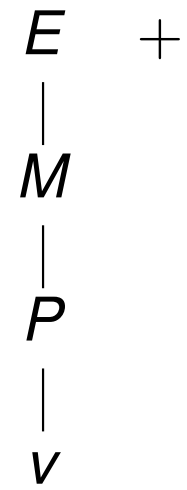
Pilha



Lista de tokens



Árvore sintática

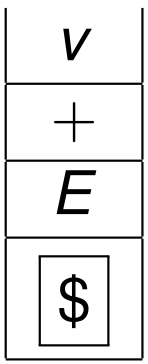


Reconhecimento de sentença

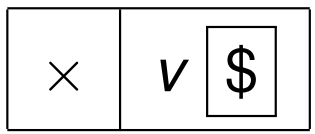
Exemplo (7)

Para $[+, v]$, deslocar:

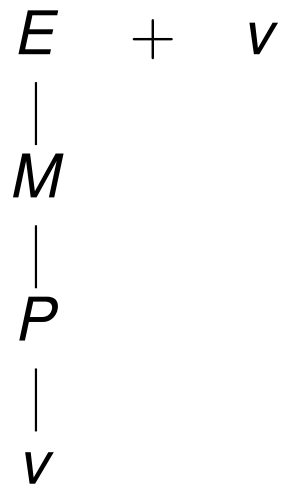
Pilha



Lista de tokens



Árvore sintática



Reconhecimento de sentença

Exemplo (8)

Para $[v, \times]$, reduzir $(P \rightarrow v)$:

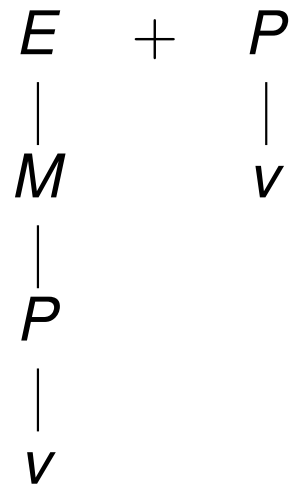
Pilha

P
$+$
E
$\boxed{\$}$

Lista de tokens

$\times$	v $\boxed{\$}$

Árvore sintática

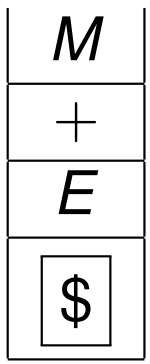


Reconhecimento de sentença

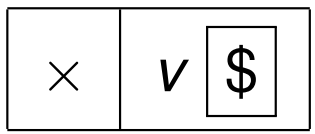
Exemplo (9)

Para $[P, \times]$, reduzir $(M \rightarrow P)$:

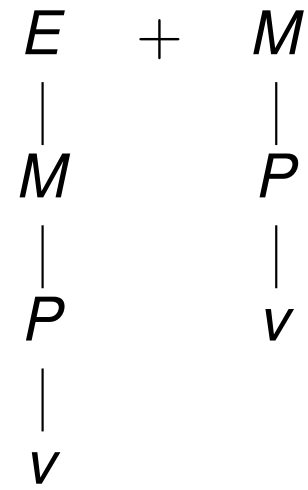
Pilha



Lista de tokens



Árvore sintática

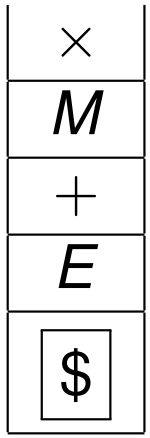


Reconhecimento de sentença

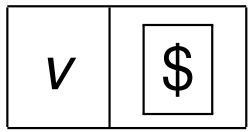
Exemplo (10)

Para $[M, \times]$, deslocar:

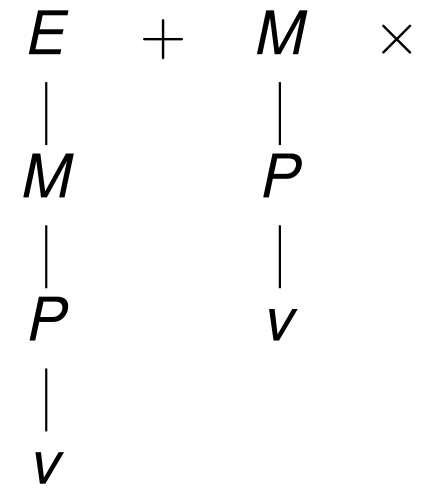
Pilha



Lista de tokens



Árvore sintática

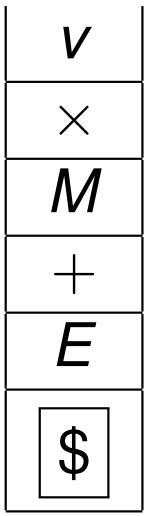


Reconhecimento de sentença

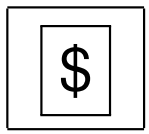
Exemplo (11)

Para $[\times, v]$, deslocar:

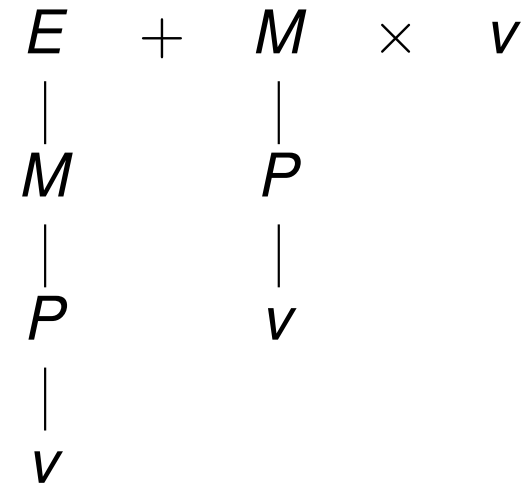
Pilha



Lista de tokens



Árvore sintática

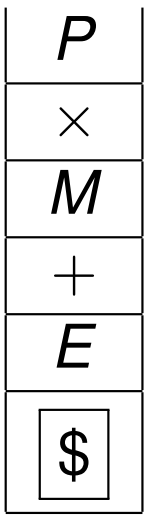


Reconhecimento de sentença

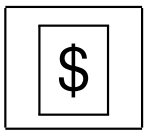
Exemplo (12)

Para $[v, \boxed{\$}]$, reduzir $(P \rightarrow v)$:

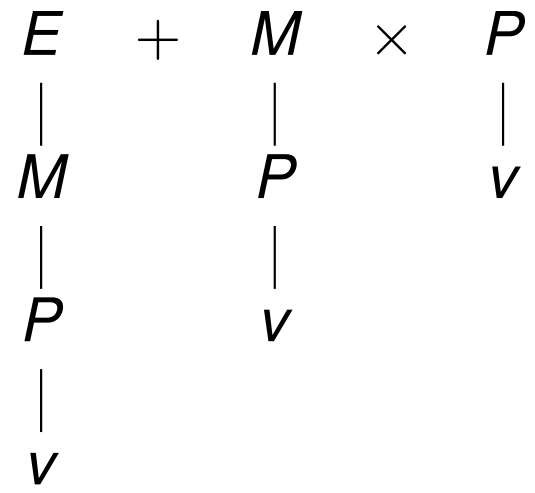
Pilha



Lista de tokens



Árvore sintática

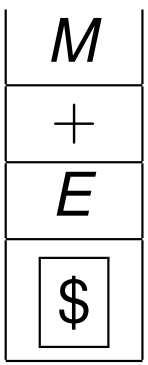


Reconhecimento de sentença

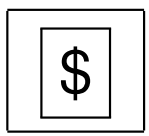
Exemplo (13)

Para $[P, \boxed{\$}]$, reduzir $(M \rightarrow M \times P)$:

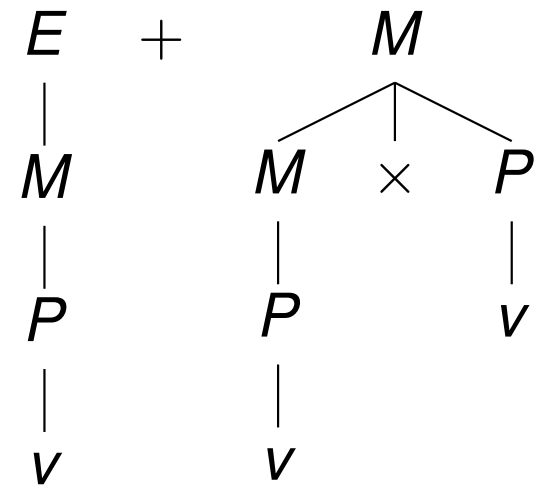
Pilha



Lista de tokens



Árvore sintática

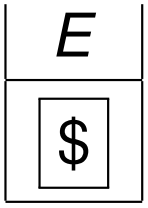


Reconhecimento de sentença

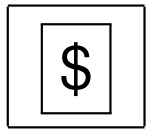
Exemplo (14)

Para $[M, \boxed{\$}]$, reduzir $(E \rightarrow E + M)$:

Pilha



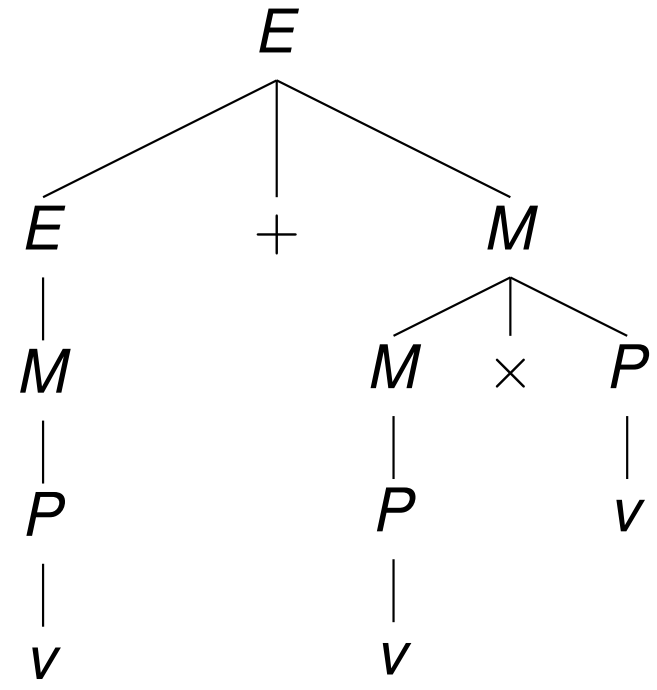
Lista de tokens



Condição de aceitação

Todos os tokens consumidos e apenas símbolo sentencial na pilha

Árvore sintática



Analisador de deslocamento e redução

- Construção ascendente da árvore sintática
- Para construção de uma tabela DR sem conflitos (duas possíveis ações numa mesma linha e coluna), gramática deve ser uma gramática de operadores
  - Nenhuma produção com dois símbolos não-terminais juntos no lado direito
  - Nenhuma produção com ε do lado direito

Analisador de deslocamento e redução

- É um analisador LR(1)
  - L Varredura da sentença da esquerda para a direita *(Left to right)*
  - R Derivação canônica mais à direita é utilizada para o reconhecimento *(Rightmost derivation)*
  - (1) Um símbolo da sentença analisado por vez para decidir a ação a ser tomada

Geradores de analisadores sintáticos

- Como para análise léxica, procedimentos para construção dos analisadores léxicos são sistemáticos
- Podem ser automatizados
- Papel das ferramentas para gerar analisadores sintáticos
  - Operam conjuntamente com analisadores léxicos

Gerador de analisador sintático

Yacc

Yet Another Compiler-Compiler

- Programa original do ambiente Unix, desenvolvido para trabalhar juntamente com o lex
- Gera uma função C que implementa um analisador sintático LR(1) a partir de uma especificação de gramática
- Como lex, deu origem a vários programas similares
  - Outras implemetações e outras linguagens

Especificação do analisador

Formato yacc

Como arquivo lex, três seções separadas por %%:

1. Definições e declarações
2. Regras da gramática e ações associadas
3. Código complementar

Especificação da gramática

Notação estilo BNF:

```
símbolo   :   expansão   ;
```

símbolo lado esquerdo da produção, um símbolo não-terminal

expansão lado direito da produção, a expansão de símbolo em termos de outros símbolos, sejam eles terminais ou não-terminais (definidos em outra produção da gramática)

Produções

- Símbolo sentencial: declaração `%start`
  - Na ausência da declaração, lado esquerdo da primeira produção é considerado o símbolo sentencial
- Se produção é recursiva, recursão à esquerda é preferível
- Expansões alternativas podem ser separadas com o símbolo |

Tokens

Os símbolos terminais das produções podem ser representados de duas formas:

- nomes simbólicos, declarados com o comando `%token`, ou
- representação direta com notação de caracteres de C, se for um único caráter

Tokens

Operadores

Símbolos terminais para operadores podem usar, ao invés de `token`, declaração com associatividade explícita:

%left	operador com associatividade à esquerda
%right	operador com associatividade à direita
%nonassoc	operador não-associativo

Estratégia para eliminar ambigüidade

- Operadores declarados na mesma linha têm mesma precedência
- Operadores declarados na última linha têm maior precedência

Manipulação das sentenças reconhecidas

Ação semântica

- Código C cuja execução está associada à aplicação da produção
- Especificado entre chaves após a expansão da produção
- Pode referenciar o valor semântico de um token da expansão
  - Notação posicional: $1 é o primeiro token, $2 o segundo...
  - Para tokens, analisador léxico deve definir esse valor por meio da variável `yylval`
  - Um valor semântico pode ser associado ao lado esquerdo ($$), para uso em outras expansões
  - Tipo padrão do valor semântico é `int`, mas pode ser alterado com a redefinição da *string* `YYSTYPE`

Tratamento de erros

error	símbolo não-terminal pré-definido que está associado a sentenças não reconhecidas pela gramática especificada
yyerrok	macro definida em C que limpa a entrada de caracteres para que a análise possa prosseguir
yyerror()	função definida pelo usuário, invocada pelo analisador para apresentação de mensagens de erro

Desenvolvimento de uma aplicação

A gramática de expressões (soma ou multiplicação) é usada neste exemplo como a semente para uma calculadora:

$$
\begin{array}{lcl}
E & \rightarrow & E + E \\
E & \rightarrow & E \times E \\
E & \rightarrow & (E) \\
E & \rightarrow & v
\end{array}
$$

Exemplo

Primeira seção: declarações C/C++ (1)

```
%{
#include <iostream>
using namespace std;
    extern "C"
    {
            int yyparse(void);
            int yylex(void);
            int yywrap()  {
                    return 1;
            }

    }
void yyerror(char *);
%}
```

Exemplo

Primeira seção: declarações yacc (2)

```
%start  entrada
%token  VALOR FIMLIN
%left   SOMA
%left   MULT
%token  ABRPAR FECPAR
%%
```

Exemplo

Segunda seção: produções e ações (3)

```
expr :   expr SOMA expr      {   $$ = $1 + $3; }
     |   expr MULT expr      {   $$ = $1 * $3; }
     |   ABRPAR expr FECPAR      {   $$ = $2;   }
     |   VALOR
     ;
```

Exemplo

Segunda seção: produções e ações (4)

```
entrada :   /* vazia */
        |   entrada result
        ;
result  :   FIMLIN
        |   expr FIMLIN
            { cout << "Resposta: " << $1 << endl; }
        |   error FIMLIN  {  yyerrok;  }
        ;
```

Exemplo

Terceira seção: código C (5)

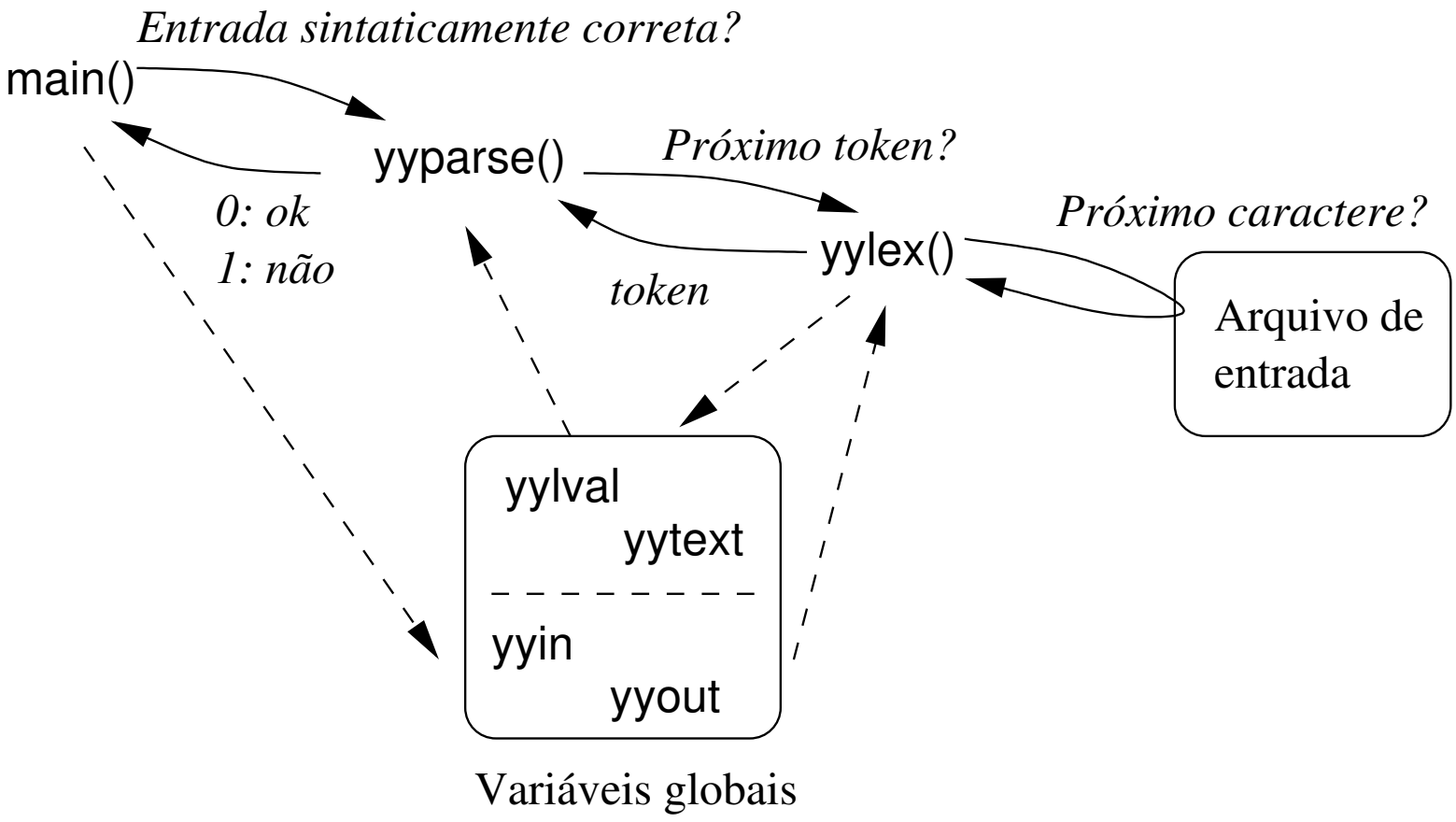
```
%%
void yyerror(char* msg) {
    extern char* yytext;
    cout << msg << ": " << yytext << endl;
}
```

Integração com lex

- Arquivo de cabeçalho produzido com opção `-d`
- Definições de valores para os nomes simbólicos associados aos tokens, declarados na primeira seção do arquivo de especificação de yacc (`acumuly.y`)
- Execução com `bison` (implementação Gnu de yacc)

```
> bison -d acumuly.y
```

Integração lex-yacc



Integração com lex

Exemplo: arquivo acumuly.l (6)

```
%{
#include "acumuly.tab.h"
extern YYSTYPE yylval;
%}
%%
[0-9]+ {  yylval = atoi(yytext);
          return VALOR;
       }
\+     { return SOMA; }
\*     { return MULT; }
\(     { return ABRPAR; }
\)     { return FECPAR; }
\n     { return FIMLIN; }
%%
```

Geração da aplicação

- Analisador léxico é um programa C, produzido com `flex`

```
> flex acumuly.l
> gcc -c lex.yy.c
```

- Compilação do analisador sintático

```
> g++ -o acc lex.yy.o acumuly.tab.c -ly -lfl
```

Exemplo

Execução (7)

```
> ./acc
1+2+4*5
Resposta: 23
2+2
Resposta: 4
1=2+3+4
=syntax error: 2
1+2
Resposta: 3
[^d]
>
```

Sugestões de leitura (Web)

- Bison: Gnu parser generator
  `http://www.gnu.org/software/bison/`