

# Tema 4

## Análise Sintáctica

Gramáticas independentes de contexto, análise sintáctica descendente e ascendente

*Compiladores+LFA, 2º semestre 2019-2020*

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Crítérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções  $\epsilon$

Eliminação de  
recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first, follow e lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica

Miguel Oliveira e Silva, Artur Pereira  
DETI, Universidade de Aveiro

# Análise sintáctica

## Análise Sintáctica

### Análise sintáctica: Estrutura de um Compilador

### Gramáticas Independentes de Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

### Definição de gramáticas

Símbolos inacessíveis

Crítérios de sanidade em  
gramáticas

### Transformações de gramáticas

Eliminação de produções  $\epsilon$

Eliminação de  
recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

### Análise sintáctica descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

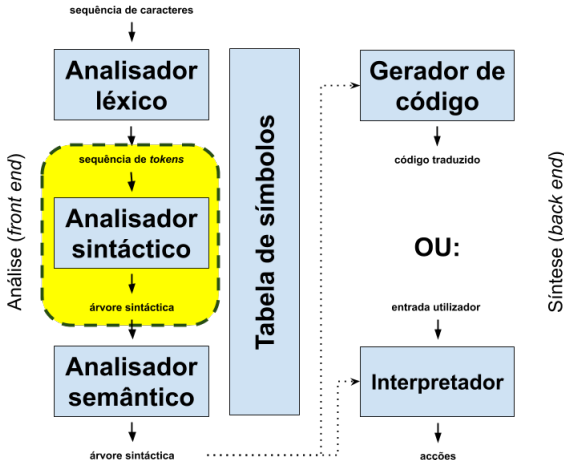
Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica

# Análise sintáctica: Estrutura de um Compilador

- Vamos agora analisar com mais detalhe a fase de análise sintáctica:



# Gramáticas Independentes de Contexto

## Análise Sintáctica

Análise sintáctica:  
Estrutura de um  
Compilador

## Gramáticas Independentes de Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

## Definição de gramáticas

Símbolos inacessíveis

Crítérios de sanidade em  
gramáticas

## Transformações de gramáticas

Eliminação de produções  $\epsilon$

Eliminação de  
recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

## Análise sintáctica descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica

# Definição de gramática

- Uma gramática é um quádruplo  $G = (T, N, S, P)$ , onde:
  - 1  $T$  é um conjunto finito não vazio de símbolos terminais;
  - 2  $N$  é um conjunto finito não vazio, disjunto de  $T$  ( $N \cap T = \emptyset$ ), de símbolos não terminais;
  - 3  $S \in N$  é o símbolo inicial;
  - 4  $P$  é um conjunto finito de produções (ou regras de rescrita), cada uma da forma  $\alpha \rightarrow \beta$ .
- $\alpha$  e  $\beta$  são designados por **cabeça da produção** e **corpo da produção**, respetivamente.
- No caso geral:

$$\alpha \in (T \cup N)^* N (T \cup N)^* \\ \beta \in (T \cup N)^*$$

isto é,  $\alpha$  é uma cadeia de símbolos terminais e não terminais contendo, pelo menos, um símbolo não terminal; e  $\beta$  é uma cadeia de símbolos terminais e não terminais.

- Uma gramática  $G = (T, N, S, P)$  diz-se independente ou livre de contexto se, para qualquer produção  $(\alpha \rightarrow \beta) \in P$ , a condição seguinte é satisfeita:

$$\alpha \in N$$

- A linguagem gerada por uma gramática independente do contexto diz-se independente do contexto.
  - Note que as gramáticas regulares são também gramáticas independentes do contexto.
- As gramáticas independentes do contexto são fechadas sob as operações de reunião, concatenação e fecho.
  - No entanto, as GIC não são fechadas sob as operações de intersecção e complementação.

## Exemplo de gramática

- Por exemplo a gramática  $G = (\{0, 1\}, \{S, A\}, S, P)$ , onde  $P$  é constituído pelas regras:

$$S \rightarrow 0S$$

$$S \rightarrow 0A$$

$$A \rightarrow 0A1$$

$$A \rightarrow \varepsilon$$

- A linguagem definida por esta gramática é:

$$L = \{0^n 1^m : n \in \mathbb{N} \wedge m \in \mathbb{N}_0 \wedge n > m\}$$

- Esta gramática é um exemplo de uma **gramática independente (ou livre) de contexto**.
- Dada a gramática  $G_a = (\{0, 1\}, \{A\}, A, P)$  (para as mesmas produções  $P$ ), que linguagem será definida por  $G_a$ ?

## Outro exemplo de gramática

- Vamos supor que queremos definir uma gramática, num alfabeto binário ( $B = \{0, 1\}$ ), que garanta que o número de uns é igual ao números de zeros. Isto é para a linguagem:

$$L = \{u \in B^* \mid \#(0, u) = \#(1, u)\}$$

- Para chegar a esta gramática podemos desde logo constatar o seguinte:
  - caso apareça um **0** terá de aparecer um **1**;
  - a ordem dos símbolos é irrelevante (não pode depender de permutações).
- Para garantir isso basta ter um conta as duas variantes:

$$S \rightarrow \varepsilon \mid 1 S 0 S \mid 0 S 1 S$$

- Esta gramática não é regular já que não é possível encostar os terminais totalmente à esquerda (ou à direita).
- É sim um outro exemplo de uma **gramática independente (ou livre) de contexto**.



# Derivações

## Análise Sintáctica

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

### Derivações

Árvore sintáctica  
Gramáticas ambíguas  
Projecto de gramáticas  
simples  
Operações sobre GIC  
Reunião  
Concatenação  
Fecho de Kleene

### Definição de gramáticas

Símbolos inacessíveis  
Critérios de sanidade em  
gramáticas

### Transformações de gramáticas

Eliminação de produções  $\epsilon$   
Eliminação de  
recursividade à esquerda  
Factorização à esquerda  
Eliminação de  
ambiguidades

### Análise sintáctica descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*  
Análise sintáctica  
descendente recursiva  
Análise sintáctica  
descendente preditiva  
Análise sintáctica

# Derivações

- Podemos tratar as produções como regras de rescrita, substituindo um símbolo não terminal pelo corpo de uma das suas produções.
- Por exemplo, se tivermos a seguinte gramática:

$$E \rightarrow E * E \mid E + E \mid -E \mid (E) \mid \mathbf{id}$$

podemos rescrever o símbolo não terminal  $E$  por qualquer uma das alternativas aplicáveis. Por exemplo:

$$E \Rightarrow -E$$

que se deve ler: “ $E$  deriva  $-E$ ”.

- Uma rescrita mais extensa poderá ser:

$$E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(\mathbf{id})$$

- Chamamos a uma sequência deste género, uma *derivação* de  $-(\mathbf{id})$  a partir de  $E$ .
- Esta derivação prova que a palavra  $-(\mathbf{id})$  é um caso particular de uma expressão  $E$ .

## Derivações: definição geral

- Dada uma palavra  $\alpha A \beta$  e uma produção  $A \rightarrow v$ , chama-se **derivação direta** à rescrita de  $\alpha A \beta$  em  $\alpha v \beta$ , denotando-se

$$\alpha A \beta \Rightarrow \alpha v \beta$$

- Dada uma palavra  $\alpha A \beta$ , com  $\beta \in T^*$ , e uma produção  $A \rightarrow v$ , chama-se **derivação direta à direita** à rescrita de  $\alpha A \beta$  em  $\alpha v \beta$ , denotando-se

$$\alpha A \beta \xRightarrow{D} \alpha v \beta$$

- Dada uma palavra  $\alpha A \beta$ , com  $\alpha \in T^*$ , e uma produção  $A \rightarrow v$ , chama-se **derivação direta à esquerda** à rescrita de  $\alpha A \beta$  em  $\alpha v \beta$ , denotando-se

$$\alpha A \beta \xRightarrow{E} \alpha v \beta$$

- Isto é, é possível fazer uma derivação directa à direita/esquerda quando o lado direito da produção tem terminais à direita/esquerda da regra a rescrever.

## Derivações: definição geral (2)

- Chama-se **derivação** a uma sucessão de zero ou mais derivações diretas, denotando-se

$$\alpha \xRightarrow{*} \beta$$

- Chama-se **derivação à direita** a uma sucessão de zero ou mais derivações diretas à direita, denotando-se

$$\alpha \xRightarrow{D^*} \beta$$

ou, equivalentemente,

$$\alpha = \alpha_0 \xRightarrow{D} \alpha_1 \xRightarrow{D} \dots \xRightarrow{D} \alpha_n = \beta$$

onde  $n$  é o comprimento da derivação.

- Chama-se **derivação à esquerda** a uma sucessão de zero ou mais derivações diretas à esquerda, denotando-se

$$\alpha \xRightarrow{E^*} \beta$$

ou, equivalentemente,

$$\alpha = \alpha_0 \xRightarrow{E} \alpha_1 \xRightarrow{E} \dots \xRightarrow{E} \alpha_n = \beta$$

onde  $n$  é o comprimento da derivação.

## Derivações: definição geral (3)

- Recuperando a gramática:

$$E \rightarrow E * E \mid E + E \mid -E \mid (E) \mid \mathbf{id}$$

- Podemos verificar que a palavra  $-(\mathbf{id} + \mathbf{id})$  é um caso particular desta gramática, porque existe (pelo menos) uma derivação:

$$E \xRightarrow{E} -E \xRightarrow{E} -(E) \xRightarrow{E} -(E+E) \xRightarrow{E} -(\mathbf{id}+E) \xRightarrow{E} -(\mathbf{id}+\mathbf{id})$$

- Isto é:  $E \xRightarrow{*} -(\mathbf{id} + \mathbf{id})$
- Em cada passo da derivação há duas opções a tomar: escolher qual o não terminal a substituir e, tendo feito esta escolha, escolher a produção desse não terminal.
- A palavra  $-(\mathbf{id} + \mathbf{id})$  tem uma derivação alternativa (que difere da anterior na penúltima rescrita) dada por:

$$E \xRightarrow{D} -E \xRightarrow{D} -(E) \xRightarrow{D} -(E+E) \xRightarrow{D} -(E+\mathbf{id}) \xRightarrow{D} -(\mathbf{id}+\mathbf{id})$$

## Exemplo de derivação

- Considere, sobre o alfabeto  $T = \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$ , a gramática seguinte

$$S \rightarrow \varepsilon \mid \mathbf{aB} \mid \mathbf{bA} \mid \mathbf{cS}$$

$$A \rightarrow \mathbf{aS} \mid \mathbf{bAA} \mid \mathbf{cA}$$

$$B \rightarrow \mathbf{aBB} \mid \mathbf{bS} \mid \mathbf{cB}$$

- Determine as derivações à direita e à esquerda da palavra **aabcbc**

- à direita:

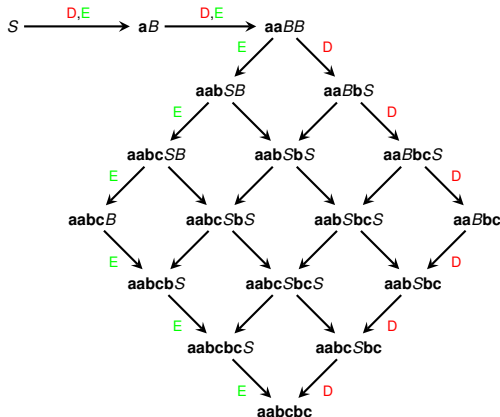
$$\begin{aligned} S &\Rightarrow \mathbf{aB} \Rightarrow \mathbf{aaBB} \Rightarrow \mathbf{aaBbS} \Rightarrow \mathbf{aaBbcS} \\ &\Rightarrow \mathbf{aaBbc} \Rightarrow \mathbf{aabSbc} \Rightarrow \mathbf{aabcbS} \Rightarrow \mathbf{aabcbc} \end{aligned}$$

- à esquerda:

$$\begin{aligned} S &\Rightarrow \mathbf{aB} \Rightarrow \mathbf{aaBB} \Rightarrow \mathbf{aabSB} \Rightarrow \mathbf{aabcbS} \\ &\Rightarrow \mathbf{aabcb} \Rightarrow \mathbf{aabcbS} \Rightarrow \mathbf{aabcbcbS} \Rightarrow \mathbf{aabcbcb} \end{aligned}$$

## Exemplo de derivação (2)

- A árvore seguinte capta as alternativas de derivação. Considera-se novamente a palavra **aabcbc** e a gramática anterior.



- Note que esta árvore contém todas as combinações possíveis de derivações à esquerda e à direita.

# Árvore sintáctica

## Análise Sintáctica

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas  
Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Crítérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções  $\epsilon$

Eliminação de  
recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

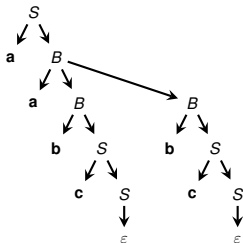
Análise sintáctica  
descendente preditiva

Análise sintáctica



# Árvore sintáctica

- Uma **árvore sintáctica**, ou árvore de derivação (*parse tree*), é uma representação de uma derivação onde os nós-ramos são elementos de  $N$  (não terminais) e os nós-folhas são elementos de  $T$  (terminais).
- A árvore sintáctica da palavra **aabcbc** na gramática anterior é:



## Análise Sintáctica

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas  
Projecto de gramáticas  
simples

Operações sobre GIC  
Reunião

Concatenação  
Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis  
Critérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções- $\epsilon$   
Eliminação de  
recursividade à esquerda  
Factorização à esquerda  
Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*  
Análise sintáctica  
descendente recursiva  
Análise sintáctica  
descendente preditiva  
Análise sintáctica

## Árvore sintáctica (2)

- A relação entre gramáticas independentes de contexto e a estrutura de dados tipo árvore é mais profunda do que pode parecer à primeira vista.
- O facto de nas GICs todas as produções terem um, e um só, símbolo não terminal na cabeça da produção, faz com que este possa ser reescrito (derivado) em função da sequência de símbolos do corpo (aplicável) da produção.
- Ou seja, numa estrutura de dados tipo árvore em que a cabeça da produção é o nó “pai”, e a sequência de símbolos do corpo da produção são os filhos.
- O facto da decisão sobre possíveis derivações depender apenas de um (e um só) símbolo não terminal, mais torna eficiente algoritmos automáticos para esse reconhecimento.
- Nas gramáticas que não são independentes de contexto (dependentes de contexto ou sem restrição) esta representação não é possível.

### Análise Sintáctica

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas  
Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Critérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções- $\epsilon$

Eliminação de  
recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Crítérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções- $\epsilon$

Eliminação de  
recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica

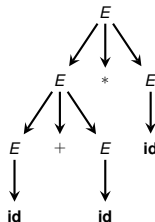
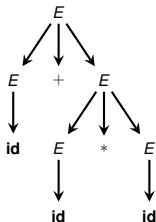
# Gramáticas ambíguas

- Recuperando novamente a gramática:

$$E \rightarrow E * E \mid E + E \mid -E \mid (E) \mid \text{id}$$

considere a entrada **id + id \* id**

- Podemos extrair duas árvores sintáticas distintas:



- A árvore da esquerda segue as regras de precedência da aritmética, e a da direita dá maior prioridade à soma.

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Crítérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções-ε

Eliminação de  
recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

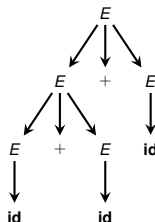
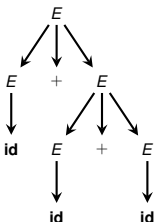
Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica

## Gramáticas ambíguas (2)

- Este problema não resulta apenas da precedência entre duas operações distintas, a associatividade aplicável à mesma operação pode também gerar ambiguidade.
- Da entrada **id + id + id** também resultam duas árvores sintáticas distintas:



- A árvore da esquerda tem associatividade à direita, e a da direita tem associatividade à esquerda.

# Gramáticas ambíguas: definição

- Diz-se que uma palavra é derivada **ambiguamente** se possuir duas ou mais árvores de derivação distintas.
- Diz-se que uma gramática é **ambígua** se possuir pelo menos uma palavra gerada ambiguamente.
- Muitas vezes é possível definir-se uma gramática não ambígua que gere a mesma linguagem do que outra gramática ambígua.
- No entanto, há gramáticas **inerentemente ambíguas**.
- Já vimos um exemplo comum resultante das expressões aritméticas.
- Vejamos outro exemplo:

$$L = \{a^i b^j c^k : i = j \vee j = k\}$$

## Análise Sintáctica

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Crítérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções-ε

Eliminação de  
recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica

# Gramáticas ambíguas: exemplo 1

- Uma gramática (GIC) para esta linguagem será  $G = (\{\mathbf{a}, \mathbf{b}, \mathbf{c}\}, \{S, AB, C, A, BC\}, S, P)$ , em que as produções ( $P$ ) são:

$$\begin{aligned} S &\rightarrow AB\ C \mid A\ BC \\ AB &\rightarrow \mathbf{a}\ AB\ \mathbf{b} \mid \varepsilon \\ BC &\rightarrow \mathbf{b}\ BC\ \mathbf{c} \mid \varepsilon \\ A &\rightarrow \mathbf{a}\ A \mid \varepsilon \\ C &\rightarrow \mathbf{c}\ C \mid \varepsilon \end{aligned}$$

- Assim, caso  $i = j = k$ , teremos duas possíveis árvores sintáticas (uma derivando por  $AB$  outra por  $BC$ ).

## Gramáticas ambíguas: exemplo 2

- Recuperando novamente a gramática:

$$E \rightarrow E * E \mid E + E \mid -E \mid (E) \mid \text{id}$$

- Se desejarmos garantir que a operação da multiplicação tem precedência relativamente à soma, sem utilizar convenções que não estejam expressas directamente na gramática (como acontece com a regra ANTLR de dar prioridade às regras definidas primeiro), será que conseguimos gerar uma gramática não ambígua?
- Vejamos se as seguintes regras funcionam:

$$E \rightarrow P + E \mid P$$

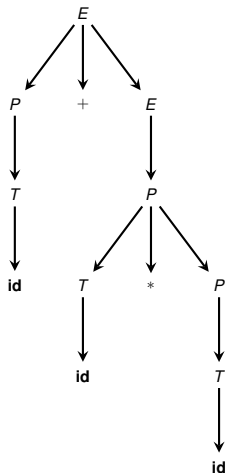
$$P \rightarrow T * P \mid T$$

$$T \rightarrow -E \mid (E) \mid \text{id}$$



## Gramáticas ambíguas: exemplo 2 (2)

- Para a entrada **id + id \* id**, teremos apenas a seguinte árvore sintáctica:



# Projecto de gramáticas simples

## Análise Sintáctica

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Crítérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções  $\epsilon$

Eliminação de  
recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica

## Exemplo de projecto de GIC: solução 1

- Sobre o conjunto de terminais  $T = \{\mathbf{a}, \mathbf{b}\}$ , determine uma gramática independente do contexto que represente a linguagem:

$$L1 = \{w \in T^* : \#(a, w) = \#(b, w)\}$$

- Anteriormente já foi resolvido um problema similar.
- Logo:

$$S \rightarrow \varepsilon \mid \mathbf{a S b S} \mid \mathbf{b S a S}$$

- A gramática é ambígua?
- Sim! (Analise a palavra **aabbab**)

### Análise Sintáctica

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Crítérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções- $\varepsilon$

Eliminação de  
recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica

# Exemplo de projecto de GIC: outras soluções

- Outra possível solução é:

$$S \rightarrow \varepsilon \mid \mathbf{a}B \mid \mathbf{b}A$$

$$A \rightarrow \mathbf{a}S \mid \mathbf{b}AA$$

$$B \rightarrow \mathbf{a}BB \mid \mathbf{b}S$$

- Ainda outra possibilidade é:

$$S \rightarrow \varepsilon \mid \mathbf{a}BS \mid \mathbf{b}AS$$

$$A \rightarrow \mathbf{a} \mid \mathbf{b}AA$$

$$B \rightarrow \mathbf{a}BB \mid \mathbf{b}$$

## Análise Sintáctica

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Crítérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções- $\varepsilon$

Eliminação de  
recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica

## Outro Exemplo de projecto de GIC

- Sobre o conjunto de terminais  $T = \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$ , determine uma gramática independente do contexto que represente a linguagem:

$$L = \{w \in T^* : \#(a, w) = \#(b, w)\}$$

- Este problema difere do anterior pelo facto de haver mais um símbolo terminal.
- Logo:

$$S \rightarrow \varepsilon \mid \mathbf{a} S \mathbf{b} S \mid \mathbf{b} S \mathbf{a} S \mid \mathbf{c} S$$

# Operações sobre GIC

## Análise Sintáctica

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Crítérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções  $\epsilon$

Eliminação de  
recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica

## Operações sobre GIC: reunião

- Sejam  $G_1 = (T_1, N_1, S_1, P_1)$  e  $G_2 = (T_2, N_2, S_2, P_2)$  duas gramáticas independentes de contexto quaisquer com  $N_1 \cap N_2 = \emptyset$

- A gramática  $G = (T, N, S, P)$  onde:

$$T = T_1 \cup T_2$$

$$N = N_1 \cup N_2 \cup \{S\}$$

$$S \notin (T \cup N_1 \cup N_2)$$

$$P = \{S \rightarrow S_1, S \rightarrow S_2\} \cup P_1 \cup P_2$$

é independente de contexto e gera a linguagem  
 $L = L(G_1) \cup L(G_2)$

- As produções de  $G_1$  e  $G_2$  são transferidas para a nova gramática sem nenhuma alteração substancial (eventualmente pode ser necessário mudar nomes de símbolos não terminais).
- A nova produção  $S \rightarrow S_i$ , com  $i = 1, 2$ , permite que  $G$  gere a linguagem  $L(G_i)$ .

## Operações sobre GIC: reunião (exemplo)

- Sobre o conjunto de terminais  $T = \{a, b, c\}$ , determine uma gramática independente de contexto que represente a linguagem

$$L = \{w \in T^* : L_1 \cup L_2\}$$

sabendo que:

$$L_1 = \{w \in T^* : \#(a, w) = \#(b, w)\}$$

$$L_2 = \{w \in T^* : \#(a, w) = \#(c, w)\}$$

- Primeiro vamos gerar as gramáticas para cada linguagem:

$$L_1 : S_1 \rightarrow \varepsilon \mid a S_1 b S_1 \mid b S_1 a S_1 \mid c S_1$$

$$L_2 : S_2 \rightarrow \varepsilon \mid a S_2 c S_2 \mid c S_2 a S_2 \mid b S_2$$

- Donde resulta a seguinte solução:

$$S \rightarrow S_1 \mid S_2$$

$$S_1 \rightarrow \varepsilon \mid a S_1 b S_1 \mid b S_1 a S_1 \mid c S_1$$

$$S_2 \rightarrow \varepsilon \mid a S_2 c S_2 \mid c S_2 a S_2 \mid b S_2$$



# Operações sobre GIC: concatenação

- Sejam  $G_1 = (T_1, N_1, S_1, P_1)$  e  $G_2 = (T_2, N_2, S_2, P_2)$  duas gramáticas independentes de contexto quaisquer com  $N_1 \cap N_2 = \emptyset$

- A gramática  $G = (T, N, S, P)$  onde:

$$T = T_1 \cup T_2$$

$$N = N_1 \cup N_2 \cup \{S\}$$

$$S \notin (T \cup N_1 \cup N_2)$$

$$P = \{S \rightarrow S_1 S_2\} \cup P_1 \cup P_2$$

é independentes de contexto e gera a linguagem

$$L = L(G_1) \cdot L(G_2)$$

- A nova produção  $S \rightarrow S_1 S_2$  justapõe palavras de  $L(G_2)$  às de  $L(G_1)$ .

# Operações sobre GIC: concatenação (exemplo)

- Sobre o conjunto de terminais  $T = \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$ , determine uma gramática independente de contexto que represente a linguagem

$$L = L_1 \cdot L_2$$

sabendo que:

$$L_1 = \{w_1 \in T^* : \#(a, w_1) = \#(b, w_1)\}$$

$$L_2 = \{w_2 \in T^* : \#(a, w_2) = \#(c, w_2)\}$$

- Solução possível:

$$S \rightarrow S_1 S_2$$

$$S_1 \rightarrow \varepsilon \mid \mathbf{a} S_1 \mathbf{b} S_1 \mid \mathbf{b} S_1 \mathbf{a} S_1 \mid \mathbf{c} S_1$$

$$S_2 \rightarrow \varepsilon \mid \mathbf{a} S_2 \mathbf{c} S_2 \mid \mathbf{c} S_2 \mathbf{a} S_2 \mid \mathbf{b} S_2$$

# Operações sobre GIC: fecho de Kleene

- Seja  $G_1 = (T_1, N_1, S_1, P_1)$  uma gramática independentes de contexto qualquer.
- A gramática  $G = (T, N, S, P)$  onde:

$$T = T_1$$

$$N = N_1 \cup \{S\}$$

$$S \notin (T \cup N_1)$$

$$P = \{S \rightarrow \varepsilon, S \rightarrow S_1 S\} \cup P_1$$

é independentes de contexto e gera a linguagem  
 $L = (L(G_1))^*$

# Operações sobre GIC: fecho de Kleene (exemplo)

- Sobre o conjunto de terminais  $T = \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$ , determine uma gramática independente de contexto que represente a linguagem

$$L = L_1^*$$

sabendo que:

$$L_1 = \{w \in T^* : \#(a, w) = \#(b, w)\}$$

- Solução possível:

$$\begin{aligned} S &\rightarrow \varepsilon \mid S_1 S \\ S_1 &\rightarrow \varepsilon \mid \mathbf{a} S_1 \mathbf{b} S_1 \mid \mathbf{b} S_1 \mathbf{a} S_1 \mid \mathbf{c} S_1 \end{aligned}$$

# Definição de gramáticas: Símbolos inacessíveis

## Análise Sintáctica

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Crítérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções  $\epsilon$

Eliminação de  
recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica

- Tal como na construção de um programa é trivial definir funções que nunca são utilizadas no programa, também nas gramáticas podemos ter regras inacessíveis.
- Seja  $G = (T, N, S, P)$  uma gramática qualquer. Um símbolo  $X$  diz-se **acessível** se for possível transformar o símbolo inicial ( $S$ ) numa expressão que contenha esse símbolo. Ou seja,

$$S \xRightarrow{*} \alpha X \beta \quad \text{com} \quad \alpha, \beta \in (N \cup T)^*$$

- Caso contrário, o símbolo não terminal diz-se **inacessível**.

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Critérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções- $\epsilon$

Eliminação de

recursividade à esquerda

Factorização à esquerda

Eliminação de

ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica

# Símbolos inacessíveis: exemplo

- Considere a gramática

$G = (\{\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}\}, \{S, C, D, X\}, S, P)$ , com as produções  $P$ :

$$S \rightarrow \varepsilon \mid \mathbf{a} S \mathbf{b} \mid \mathbf{c} C \mathbf{c}$$

$$C \rightarrow \mathbf{c} S \mathbf{c}$$

$$D \rightarrow \mathbf{d} X \mathbf{d}$$

$$X \rightarrow C C$$

- É impossível transformar  $S$  numa expressão que contenha  $D$ ,  $\mathbf{d}$  ou  $X$ , pelo que esses símbolos são inacessíveis.
- Os restantes símbolos são acessíveis.

## Eliminação de símbolos inacessíveis

- Podemos definir um algoritmo para descobrir os símbolos acessíveis (logo, que permita a eliminação dos restantes).
- O conjunto de símbolos acessíveis  $V_A$  pode ser determinado com o seguinte algoritmo:

```
 $V_A = \{S\}$   
 $N_X = V_A$   
while  $N_X \neq \phi$  do  
   $A = \text{element-of}(N_X)$   
   $N_X = N_X \setminus \{A\}$   
  foreach  $(A \rightarrow \alpha) \in P$  do  
    foreach  $x \in \alpha$  do  
      if  $x \notin V_A$  then  
         $V_A = V_A \cup \{x\}$   
        if  $x \in N$  then  
           $N_X = N_X \cup \{x\}$   
        end // if  
      end // if  
    end // foreach  
  end // foreach  
end // while
```

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Crítérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções- $\epsilon$

Eliminação de

recursividade à esquerda

Factorização à esquerda

Eliminação de

ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica



# Definição de gramáticas: Critérios de sanidade

## Análise Sintáctica

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Critérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções  $\epsilon$

Eliminação de  
recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica

# Recursividade: Condições de Sanidade

- Por agora deve começar a ser bem evidente duas características essenciais das gramáticas:
  - 1 são expressas de forma **declarativa**;
  - 2 são muitas vezes, directa ou indirectamente, **recursivas**.
- Como exemplo, a gramática seguinte tem regras recursivas (directa e indirectamente):

```
instruction  →  ε | conditional | ...
conditional →  if expression then instruction |
               if expression then instruction else instruction
expression  →  expression '+' expression |
               ...
               ...
```

- A regra *expression* é directamente recursiva, e a regra *instruction* é indirectamente recursiva.
- Não é assim de estranhar que deva ser recuperada para o campo das gramáticas as **condições de sanidade** aplicáveis às funções recursivas (independentemente das gramáticas serem implementadas por funções).

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Critérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções-ε

Eliminação de  
recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica

# Recursividade: Condições de Sanidade (2)

- Uma definição recursiva útil requer que:
  - 1 Exista pelo menos uma alternativa não recursiva (**CASO(S) LIMITE**);
  - 2 Todas as alternativas recursivas ocorram num contexto diferente do original (**VARIABILIDADE**);
  - 3 Para todas as alternativas recursivas, a mudança do contexto (2) levam-nas mais próximo de, pelo menos, uma alternativa não recursiva (1) (**CONVERGÊNCIA**).
- As condições (1) e (2) são **necessárias**. As três juntas são **suficientes** para garantir a terminação da recursão.
- A aplicação destes conceitos às gramáticas, permite que se definam condições de sanidade a elas aplicadas.
- Como resultado destas condições, para garantir que uma qualquer **gramática**  $G = (T, N, S, P)$  **faça sentido**, todos os seus símbolos não terminais ( $Z$ ) têm de poder ser derivados numa sequência de símbolos terminais.
- Isto é:  $Z \Rightarrow^* u \wedge u \in T^* \quad \text{com} \quad Z \in N$

# Recursividade: Condições de Sanidade (3)

## ① CASO(S) LIMITE:

- Para cumprir esta condição basta garantir que cada símbolo não terminal tenha sempre directa ou indirectamente uma alternativa não recursiva.

## ② VARIABILIDADE:

- Para esta condição é necessário que a regra não tenha nenhum alternativa em que esteja definida (directa ou indirectamente) apenas à custa de si própria.
- Isto é, só podemos voltar à mesma regra depois de existir pelo menos uma derivação diferente da regra.

## ③ CONVERGÊNCIA:

- Às condições anteriores acresce a condição de só podermos voltar à mesma regra depois de existir pelo menos uma derivação que garanta o consumo (antes ou depois) de pelo menos um símbolo terminal.
  - Para garantir que a regra como um todo faça sentido, esta condição tem de ser aplicável a todas as definições alternativas da regra.
- Uma gramática fará sentido, caso estas condições se apliquem ao símbolo inicial.

## Recursividade: Condições de Sanidade (4)

- Como exemplo, considere a seguinte gramática  $G = (\{\mathbf{a}, \mathbf{b}, \mathbf{c}\}, \{S, U, V, W, X\}, S, P)$ , com  $P$ :

$$S \rightarrow \mathbf{a} S \mathbf{b} \mid U \mid V \mid W$$
$$U \rightarrow \mathbf{c}$$
$$V \rightarrow \mathbf{a} V$$
$$W \rightarrow X$$
$$X \rightarrow X$$

- Requisito ①
  - as regras  $\{S, U\}$  cumprem;
  - as regras  $\{V, W, X\}$  não cumprem.
- Requisito ②
  - as regras  $\{S, U, V, W\}$  cumprem;
  - a regra  $\{X\}$  não cumpre.
- Requisito ③
  - a regra  $\{U\}$  cumpre;
  - nenhuma das restantes cumpre.
- Como o símbolo inicial ( $S$ ) não cumpre as três condições, não há a garantia de que esta gramática faça sentido.

# Transformações de gramáticas

## Análise Sintáctica

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Crítérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções- $\epsilon$

Eliminação de  
recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica

# Transformações de gramáticas

- Em geral, existem inúmeras gramáticas capazes de reconhecer a mesma linguagem.
- Duas gramáticas dizem-se *equivalentes* se reconhecerem exactamente a mesma linguagem.
- Assim sendo, é possível modificar gramáticas sem alterar a linguagem reconhecida.
- Estas transformações podem ser justificadas pelas seguintes razões:
  - Simplificação da gramática;
  - Eliminar ambiguidades da gramática;
  - Reduzir a redundância da gramática;
  - Facilitar a interpretação ou a geração de código;
  - Permitir ou facilitar a sua implementação automática.
- Já vimos anteriormente algoritmos para transformar gramáticas por forma a eliminar símbolos inatingíveis.
- Vamos agora analisar outras transformações.

## Análise Sintáctica

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Crítérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções-ε

Eliminação de

recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica

# Eliminação de produções- $\epsilon$

## Análise Sintáctica

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Crítérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções- $\epsilon$

Eliminação de

recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica



# Eliminação de produções- $\varepsilon$

- Uma produção- $\varepsilon$  é uma produção do tipo  $Z \rightarrow \varepsilon$ , para um qualquer símbolo não terminal  $Z$ .
- Se tivermos uma linguagem independente de contexto que não contém a palavra vazia (i.e. que não a aceita), então é sempre possível transformar a sua gramática numa gramática equivalente sem este tipo de produções.
- Considere a gramática:

$$\begin{aligned} S &\rightarrow 0S \mid 1U \\ U &\rightarrow \varepsilon \mid 0U \mid 1S \end{aligned}$$

que descreve a linguagem formada pelas palavras definidas sobre o alfabeto  $\{0, 1\}$ , com um número ímpar de 1s.

- A existência da produção- $\varepsilon$  faz com que as produções  $S \rightarrow 1U$  e  $U \rightarrow 0U$  possam produzir as derivações  $S \Rightarrow 1$  e  $U \Rightarrow 0$ .

## Análise Sintáctica

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Crítérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções- $\varepsilon$

Eliminação de

recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica

# Eliminação de produções- $\varepsilon$ (2)

- Assim, podemos eliminar esta produção acrescentando as produções  $S \rightarrow 1$  e  $U \rightarrow 0$ :

$$S \rightarrow 0S \mid 1U \mid 1$$

$$U \rightarrow 0U \mid 1S \mid 0$$

- Em geral, o papel da produção  $Z \rightarrow \varepsilon$  sobre uma produção  $U \rightarrow \alpha Z \beta$  pode ser substituído pela inclusão, como alternativa, da produção  $U \rightarrow \alpha \beta$

# Eliminação de recursividade à esquerda

## Análise Sintáctica

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Crítérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções- $\epsilon$

Eliminação de  
recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica

# Eliminação de recursividade à esquerda

- Diz-se que uma gramática é **recursiva à esquerda** se possuir um símbolo não terminal  $Z$  que admita uma derivação do tipo  $Z \xRightarrow{+} Z\beta$
- Isto é, se for possível, em um ou mais passos de derivação, transformar uma expressão  $Z$  numa expressão que tem  $Z$  no início.
- A gramática seguinte é recursiva à esquerda:

$$\begin{aligned} S &\rightarrow UV \\ U &\rightarrow \varepsilon \mid S+ \\ V &\rightarrow a \mid b \mid (S) \end{aligned}$$

- A derivação  $S \Rightarrow UV \Rightarrow S+V$  mostra que é possível transformar  $S$  numa expressão com  $S$  à esquerda.
- Logo esta gramática tem recursividade à esquerda associada ao símbolo não terminal  $S$ .

## Eliminação de recursividade à esquerda (2)

- Se a recursividade à esquerda se faz com apenas um passo na derivação, então diz-se que a recursividade é **imediate** (ou directa).
- Esta situação só ocorre se a gramática possuir uma ou mais produções do tipo  $Z \rightarrow Z\beta$ .
- Na gramática seguinte a recursividade é imediata:

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

- A eliminação da recursividade à esquerda imediata é simples.
- Para tal basta introduzir um novo símbolo não terminal, e converter a recursividade imediata à esquerda para recursividade à direita.

## Eliminação de recursividade à esquerda (3)

- Considere a seguinte gramática:

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

- Se observarmos as derivações possíveis a partir de  $A$ :

$$A \Rightarrow \beta \quad A \Rightarrow A\alpha \Rightarrow \beta\alpha \quad A \Rightarrow A\alpha \Rightarrow A\alpha\alpha \dots$$

- Constatamos que para  $n$  passos ( $n > 0$ ), temos:  
 $A \Rightarrow \beta\alpha^{n-1}$
- Estas palavras podem ser obtidas com a seguinte gramática equivalente:

$$A \rightarrow \beta X$$

$$X \rightarrow \varepsilon \mid \alpha X$$

- Constata-se assim que se converteu a recursividade à esquerda para uma recursividade à direita.

## Eliminação de recursividade à esquerda (3)

- Este algoritmo é facilmente generalizável a situações em que há mais do que uma produção com recursividade imediata à esquerda.

$$A \rightarrow \beta_1 \mid \beta_2 \mid \cdots \mid \beta_m \mid A\alpha_1 \mid A\alpha_2 \mid \cdots \mid A\alpha_n$$

- Então teremos:

$$A \rightarrow \beta_1 X \mid \beta_2 X \mid \cdots \mid \beta_m X$$

$$X \rightarrow \varepsilon \mid \alpha_1 X \mid \alpha_2 X \mid \cdots \mid \alpha_n X$$

- O algoritmo pode ser generalizado para eliminar recursividade à esquerda que não seja imediata.

# Eliminação de recursividade à esquerda: exemplo

- Considere a seguinte gramática:

$$E \rightarrow E + T \mid T$$

$$T \rightarrow T * F \mid F$$

$$F \rightarrow (E) \mid \text{id}$$

- Esta gramática tem recursividade à esquerda, aplicando o algoritmo para a transformar numa gramática com recursividade à direita:

$$E \rightarrow T E'$$

$$E' \rightarrow + T E' \mid \varepsilon$$

$$T \rightarrow F T'$$

$$T' \rightarrow * F T' \mid \varepsilon$$

$$F \rightarrow (E) \mid \text{id}$$



# Factorização à esquerda

## Análise Sintáctica

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Crítérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções  $\epsilon$

Eliminação de  
recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica

## Factorização à esquerda

- A factorização à esquerda é uma transformação da gramática que é útil para produzir gramáticas adaptadas a análise sintáctica descendente preditiva.
- Consiste numa operação similar à transformação aritmética de “pôr em evidência” um factor comum.
- No caso, sempre que duas ou mais produções têm o mesmo prefixo (sequência de símbolos terminais e/ou não terminais), então podemos transformar essa gramática por forma a eliminar essa redundância.
- Como exemplo, vamos considerar a seguinte gramática:

$$\begin{aligned} \text{stat} &\rightarrow \text{if expr then stat else stat} \mid \\ &\quad \text{if expr then stat} \end{aligned}$$

- Se observarmos o *token if* não podemos desde logo decidir por qual das duas produções alternativas devemos seguir.
- O problema resolve-se com a seguinte transformação:

$$\begin{aligned} \text{stat} &\rightarrow \text{if expr then stat } U \mid \\ U &\rightarrow \epsilon \mid \text{else stat} \end{aligned}$$

### Análise Sintáctica

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Crítérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções  $\epsilon$

Eliminação de  
recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica

# Factorização à esquerda: algoritmo

- O algoritmo geral para realizar esta transformação é o seguinte:
- Para cada não terminal  $A$ , descobrir o maior prefixo  $\alpha$  comum a duas ou mais alternativas.
- Se  $\alpha \neq \varepsilon$ , então a produção:

$$A \rightarrow \alpha \beta_1 \mid \alpha \beta_2 \mid \cdots \mid \alpha \beta_n \mid \gamma$$

- pode ser transformada nas seguintes produções:

$$A \rightarrow \alpha X \mid \gamma$$

$$X \rightarrow \beta_1 \mid \beta_2 \mid \cdots \mid \beta_n$$

- Esta transformação deve ser repetida até que não se encontrem prefixos comuns.

# Eliminação de ambiguidades

## Análise Sintáctica

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Crítérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções  $\epsilon$

Eliminação de  
recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

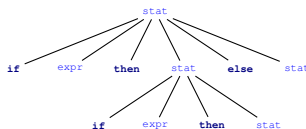
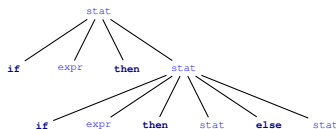
Análise sintáctica

# Eliminação de ambiguidades

- A gramática seguinte é ambígua.

stat  $\rightarrow$  if expr then stat  
| if expr then stat else stat  
| ... (outras instruções)

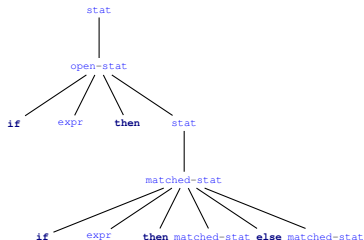
- Na presença de uma entrada tipo **if if else** vão existir duas árvores sintácticas distintas consoante se considera o **else** como sendo do primeiro ou do segundo **if**.



## Eliminação de ambiguidades (2)

- Podemos reescrever esta gramática seguindo a ideia de que uma instrução (stat) que apareça entre um **then** e um **else** tem de ser terminado:

stat  $\rightarrow$  matched-stat | open-stat  
matched-stat  $\rightarrow$  **if** expr **then** matched-stat **else** matched-stat  
|  $\dots$  (*outras instruções*)  
open-stat  $\rightarrow$  **if** expr **then** stat  
| **if** expr **then** matched-stat **else** open-stat



# Análise sintáctica descendente

## Análise Sintáctica

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Crítérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções  $\epsilon$

Eliminação de  
recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica

# Análise sintáctica descendente

- A análise sintáctica descendente pode ser vista como sendo o problema de construir a árvore sintáctica para a sequência de *tokens* de entrada, partindo da raiz (símbolo inicial), e criando os nós da árvore em pré-ordem.
- De forma análoga, podemos ver este tipo de análise como sendo a descoberta da derivação mais à esquerda para a sequência de entrada.
- Recuperando a seguinte gramática:

$$E \rightarrow T E'$$

$$E' \rightarrow + T E' \mid \varepsilon$$

$$T \rightarrow F T'$$

$$T' \rightarrow * F T' \mid \varepsilon$$

$$F \rightarrow ( E ) \mid \text{id}$$

- Podemos construir a árvore sintáctica para a entrada:  
**id + id \* id**

$$\begin{aligned} E &\Rightarrow T E' \Rightarrow F T' E' \Rightarrow \text{id} T' E' \Rightarrow \text{id} E' \Rightarrow \text{id} + T E' \Rightarrow \text{id} + F T' E' \Rightarrow \text{id} + \text{id} T' E' \\ &\Rightarrow \text{id} + \text{id} * F T' E' \Rightarrow \text{id} + \text{id} * \text{id} T' E' \Rightarrow \text{id} + \text{id} * \text{id} E' \Rightarrow \text{id} + \text{id} * \text{id} \end{aligned}$$

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Crítérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções- $\varepsilon$

Eliminação de

recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

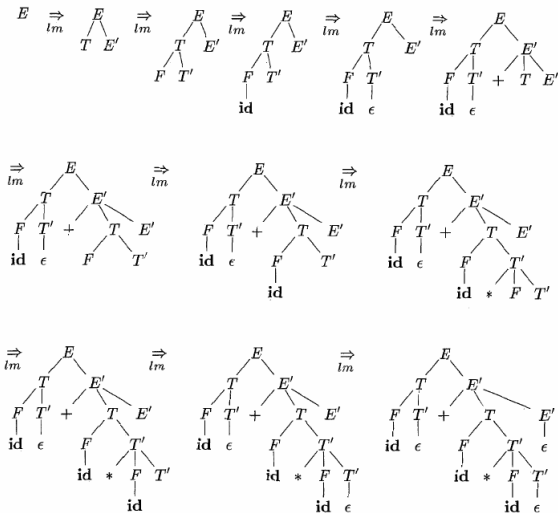
Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica



# Análise sintáctica descendente (2)



## Análise Sintáctica

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Crítérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções- $\epsilon$

Eliminação de  
recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica

## Análise sintáctica descendente (3)

- Existem várias aproximações à análise sintáctica descendente.
- Uma delas – denominada por **análise sintáctica descendente recursiva** – é uma aproximação genérica, mas pode requerer um algoritmo de *backtracking* (tentativa/erro) para descobrir a produção correcta a ser aplicada a cada novo *token* na entrada.
- Outra aproximação mais prática – denominada por **análise sintáctica descendente preditiva** – é um caso especial deste tipo de análise que não requer *backtracking*.
- Vamos estudar três conjuntos de análise – *first*, *follow* e *lookahead* – que simplificam o desenvolvimento destes analisadores.
- A análise sintáctica descendente preditiva escolhe a produção a ser aplicada vendo um número fixo de *tokens* na entrada que ainda não foram consumidos.

### Análise Sintáctica

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas  
Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Crítérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções-ε

Eliminação de  
recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica

## Análise sintáctica descendente (4)

- Se o número de *tokens* requerido for apenas um, este tipo de analisadores chamam-se  $LL(1)$ .
- O primeiro  $L$  indica que a entrada é analisada da esquerda para a direita; O segundo  $L$  significa que se escolhe sempre a derivação à esquerda.
- Para  $k$  *tokens* os analisadores chamam-se  $LL(k)$ .
- A construção deste tipo de analisadores pode ser feita **recursivamente por funções** – onde a cada regra não terminal se associa uma função –, ou de uma forma **iterativa através de uma tabela** e explicitando e com uma estrutura de dados tipo pilha.

### Análise Sintáctica

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Crítérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções- $\epsilon$

Eliminação de  
recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica

## Os conjuntos de análise *first*, *follow* e *lookahead*

- Para ajudar na construção de analisadores sintácticos descendentes preditivos, é conveniente a definição das funções – *first* e *follow* – associadas a símbolos da gramática.
- Estas funções calculam e devolvem conjuntos de símbolos terminais.
- Durante o reconhecimento descendente estas funções facilitam (ou mesmo, determinam) a escolha das produções a aplicar tendo em conta os próximos *token* de entrada.
- Para lidar com erros sintácticos, a função *follow* pode ser utilizada como mecanismo de re-sincronização do analisador sintáctico.
- Seja  $\alpha \in (T \cup N)^*$ , a função *first*( $\alpha$ ) devolve o conjunto de símbolos terminais ( $\in T$ ), que podem começar sequências de *tokens* derivadas de  $\alpha$ .
- Se  $\alpha \xRightarrow{*} \varepsilon$ , então  $\varepsilon$  também fará parte desse conjunto.
- Assim um símbolo terminal **x** pertence a este conjunto se e só se  $\alpha \xRightarrow{*} \mathbf{x} \beta$

## Os conjuntos de análise *first*, *follow* e *lookahead* (2)

- A função *follow*(*A*), para um símbolo não terminal *A*, devolve o conjunto de símbolos terminais ( $\in T$ ), que podem aparecer imediatamente a seguir a *A* (i.e. à sua direita).
- Assim um símbolo terminal *x* pertence a este conjunto se e só se  $S \xRightarrow{*} \alpha A x \beta$ , para quaisquer sequências  $\alpha$  e  $\beta$ .
- Chama-se a atenção de que o *token* especial “fim de ficheiro” (EOF ou \$) pode aparecer no conjunto *follow*(*A*) se este não terminal for o último da gramática.
- Para calcular a função *first*(*X*), para qualquer símbolo *X* da gramática podemos utilizar o seguinte algoritmo:
  - 1 Se *X* é um símbolo terminal, então:  $first(X) = \{X\}$ .
  - 2 Se *X* é um símbolo não terminal e  $X \rightarrow Y_1 Y_2 \dots Y_k$  é uma produção, para algum  $k \geq 1$ , então adicionar *x* a *first*(*X*) se, para algum valor de *i*, *x* está em *first*(*Y<sub>i</sub>*) e se  $\epsilon$  está em todos os  $first(Y_1), \dots, first(Y_{i-1})$ . Isto é,  $Y_1 \dots Y_{i-1} \xRightarrow{*} \epsilon$ . Se  $\epsilon$  está em *first*(*Y<sub>j</sub>*), para  $j = 1, 2, \dots, k$ , então adicionar  $\epsilon$  a *first*(*X*).
  - 3 Se  $X \rightarrow \epsilon$  é uma produção, então adicionar  $\epsilon$  a *first*(*X*).

# Os conjuntos de análise *first*, *follow* e *lookahead* (3)

- Podemos agora calcular a função *first* para qualquer sequência de símbolos  $X_1 X_2 \cdots X_n$  de seguinte forma:
  - Adicionar todos os símbolos, excepto  $\varepsilon$ , de  $first(X_1)$  a  $first(X_1 X_2 \cdots X_n)$ .
  - Se  $\varepsilon$  pertencer a  $first(X_1)$ , adicionar todos os símbolos, excepto  $\varepsilon$ , de  $first(X_2)$  a  $first(X_1 X_2 \cdots X_n)$ .
  - E assim sucessivamente.
  - Finalmente, adicionar  $\varepsilon$  a  $first(X_1 X_2 \cdots X_n)$  se  $\varepsilon$  pertencer a todos os  $first(X_i)$ ,  $i = 1, \dots, n$
- Para calcular a função  $follow(X)$ , para qualquer símbolo não terminal  $X$  da gramática aplicam-se o seguinte algoritmo:
  - Colocar “fim de ficheiro” em  $follow(S)$  (sendo  $S$  o símbolo inicial).
  - Se existir uma produção  $A \rightarrow \alpha X \beta$ , então adicionar  $first(\beta) \setminus \{\varepsilon\}$  a  $follow(X)$ .
  - Se existir uma produção  $A \rightarrow \alpha X$  ou uma produção  $A \rightarrow \alpha X \beta$  onde  $\varepsilon \in first(\beta)$ , então adicionar  $follow(A)$  a  $follow(X)$ .

# Os conjuntos de análise *first*, *follow* e *lookahead*: exemplo

- Recuperando novamente a gramática:

$$E \rightarrow T E'$$

$$E' \rightarrow + T E' \mid \epsilon$$

$$T \rightarrow F T'$$

$$T' \rightarrow * F T' \mid \epsilon$$

$$F \rightarrow ( E ) \mid \text{id}$$

temos:

$$\text{first}(E) = \text{first}(T) = \text{first}(F) = \text{first}(( E )) \cup \text{first}(\text{id}) = \{ (, \text{id} \}$$

$$\text{first}(E') = \text{first}(+ T E') \cup \text{first}(\epsilon) = \{ +, \epsilon \}$$

$$\text{first}(T') = \text{first}( * F T' ) \cup \text{first}(\epsilon) = \{ *, \epsilon \}$$

$$\text{follow}(E) = \{ \text{EOF} \} \cup \{ \} = \{ \}, \text{EOF} \}$$

$$\text{follow}(E') = \text{follow}(E) \cup \text{follow}(E') = \text{follow}(E) = \{ \}, \text{EOF} \}$$

$$\text{follow}(T) = (\text{first}(E') \setminus \{ \epsilon \}) \cup \text{follow}(E') = \{ +, \}, \text{EOF} \}$$

$$\text{follow}(T') = \text{follow}(T) \cup \text{follow}(T') = \text{follow}(T) = \{ +, \}, \text{EOF} \}$$

$$\text{follow}(F) = (\text{first}(T') \setminus \{ \epsilon \}) \cup \text{follow}(T') = \{ +, *, \}, \text{EOF} \}$$

# O conjunto de análise *lookahead*

- O conjunto *lookahead* (ou de antevisão) aplica-se às produções de uma gramática, e envolve as funções *first* e *follow*.
- É definido por:

$$\text{lookahead}(A \rightarrow \alpha) = \begin{cases} \text{first}(\alpha) & \text{se } \alpha \text{ não deriva } \varepsilon \\ (\text{first}(\alpha) \setminus \{\varepsilon\}) \cup \text{follow}(A) & \text{se } \alpha \xRightarrow{*} \varepsilon \end{cases}$$

- Ou seja, é o conjunto de símbolos terminais (sem a palavra vazia, mas eventualmente com o EOF) que iniciam uma produção.



# Análise sintáctica descendente recursiva

## Análise Sintáctica

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Crítérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções  $\epsilon$

Eliminação de  
recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica

# Análise sintáctica descendente recursiva

- Um analisador deste tipo consiste num conjunto de funções, uma para cada não terminal.
- Para uma produção do tipo  $A \rightarrow X_1 X_2 \dots X_n$  temos a seguinte implementação para a função que lhe está associada:

```
void A() {  
    //  $A \rightarrow X_1 X_2 \dots X_n$   
    for (i = 1 to n) {  
        if ( $X_i$  é um símbolo não terminal)  
            call function  $X_i()$ ;  
        else if (símbolo de entrada reconhecido por  $X_i$ )  
            avançar a entrada para o próximo símbolo  
        else  
            erro de reconhecimento  
    }  
}
```

- Note que com este algoritmo uma produção do tipo  $A \rightarrow A \dots$  gera recursão infinita, razão pela qual não pode haver recursividade à esquerda nas regras neste tipo de analisadores.

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Crítérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções- $\epsilon$

Eliminação de

recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica

# Análise sintáctica descendente recursiva (2)

- Um analisador descendente recursivo genérico pode requerer *backtracking*, isto é, pode ser necessário andar para trás no consumo de *tokens* de entrada até se descobrir uma derivação correcta.
- No entanto, para a larga maioria das construções de linguagens de programação os analisadores descendentes não requerem *backtracking*.

## Análise Sintáctica

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Crítérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções  $\epsilon$

Eliminação de  
recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica

# Análise sintáctica descendente preditiva

## Análise Sintáctica

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Crítérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções- $\epsilon$

Eliminação de  
recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica

# Análise sintáctica descendente preditiva

- Podemos construir analisadores descendentes que não requerem *backtracking*.
- São designados por **analisadores descendentes preditivos**.
- Este tipo de analisadores designam-se por  $LL(k)$ , em que  $k$  é o número de *tokens* à frente que o algoritmo requer ( $k \geq 1$ ).
- Em particular, a classe dos analisadores  $LL(1)$  é suficientemente expressiva para cobrir a maioria das construções de linguagens de programação.
- No entanto, é necessário algum cuidado na definição das regras, já que os analisadores  $LL(1)$  não aceitam gramáticas ambíguas ou com recursividade à esquerda.

## Análise Sintáctica

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Crítérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções- $\epsilon$

Eliminação de  
recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica

# Análise sintáctica descendente preditiva (2)

- Uma gramática é  $LL(1)$  se e só se, para produções do tipo  $A \rightarrow \alpha \mid \beta$ , as condições seguintes são verdadeiras:
  - 1 Para qualquer símbolo terminal **a**, no máximo, apenas uma das sequências  $\alpha$  ou  $\beta$  podem derivar palavras que comecem por **a**.
  - 2 No máximo, apenas uma das sequências  $\alpha$  ou  $\beta$  podem derivar  $\varepsilon$ .
  - 3 Se  $\beta \xRightarrow{*} \varepsilon$ , então  $\alpha$  não deriva nenhuma sequência começando com um terminal existente em  $follow(A)$ . Da mesma forma, se  $\alpha \xRightarrow{*} \varepsilon$ , então  $\beta$  não deriva nenhuma sequência começando com um terminal existente em  $follow(A)$ .

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Críticos de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções- $\varepsilon$

Eliminação de  
recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica

## Análise sintáctica descendente preditiva (3)

- Considerando a gramática  $G = (T, N, S, P)$ , e todos os não-terminais  $A \in N$  com produções  $P : A \rightarrow \alpha \mid \beta$ , as primeiras duas condições são equivalentes à condição:

$$\text{first}(\alpha) \cap \text{first}(\beta) = \emptyset$$

- A terceira condição, é equivalente a:

$$\varepsilon \in \text{first}(\beta) \implies \text{first}(\alpha) \cap \text{follow}(A) = \emptyset$$

(e *vice-versa* trocando  $\beta$  por  $\alpha$ )

- Em síntese, uma gramática será  $LL(1)$  se um *token* na entrada for sempre suficiente para decidir sem ambiguidade o que fazer.

# Análise sintáctica descendente preditiva: tabela de *parsing*

- Podemos construir uma tabela de *parsing*  $M$  para analisadores descendentes preditivos  $LL(1)$  que indique as produções a escolher como função do símbolo não-terminal onde estamos e do próximo *token* na entrada.
- Algoritmo: Para cada produção  $A \rightarrow \alpha$  da gramática fazer o seguinte:
  - Para cada símbolo terminal  $a$  existente em  $first(A)$ , adicionar  $A \rightarrow \alpha$  a  $M[A, a]$
  - Se  $\varepsilon \in first(A)$ , então para cada terminal  $b$  existente em  $follow(A)$ , adicionar  $A \rightarrow \alpha$  a  $M[A, b]$ . Se  $\varepsilon \in first(A) \wedge EOF \in follow(A)$ , então adicionar  $A \rightarrow \alpha$  a  $M[A, EOF]$ .

## Análise Sintáctica

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Crítérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções- $\varepsilon$

Eliminação de  
recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica



# Análise sintáctica descendente preditiva: exemplo

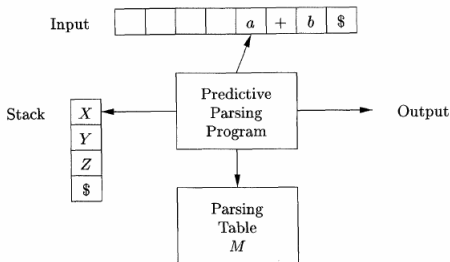
- A gramática apresentada a seguir é  $LL(1)$ :

$E \rightarrow T E'$	$first(E) = first(T) = first(F) = \{ (, id \}$
$E' \rightarrow + T E'$	$first(E') = \{ +, \epsilon \}$
$E' \rightarrow \epsilon$	$first(T') = \{ *, \epsilon \}$
$T \rightarrow F T'$	$follow(E) = follow(E') = \{ ), EOF \}$
$T' \rightarrow * F T'$	$follow(T) = follow(T') = \{ +, ), EOF \}$
$T' \rightarrow \epsilon$	$follow(F) = \{ +, *, ), EOF \}$
$F \rightarrow ( E )$	
$F \rightarrow id$	

- A tabela de *parsing* associa produções à relação de símbolos não-terminais com os *tokens* de entrada:

NON - TERMINAL	INPUT SYMBOL					
	id	+	*	(	)	\$
$E$	$E \rightarrow T E'$			$E \rightarrow T E'$		
$E'$		$E' \rightarrow + T E'$			$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$
$T$	$T \rightarrow F T'$			$T \rightarrow F T'$		
$T'$		$T' \rightarrow \epsilon$	$T' \rightarrow * F T'$		$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$
$F$	$F \rightarrow id$			$F \rightarrow ( E )$		

# Análise sintáctica descendente preditiva não recursiva



- Podemos assim construir um analisador preditivo sem funções recursivas, implementando-o com uma tabela e uma pilha.

## Análise Sintáctica

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Critérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções- $\epsilon$

Eliminação de  
recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica

# Análise sintáctica descendente preditiva não recursiva (2)

- Para a gramática anterior e a entrada  $id_1 + id_2 * id_3$ :

<i>reconhecido</i>	<i>pilha</i>	<i>entrada</i>	<i>acção</i>
	$E\$$	$id_1 + id_2 * id_3\$$	$E \rightarrow TE'$
	$TE'\$$	$id_1 + id_2 * id_3\$$	$T \rightarrow FT'$
	$FT' E'\$$	$id_1 + id_2 * id_3\$$	$F \rightarrow id$
	$id T' E'\$$	$id_1 + id_2 * id_3\$$	<i>reconhece id</i>
$id_1$	$T' E'\$$	$+ id_2 * id_3\$$	$T' \rightarrow \epsilon$
$id_1$	$E'\$$	$+ id_2 * id_3\$$	$E' \rightarrow +TE'$
$id_1$	$+TE'\$$	$+ id_2 * id_3\$$	<i>reconhece +</i>
$id_1 +$	$TE'\$$	$id_2 * id_3\$$	$T \rightarrow FT'$
$id_1 +$	$FT' E'\$$	$id_2 * id_3\$$	$F \rightarrow id$
$id_1 +$	$id T' E'\$$	$id_2 * id_3\$$	<i>reconhece id</i>
$id_1 + id_2$	$T' E'\$$	$* id_3\$$	$T' \rightarrow *FT'$
$id_1 + id_2$	$*FT' E'\$$	$* id_3\$$	<i>reconhece *</i>
$id_1 + id_2 *$	$FT' E'\$$	$id_3\$$	$F \rightarrow id$
$id_1 + id_2 *$	$id T' E'\$$	$id_3\$$	<i>reconhece id</i>
$id_1 + id_2 * id_3$	$T' E'\$$	$\$$	$T' \rightarrow \epsilon$
$id_1 + id_2 * id_3$	$E'\$$	$\$$	$E' \rightarrow \epsilon$
$id_1 + id_2 * id_3$	$\$$	$\$$	<i>aceita</i>

# Análise sintáctica descendente em ANTLR4

## Análise Sintáctica

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações  
Árvore sintáctica  
Gramáticas ambíguas  
Projecto de gramáticas  
simples  
Operações sobre GIC  
Reunião  
Concatenação  
Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis  
Critérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções- $\epsilon$   
Eliminação de  
recursividade à esquerda  
Factorização à esquerda  
Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*  
Análise sintáctica  
descendente recursiva  
Análise sintáctica  
descendente preditiva  
Análise sintáctica

# Análise sintáctica descendente em ANTLR4

- Como foi já referido por várias vezes, o ANTLR4 é um analisador descendente (em oposição ao `yacc/bison` que é um analisador ascendente).
- No entanto, a tecnologia de compilação implementada no ANTLR4 tem algumas diferenças relativamente aos analisadores descendentes preditivos apresentados.
- Uma delas, é o facto de ser caracterizado como sendo um analisador preditivo da  $LL(*)$  adaptativo.
- Não requer *backtracking*, mas pode observar os *tokens* à frente na entrada até ao fim desta, se necessário for.
- Por outro lado, faz uso de uma análise dinâmica da gramática (em tempo de execução), antes do reconhecimento começar.
- Na prática, isto significa que o ANTLR4 aceita gramáticas bem mais genéricas do que as normalmente associadas aos analisadores  $LL(k)$ .

## Análise Sintáctica

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Crítérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções- $\epsilon$

Eliminação de  
recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica

## Análise sintáctica descendente em ANTLR4 (2)

- Outra característica de grande interesse prático do ANTLR4, é o facto de aceitar gramáticas com recursividade directa à esquerda.
- Torna-se assim possível definir gramáticas do tipo:

```
grammar Expr;  
  
main: stat* EOF;  
  
stat: expr;  
  
expr: expr '*' expr # Mult  
    | expr '+' expr # Add  
    | INT           # Int  
    ;  
  
INT: [0-9]+;  
WS: [ \t\r\n]+ -> skip;
```

- O que o ANTLR4 faz é automaticamente reescrever as regras com recursividade directa à esquerda em regras equivalentes sem essa característica (utilizando um algoritmo que pode ser similar ao apresentado atrás).
- Uma limitação desta característica é facto da recursividade à esquerda ter de ser directa.

# Análise sintáctica descendente em ANTLR4 (3)

- O ANTLR4 tem outras características que o diferenciam, e tornam a sua classificação como analisador descendente um pouco mais complicada.
- É o caso dos predicados semânticos que permitem que a gramática a utilizar possa mudar dinamicamente consoante quaisquer condições expressas na linguagem destino (Java por omissão).
- Esta característica faz com que o ANTLR4 possa, durante a análise sintáctica, implementar construções típicas de gramáticas dependentes de contexto.

## Análise Sintáctica

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Crítérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções-ε

Eliminação de  
recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica

# Análise sintáctica ascendente

## Análise Sintáctica

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Crítérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções  $\epsilon$

Eliminação de  
recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica



# Análise sintáctica ascendente

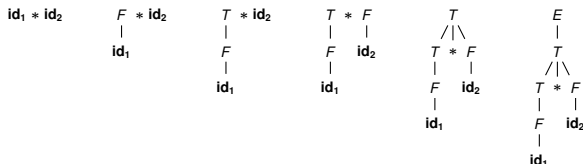
- A análise sintáctica ascendente pode ser vista como sendo o problema de construir a árvore sintáctica partindo das folhas (*tokens*), e criando os nós da árvore de baixo para cima até à raiz (símbolo inicial).
- Considere a seguinte gramática:

$$E \rightarrow E + T \mid T$$

$$T \rightarrow T * F \mid F$$

$$F \rightarrow (E) \mid \text{id}$$

- Vamos proceder à análise sintáctica processando a entrada **id<sub>1</sub> \* id<sub>2</sub>** da esquerda para direita e aplicando de imediato a produção que a vai aproximando do topo.



- A análise ascendente pode também ser vista como o processo de **reduzir** a sequência de *tokens* de entrada ao símbolo inicial.
- Em cada passo de redução, a subsequência de símbolos (terminais e/ou não terminais) correspondente ao corpo de uma produção, é substituída pela cabeça da produção.
- A decisão chave neste tipo de análise sintáctica, reside em quando aplicar uma redução e a escolha da produção a aplicar.
- A sequência de reduções aplicadas ao exemplo anterior, gerou a seguinte sequência de símbolos:

$$\mathbf{id_1 * id_2, F * id_2, T * id_2, T * F, T, E}$$

- Cada elemento desta sequência corresponde à raiz das árvores atrás apresentadas.

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Crítérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções  $\epsilon$

Eliminação de  
recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Crítérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções- $\epsilon$

Eliminação de  
recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica

- Por definição, uma redução é a operação inversa da (respectiva) derivação.
- Assim se reescrevermos a sequência seguindo as derivações (do símbolo inicial até à entrada):

$$E \Rightarrow T \Rightarrow T * F \Rightarrow T * \mathbf{id}_2 \Rightarrow F * \mathbf{id}_2 \Rightarrow \mathbf{id}_1 * \mathbf{id}_2$$

- Constatamos assim que esta análise corresponde à derivação à direita da sequência de entrada.

# Análise sintáctica por deslocamento e redução

## Análise Sintáctica

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Crítérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções  $\epsilon$

Eliminação de  
recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica

# Análise sintáctica por deslocamento e redução

- A análise sintáctica por deslocamento e redução é uma forma de análise ascendente na qual uma pilha vai armazenando símbolos da gramática (por reduzir), e um *buffer* de entrada contém os *tokens* ainda por processar.
- O topo dessa pilha conterá sempre a próxima sequência de símbolos a reduzir.
- Para facilitar a visualização deste processo iremos utilizar o símbolo \$ para indicar o *token* especial *fim de ficheiro*.
- Podemos assim visualizar o processo de análise ascendente com uma tabela onde se mostra o conteúdo da pilha, o *buffer* de entrada e a acção a tomar.
- Podemos efectuar as seguintes acções:
  - 1 **redução**: o topo da pilha é substituído pela cabeça da produção (respectiva);
  - 2 **deslocamento**: o próximo *token* da entrada é colocado no topo da pilha.
  - 3 **aceitar**: entrada reconhecida com sucesso
  - 4 **rejeição**: erro sintáctico

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Crítérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções-ε

Eliminação de  
recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica

## Análise sintáctica por deslocamento e redução (2)

- Pegando novamente no exemplo dado, teremos a seguinte tabela:

<i> pilha</i>	<i> entrada</i>	<i> acção</i>
\$	<b>id<sub>1</sub> * id<sub>2</sub></b> \$	deslocamento
\$ <b>id<sub>1</sub></b>	<b>*</b> <b>id<sub>2</sub></b> \$	redução por $F \rightarrow \text{id}$
\$ $F$	<b>*</b> <b>id<sub>2</sub></b> \$	redução por $T \rightarrow F$
\$ $T$	<b>*</b> <b>id<sub>2</sub></b> \$	deslocamento
\$ $T *$	<b>id<sub>2</sub></b> \$	deslocamento
\$ $T * \text{id}_2$	\$	redução por $F \rightarrow \text{id}$
\$ $T * F$	\$	redução por $T \rightarrow T * F$
\$ $T$	\$	redução por $E \rightarrow T$
\$ $E$	\$	aceitar

# Análise sintáctica por deslocamento e redução (3)

- Uma vez que estamos na presença de derivações à direita (embora feitas “ao contrário”, partindo das folhas da árvore para a raiz), é garantido que, a existir, o corpo da produção aparecerá sempre no topo da pilha.
- Vamos implementar um analisador ascendente por deslocamento/redução para processar expressões aritméticas simples.
- Para simplificar, vamos considerar que um *token* é um carácter (os números são apenas um dígito).
- A gramática pretendida é:  
$$e \rightarrow (e) \mid e + e \mid e - e \mid exe \mid e/e \mid D$$

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Crítérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções-ε

Eliminação de  
recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica

# Conflitos

## Análise Sintáctica

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Crítérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções- $\epsilon$

Eliminação de  
recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica



- Existem gramáticas independentes de contexto para as quais a análise sintáctica por deslocamento e redução não funciona.
- Nessas gramáticas, mesmo com o conhecimento do conteúdo actual da pilha e da entrada, o analisador não consegue decidir se deve fazer uma redução ou um deslocamento (conflito do tipo *shift/reduce*), ou não consegue decidir qual das reduções possíveis deve fazer (conflito do tipo *reduce/reduce*).
- Como exemplo deste tipo de situações temos as gramáticas ambíguas.

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Crítérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções-ε

Eliminação de  
recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica

## Conflitos: *shift/reduce*

- Um exemplo de conflitos do tipo *shift/reduce* é a seguinte gramática da instrução condicional:

```
stat  →  if expr then stat
      |  if expr then stat else stat
      |  other
```

- Quando o analisador estiver no estado:

<i>pilha</i>	<i>entrada</i>
\$ ... <b>if</b> expr <b>then</b> stat	<b>else</b> ... \$

pode fazer quer um deslocamento, quer uma redução.

- Uma possibilidade simples para resolver este tipo de conflitos é privilegiar uma dessas acções (por exemplo, o deslocamento).

## Conflitos: *reduce/reduce*

- Vejam os agora a seguinte gramática:

```
stat  →  id ( parameterList )
      |  id := expr
parameterList → parameterList , parameter
              | parameter
parameter    →  id
expr        →  id
              |  id ( exprList )
exprList    →  exprList , expr
              |  expr
```

- Quando o analisador estiver no estado:

<i>pilha</i>	<i>entrada</i>
\$ ... <b>id</b> ( <b>id</b>	, <b>id</b> ... \$

temos outro tipo de conflito (*reduce/reduce*) já que o analisador não consegue decidir como reduzir o **id** que está no topo da pilha (parameter ou expr?).

# Construção de um analisador ascendente

## Análise Sintáctica

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Crítérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções  $\epsilon$

Eliminação de  
recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica

# Construção de um analisador ascendente

- O tipo de analisadores ascendentes mais em uso são os chamados  $LR(k)$ .
- O primeiro  $L$  indica que a entrada é analisada da esquerda para a direita, e o  $R$  significa que se escolhe sempre a derivação à direita.
- O  $k$  é o número de *tokens* de entrada de antevisão (*lookahead*).
- Os casos práticos de interesse tem valores de  $k$  iguais a 0 ou a 1.
- Os analisadores  $LR$  são baseados em tabelas (à semelhança dos analisadores  $LL$  não recursivos).

## Análise Sintáctica

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Crítérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções- $\epsilon$

Eliminação de  
recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica

## Construção de um analisador ascendente (2)

- Este tipo de analisadores sintácticos têm as seguintes características:
  - É possível construir analisadores para virtualmente todo o tipo de linguagens para as quais existem gramáticas independentes de contexto;
  - O método de análise *LR* é o método por deslocamento/redução sem *backtracking* mais genérico conhecido;
  - Permite a detecção de erros sintácticos assim que possível numa análise da entrada sequencial da esquerda para direita;
  - Em teoria, a classe de gramáticas passível de ser analisada supera a dos analisadores *LL* preditivos.
- Uma das suas limitações é ser mais complicado ter acesso ao contexto (*top-down*) envolvido no processo de reconhecimento de uma regra (dificultando a passagem de informação para baixo).
- Outro “problema” é a complexidade no seu desenvolvimento, pelo que requer uma ferramenta para esse fim (*yacc/bison*).

### Análise Sintáctica

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas  
Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Critérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções-ε

Eliminação de  
recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica

## Analizador ascendente LR simples (SLR)

- Como é que um analisador por deslocamento/redução decide por uma das acções?
- Por exemplo, na gramática apresentadas com expressões aritméticas, se a pilha contiver  $T$  e o próximo *token* for  $*$ , como é que o analisador sabe que o topo da pilha não é para reduzir ( $E \rightarrow T$ ), mas sim deslocar esse símbolo para a pilha?
- Um analisador *SLR* toma estas decisões registando o estado onde está (*item*) no reconhecimento (como um autómato).
- Por exemplo, a produção  $A \rightarrow X Y Z$  tem 4 itens possíveis (o ponto indica o estado de reconhecimento):

$$A \rightarrow \cdot X Y Z$$

$$A \rightarrow X \cdot Y Z$$

$$A \rightarrow X Y \cdot Z$$

$$A \rightarrow X Y Z \cdot$$

- A produção vazia  $A \rightarrow \varepsilon$  tem apenas um item:  $A \rightarrow \cdot$

## Analizador ascendente LR simples (2)

- Intuitivamente, um item mostra quando da produção é que já foi reconhecida e o que é esperado a seguir.
- Assim o item  $A \rightarrow \cdot X Y Z$  mostra que esperamos vir a reconhecer uma derivação de  $X Y Z$ .
- Já o item  $A \rightarrow X \cdot Y Z$  mostra que acabamos de “consumir” um sequência derivada de  $X$  e esperamos uma derivação de  $Y Z$ .
- O item  $A \rightarrow X Y Z \cdot$  indica que o corpo da produção  $A$  está completo pelo que podemos proceder à sua redução.

### Análise Sintáctica

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Critérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções  $\epsilon$

Eliminação de  
recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica



# Analizador ascendente LR(0) canónica

- Uma colecção de conjuntos de  $LR(0)$  – designada  $LR(0)$  canónica – fornece a base para se construir um autómato finito determinista que é utilizado nesta decisão.
- Este autómato é chamado autómato  $LR(0)$ .
- Cada estado deste autómato representa um conjunto de itens nessa colecção  $LR(0)$  canónica.
- Para construir a colecção de conjuntos de itens, vamos definir uma gramática aumentada e duas novas funções: CLOSURE (fecho) e GOTO
- Se uma gramática  $G$  tiver  $S$  como símbolo inicial, então  $G'$  será a sua gramática aumentada, em que passará a existir um novo símbolo inicial ( $S'$ ) definido pela produção  $S' \rightarrow S$
- Este novo símbolo serve para indicar a aceitação da entrada.

## Análise Sintáctica

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas  
Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Crítérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções- $\epsilon$

Eliminação de  
recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica

# Analizador ascendente LR(0) canónica: função CLOSURE

Análise Sintáctica

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas  
Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Crítérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções- $\epsilon$

Eliminação de  
recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica

- Se  $I$  é um conjunto de itens para uma gramática  $G$ , então  $CLOSURE(I)$  é o conjunto de itens construído da seguinte forma:
  - 1 Todos os elementos de  $I$  pertencem a  $CLOSURE(I)$ .
  - 2 Se  $A \rightarrow \alpha \cdot B \beta$  for um dos itens de  $CLOSURE(I)$  e existir a produção  $B \rightarrow \gamma$ , então adicionar o item  $B \rightarrow \cdot \gamma$  a  $CLOSURE(I)$  caso ainda lá não exista. Aplicar esta regra até não existirem mais itens a ser adicionados a  $CLOSURE(I)$ .

# Analizador ascendente LR(0) canónica: função CLOSURE

## (2)

- Considere a seguinte gramática aumentada (com  $E'$  como símbolo inicial):

$$E' \rightarrow E$$

$$E \rightarrow E + T \mid T$$

$$T \rightarrow T * F \mid F$$

$$F \rightarrow (E) \mid \text{id}$$

- Se  $I$  é o conjunto de itens apenas com o elemento  $\{[E' \rightarrow \cdot E]\}$ , então:

$$\text{CLOSURE}(I) = \{[E' \rightarrow \cdot E], [E \rightarrow \cdot E + T], [E \rightarrow \cdot T], [T \rightarrow \cdot T * F], [T \rightarrow \cdot F], [F \rightarrow \cdot (E)], [F \rightarrow \cdot \text{id}]\}$$

# Analizador ascendente LR(0) canónica: função GOTO

- Se  $I$  é um conjunto de itens para uma gramática  $G$  e  $X$  um símbolo de  $G$ , então  $\text{GOTO}(I, X)$  é o fecho (*closure*) do conjunto de todos os itens  $[A \rightarrow \alpha X \cdot \beta]$  desde que  $[A \rightarrow \alpha \cdot X \beta]$  pertença a  $I$ .
- Se  $I$  é o conjunto de itens  $\{[E' \rightarrow E \cdot], [E \rightarrow E \cdot + T]\}$ , então:

$$\text{GOTO}(I, +) = \{[E \rightarrow E + \cdot T], [T \rightarrow \cdot T * F], [T \rightarrow \cdot F], [F \rightarrow \cdot (E)], [F \rightarrow \cdot \text{id}]\}$$

## Análise Sintáctica

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Crítérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções- $\epsilon$

Eliminação de  
recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica

# Analizador ascendente LR(0) canónica: algoritmo

- O seguinte algoritmo permite a criação do autómato LR(0) para a gramática aumentada  $G'$ :
  - 1 O estado inicial será  $CLOSURE([S' \rightarrow \cdot S])$ .
  - 2 Sempre que um novo estado é adicionado, verificar as transições que esse estado pode ter por ocorrência de símbolos (terminais ou não terminais).
  - 3 Sempre que dessas transições resultar um estado inexistente, criar esse novo estado, e repetir o processo.
- Note que este procedimento tem algumas semelhanças formais com a transformação de AFND em AFD (análise lexical, autómatos finitos).

## Análise Sintáctica

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Crítérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções- $\epsilon$

Eliminação de  
recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first, follow e lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica

## Analizador ascendente LR(0) canónica: exemplo

- Recuperando a gramática aumentada de expressões aritméticas:

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

- O estado inicial será:

$$\begin{aligned}I_0 &= \text{CLOSURE}(\{[E' \rightarrow \cdot E]\}) \\&= \{[E' \rightarrow \cdot E], [E \rightarrow \cdot E + T], [E \rightarrow \cdot T], [T \rightarrow \cdot T * F], [T \rightarrow \cdot F], [F \rightarrow \cdot (E)], [F \rightarrow \cdot \text{id}]\}\end{aligned}$$

- Este estado pode evoluir pelos símbolos  $E, T, F, (, \text{id}$ , logo:

$$\begin{aligned}I_1 &= \text{CLOSURE}(\text{GOTO}(I_0, E)) = \{[E' \rightarrow E \cdot], [E \rightarrow E \cdot + T]\} \\I_2 &= \text{CLOSURE}(\text{GOTO}(I_0, T)) = \{[E \rightarrow T \cdot], [T \rightarrow T \cdot * F]\} \\I_3 &= \text{CLOSURE}(\text{GOTO}(I_0, F)) = \{[T \rightarrow F \cdot]\} \\I_4 &= \text{CLOSURE}(\text{GOTO}(I_0, ( )) = \{[F \rightarrow ( \cdot E)], [E \rightarrow \cdot E + T], [E \rightarrow \cdot T], \\&\quad [T \rightarrow \cdot T * F], [T \rightarrow \cdot F], [F \rightarrow \cdot (E)], [F \rightarrow \cdot \text{id}]\} \\I_5 &= \text{CLOSURE}(\text{GOTO}(I_0, \text{id})) = \{[F \rightarrow \text{id} \cdot]\}\end{aligned}$$

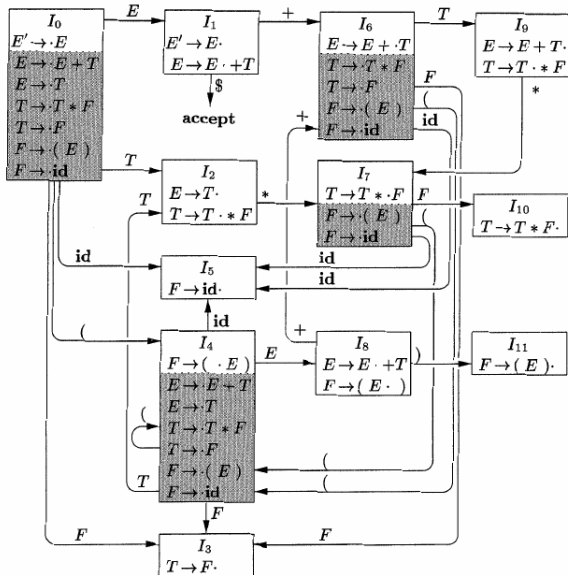
## Analizador ascendente LR(0) canónica: exemplo (2)

- As evoluções para os novos estados são:

$$\begin{aligned} \text{GOTO}(I_1, \$) &= \text{aceitar} \\ I_6 &= \text{CLOSURE}(\text{GOTO}(I_1, +)) \\ &= \{[E \rightarrow E + \cdot T], [T \rightarrow \cdot T * F], [T \rightarrow \cdot F], [F \rightarrow \cdot (E)], [F \rightarrow \cdot \text{id}]\} \\ I_7 &= \text{CLOSURE}(\text{GOTO}(I_2, *)) = \{[T \rightarrow T * \cdot F], [F \rightarrow \cdot (E)], [F \rightarrow \cdot \text{id}]\} \\ I_8 &= \text{CLOSURE}(\text{GOTO}(I_4, E)) = \{[E \rightarrow E \cdot + T], [F \rightarrow (E \cdot)]\} \\ \text{CLOSURE}(\text{GOTO}(I_4, T)) &= \{[E \rightarrow T \cdot], [T \rightarrow T \cdot * F]\} = I_2 \\ \text{CLOSURE}(\text{GOTO}(I_4, F)) &= \{[T \rightarrow F \cdot]\} = I_3 \\ \text{CLOSURE}(\text{GOTO}(I_4, ( )) &= I_4 \\ \text{CLOSURE}(\text{GOTO}(I_4, \text{id})) &= \{[F \rightarrow \text{id} \cdot]\} = I_5 \\ I_9 &= \text{CLOSURE}(\text{GOTO}(I_6, T)) = \{[E \rightarrow E + T \cdot], [T \rightarrow T * \cdot F]\} \\ \text{CLOSURE}(\text{GOTO}(I_6, F)) &= \{[T \rightarrow F \cdot]\} = I_3 \\ \text{CLOSURE}(\text{GOTO}(I_6, ( )) &= I_4 \\ \text{CLOSURE}(\text{GOTO}(I_6, \text{id})) &= \{[F \rightarrow \text{id} \cdot]\} = I_5 \\ I_{10} &= \text{CLOSURE}(\text{GOTO}(I_7, F)) = \{[T \rightarrow T * \cdot F]\} \\ \text{CLOSURE}(\text{GOTO}(I_7, ( )) &= I_4 \\ \text{CLOSURE}(\text{GOTO}(I_7, \text{id})) &= \{[F \rightarrow \text{id} \cdot]\} = I_5 \\ I_{11} &= \text{CLOSURE}(\text{GOTO}(I_8, ( )) = \{[F \rightarrow (E \cdot)]\} \\ \text{CLOSURE}(\text{GOTO}(I_9, * )) &= I_7 \end{aligned}$$

# Analizador ascendente LR(0) canónica: exemplo - diagrama de transições

Donde resulta o seguinte diagrama de transições do autómato:

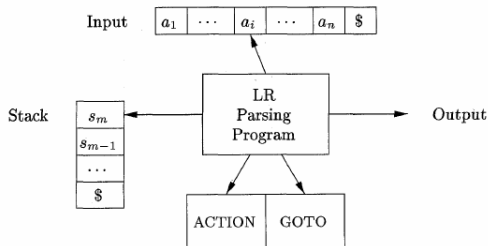




# Analizador ascendente LR(0) canónica: exemplo - diagrama de transições (2)

- O conjunto de itens de cada estado pode ser dividido em dois:
  - 1 Itens nucleares: o item inicial e todos os itens em cujos o ponto não está à esquerda;
  - 2 Itens não-nucleares: todos os itens restantes (i.e. todos os itens em que o ponto está à esquerda, excepto o item inicial).
- Apenas é necessário guardar os itens nucleares (os outros, representados a sombreado na figura, estão na transição dos estados).

# Analisador ascendente LR: algoritmo de reconhecimento



- A pilha contém os estados do autômato.
- A tabela de *parsing* tem duas partes: (1) uma função ACTION; (2) e uma função GOTO.
  - A função ACTION tem como argumentos um estado do autômato e um símbolo terminal (acrescida com o EOF).
  - O seu valor pode ter 4 formas: (1) deslocamento e transição para outro estado; (2) redução por uma produção; (3) aceitar a entrada; (4) rejeitar a entrada (erro).
  - A função GOTO indica as transições de estados no autômato em função de símbolos não terminais.

# Analizador ascendente LR(0) canónica: exemplo - tabela de parsing

estado	acção						GOTO		
	id	+	*	(	)	\$	E	T	F
0	S <sub>5</sub>				S <sub>4</sub>		1	2	3
1		S <sub>6</sub>				aceitar			
2		$r_{E \rightarrow T}$	S <sub>7</sub>		$r_{E \rightarrow T}$	$r_{E \rightarrow T}$			
3		$r_{T \rightarrow F}$	$r_{T \rightarrow F}$		$r_{T \rightarrow F}$	$r_{T \rightarrow F}$			
4	S <sub>5</sub>				S <sub>4</sub>		8	2	3
5		$r_{F \rightarrow \text{id}}$	$r_{F \rightarrow \text{id}}$		$r_{F \rightarrow \text{id}}$	$r_{F \rightarrow \text{id}}$			
6	S <sub>5</sub>				S <sub>4</sub>			9	3
7	S <sub>5</sub>				S <sub>4</sub>				10
8		S <sub>6</sub>			S <sub>11</sub>				
9		$r_{E \rightarrow E + T}$	S <sub>7</sub>		$r_{E \rightarrow E + T}$	$r_{E \rightarrow E + T}$			
10		$r_{T \rightarrow T * F}$	$r_{T \rightarrow T * F}$		$r_{T \rightarrow T * F}$	$r_{T \rightarrow T * F}$			
11		$r_{F \rightarrow (E)}$	$r_{F \rightarrow (E)}$		$r_{F \rightarrow (E)}$	$r_{F \rightarrow (E)}$			

- As acções  $s_i$  indicam deslocamento para o estado  $i$ , e  $r_p$  redução pela produção  $p$ .

# Analizador ascendente LR(0) canónica: exemplo

- Na aplicação do analisador, a pilha irá agora registar estados (em vez de símbolos).
- Sempre que exista uma transição com o *token* de entrada optar-se-á pelo seu deslocamento, colocando na pilha o estado para onde a transição é feita.
- Caso contrário, opta-se pela redução, fazendo o número de pops na pilha igual ao número de símbolos do corpo da produção reduzida, e colocando na pilha a cabeça dessa produção.
- Para ilustrar o funcionamento do analisador, vamos aplicá-lo à entrada **id \* id**

## Análise Sintáctica

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Críticos de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções-ε

Eliminação de  
recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica

# Analizador ascendente LR(0) canónica: exemplo - aplicação

	<i>pilha</i>	<i>símbolos</i>	<i>entrada</i>	<i>acções</i>
(1)	0	\$	<b>id<sub>1</sub></b> * <b>id<sub>2</sub></b> \$	deslocamento, <i>push</i> 5
(2)	0 5	\$ <b>id<sub>1</sub></b>	* <b>id<sub>2</sub></b> \$	redução por $F \rightarrow \text{id}, 1 \times \text{pop}$
(3)	0	\$ <i>F</i>	* <b>id<sub>2</sub></b> \$	<i>push(goto(0, F))</i>
(4)	0 3	\$ <i>F</i>	* <b>id<sub>2</sub></b> \$	redução por $T \rightarrow F, 1 \times \text{pop}$
(5)	0	\$ <i>T</i>	* <b>id<sub>2</sub></b> \$	<i>push(goto(0, T))</i>
(6)	0 2	\$ <i>T</i>	* <b>id<sub>2</sub></b> \$	deslocamento, <i>push</i> 7
(7)	0 2 7	\$ <i>T</i> *	<b>id<sub>2</sub></b> \$	deslocamento, <i>push</i> 5
(8)	0 2 7 5	\$ <i>T</i> * <b>id<sub>2</sub></b>	\$	redução por $F \rightarrow \text{id}, 1 \times \text{pop}$
(9)	0 2 7	\$ <i>T</i> * <i>F</i>	\$	<i>push(goto(7, F))</i>
(10)	0 2 7 10	\$ <i>T</i> * <i>F</i>	\$	redução por $T \rightarrow T * F, 3 \times \text{pop}$
(11)	0	\$ <i>T</i>	\$	<i>push(goto(0, T))</i>
(12)	0 2	\$ <i>T</i>	\$	redução por $E \rightarrow T, 1 \times \text{pop}$
(13)	0	\$ <i>E</i>	\$	<i>push(goto(0, E))</i>
(14)	0 1	\$ <i>E</i>	\$	aceitar

# Outros analisadores ascendentes LR

- Os analisadores ascendentes gerados por ferramentas tipo *yacc/bison* não são analisadores *LR(0)* simples.
- Ao contrário deste, esses analisadores fazem uso da antevisão do próximo token de entrada e são designados por *LALR(1)*.
- A utilização dessa informação permite uma melhor escolha das transições a serem feitas, e alargam enormemente o leque de gramáticas independentes de contexto que podem ser implementadas.
- A sua implementação directa é muito trabalhosa, pelo que não iremos detalhar a sua implementação.

## Análise Sintáctica

Análise sintáctica:  
Estrutura de um  
Compilador

Gramáticas  
Independentes de  
Contexto

Derivações

Árvore sintáctica

Gramáticas ambíguas

Projecto de gramáticas  
simples

Operações sobre GIC

Reunião

Concatenação

Fecho de Kleene

Definição de  
gramáticas

Símbolos inacessíveis

Crítérios de sanidade em  
gramáticas

Transformações de  
gramáticas

Eliminação de produções- $\epsilon$

Eliminação de  
recursividade à esquerda

Factorização à esquerda

Eliminação de  
ambiguidades

Análise sintáctica  
descendente

Os conjuntos de análise  
*first*, *follow* e *lookahead*

Análise sintáctica  
descendente recursiva

Análise sintáctica  
descendente preditiva

Análise sintáctica