Parsing preditivo usando uma tabela de parsing

$E \to TE'$
$E' \to +E \mid \varepsilon$
$T \to FT'$
$T' \to *T \mid \varepsilon$
$F \to (E) \mid id$

- A partir da gramática podemos construir uma tabela de parsing (veremos como mais adiante)

M:

	id	+	*	(	)	$
E	TE'			TE'		
E'		$+E$			ε	ε
T	FT'			FT'		
T'		ε	$*T$		ε	ε
F	id			(E)		

Símbolos não terminais

Símbolo terminais e $ que significa o fim to input

Significado das entradas da tabela

- Ao expandir o símbolo não terminal X, olha-se para o token que está no input, t, e expandimos X para aquilo que está indicado na entrada M[X, t]

- Exemplos:

símbolo não terminal	input token	acção
E	(	$E \Rightarrow TE'$
E	*	erro
E'	)	$E' \Rightarrow \varepsilon$

- Tendo a tabela de parsing, podemos implementar o parser preditivo.
  - podemos usar uma implementação recursiva (como na última aula)
  - ou uma implementação não recursiva baseada num stack.

- O parser toma decisões baseadas no símbolo que está no topo do stack (seja ele $\underline{X}$), e no token current (seja ele $\underline{a}$).

- O stack é inicializado com $S, onde S é o símbolo inicial da gramática

- 3 casos:

X é um token:

1) Se $X = a$ e $a = \$$, então o parser termina com ~~sucesso~~ sucesso.

2) Se $X = a$ e $a \neq \$$, o parser faz pop de X e avança o input para o próximo token.

3) Se X é um símbolo não terminal, o parser consulta a entrada $M[X, a]$

   - se entrada for $X \rightarrow Y_1 Y_2 \cdots Y_k$ então faz pop de X, seguido de push de $Y_k, Y_{k-1}, \cdots, Y_1$, com Y_1 no topo do stack.
   - se entrada for vazia, o parser emite uma mensagem de erro.

Exemplo

input: id + id * id

Stack	Input	Output
$E	id + id * id $	
$E'T	id + id * id $	E → TE'
$E'T'F	id + id * id $	T → FT'
$E'T'id	id + id * id $	F → id
$E'T'	+ id * id $	match (id)
$E'	+ id * id $	T' → ε
$E+	+ id * id $	E' → +E
$E	id * id $	match (+)
$E'T	id * id $	E → TE'
$E'T'F	id * id $	T → FT'
$E'T'id	id * id $	F → id
$E'T'	* id $	match (id)
$E'T*	* id $	T' → *T
$E'T	id $	match (*)
$E'T'F	id $	T → FT'
$E'T'id	id $	F → id
$E'T'	$	match (id)
$E'	$	T' → ε
$	$	E' → ε
		ACCEPT

Como construir a tabela de parsing?

- As entradas da tabela especificam para cada símbolo não terminal e para cada token, qual a regra da gramática que deve ser usada.

- Um token t pode prever uma regra por 2 motivos:

  1) O lado dto. da regra pode gerar uma string que comece por $\underline{t}$.

  2) O lado dto. da regra não gera nada (i.e, ε ou uma string de símbolos não terminais que recursivamente geram ε) e $\underline{t}$ pode começar o que vem a seguir.

- FIRST(A) ⇝ conjunto de tokens que podem estar ao início de A.

Esta definição é ligeiramente diferente do que está no livro do dragão. Estou a seguir a descrição que está no livro "Programming Language Pragmatics" de Michael Scott. O livro do dragão considera que ε pode fazer parte de FIRST.

- FOLLOW(A) ⇝ conjunto de tokens que pode aparecer imediatamente após A.

- O conjunto de tokens que prevê a regra $A \to \beta$ é FIRST(β), mais FOLLOW(A) se $\beta \overset{*}{\Rightarrow} \varepsilon$

Vamos definir $EPS(\beta) = \begin{cases} \text{true} & \text{se } \beta \overset{*}{\Rightarrow} \varepsilon \\ \text{false} & \text{caso contrário} \end{cases}$

- Colocamos a gramática na forma BNF.

$E \to TE'$
$E' \to +E$
$E' \to \varepsilon$
$T \to FT'$
$T' \to *T$
$T' \to \varepsilon$
$F \to (E)$
$F \to$ ~~(id)~~ id

$\$ \in FOLLOW(E)$
$+ \in FIRST(E')$
$EPS(E') = true$

$* \in FIRST(T')$
$EPS(T') = true$

$(\in FIRST(F),) \in FOLLOW(E)$

$id \in FIRST(F)$

- Calculamos factos óbvios a partir de pares de símbolos adjacentes no lado dto de cada regra. Obtém-se o seguinte:

	FIRST	FOLLOW	EPS
E		$\$$)	
E'	+		true
T			
T'	*		true
F	(id		

- Agora faz-se um novo passo pelas regras da gramática de modo a deduzir mais coisas.

- Regras:

$$X \rightarrow A\ B\ C\ D$$

- FIRST(A) ⊂ FIRST(X)
  se EPS(A) = true então
    FIRST(B) ⊂ FIRST(X)
  se EPS(A) = true e EPS(B) = true então
    FIRST(c) ⊂ FIRST(X)
  ...

- FIRST(B) ⊂ FOLLOW(A)
  se EPS(B) = true então
    FIRST(c) ⊂ FOLLOW(A)
  ...
  ...

- FOLLOW(X) ⊂ FOLLOW(D)
  se EPS(D) = true então
    FOLLOW(X) ⊂ FOLLOW(c)
  ...

①	FIRST(T) ⊂ FIRST(E)	porque	$E \to TE'$
②	FIRST(E') ⊂ FOLLOW(T)		$E \to TE'$
③	FOLLOW(E) ⊂ FOLLOW(T)		$E \to TE'$ e EPS(E') = true
④	FOLLOW(E) ⊂ FOLLOW(E')		$E \to TE'$
⑤	FOLLOW(E') ⊂ FOLLOW(E)		$E' \to +E$
⑥	FIRST(F) ⊂ FIRST(T)		$T \to FT'$
⑦	FIRST(T') ⊂ FOLLOW(F)		$T \to FT'$
⑧	FOLLOW(T) ⊂ FOLLOW(T')		$T \to FT'$
⑨	FOLLOW(T) ⊂ FOLLOW(F)		$T \to FT'$ e EPS(T') = true
⑩	FOLLOW(T') ⊂ FOLLOW(T)		$T' \to *T$

Após uma 2ª passagem obteríamos o seguinte (ver tabela na próxima página. As entradas a azul são do 1º passo.)

	FIRST	FOLLOW	EPS
E		$)	
E'	+	④ ④ $)	true
T	⑥ ⑥ (id	② ③ ⑤ + $)	
T'	*	⑧ ⑧ ⑧ + $)	true
F	(id	⑦ ⑨ ⑨ ⑨ * + $)	

- Volta-se a fazer mais um passo para ver se podemos acrescentar algo. (Tudo o que é conhecido dos passos anteriores está a azul.)

	FIRST	FOLLOW	EPS
E	① ① (id	$)	
E'	+	$)	true
T	(id	+ $)	
T'	*	+ $)	true
F	(id	* + $)	

- Continua-se a fazer passos até que não se consiga acrescentar mais nada.

- No exemplo dado, se fizermos um 4º passo não conseguimos acrescentar nada, pelo que o algoritmo pára.

Pode-se agora saber os tokens que prevêm uma regra da gramática.

$$PREDICT(A \to \beta) = FIRST(\beta) \cup (FOLLOW(A) \text{ se } \underset{= true}{EPS(\beta)})$$

	PREDICT SET	
$E \rightarrow TE'$	(id	FIRST (T)
$E' \rightarrow +E$	+	FIRST (+)
$E' \rightarrow \varepsilon$	$)	FOLLOW (E')
$T \rightarrow FT'$	(id	FIRST (F)
$T' \rightarrow *T$	*	FIRST (*)
$T' \rightarrow \varepsilon$	+ $)	FOLLOW (T')
$F \rightarrow (E)$	(	FIRST (( )
$F \rightarrow id$	id	FIRST (id)

Estas são as entradas da tabela de parsing que vimos anteriormente.