Compilação: Erros

Detecção de Erros:

* Analisadores Top-Down

  - Preditivo Tabular (LL)
  - “Feito a mão”

* Analisadores Botton-Up:

  - Shift-Reduce (SLR)

* Erros no Lex
* Erros no Yacc
* Erros na Definição da Gramática!

Recuperação de Erros:

* Analisadores Top-Down (LL) – Modo Pânico / Recuperação Local

* Analisadores Botton-Up (SLR) - Ações de recuperação

* Recuperação de erros: análise de expressões com precedência de operadores

Compilação: Erros

Detecção de Erros:

* Analisadores Top-Down

  - Preditivo Tabular (LL)
  - “Feito a mão”

* Analisadores Botton-Up:

  - Shift-Reduce (SLR)

* Erros no Lex
* Erros no Yacc
* Erros na Definição da Gramática!

- Parar a análise no primeiro erro
- Mensagens de erro precisas
- Compiladores famosos...
  Turbo Pascal e Microsoft C (1990)

Recuperação de Erros:

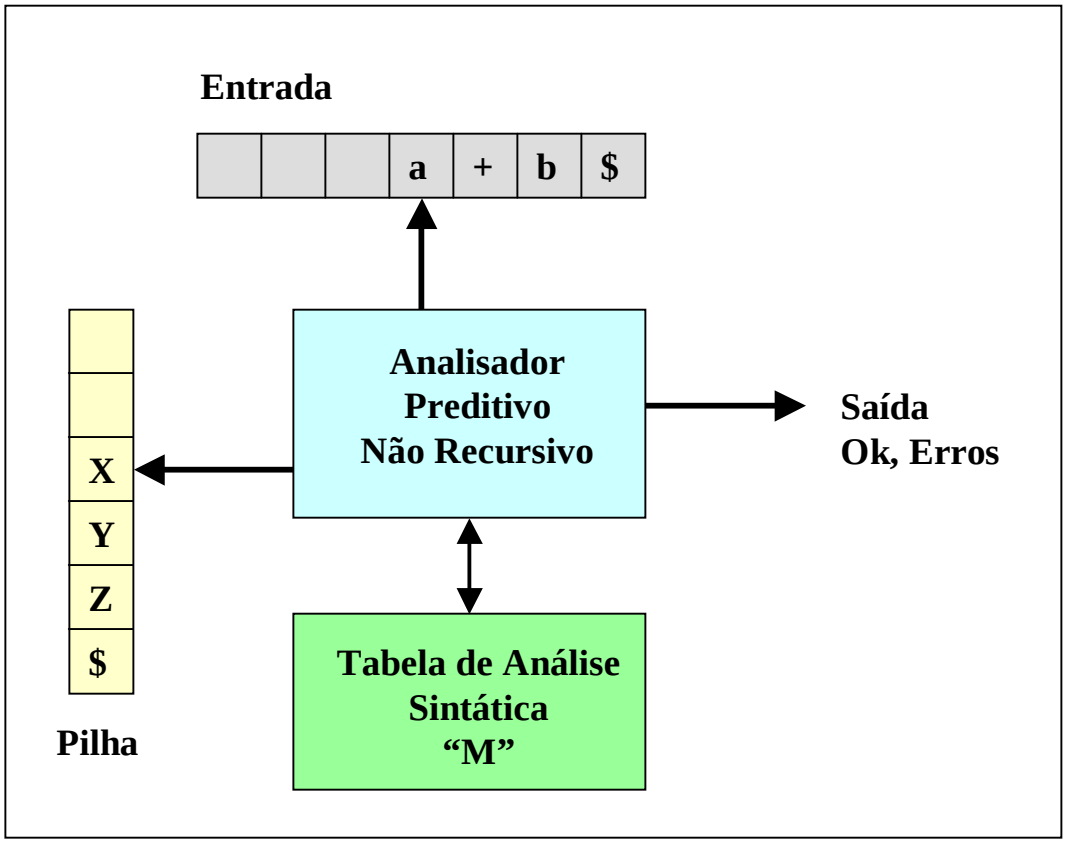
* Analisadores Top-Down (LL) – Modo Pânico / Recuperação Local

* Analisadores Botton-Up (SLR) - Ações de recuperação

* Recuperação de erros: análise de expressões com precedência de operadores

Compilação: DETECÇÃO de Erros - Análise Top Down (LL)

Analisador Preditivo Tabular



Analisador Preditivo Não-Recursivo:

Composto por:

- **Uma "fita de entrada"**
- **Uma pilha auxiliar**
- **Uma tabela de derivação preditiva (Tabela Sintática)**

Funcionamento:

Similar ao Analisador Preditivo Recursivo. Necessita que seja gerada previamente a Tabela Sintática.

É possível construir um analisador preditivo não-recursivo mantendo explicitamente uma pilha, ao invés de implicitamente através de chamadas recursivas

Compilação: DETECÇÃO de Erros - Análise Top Down (LL)

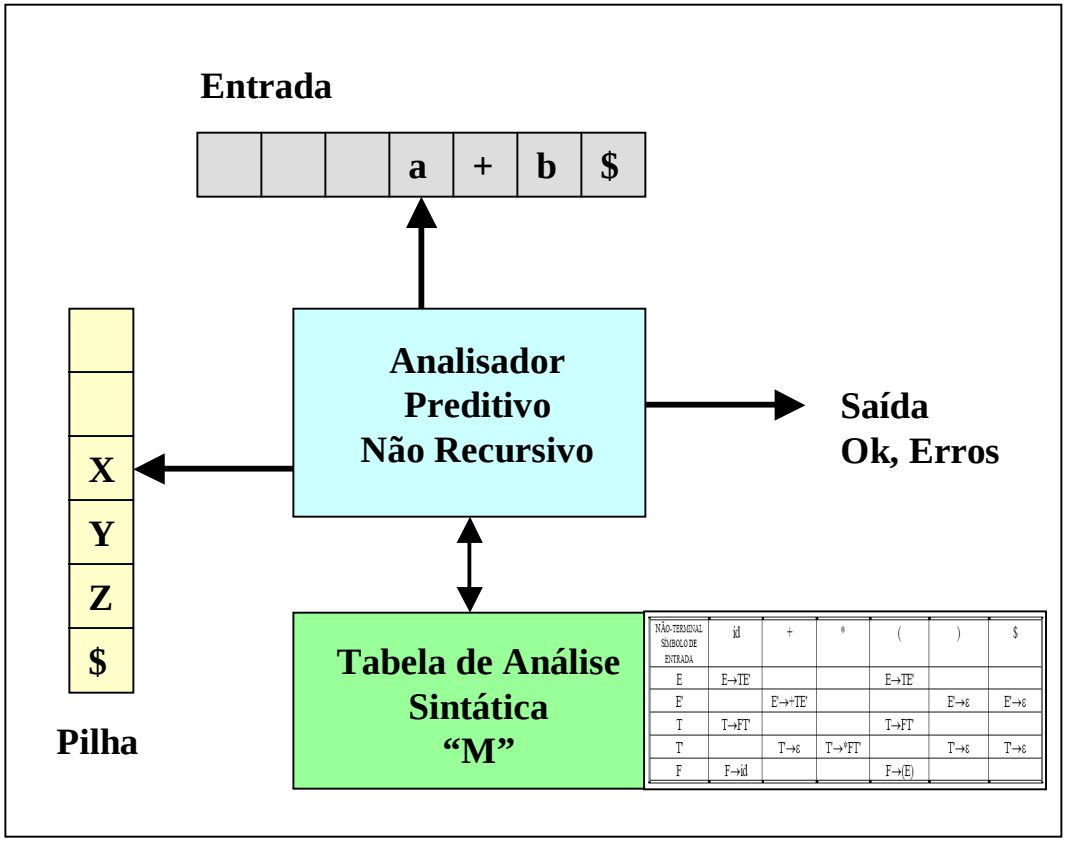
Analisador Preditivo Tabular

Análise Sintática:

O analisador é controlado por um programa que considera X, o símbolo no topo da pilha, e *a* é um símbolo terminal da entrada. Estes dois símbolos determinam a ação do analisador. Há três possibilidades então:

1. **Se X = *a* = $, o analisador encerra o reconhecimento, com sucesso.**
2. **Se X = *a* ≠ $, o analisador desempilha X da pilha e avança o ponteiro de entrada ao próximo símbolo de entrada.**
3. **Se X é um não-terminal, o programa consulta a entrada M[X,*a*] da tabela de derivação M. Esta entrada pode ser uma produção de X ou um erro.**
   Se, por exemplo, M[X,*a*] = { X⇒UVW }, o analisador substitui X no topo da pilha por WVU (com U no topo).

Como saída, assume-se que o analisador apenas imprime a produção utilizada. Se M[X,a] = erro, o analisador chama a rotina de recuperação de erro.



NÃO-TERMINAL / SÍMBOLO DE ENTRADA	id	+	*	(	)	$
E	E→TE'			E→TE'		
E'		E'→+TE'			E'→ε	E'→ε
T	T→FT'			T→FT'		
T'		T'→ε	T'→*FT'		T'→ε	T'→ε
F	F→id			F→(E)		

Tabela de Análise Sintática:

É uma matriz M com *n* linhas e *t+1* colunas, onde *n* é o número de símbolos não-terminais, e *t* é o número de símbolos terminais (a coluna extra corresponde ao Símbolo $)

DETECÇÃO de Erros

Analisador Preditivo Tabular

Regras de Produção

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

Terminais:
{ id, +, *, (,) }

Não-Terminais:
{ E, E´, T, T´, F }

Entrada:
Id + Id * Id

Algoritmo:
Price & Toscani, pg.47

PILHA	ENTRADA	SAÍDA
$E	id+id*id$	
$E'T	id+id*id$	E → TE'
$E'T'F	id+id*id$	T → FT'
$E'T'id	id+id*id$	F → id
$E'T'	+id*id$	
$E'	+id*id$	T' → ε
$E'T+	+id*id$	E'→ +TE'
$E'T	id*id$	
$E'T'F	id*id$	T → FT'
$E'T'id	id*id$	F → id
$E'T'	*id$	
$E'T'F*	*id$	T' → *FT'
$E'T'F	id$	
$E'T'id	id$	F → id
$E'T'	$	
$E'	$	T' → ε
$	$	E' → ε

NÃO-TERMINAL SÍMBOLO DE ENTRADA	id	+	*	(	)	$
E	E→TE'			E→TE'		
E'		E'→+TE'			E'→ε	E'→ε
T	T→FT'			T→FT'		
T'		T'→ε	T'→*FT'		T'→ε	T'→ε
F	F→id			F→(E)		

Gramática LL(1)

DETECÇÃO de Erros

Analisador Preditivo Tabular

Regras de Produção

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

Terminais:
{ id, +, *, (,) }

Não-Terminais:
{ E, E´, T, T´, F }

Entrada:
Id + + Id * Id

Algoritmo:
Price & Toscani, pg.47

PILHA	ENTRADA	SAÍDA
$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 $	
$E'	+ + ID * ID $	T' → ε
$E'T+	+ + ID * ID $	E' → +TE'
$E'T	+ ID * ID $	ERRO!
$E'T'F	ID * ID $	T → FT'
$E'T'id	ID * ID $	F → id
$E'T'	* ID $	
$E'T'F*	*id$	T' → *FT'
$E'T'F	id$	
$E'T'id	id$	F → id
$E'T'	$	
$E'	$	T' → ε
$	$	E' → ε

NÃO-TERMINAL SÍMBOLO DE ENTRADA	id	+	*	(	)	$
E	E→TE'			E→TE'		
E'		E'→+TE'			E'→ε	E'→ε
T	T→FT'	ERRO!		T→FT'		
T'		T'→ε	T'→*FT'		T'→ε	T'→ε
F	F→id			F→(E)		

Gramática LL(1)

Compilação: DETECÇÃO de Erros - Análise Top Down (LL)

Analisador Descendente Recursivo – "Feito a mão"

Predictive Parsing + Hand Coding = Recursive Descent Parser

- One procedure per nonterminal
- That procedure examines the current input symbol
- Recursively calls procedures for RHS of chosen production
- Procedures return true if parse succeeded, false otherwise

Slides From: OCW – Oppen CourseWare MIT / 6.035 Lecture

Compilação: DETECÇÃO de Erros - Análise Top Down (LL)

Analisador Descendente Recursivo – "Feito a mão"

Term → Int *Term'*
Term' → * Int *Term'*
Term' → / Int *Term'*
Term' → ε

Example

```
Term()
    if (token = Int n) token = NextToken(); return(TermPrime())
    else return(false)
TermPrime()
    if (token = *) || (token = /)
        token = NextToken();
        if (token = Int n) token = NextToken(); return(TermPrime())
        else return(false)
    else return(true)
```


Slides From: OCW – Oppen CourseWare MIT / 6.035 Lecture

Compilação: DETECÇÃO de Erros - Análise Top Down (LL)

Analisador Descendente Recursivo – "Feito a mão"

Term → Int *Term'*
Term' → * Int *Term'*
Term' → / Int *Term'*
Term' → ε

Example

```
Term()
    if (token = Int n) token = NextToken(); return(TermPrime())
    else return(false)        ERRO!
TermPrime()
    if (token = *) || (token = /)
        token = NextToken();
        if (token = Int n) token = NextToken(); return(TermPrime())
        else return(false)    ERRO!
    else return(true)
```

Martin Rinard 58 6.035 ©MIT Fall 2000

Slides From: OCW – Oppen CourseWare MIT / 6.035 Lecture

Compilação: DETECÇÃO de Erros - Análise Botton-Up (SLR)

Analisador Shift-Reduce

PILHA	ENTRADA	AÇÃO
(1) $	$id_1 + id_2 * id_3\$$	shift
(2) $id1	$+ id_2 * id_3\$$	reduce por E → id
(3) $E	$+ id_2 * id_3\$$	shift
(4) $E +	$id_2 * id_3\$$	shift
(5) $E + id2	$* id_3\$$	reduce por E → id
(6) $E + E	$* id_3\$$	shift
(7) $E + E *	$id_3\$$	shift
(8) $E + E * id3	$\$$	reduce por E → id
(9) $E + E * E	$\$$	reduce por E → E * E
(10) $E + E	$\$$	reduce por E → E + E
(11) $E	$\$$	accept

Sentença

$id_1+id_2* id_3$

Gramática

$E \rightarrow E + E$
$E \rightarrow E * E$
$E \rightarrow (E)$
$E \rightarrow id$

Tipos de Ações

1. *shift*: o próximo símbolo de entrada é colocado no topo da pilha.
2. *reduce*: o analisador reconhece o lado direito do *handle* que está no topo da pilha, devendo então pesquisar o lado esquerdo e decidir que não-terminal será utilizado para substituí-lo.
3. *accept*: o analisador identifica um estado de final de análise, com sucesso.
4. *error*: o analisador identifica um erro de sintaxe e chama (se houver) uma rotina de recuperação de erro.

Compilação: DETECÇÃO de Erros - Análise Botton-Up (SLR)

Analisador Shift-Reduce

Parse Table Example

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

Shift to s*n*
- Push input token into the symbol stack
- Push s*n* into state stack
- Advance to next input symbol

Reduce (*n*)
- Pop both stacks as many times as the number of symbols on the RHS of rule *n*
- Push LHS of rule *n* into symbol stack

Reduce (*n*) (continued)
- Look up
  - Table[top of the state stack][top of symbol stack]
- Push that state (in goto part of table) onto state stack

State Stack	Symbol Stack	Input	Grammar
		(())	$S \rightarrow X\$$ (1)
			$X \rightarrow (X)$ (2)
s0	X		$X \rightarrow (\ )$ (3)

Slides From: OCW – Oppen CourseWare MIT / 6.035 Lecture

Compilação: DETECÇÃO de Erros - Análise Botton-Up (SLR)

Analisador Shift-Reduce

Parse Table Example

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	ERRO!	ERRO!	goto s1
s1	ERRO!	ERRO!	accept	
s2	shift to s2	shift to s5	ERRO!	goto s3
s3	ERRO!	shift to s4	ERRO!	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

Shift to s*n*
- Push input token into the symbol stack
- Push s*n* into state stack
- Advance to next input symbol

Reduce (*n*)
- Pop both stacks as many times as the number of symbols on the RHS of rule *n*
- Push LHS of rule *n* into symbol stack

Reduce (*n*) (continued)
- Look up
  - Table[top of the state stack][top of symbol stack]
- Push that state (in goto part of table) onto state stack

State Stack: s0

Symbol Stack: *X*

Input: (())

Grammar:

$S \rightarrow X\$$ (1)

$X \rightarrow (X)$ (2)

$X \rightarrow ( )$ (3)

O que fazer quando acha um erro?!?

Slides From: OCW – Oppen CourseWare MIT / 6.035 Lecture

Compilação: DETECÇÃO de Erros - Análise Botton-Up (SLR)

Analisador Shift-Reduce

Parse Table Example

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	ERRO!	ERRO!	goto s1
s1	ERRO!	ERRO!	accept	
s2	shift to s2	shift to s5	ERRO!	goto s3
s3	ERRO!	shift to s4	ERRO!	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

Shift to s*n*
- Push input token into the symbol stack
- Push s*n* into state stack
- Advance to next input symbol

Reduce (*n*)
- Pop both stacks as many times as the number of symbols on the RHS of rule *n*
- Push LHS of rule *n* into symbol stack

Reduce (*n*) (continued)
- Look up
  - Table[top of the state stack][top of symbol stack]
- Push that state (in goto part of table) onto state stack

State Stack	Symbol Stack	Input	Grammar
		(())	$S \rightarrow X\$$ (1)
			$X \rightarrow (X)$ (2)
s0	*X*		$X \rightarrow ( )$ (3)

O que fazer quando acha um erro?!?

Se parar: Mensagem Precisa!

Slides From: OCW – Oppen CourseWare MIT / 6.035 Lecture

Compilação: Recuperação de Erros

Recuperação de Erros

* O que é ?

  - Quando um compilador detecta um erro de sintaxe é desejável que ele tente continuar o processo de análise de modo a detectar outros erros.

* Qual a sua função ?

  - Na recuperação de erros, tenta-se colocar o analisador em um estado tal que o restante da sentença de entrada ainda possa ser analisada. Este processo pode envolver: a modificação da entrada (excluir tokens), a modificação da pilha, ou de ambos.

* Problema: gerar mais erros do que se não tivesse tratado...

* Tipos de recuperação de erro

  - **Modo Pânico**: forma mais simples de detecção na qual o analisador lê e descarta símbolos da entrada até encontrar um token de sincronização;
  - **Modo Recuperação Local**: recupera o erro fazendo alterações sobre um símbolo, desprezando um token de entrada, ou substituindo-o por outro, ou inserindo um novo token, ou ainda removendo um símbolo da pilha.

Compilação: Recuperação de Erros

Recuperação de Erros

* Exemplos simples:

A := B + C <= Erro!
While A < X
Do A:= A+1;

A : = B C + D; <= Erro!

* Qualidade da solução...

Um algoritmo de recuperação pode tentar reiniciar a análise inserindo um ";" após o identificador "C"... OK

Um algoritmo de recuperação pode tentar reiniciar a análise inserindo um ";" após o identificador "B"... Mais problemas!!!

- A qualidade de uma rotina de recuperação de erros é medida pela precisão com que ela re-sincroniza a análise e é inversamente proprocional à sua capacidade de gerar falsos erros

Compilação: RECUPERAÇÃO de Erros - Análise Top Down (LL)

Analisador Preditivo Tabular

Regras de Produção

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

Terminais:
{ id, +, *, (,) }

Não-Terminais:
{ E, E´, T, T´, F }

Entrada:
Id + Id * Id

Entrada:
Id + + Id * Id

Recuperação?

PILHA	ENTRADA	SAÍDA
$E	id+id*id$	
$E'T	id+id*id$	E → TE'
$E'T'F	id+id*id$	T → FT'
$E'T'id	id+id*id$	F → id
$E'T'	+id*id$	
$E'	+id*id$	T' → ε
$E'T+	+id*id$	E'→ +TE'
$E'T	id*id$	
$E'T'F	id*id$	T → FT'
$E'T'id	id*id$	F → id
$E'T'	*id$	
$E'T'F*	*id$	T' → *FT'
$E'T'F	id$	
$E'T'id	id$	F → id
$E'T'	$	
$E'	$	T' → ε
$	$	E' → ε

NÃO-TERMINAL SÍMBOLO DE ENTRADA	id	+	*	(	)	$
E	E→TE'			E→TE'		
E'		E'→+TE'			E'→ε	E'→ε
T	T→FT'			T→FT'		
T'		T'→ε	T'→*FT'		T'→ε	T'→ε
F	F→id			F→(E)		

Compilação: RECUPERAÇÃO de Erros - Análise Top Down (LL)

Analisador Preditivo Tabular

Regras de Produção

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

Terminais:
{ id, +, *, (,) }

Não-Terminais:
{ E, E´, T, T´, F }

Entrada:
Id + Id * Id

Entrada:
Id + + Id * Id

Recuperação...

Modo Pânico:

=> Despreza símbolos da entrada até encontrar um símbolo de sincronização

=> Método bastante simples...

=> Resultado também é simplório...

NÃO-TERMINAL SÍMBOLO DE ENTRADA	id	+	*	(	)	$
E	E→TE'			E→TE'		
E'		E'→+TE'			E'→ε	E'→ε
T	T→FT'			T→FT'		
T'		T'→ε	T'→*FT'		T'→ε	T'→ε
F	F→id			F→(E)		

Compilação: RECUPERAÇÃO de Erros - Análise Top Down (LL)

Analisador Preditivo Tabular

Regras de Produção

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

Terminais:
{ id, +, *, (,) }

Não-Terminais:
{ E, E´, T, T´, F }

Entrada:
Id + Id * Id

Entrada:
Id + + Id * Id

Modo Pânico:

- Na tabela do analisador preditivo tabular, as entradas em branco representam situações de erro;
- Alteramos a tabela do reconhecedor de modo a desprezar símbolos de entrada até encontrar um token de sincronização;
- O conjunto de tokens de sincronização para um símbolo-não terminal A é formado pelos símbolos terminais contidos em Follow(A). Exemplo:
  Follow (E) = Follow (E') = {), $ }
  Follow (T) = Follow (T') = { +,), $ }
  Follow (F) = { +, *,), $ }

Recuperação... nova Tabela do Reconhecedor LL com extensão para tratar erros

	id	+	*	(	)	$
E	E → TE′			E→TE′	sinc	sinc
E′		E′→+TE′			E′ → ε	E′ → ε
T	T → FT′	sinc		T → FT′	sinc	sinc
T′		T′→ε	T′→*FT′		T′→ ε	T′→ ε
F	F → id	sinc	sinc	F → (E)	sinc	sinc

Compilação: RECUPERAÇÃO de Erros - Análise Top Down (LL)

Analisador Preditivo Tabular

Regras de Produção

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

Terminais:
{ id, +, *, (,) }

Não-Terminais:
{ E, E´, T, T´, F }

Entrada:
Id + Id * Id

Entrada:
Id + + Id * Id

Modo Pânico:

- se a entrada estiver vazia, lê-se o próximo token;
- se a entrada é *sinc*, desempilha o não-terminal do topo;
- se o token do topo não é igual ao símbolo da entrada, desempilha.

- O conjunto de tokens de sincronização para um símbolo-não terminal A é formado pelos símbolos terminais contidos em Follow(A). Exemplo:
Follow (E) = Follow (E') = {), $ }
Follow (T) = Follow (T') = { +,), $ }
Follow (F) = { +, *,), $ }

Recuperação... nova Tabela do Reconhecedor LL com extensão para tratar erros

	id	+	*	(	)	$
E	E → TE′			E→TE′	sinc	sinc
E′		E′→+TE′			E′ → ε	E′ → ε
T	T → FT′	sinc		T → FT′	sinc	sinc
T′		T′→ε	T′→*FT′		T′→ ε	T′→ ε
F	F → id	sinc	sinc	F → (E)	sinc	sinc

Compilação: RECUPERAÇÃO de Erros - Análise Top Down (LL)

Analisador Preditivo Tabular

Regras de Produção

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

Terminais:
{ id, +, *, (,) }

Não-Terminais:
{ E, E´, T, T´, F }

Entrada:
Id + Id * Id

Entrada:
Id + + Id * Id

Recuperação... nova Tabela

Modo Pânico:

- se a entrada estiver vazia, lê-se o próximo token;
- se a entrada é *sinc*, desempilha o não-terminal do topo;
- se o token do topo não é igual ao símbolo da entrada, desempilha.

PILHA	ENTRADA	SAÍDA
$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 $	
$E'	+ + ID * ID $	T' → ε
$E'T+	+ + ID * ID $	E' → +TE'
$E'T	+ ID * ID $	ERRO!
$E'	+ ID * ID $	E' => +TE'
$E'T+	+ ID * ID $	
$E'T	ID * ID $	T => FT'

	id	+	*	(	)	$
E	E → TE′			E→TE′	sinc	sinc
E′		E′→+TE′			E′ → ε	E′ → ε
T	T → FT′	sinc		T → FT′	sinc	sinc
T′		T′→ε	T′→*FT′		T′→ ε	T′→ ε
F	F → id	sinc	sinc	F → (E)	sinc	sinc

Compilação: RECUPERAÇÃO de Erros - Análise Top Down (LL)

Analisador Preditivo Tabular

Regras de Produção

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

Terminais:
{ id, +, *, (,) }

Não-Terminais:
{ E, E´, T, T´, F }

Entrada:
Id + Id * Id

Entrada:
Id + + Id * Id

Recuperação...

Modo Recuperação Local:

=> Despreza símbolos da entrada, substitui ou insere um símbolo a cada erro detectado (na inserção tem que evitar laço infinito)

=> Método mais sofisticado

=> Criação da Tabela do Reconhecedor com a inclusão de situações específicas para tratar diferentes erros (expande tabela acrescentando linhas para os terminais também)

NÃO-TERMINAL SÍMBOLO DE ENTRADA	id	+	*	(	)	$
E	E→TE'			E→TE'		
E'		E'→+TE'			E'→ε	E'→ε
T	T→FT'			T→FT'		
T'		T'→ε	T'→*FT'		T'→ε	T'→ε
F	F→id			F→(E)		

Compilação: RECUPERAÇÃO de Erros - Análise Top Down (LL)

Analisador Preditivo Tabular

Regras de Produção

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

Terminais:
{ id, +, *, (,) }

Não-Terminais:
{ E, E´, T, T´, F }

Modo Recuperação Local:

=> Criação da Tabela do Reconhecedor com a inclusão de situações específicas para tratar diferentes erros (expande tabela acrescentando linhas para os terminais também)

=> Em branco: situações que não ocorrem

NÃO-TERMINAL SÍMBOLO DE ENTRADA	id	+	*	(	)	$
E	E→TE'	**ERRO1**	**ERRO1**	E→TE'	**ERRO1**	**ERRO1**
E'	**Descarta**	E'→+TE'	**Descarta**	**Descarta**	E'→ε	E'→ε
T	T→FT'	**ERRO1**	**ERRO1**	T→FT'	**ERRO1**	**ERRO1**
T'	**Descarta**	T'→ε	T'→*FT'	**Descarta**	T'→ε	T'→ε
F	F→id	**ERRO1**	**ERRO1**	F→(E)	**ERRO1**	**ERRO1**
id	**Desempilha**					
+		**Desempilha**				
*****			**Desempilha**			
(				**Desempilha**		
)	**ERRO2**	**ERRO2**	**ERRO2**	**ERRO2**	**Desempilha**	**ERRO2**
$	**ERRO3**	**ERRO3**	**ERRO3**	**ERRO3**	**ERRO3**	**ACEITA!**

Compilação: RECUPERAÇÃO de Erros - Análise Top Down (LL)

Analisador Preditivo Tabular

Regras de Produção

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

Terminais:
{ id, +, *, (,) }

Não-Terminais:
{ E, E´, T, T´, F }

Modo Recuperação Local:

Rotinas de erro típicas para o exemplo abaixo:
ERRO1: insere token id na entrada e msg "operando esperado"
ERRO2: desempilha o símbolo ")" e msg "parêntese esperado"
ERRO3: descarta o token da entrada e msg "fim do arquivo encontrado"
Descarta: retira da pilha um não terminal
Desempilha: retira da pilha um terminal

NÃO-TERMINAL SÍMBOLO DE ENTRADA	id	+	*	(	)	$
E	E→TE'	**ERRO1**	**ERRO1**	E→TE'	**ERRO1**	**ERRO1**
E'	**Descarta**	E'→+TE'	**Descarta**	**Descarta**	E'→ε	E'→ε
T	T→FT'	**ERRO1**	**ERRO1**	T→FT'	**ERRO1**	**ERRO1**
T'	**Descarta**	T'→ε	T'→*FT'	**Descarta**	T'→ε	T'→ε
F	F→id	**ERRO1**	**ERRO1**	F→(E)	**ERRO1**	**ERRO1**
id	**Desempilha**					
+		**Desempilha**				
*****			**Desempilha**			
(				**Desempilha**		
)	**ERRO2**	**ERRO2**	**ERRO2**	**ERRO2**	**Desempilha**	**ERRO2**
$	**ERRO3**	**ERRO3**	**ERRO3**	**ERRO3**	**ERRO3**	**ACEITA!**

Compilação: RECUPERAÇÃO de Erros - Análise Botton-Up (SLR)

Analisador Shift-Reduce

Parse Table Example

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

Shift to s*n*
- Push input token into the symbol stack
- Push s*n* into state stack
- Advance to next input symbol

Reduce (*n*)
- Pop both stacks as many times as the number of symbols on the RHS of rule *n*
- Push LHS of rule *n* into symbol stack

Reduce (*n*) (continued)
- Look up
  - Table[top of the state stack][top of symbol stack]
- Push that state (in goto part of table) onto state stack

State Stack	Symbol Stack	Input	Grammar
		(())	$S \rightarrow X\$$ (1)
			$X \rightarrow (X)$ (2)
			$X \rightarrow (\ )$ (3)
s0	X		

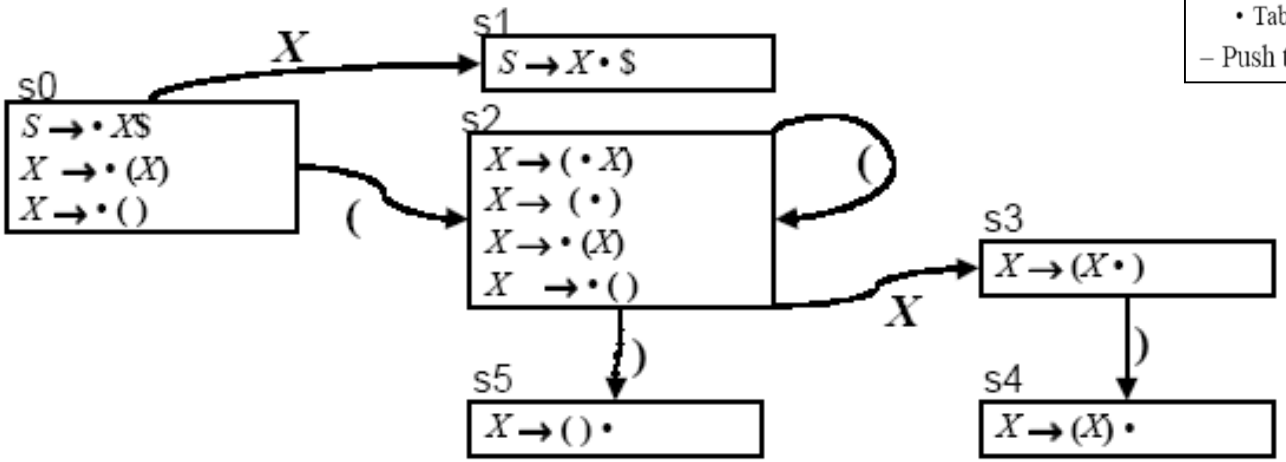
Slides From: OCW – Oppen CourseWare MIT / 6.035 Lecture

Compilação: RECUPERAÇÃO de Erros - Análise Botton-Up (SLR)

Analisador Shift-Reduce

Building Parse Table Example

State	ACTION (	)	$	Goto X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	



Shift to s*n*
- Push input token into the symbol stack
- Push s*n* into state stack
- Advance to next input symbol

Reduce (*n*)
- Pop both stacks as many times as the number of symbols on the RHS of rule *n*
- Push LHS of rule *n* into symbol stack

Reduce (*n*) (continued)
- Look up
  - Table[top of the state stack][top of symbol stack]
- Push that state (in goto part of table) onto state stack

Grammar

$S \rightarrow X\$$ (1)

$X \rightarrow (X)$ (2)

$X \rightarrow ( )$ (3)

=> Criação da Tabela do Reconhecedor com a inclusão de situações específicas para tratar cada tipo de erro (só ocorrem ao ler o próximo token)

Compilação: RECUPERAÇÃO de Erros

Analisador Descendente Recursivo – "Feito a mão"

Term → Int *Term'*
Term' → * Int *Term'*
Term' → / Int *Term'*
Term' → ε

Example

```
Term()
    if (token = Int n) token = NextToken(); return(TermPrime())
    else return(false)
TermPrime()
    if (token = *) || (token = /)
          token = NextToken();
          if (token = Int n) token = NextToken(); return(TermPrime())
          else return(false)
    else return(true)
```

O que fazer quando acha um erro?!?