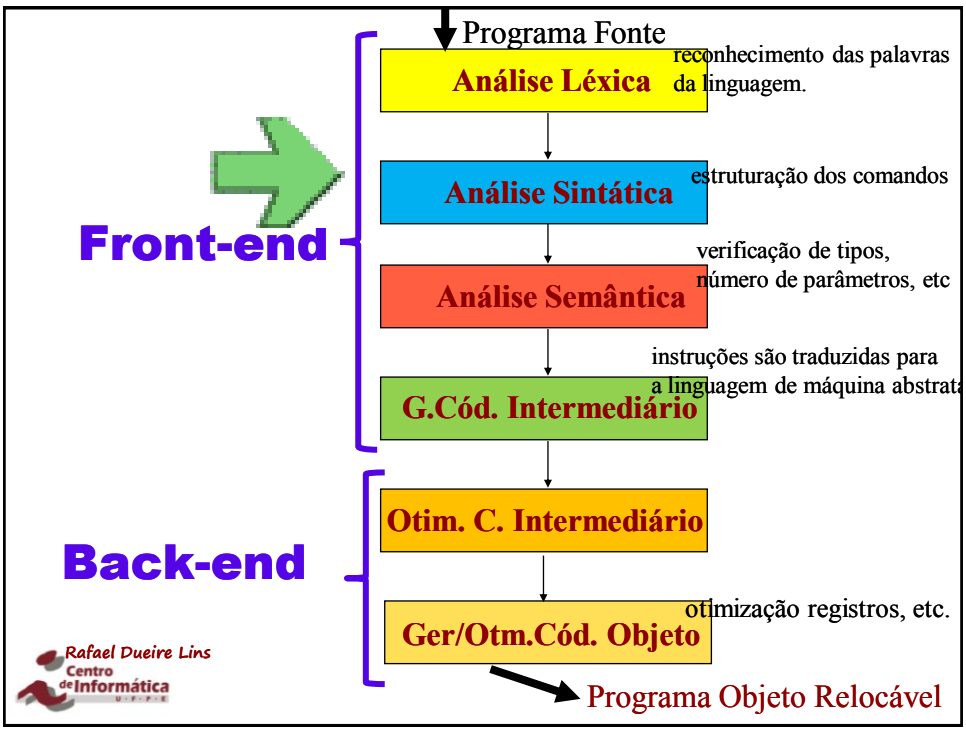
Análise Sintática

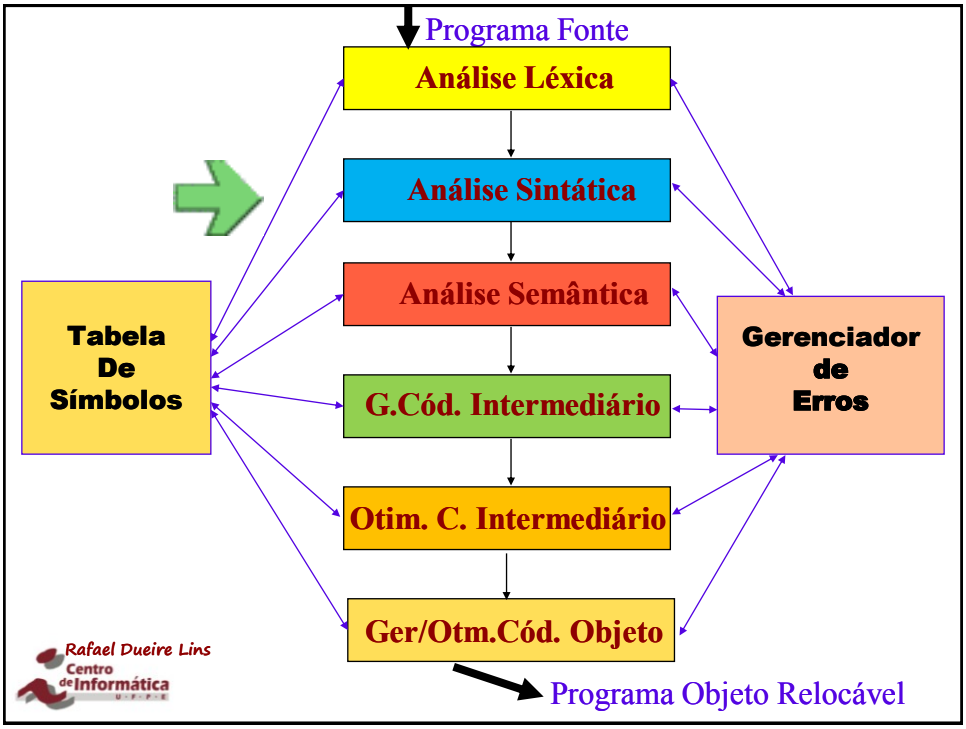
Rafael Dueire Lins

A análise sintática ou parsing é a segunda fase da compilação.

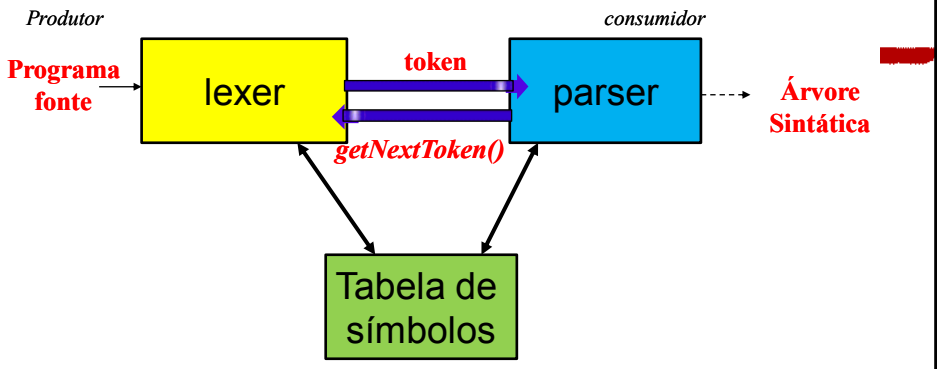
Rafael Dueire Lins
Centro de Informática UFPE



Rafael Dueire Lins
Centro de Informática UFPE



Relacionamento Léxico-Sintático



Separação Lexer-Parser:

- **Simplicidade:** misturar análise léxica com sintática torna o *parser* mais complicado
- **Eficiência:** separação possibilita construir processadores léxicos e sintáticos mais eficientes.

Rafael Dueire Lins
Centro de Informática

Gramática
Sintaxe de uma linguagem

- Define os *strings* estruturalmente válidos de uma linguagem.
- *Parsing* correto:
  Todos as sentenças estão “bem-formadas”.
- O programa pode ainda...
  - Não passar na checagem de tipos (<70 % erros)
  - Conter erros na lógica!!!

Rafael Dueire Lins
Centro de Informática UFPE

Gramática
Especificação da Sintaxe

- Dever ser precisa e fácil de entender
  - Precisa: não ambígua
  - Fácil de entender: deixa evidente a sintaxe da linguagem
- **Gramáticas livre de contexto.**
- **Forma de Backus-Naur** (BNFs) é um formalismo adequado
  - Fácil de especificar/manter/entender

Rafael Dueire Lins
Centro de Informática UFPE

Gramática
Especificação da Sintaxe

- **Terminais:**
  - Símbolos básicos com os quais *strings* são formados.
  - Token ≡ terminal (if, then, else, while, etc.)
- **Não-terminais:**
  - Variáveis sintáticas – conjuntos de strings.
  - Símbolo inicial.
- **Produções:**
  - Modo como terminais e não-terminais formam *strings*.

Rafael Dueire Lins
Centro de Informática UFPE

Gramática
Convenções Notacionais

- **Terminais:**
  - a, b, c, ... (minúsculas iniciais do alfabeto)
  - +, -, *, (operadores)
  - (,), “,”, ; (símbolos de pontuação)
  - 0, 1, 2, ..., 9 (dígitos)
  - Strings em negrito (e.g.: **id**, **if**, **then**)
- **Não-terminais:**
  - A, B, C, ... (maiuscúlas iniciais do alfabeto)
  - S – geralmente o símbolo inicial.
  - Nomes minúsculos em itálico (*expr*, *stmt*, ...)
- **Símbolos da Gramática:**
  - X, Y, Z - terminais ou não-terminais

Rafael Dueire Lins
Centro de Informática UFPE

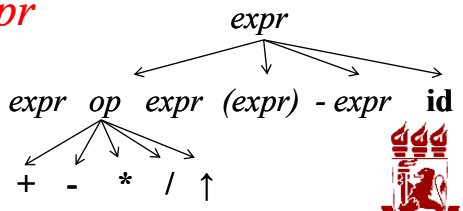
Gramática
Convenções Notacionais

- ***Strings* de Terminais:**
  u, v, ..., z, ... (minúsculas finais do alfabeto)
- ***Strings* de símbolos da gramática:**
  $\alpha, \beta, \gamma, \delta, \eta, \ldots$ (minúsculas do alfabeto grego)
- **Alternativas:**
  $A \rightarrow \alpha_1, A \rightarrow \alpha_2, A \rightarrow \alpha_3, A \rightarrow \alpha_n,$
  $A \rightarrow \alpha_1 | \alpha_2 | \alpha_3 | \ldots | \alpha_n,$
- **Símbolo inicial:**
  Se nada dito em contrário, o lado esquerdo da primeira produção.

Gramática
Exemplo

expr → *expr op expr* | (*expr*) | - *expr* | **id**
op → **+** | **-** | ***** | **/** | ↑

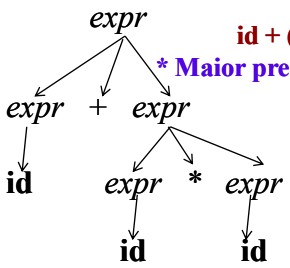
- **Terminais:**
  + - * / **id** ↑
- **Não-terminais:** *expr, op.*
- **Símbolo inicial:** *expr*



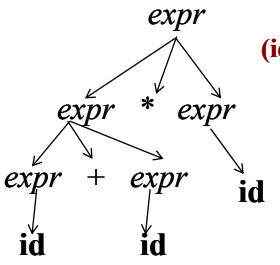
Gramática Ambígua
Gera Mais de Uma Árvore Sintática

expr → *expr op expr* | (*expr*) | - *expr* | id
op → + | - | * | / | ↑

id + id * id



id + (id * id)
* Maior precedência que +



(id + id) * id

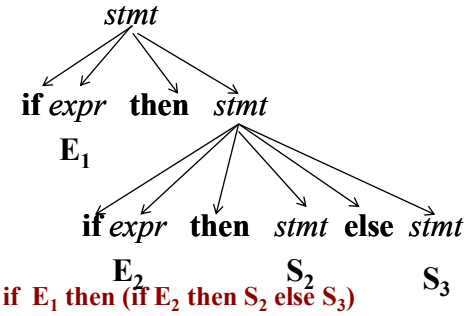
Rafael Dueire Lins
Centro de Informática UFPE

Ambiguidade deve ser evitada!!!

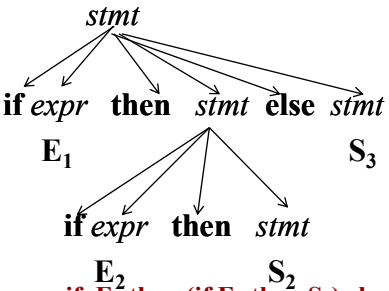
Gramática Ambígua
Gera Mais de uma Árvore Sintática

stmt → if *expr* then *stmt*
| if *expr* then *stmt* *else* *stmt*
| other

if E_1 then if S_1 then S_2 else S_3



if E_1 then (if E_2 then S_2 else S_3)
preferível



if E_1 then (if E_2 then S_2) else S_3

Recursão à esquerda

- Dificuldade em saber quando parar de aplicar uma produção

$S \rightarrow \mathbf{a}A\mathbf{e}$

$A \rightarrow A\mathbf{bc} \mid \mathbf{b}$

$B \rightarrow \mathbf{d}$

$\mathbf{a}\underline{A}\mathbf{e}$

$\rightarrow \mathbf{a}\underline{A}\mathbf{bc}B\mathbf{e}$

$\rightarrow \mathbf{a}\underline{A}\mathbf{bcbc}B\mathbf{e}$

$\rightarrow \mathbf{a}\underline{A}\mathbf{bcbcbc}B\mathbf{e}$

$\rightarrow \ldots$

Rafael Dueire Lins
Centro de Informática UFPE

Expressões Regulares vs. Gramáticas Livres de Contexto

- Toda ER pode ser descrita como uma GLC.
- Conversão automática ER→GLC.
- Por que usar ER para a parte léxica?
  - Regras léxicas são muito simples!
  - ER são mais concisas e intuitivas.
  - ER permitem analisadores ótimos!
  - Modularidade e Mantainabilidade!
- Estruturas aninhadas não podem ser expressas por ER (não há memória).

Construtores Não-Livres-de-Contexto

- Algumas linguagens não podem ser especificadas por gramáticas somente.
- $L_1 = \{wcw \mid w \text{ em } (a|b)^*\}$

  declaração – uso do identificador

  $L_2 = \{a^n b^m c^n d^m \mid n >= 1 \text{ e } m >= 1\}$

  contagem dos parâmetros em uma chamada de subrotina

  $L_3 = \{a^n b^n c^n \mid n >= 0\}$

  contagem dos parâmetros
- Em alguns casos pode-se obter GLCs equivalentes!!

Tipos de *Parsers*

- Universais:
  - todo tipo de gramática.
  - Cocke-Younger[1967]-Kasami [1965] e Earley [1970]
  - Ineficientes
- Top-down ou Bottom-up:
  - Sub-classes de gramáticas tais como LL e LR.
  - LR podem ser automatizados.

Top-down parsing

- Da raiz para as folhas.
- Procura sequência de derivações mais à esquerda para se obter um *string* de entrada
- O parsing do *string* **abbcbcde** é feito pela sequência de derivações abaixo:

$S \rightarrow \mathbf{a}A\mathbf{e}$
$A \rightarrow A\mathbf{bc} \mid \mathbf{b}$
$B \rightarrow \mathbf{d}$

$S \rightarrow \mathbf{a}\underline{A}\mathbf{e}$
$\rightarrow \mathbf{a}\underline{A}\mathbf{bc}\mathbf{e}$
$\rightarrow \mathbf{a}\underline{A}\mathbf{bcbc}\mathbf{e}$
$\rightarrow \mathbf{abbcbc}\underline{\mathbf{B}}\mathbf{e}$
$\rightarrow \mathbf{abbcbcde}$

Rafael Dueire Lins
Centro de Informática UFPE

Top-down parsing

- Inicie com símbolo raiz
- Consuma *tokens* da esquerda para à direita
- Decida que produção aplicar

$S \rightarrow \mathbf{a}A\mathbf{e}$
$A \rightarrow A\mathbf{bc} \mid \mathbf{b}$
$B \rightarrow \mathbf{d}$

a	b	b	c	b	c	d	e		S

Rafael Dueire Lins
Centro de Informática UFPE

Top-down parsing

- Inicie com símbolo raiz
- Consuma tokens da esquerda para direita
- Decida que produção aplicar

$S \rightarrow$ **a***AB***e**
$A \rightarrow$ ***A*bc | b**
$B \rightarrow$ **d**

a	b	b	c	b	c	d	e

a*AB***e**

Rafael Dueire Lins
Centro de Informática UFPE

Top-down parsing

- Inicie com símbolo raiz
- Consuma tokens da esquerda para direita
- Decida que produção aplicar

$S \rightarrow$ **a***AB***e**
$A \rightarrow$ ***A*bc | b**
$B \rightarrow$ **d**

escolha!

a	b	b	c	b	c	d	e

AB
e

Rafael Dueire Lins
Centro de Informática UFPE

Top-down parsing

- Inicie com símbolo raiz
- Consuma tokens da esquerda para direita
- Decida que produção aplicar

S → **a***AB***e**
A → ***A*bc** | **b**
B → **d**

a	b	b	c	b	c	d	e

b*B***e**

Rafael Dueire Lins
Centro de Informática UFPE

Top-down parsing

- Inicie com símbolo raiz
- Consuma tokens da esquerda para direita
- Decida que produção aplicar

S → **a***AB***e**
A → ***A*bc** | **b**
B → **d**

a	b	b	c	b	c	d	e

***B*e**

✘ **Erro. Backtrack!**

Rafael Dueire Lins
Centro de Informática UFPE

Top-down parsing

- Inicie com símbolo raiz
- Consuma tokens da esquerda para direita
- Decida que produção aplicar

$S \rightarrow \mathbf{a}AB\mathbf{e}$
$A \rightarrow A\mathbf{bc} \mid \mathbf{b}$
$B \rightarrow \mathbf{d}$

a	b	b	c	b	c	d	e

ABe

Faz outra escolha!

Restaura estado anterior a última escolha

Rafael Dueire Lins
Centro de Informática UFPE

Top-down parsing

- Inicie com símbolo raiz
- Consuma tokens da esquerda para direita
- Decida que produção aplicar

$S \rightarrow \mathbf{a}AB\mathbf{e}$
$A \rightarrow A\mathbf{bc} \mid \mathbf{b}$
$B \rightarrow \mathbf{d}$

a	b	b	c	b	c	d	e

*A***bc***B***e*

Rafael Dueire Lins
Centro de Informática UFPE

Top-down parsing

- Inicie com símbolo raiz
- Consuma tokens da esquerda para direita
- Decida que produção aplicar

$S \rightarrow \mathbf{a}AB\mathbf{e}$
$A \rightarrow A\mathbf{bc} \mid \mathbf{b}$
$B \rightarrow \mathbf{d}$

escolha novamente!

a	b	b	c	b	c	d	e

*A*bc*B*e

Símbolo *A* novamente na posição mais à esquerda

Rafael Dueire Lins
Centro de Informática UFPE

Top-down parsing

- Inicie com símbolo raiz
- Consuma tokens da esquerda para direita
- Decida que produção aplicar

$S \rightarrow \mathbf{a}AB\mathbf{e}$
$A \rightarrow A\mathbf{bc} \mid \mathbf{b}$
$B \rightarrow \mathbf{d}$

Outra escolha!

a	b	b	c	b	c	d	e

bbc*B*e

Rafael Dueire Lins
Centro de Informática UFPE

Top-down parsing

- Inicie com símbolo raiz
- Consuma tokens da esquerda para direita
- Decida que produção aplicar

S → **a***AB***e**
A → *A***bc** | **b**
B → **d**

a	b	b	c	b	c	d	e

bc*Be*

Rafael Dueire Lins
Centro de Informática UFPE

Top-down parsing

- Inicie com símbolo raiz
- Consuma tokens da esquerda para direita
- Decida que produção aplicar

S → **a***AB***e**
A → *A***bc** | **b**
B → **d**

a	b	b	c	b	c	d	e

Be

✘ **Erro. Backtrack!**

Rafael Dueire Lins
Centro de Informática UFPE

Top-down parsing

- Inicie com símbolo raiz
- Consuma tokens da esquerda para direita
- Decida que produção aplicar

S → **a***AB***e**
A → ***A*bc** | **b**
B → **d**

a	b	b	c	b	c	d	e

***A*bc*B*e**

Restaura estado anterior a última escolha

Rafael Dueire Lins
Centro de Informática UFPE

Top-down parsing

- Inicie com símbolo raiz
- Consuma tokens da esquerda para direita
- Decida que produção aplicar

S → **a***AB***e**
A → ***A*bc** | **b**
B → **d**

a	b	b	c	b	c	d	e

***A*bcbc*B*e**

Faz outra escolha!

Rafael Dueire Lins
Centro de Informática UFPE

Top-down parsing

- Inicie com símbolo raiz
- Consuma tokens da esquerda para direita
- Decida que produção aplicar

S → **a***AB***e**
A → ***A*bc** | **b**
B → **d**

a	**b**	**b**	**c**	**b**	**c**	**d**	**e**	**bbcbc*Be***

Top-down parsing

- Inicie com símbolo raiz
- Consuma tokens da esquerda para direita
- Decida que produção aplicar

S → **a***AB***e**
A → ***A*bc** | **b**
B → **d**

a	**b**	**b**	**c**	**b**	**c**	**d**	**e**	***Be***

Top-down parsing

- Inicie com símbolo raiz
- Consuma tokens da esquerda para direita
- Decida que produção aplicar

$S \rightarrow \mathbf{a}AB\mathbf{e}$
$A \rightarrow A\mathbf{bc} \mid \mathbf{b}$
$B \rightarrow \mathbf{d}$

a	b	b	c	b	c	d	e	de

Rafael Dueire Lins
Centro de Informática UFPE

Top-down parsing

- Inicie com símbolo raiz
- Consuma tokens da esquerda para direita
- Decida que produção aplicar

$S \rightarrow \mathbf{a}AB\mathbf{e}$
$A \rightarrow A\mathbf{bc} \mid \mathbf{b}$
$B \rightarrow \mathbf{d}$

a	b	b	c	b	c	d	e	

Rafael Dueire Lins
Centro de Informática UFPE

Top-down parsing

- Inicie com símbolo raiz
- Consuma tokens da esquerda para direita
- Decida que produção aplicar

$S \rightarrow aABe$
$A \rightarrow Abc \mid b$
$B \rightarrow d$

a	b	b	c	b	c	d	e	

derivação correspondente!

$S \rightarrow a\underline{A}Be$
$\rightarrow a\underline{A}bcBe$
$\rightarrow a\underline{A}bcbcBe$
$\rightarrow abbcbc\underline{B}e$
$\rightarrow abbcbcde$

Rafael Dueire Lins
Centro de Informática UFPE

Classificação de *parsers*

Top-down parsers operam desta forma!

Número de tokens de lookahead lidos porém não consumidos

LL(k)

Lê entrada da esquerda (L) para a direita

Procura derivação mais à esquerda

Rafael Dueire Lins
Centro de Informática UFPE

Resultados gerais

Pode-se construir parser LL(k=1) para uma gramática não- ambígua.

Gramáticas ambíguas pedem parsers com valor de k > 1.

Existem algoritmos para gerar parsers top-down de gramáticas ambíguas (e também com recursão à esquerda).

Rafael Dueire Lins
Centro de Informática UFPE

Predictive parsing

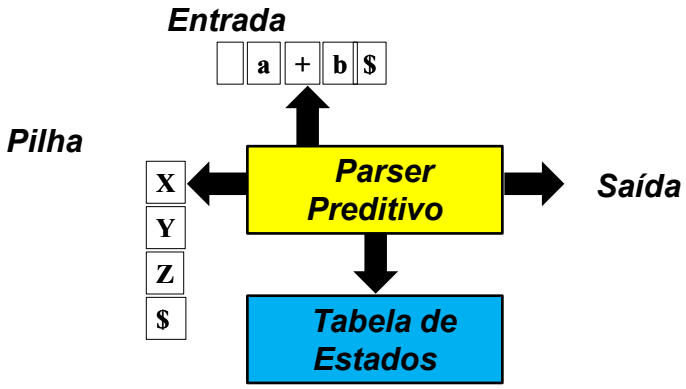
- É um parser **top-down**
- É um parser que não requer *backtracking*
  - Simples de construir manualmente
  - Mas, requer modificação na gramática para tratar recursão à esquerda e ambigüidade
- Eliminação de recursão à esquerda (fatoração):

$S \rightarrow \mathbf{a}AB\mathbf{e}$
$A \rightarrow A\mathbf{bc} \mid \mathbf{b}$
$B \rightarrow \mathbf{d}$

$S \rightarrow \mathbf{a}AB\mathbf{e}$
$A \rightarrow \mathbf{b}K$
$K \rightarrow \mathbf{bc}K \mid \in$
$B \rightarrow \mathbf{d}$

Rafael Dueire Lins
Centro de Informática UFPE

Non-Recursive Predictive Parsing



Rafael Dueire Lins
Centro de Informática UFPE

Bottom-up parsing

- Usa as produções de forma invertida até chegar no símbolo inicial.
- O *string* **abbcbcde** é caracterizado pela sequência de derivações abaixo.

$S \rightarrow aABe$
$A \rightarrow Abc \mid b$
$B \rightarrow d$

abbcbcde
→ **a*A*bcbcde**
→ **a*A*bcde**
→ **a*A*de**
→ **a*A*B*e***
→ *S*

Rafael Dueire Lins
Centro de Informática UFPE

Shift-reduce parsers

- Usa uma pilha e uma tabela
  - Pilha: armazena tokens para salvar contexto
  - Tabela: determina as opções atuais de ação
- Possíveis ações:
  - Shift (push): coloca tokens na pilha
  - Reduce: determina uma produção
  - Accept: finaliza. reconhece string.
  - Error: Nenhuma ação é possível

Shift-Reduce Parsers

E→E+E|E*E|(E)|id

Pilha	Entrada	Ação
$	id_1 + id_2 * id_3 $	shift
$$id_1$	+ id_2 * id_3 $	reduce by E→id
$E	+ id_2 * id_3 $	shift
$E +	id_2 * id_3 $	shift
$E + id_2	* id_3 $	reduce by E→id
$E + E	* id_3 $	shift
$E + E *	id_3 $	shift
$E + E * id_3	$	reduce by E→id
$E + E * E	$	reduce by E→E*E
$E + E	$	reduce by E→E+E
E	$	accept

Conflitos Shift-Reduce

- Há GLC onde parsers shift-reduce não podem ser usados.
- Surgem conflitos e o parser não consegue decidir se shift ou reduce.
- Tais gramáticas são ditas não LR.
- Gramáticas ambíguas são não-LR.
- A gramática com o else pendente não pode ser processada por parser LR.

Parsers LR

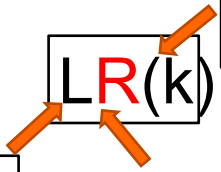
- Eficientes
- Reconhecem quase todos os construtores de linguagens de programação expressos por GLC.
- O mais geral sem *backtracking*.
- A classe de gramáticas tratadas é um superconjunto próprio dos parsers preditivos.

Parsers LR

- Pode detectar erros sintáticos o mais cedo possível.
- Difíceis de serem gerados manualmente!!!
- Facilmente automatizáveis a partir de GLC!
- Técnicas de construção:
  - SLR (Simple LR) - +Fácil implementar, - Poder
  - LR Canonico - +Custo implementação, +Poder
  - LALR (Look-ahead) – intermediário Custo, Poder

Classificação de *parsers*

Bottom-up parsers operam desta forma!

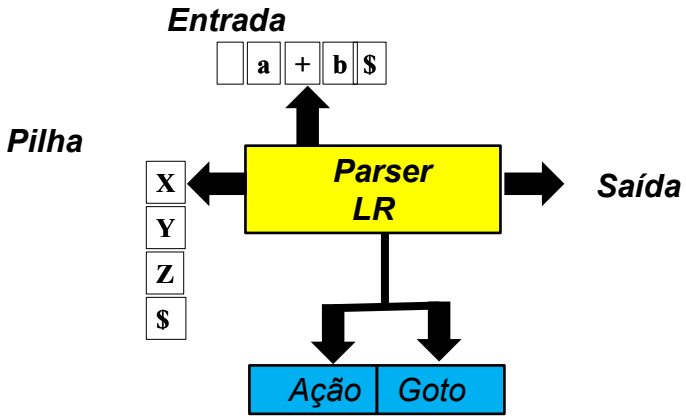


Número de tokens de *lookahead* lidos porém não consumidos

Lê entrada da esquerda (L) para direita

Procura a derivação mais à direita

Non-Recursive Predictive Parsing



Rafael Dueire Lins
Centro de Informática UFPE

Usando Gramáticas Ambíguas

Precedência e associatividade dos operadores

A gramática ambígua:

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

Pode ser re-escrita como a não-ambígua:

$E \rightarrow E+T \mid T$

$T \rightarrow T*F \mid F$

$F \rightarrow (E)|id$

Onde + tem menor precedência que *

Rafael Dueire Lins
Centro de Informática UFPE

Usando Gramáticas Ambíguas

Ambiguidade do "*Dangling-else*"

A gramática ambígua:

stmt → **if** *expr* **then** *stmt*
| **if** *expr* **then** *stmt* **else** *stmt*
| **other**

Deve-se sempre dar prioridade ao *shift* juntando o **else** ao primeiro **if_then**.

Rafael Dueire Lins
Centro de Informática UFPE

Usando Gramáticas Ambíguas

Produções Especiais

A gramática ambígua:

E→E^E_E | E_E | E^E | {E} | c

A^B_C → A^{B}_{C} A^{B_C}

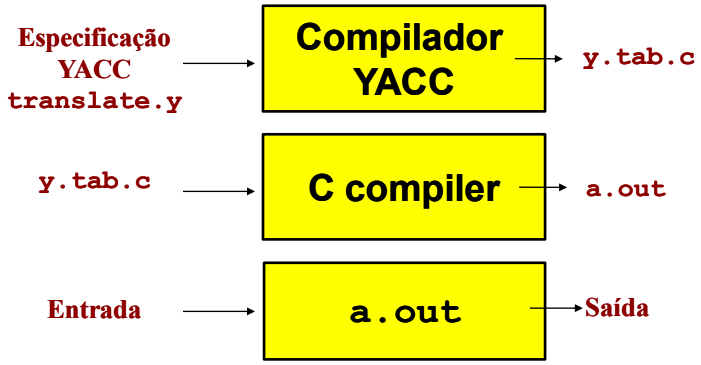
Não pode ser feita não-ambígua com precedência e associatividade.

Uma nova produção deve ser incorporada:

A^B_C → A^{B}_{C}

Rafael Dueire Lins
Centro de Informática UFPE

YACC
Yet Another Compiler Compiler



Rafael Dueire Lins
Centro de Informática UFPE

S.C.Johnson - 1970

YACC
Yet Another Compiler Compiler

- Um programa YACC tem três partes:
  - Declarações
  - %%
  - Regras de tradução
  - %%
  - Rotinas auxiliares
- A Gramática de expressões aritméticas:

  E→E+T | T

  T→T*F| F

  F→(E)|digit

Rafael Dueire Lins
Centro de Informática UFPE

Calculadora em YACC

```
%{
#include <ctype.h>
%}
%token DIGIT
%%
line    : expr '\n'          { printf("%d\n", $1); }
        ;
expr    : expr '+' term      { $$ = $1 + $3; }
        | term
        ;
term    : term '*' factor    { $$ = $1 * $3; }
        | factor
        ;
factor  : '(' expr ')'       { $$ = $2; }
        | DIGIT
        ;
%%
yylex() {
    int c;
    c = getchar();
    if (isdigit(c)) {
        yylval = c-'0';
        return DIGIT;
    }
    return c;
}
```

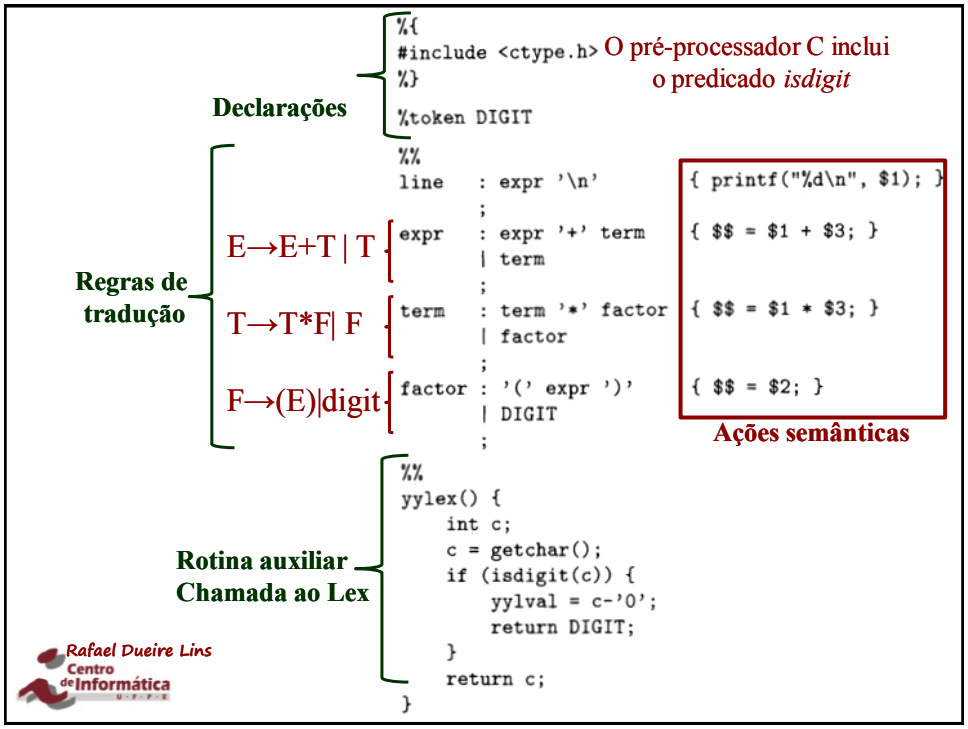
O pré-processador C inclui o predicado *isdigit*

E→E+T | T

T→T*F| F

F→(E)|digit

Rafael Dueire Lins
Centro de Informática UFPE



Rafael Dueire Lins
Centro de Informática UFPE

Conflitos no YACC

Yet Another Compiler Compiler

- Conflitos *reduce/reduce* são resolvidos pela ordem em que aparecem as produções.
- Conflitos *shift/reduce* sempre resolvidos como *shift* ("*dangling-else*").
- %left ´+´ ´-´ (mesma precedência e associativo à esquerda)
- %right ´^´ (associativo à direita)
- %nonassoc ´<´ (não-associativo)
- Tokens tem precedência crescente na ordem em que aparecem.

Comparação Top-down e Bottom-up

- Em geral, bottom-up é mais poderoso
  - Coloca menos restrições na gramática
- Bem mais trabalhoso de se escrever e manter manualmente. Porém...
  - Yacc e Bison geram parser de gramática LALR(1), um subconjunto LR(1)