Compiladores

Análise Sintática

Bruno Lopes

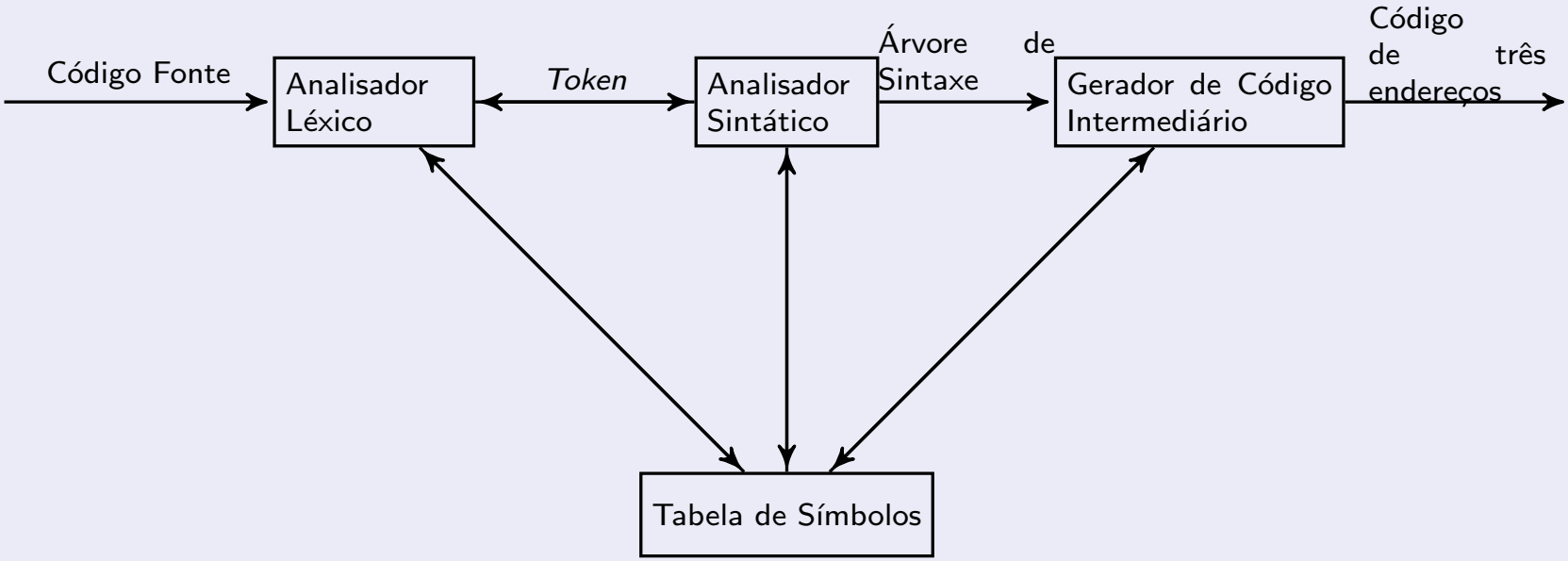
Instituto de Computação
uff

Front-end

- Lida com a linguagem de entrada
- Teste de pertinência: código fonte $\in$ linguagem fonte?
- Programa está bem formado?
  - Sintaticamente?
  - Semanticamente?
- Cria um código intermediário

Instituto de Computação

Front-end



Instituto de Computação

Parser

Construção

Converter uma especificação de linguagem em código.

1. Gramática Livre de Contexto
2. Autômato de Pilha
3. Transformar em código

Instituto de Computação

Análise Sintática

Determina a estrutura sintática

Verifica se a entrada está bem formada

Entrada

Sequência de *tokens*

Árvore sintática

Representação intermediária

Instituto de Computação

Análise Sintática

Determina a estrutura sintática

Verifica se a entrada está bem formada

Entrada

Sequência de *tokens*

Árvore sintática

Representação intermediária

Instituto de Computação

Análise Sintática

Determina a estrutura sintática

Verifica se a entrada está bem formada

Entrada

Sequência de *tokens*

Árvore sintática

Representação intermediária

Instituto de Computação

Análise Sintática

Requisitos

- Modelo matemático da sintaxe
- Algoritmo

Por que não usar ERs?

$a^n b^n$

Instituto de Computação

Análise Sintática

Requisitos

- Modelo matemático da sintaxe
- Algoritmo

Por que não usar ERs?

$a^n b^n$

Instituto de Computação

Análise Sintática

- Sintaxe definida por regras
- Regras são recursivas
- Gramáticas Livres de Contexto
- BNF

Instituto de Computação

Gramáticas Livres de Contexto

$G = \langle V, T, P, S \rangle$

- V Não-terminais
- T Terminais
- P Conjunto de regras de produção
- S Símbolo inicial

Gramática X Cadeia X Derivação X *Parsing*

Instituto de Computação

Gramáticas Livres de Contexto

$G = \langle V, T, P, S \rangle$

- V Não-terminais
- T Terminais
- P Conjunto de regras de produção
- S Símbolo inicial

Gramática X Cadeia X Derivação X *Parsing*

Instituto de Computação

Gramáticas e derivações

- Recurssão
- BNF X EBNF
- Árvore sintática
- Árvore sintática abstrata
- Árvore sintática mais à direita
- Árvore sintática mais à esquerda
- Gramática ambígua

Gramática ambígua

Compiladores diferentes podem fazer interpretações diferentes!

Instituto de Computação

Gramáticas e derivações

- Recurssão
- BNF X EBNF
- Árvore sintática
- Árvore sintática abstrata
- Árvore sintática mais à direita
- Árvore sintática mais à esquerda
- Gramática ambígua

Gramática ambígua

Compiladores diferentes podem fazer interpretações diferentes!

Instituto de Computação

Gramática ambígua

Como detectar e remover ambiguidade?

- Detectar: indecidível
- Remover: não existe algoritmo

Expressões?
Else pendente?
Aninhamentos?
Como se implementa?

Instituto de Computação

Gramática ambígua

Como detectar e remover ambiguidade?

- Detectar: indecidível
- Remover: não existe algoritmo

Expressões?
Else pendente?
Aninhamentos?
Como se implementa?

Instituto de Computação

Gramática ambígua

Como detectar e remover ambiguidade?

Detectar: indecidível

Remover: não existe algoritmo

Expressões?
Else pendente?
Aninhamentos?
Como se implementa?

Instituto de Computação

Linguagem Tiny

```
program -> stmt-sequence
stmt-sequence -> stmt-sequence ; statement | statement
statement -> if-stmt | repeat-stmt | assign-stmt | read-stmt
if-stmt -> if exp then stmt-sequence end
         | if exp then stmt-sequence else stmt-sequence end
repeat-stmt -> repeat stmt-sequence until exp
assign-stmt -> identifier := exp
read-stmt -> read identifier
write-stmt -> write exp | write-stmt
exp -> simple-exp comp-op simple-exp | simple-exp
comp-op -> < | =
simple-exp -> simple-exp addop term | term
addop -> + | -
term -> term mulop factor | factor
mulop -> * | /
factor -> ( exp ) | number | identifier
```

Instituto de Computação

Linguagem Tiny

```
{ Fatorial na linguagem Tiny }
read x;
if 0 < x then
  fact := 1;
  repeat
    fact := fact * x;
    x := x - 1
  until x = 0
  write fact
end
```

Instituto de Computação

Linguagem Tiny

```
read u;
read v; { input: two integers }
if v = 0 then v := 0
else
  repeat
    temp := v;
    v := u - u /v * v;
    u := temp
  until v = 0
end;
write u;
```

Instituto de Computação

Construção de *parsers*

- *Top-down*
- *Bottom-up*

Instituto de Computação

Top-down

1. Construa o nó raiz da árvore
2. Repita, até que o nível inferior da árvore corresponda a string de entrada
    1. Em um nó da árvore rotulado com um símbolo não terminal A, selecione uma produção com A no lado esquerdo, e construa uma sub-árvore para cada símbolo no lado direito da produção
    2. Quando um símbolo terminal é adicionado na borda da árvore e ele não corresponde a entrada, retroceda
    3. Encontre o próximo nó a ser expandido

Instituto de Computação

```
0. goal -> exp
1. exp -> exp + term
2.      | exp - term
3.      | term
4. term ->   term * factor
5.             | term / factor
6.           | factor
7.  factor -> ( exp )
8.           | number
9.           | id
```

Entrada

```
x - 2 * y
```

Instituto de Computação

Top-down: implementação

- Segue a sintaxe da EBNF
- Entrada: Vetor com tokens
- Mantém: índice para o token atual

Instituto de Computação

Top-down: implementação

Terminais

- Testa token atual
- Caso token seja compatível, avança; senão falha

Instituto de Computação

Top-down: implementação

Sequência

- Testa cada termo
- Falha no primeiro incompatível

Instituto de Computação

Top-down: implementação

Produções alternativas

- Guarda o índice atual
- Testa a primeira opção
- Se falhar, volta para o índice guardado e testa a segunda opção; assim por diante

Instituto de Computação

Top-down: implementação

Repetição

- Repete os passos até algum termo falhar

Instituto de Computação

Top-down: implementação

Não terminal

- Procedimento separado

Instituto de Computação

```
#include <stdio.h>
#include <stdlib.h>

char token;
```

Instituto de Computação

```
void error(void){
  fprintf(stderr, "Error\n");
exit(1);
}
void match(char expectedToken) {
  if (token == expectedToken) token = getchar();
  else error();
}
```

Instituto de Computação

```
int exp(void){
  int temp = term();
  while ((token == "+") || (token == "-"))
    switch (token){
      case '+': match('+'); temp+= term;  break;
    case '-': match('-'); temp-= term;  break;
    }
  return temp;
}
```

Instituto de Computação

```
int term(void){
  int temp = factor();
  while (token == "*"){
    match('*');
    temp*= factor;
  }
  return temp;
}
```

Instituto de Computação

```
int factor(void){
  int temp;
  if (token == '(') {
    match('('); temp = exp();  match(')'); }
  else if (isdigit(token)){
    ungetc(token, stdin); scanf("%d", &temp);
    token = getchar();
  }
  else error();
  return temp;
}
```

Instituto de Computação

```
int main(){
  int result;
  token = getchar();
  result = exp();
  if (token == '\n')
    printf("Result = %d\n", result);
  else error();
  return 0;
}
```

Instituto de Computação

Analisador descendente recursivo

- Escolhas de produções erradas levam a desperdício de tempo!
- Parsers descendentes não conseguem manipular gramáticas recursivas à esquerda
- Recursões a serem manipuladas por parsers descendentes devem ser recursivas à direita (como converter?)

Instituto de Computação

```
exp -> exp + term | exp - term |  term
se torna
exp -> term exp' | term
exp'-> + term exp' | - term exp'
```

Instituto de Computação