Compiladores

Análise Léxica

Bruno Lopes

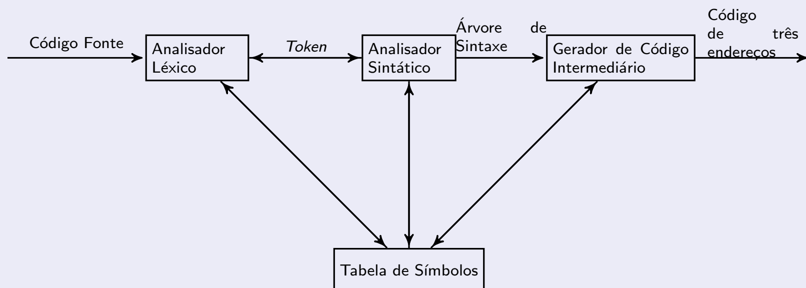
Instituto de Computação UFF

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 uff

Front-end



Scanner

Construção

Converter uma especificação de linguagem em código.

1. Escrever expressão regular para representar a linguagem
2. Construir NFA para reconhecer a linguagem, a partir das ERs
3. Transformar NFA em DFA
4. Minimizar DFA
5. Transformar em código

Instituto de Computação uff

Código simples

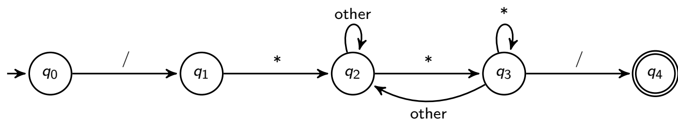
```
//estado 1
if (the next character is a letter)
  advance the input;
  // estado 2
  while (the next character is a letter or digit)
    // permanece no estado 2
    advance the input
    // vai para o estado 3, sem avançar a entrada
    accept;
else
  error;
```

Instituto de Computação uff

Autômatos Finitos: implementação

- Como traduzir qualquer DFA?
- Como ficará a complexidade do código se existirem muitos estados em caminhos arbitrários?

Instituto de Computação



Instituto de Computação uff

```
if (next character is "/")
  advance the input;
  if (next character is "*"
    advance the input;
    done =  false;
    while (! done )
      while (next character is not "*")
        advance the input;
      advance the input;
      while (next character is "*")
advance the input;
      if (next character is "/")
        done = true;
      advance the input
    accept;
```

Instituto de Computação

É possível melhorar

- Usar uma variável para manter o estado corrente
- Usar um case dentro do loop
  - um comando para testar o estado corrente
  - um comando para testar o caracter de entrada

Instituto de Computação uff

```
state = 1
while (state = 1 or state = 2)
  case state:
    1: case input character
      letter: advance the input; state = 2
      else state = ... (erro, p.ex.)
    2: case input character
      letter or digit: advance the input; state = 2
      else state = 3
if state = 3; accept
```

Instituto de Computação

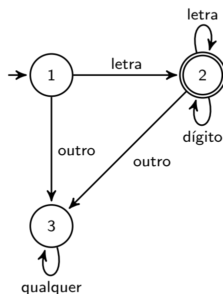
DFA como uma tabela de transições

	caracteres no alfabeto	
estados	estados representando transições com o caracter	aceitação?

Instituto de Computação uff

Tabela de transição

	letra	dígito	outro	aceitação
1	2	—	3	não
2	2	2	3	sim
3	3	3	3	não



Instituto de Computação uff

Código dirigido à tabela

```
state = 1
ch = next input character
while (not Accept[state] and not error(state))
  newState = T[state, ch];
  if (Advance[state, ch])
    ch = next input character
    state = newState;
if (Accept[state])
  accept;
```

Instituto de Computação uff

Vantagens

- código reduzido
- fácil de manter
- genérico

Desvantagens

Tabela pode ficar muito grande

Instituto de Computação uff

Linguagem Tiny

- Variáveis inteiras
- Declaração a partir da atribuição de valores
- Controle com if e repeat
- if termina com end e tem um else opcional
- Comentários entre chaves
- Expressões aritméticas e booleanas

Instituto de Computação uff

Scanner para a Tiny

Tokens

- palavras reservadas: if, then, else, end, repeat, until, read, write
- Símbolos especiais: +, -, *, /, =, ¡ , (,), ;, :=
- números: 1 ou mais dígitos
- identificadores: 1 ou mais letras

Scanner para a Tiny

- Comentários entre chaves
- espaços em branco são ignorados
- Princípio de reconhecimento da string mais longa
- Palavras reservadas tratadas como identificador (categoria sintática?)
- ERs simples

DFA?

Instituto de Computação

Scanner para a Tiny

- Comentários entre chaves
- espaços em branco são ignorados
- Princípio de reconhecimento da string mais longa
- Palavras reservadas tratadas como identificador (categoria sintática?)
- ERs simples

DFA?

Instituto de Computação uff

```
{Exemplo de programa em Tiny - Fatorial}
read x; {inteiro de entrada}
if x > 0 then {expressão booleana}
fact := 1;
repeat
fact := fact * x;
x := x + 1;
until x = 0;
write fact;
end
```

Instituto de Computação uff

Geradores de *Scanner*

- Lex and Flex seguem o caminho usual
- Algoritmos são bem conhecidos e entendidos
- Problema chave: interface para o parse

Instituto de Computação uff

Lex

- Gerador de analisador léxico
- Entrada: ERs
- Saída: lex.yy.c

Instituto de Computação uff

Formato de entrada

```
{definitions}
%%
{rules}
%%
{auxiliary routines}
```

Instituto de Computação uff

Lex com Tiny

- arquivo de entrada para o lex com as especificações da linguagem
- executa lex com esse arquivo de entrada: produz lexyy.c com o analisador léxico de Tiny
- compila lexyy.c
- executa com um programa escrito em Tiny
- Devolve os tokens do programa

Instituto de Computação uff

JFlex

Gerador de analisador léxico Java, inspirado no Lex

```
código Java (entra no arquivo de classe do scanner,  mas fora
%%
opções e declarações (como o scanner é gerado)
%%
regras do scanner  (ERs e ações)
```

Instituto de Computação uff

Regras e ações

Formato

- EXPRESSAOREGULAR CÓDIGO JAVA
- O código é inserido no método do scanner que fornece um token
- para pegar o valor do lexema, use yytext()
- Deve incluir um return
- <<EOF>> indica fim do arquivo

```
[0-9]+ { return new Token(Token.NUM, yytext(), yyline); }
```

Instituto de Computação uff

Regras e ações

Ao consumir a entrada, o scanner determina a porção regular mais longa correspondente

- Se existir mais de uma, ele escolhe a que foi definida primeiro
- Se não existir nenhuma, dá uma mensagem de erro

Instituto de Computação

Estados

Ao consumir a entrada, o *scanner* determina a porção regular mais longa correspondente

- Usados para restringir o conjunto de expressões regulares que podem ser associadas a uma entrada
- Podem ser exclusivos ou não
- Expressão regular será correspondente a uma entrada quando
  - conjunto de estados léxicos inclui o estado ativo do *scanner*
  - conjunto de estados está vazio e o estado léxico não é exclusivo

Instituto de Computação uff

Estados

- Estado corrente ativo pode ser alterado com o método yybegin()
- Estado inicial é o YYINITIAL (inclusivo), default
- Uma lista de estados pode ser associada com uma ER

Instituto de Computação

Exemplo

```
%states A, B
%xstates C
%%
// não tem estado listado: match com todos os estados,
// exceto com C (exclusivo)
expr1                   { yybegin(A); action }
<YYINITIAL, A> expr2    { action }
<A> {
  expr3                 { action }
  <B,C> expr4           { action }
```

Instituto de Computação uff

Uso de estados: regras léxicas distintas para entradas iguais

- HTML: regras léxicas para elementos que aparecem dentro ou fora de uma tag

Entradas que não podem ser reconhecidas por expressões regulares

- aninhamento de comentários
- /* começa o primeiro /* aqui tem outro */ termina o primeiro */

Instituto de Computação uff

Exemplo

```
%state TAG        // seção de declarações
%%
<YYINITIAL> {//seção de regras
[<] { yybegin(TAG); return new Token('<'); }
[^<]+ { return new Token(TOKEN.PALAVRA, yytext()); }
}
<TAG> {
[>] { yybegin(YYINITIAL); return new Token('>');
... outras regras ...
}
```

Instituto de Computação uff

- Como estados poderiam ser usados para controlar comentários aninhados?
- Implemente a entrada para o JFlex e o que mais for necessário para reconhecer uma linguagem de calculadora. Além de números inteiros e com ponto flutuante, os operadores permitidos são (,),+,-,*,/,//,**. Qualquer outro símbolo deve gerar um erro

Instituto de Computação