Análise léxica

Professor Eraldo Pereira Marinho

Análise léxica - varredura

Varredura (scanning) é o processo de leitura sequencial de caracteres, da esquerda para direita, a partir de um arquivo texto, geralmente na codificação ASCII

O processo requer um buffer, geralmente abstraído pelo ponteiro da estrutura FILE do arquivo de cabeçalho <stdio.h>; tal buffer permite o fluxo contínuo, sem interrupções em tempo significativo, de caracteres que serão lidos pela rotina que emula um autômato finito não determinístico (AFN)

A varredura é feita através de sucessivos chamados à função int getc(FILE *);

Transições épsilon, ou possíveis retrocessos (backtracking) são emulados através do uso da função void ungetc(int, FILE *);

Varredura

- Em princípio, é possível simular em C qualquer autômato finito. Contudo, para nossa conveniência e fácil portabilidade de projeto, é interessante se adotar certas convenções de design de analisadores léxicos

1. Sempre que possível, elimine transições múltiplas, tornando as transições como funções parciais – se isso não for possível para fins práticos, utilize backtracking;
2. Transições épsilon são emuladas através do uso consecutivo de getc e ungetc;
3. Os analisadores não podem possuir estado de rejeição ou de transição vazia, Ø; ao invés, considera-se um estado de aceitação confirmativo ou um estado de aceitação da palavra vazia – isso ficará claro com exemplos.

Analisador léxico como um AFN

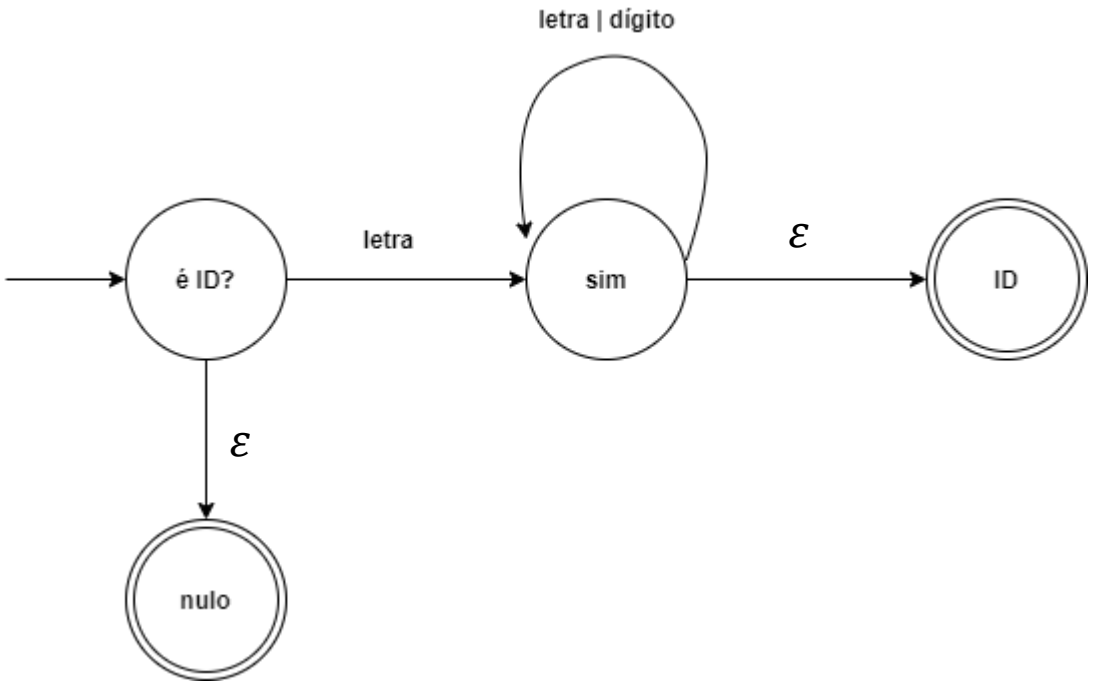

Diagrama de transição para um reconhecedor de identificadores Pascal (*case sensitive*); o estado de aceitação rotulado como “nulo” corresponde ao não reconhecimento de um padrão sem prejuízo de símbolo de entrada, através de uma transição épsilon, o que só é possível pela ação sequencial de getc e ungetc

Analisador léxico como um AFN codificado em C

```
1:  int isID (FILE *tape) {
2:          int ch = getc (tape);
3:          if ( isalpha (ch) ) {
4:                  while ( isalnum (ch = getc (tape) ) );
5:                  ungetc (ch, tape);
6:                  return ID;
7:          }
8:          ungetc (ch, tape);
9:          return 0;
10: }
```

Analisador léxico - exemplo

Façamos um simples teste para ler a sequência “ax2yz4 + 3”

O programa começa com o cursor de caracteres do buffer (FILE *tape) sobre o “a”, ou seja “ax2yz4 + 3”. Neste caso, a variável ch será carregada com o simples ASCII, ch = ‘a’, e o cursor avança para a próxima posição no buffer: “ax2yz4 + 3”. Dentro do while-loop, todos os demais símbolos serão lidos em ch, até o branco (código ASCII 32), ch = ‘ ‘. Nesta última posição, a configuração instantânea do buffer seria “ax2yz4 + 3”, com o cursor posicionado no ‘+’. Contudo, o branco não pertence ao padrão ID. Logo, este último será devolvido ao buffer, através de ungetc, ficando “ax2yz4_+ 3”, ou seja, com o cursor no branco subsequente ao 4.

Portanto, funcionou. Se o cursor estivesse sobre qualquer coisa que não fosse o início de um identificador, então a máquina iria diretamente para a linha de código “return 0;”

Ignorando espaços

Como ignorar os espaços?

Ao invés de sairmos à procura de um cartão de referência de código ASCII, para encontrarmos os valores numéricos de espaços (branco, quebra de linha etc.), simplesmente utilizamos a função predicado int isspace(FILE *); que fica no arquivo de inclusão <ctype.h>, bem como as outras isalpha e isalnum.

Nem precisamos desenhar tal máquina de Turing para este propósito e vamos diretamente ao código C:

```
1: void skipspaces (FILE *tape) {
2:         int ch;
3:         while ( isspace (ch = getc(tape)) );
4:         ungetc (ch, tape);
5: }
```

A função gettoken

Como já previamente comentado, esta é a interface entre o lexer e o parser. Podemos dizer que int gettoken(FILE *source); é o front-end do analisador léxico (lexer).

Sua implementação é relativamente simples e pode ser codificada como no exemplo seguinte:

```
1: int gettoken (FILE *source) {
2:          int token;
3:          skipspaces (source);
4:          if ( (token = isID (source)) ) return token;
5:          if ( (token = isNUM (source)) ) return token;
6:          ...
7:          token = getc (source);
8:          return token;
9: }
```

Questionário

1. Que é o processo de varredura e qual sua importância na análise léxica?
2. Quais exigências ou restrições são feitas quanto à maneira de implementar AFNs para funcionarem como analisadores léxicos?
3. Como se simula a transição épsilon na codificação C?
4. Modifique o comando skipspaces para que este contabilize o número de linhas avistados. Defina uma variável global int linenum = 1;
5. Qual a necessidade da linha 7 no template da função gettoken?

Exercícios

1. Apresente uma expressão regular para definir números octais. O número 0 é octal? Justifique sua resposta.
2. Implemente uma função predicado, denominada int isOCT(FILE *tape), que retorne a constante OCT (use #define) se o padrão lido sequencialmente de tape for octal e retorne 0 (zero) caso contrário.
3. Apresente um diagrama de estados para abstrair a função do Exercício 2. De que modo as transições épsilon são emuladas pelo analisador léxico do referido exercício?
4. Apresente uma expressão regular para definir números hexadecimais.
5. Análogo ao Exercício 2, implemente uma função predicado, int isHEX(FILE *tape), para decidir se uma cadeia, oriunda de tape, é ou não um hexadecimal.
6. Análogo ao Exercício 3, apresente um diagrama de estados para reconhecer hexadecimais, da forma que o analisador do Exercício 5 procede.

Exercícios

7. O padrão de ponto fixo é um decimal inteiro, seguindo de ponto decimal e seguido de zero ou mais dígitos. Por outro lado, um ponto fixo pode iniciar com um ponto decimal seguido de 1 ou mais dígitos. Por exemplo, 3., 3.14 e .14 são pontos fixos. Escreva uma expressão regular para denotar a representação de ponto fixo.

8. Apresente um diagrama de estados para ilustrar o reconhecimento do padrão ponto fixo.

9. Implemente uma função predicado C para reconhecer o padrão ponto fixo, retornando um inteiro positivo se reconhecer e retornando zero caso contrário. Neste último caso, a função não pode consumir símbolos de entrada.

10. O padrão ponto flutuante é definido como uma cadeia de ponto fixo, podendo vir seguida de notação exponencial (notação científica), ou pode ser um inteiro seguido de notação exponencial. A notação exponencial é definida como $EE = [eE][' + '' - ']?\,[0 - 9]^{+}$. Escreva uma expressão regular, podendo recorrer a macros, para denotar números em ponto flutuante.

11. Implemente uma função predicado que retorne DEC (por exemplo, #define DEC 1024) se a cadeia de entrada for de um inteiro decimal (sem sinal), retorne FLT (por exemplo, #define DEC 1025) se for ponto flutuante, e retorne zero, sem consumo de fita de entrada, se não for nenhum desses padrões. Dica: crie uma função predicado para verificar se o sufixo está em notação científica.

Exercícios

12. Escreva uma expressão regular, podendo recorrer a macros, para denotar números em algarismos romanos. Assuma como alfabeto o conjunto $\{i, v, x, l, c, d, m\}$. Por exemplo, 2022 é escrito como $mmxxii$. Ignore a possibilidade de ter representação de números superiores a 3999.