Primeira lista de exercícios (Compiladores) - Análise Léxica

Davi Ventura Cardoso Perdigão - 82148

1- Descreva o que são os modelos de análise e síntese relacionados ao processo de compilação? Cite as fases que compõe cada um dos modelos.

– Análise: quebrar o programa fonte em diversas construções, criando uma representação intermediária. Analisa a corretude do programa fonte.

– Síntese: gerar o programa destino a partir de uma representação intermediária.

A **compilação** é acompanhada de diversos processos auxiliares até que se gere um programa executável, sendo eles: **pré-processador** *(includes)* **-> compilador** *(conversão Assembly)* **-> montador** *(Assembly em binário)* **-> linkeditor** *(estrutura os binários gerados até então)* **-> carregador** *(carrega o código pra memória e os endereços relativos passam a ser absolutos)***.**

2- Cite e descreva 2 processos que são de utilização facultativa no projeto de um compilador?

Alguns compiladores incluem a geração de uma **representação intermediária para o programa fonte**. Trata-se de um código para uma máquina abstrata e deve ser fácil de produzir e traduzir no programa objeto.

Outra tarefa facultativa seria o **otimizador de código**. O processo de otimização de código consiste em melhorar o código intermediário de tal forma que o programa objeto resultante seja mais rápido em tempo de execução.

3- Descreva de forma sucinta o que significam os termos abaixo:

- **Análise Léxica:** É a primeira fase de um compilador, responsável pela leitura da entrada do fluxo de caracteres convertendo-os em tokens a ser analisado pelo analisador sintático (parser). Conhecidos como scanners, entregam juntamente com cada token o seu lexema e os atributos associados.
- **Assembly:** Assembly é uma linguagem de montagem (baixo nível). Ou seja, diferente da maioria das outras linguagens, que são compiladas e/ou interpretadas, programar em Assembly é escrever um código que é diretamente entendido pelo hardware, como os microprocessadores e microcontroladores. Assembly é erróneo e pouco produtivo.
- **Código Relativo:** O compilador usa um “montador” para produzir um código de máquina relativo (realocável) que é passado direto ao linkeditor.
- **Código Absoluto:** determina sua posição usando um ponto de referência.
- **Lexeme:** Sequência de caracteres que compõe um único token. Um padrão reconhece as cadeias de tal conjunto, ou seja, reconhece os lexemas que pertencem ao padrão de um token.
- **Token:** Os tokens usualmente são conhecidos pelo seu lexema e atributos adicionais. Um mesmo token pode ser produzido por várias cadeias de entrada. Tal conjunto de cadeias é descrito por uma regra padrão, associada a tais tokens. O padrão reconhece as cadeias de tal conjunto, ou seja, reconhece os lexemas que pertencem ao padrão de um token.
- **Parser:** analisador sintático ou gramatical.
- **Montador:** converte assembly para linguagem de máquina, na proporção de uma instrução para uma.
- **Loader:** altera os endereços relativos para absolutos, carregando o programa e os dados na memória.

- **Tabela de Símbolos:** uma estrutura de dados, utilizada em compiladores para o armazenamento de informações de identificadores, tais como constantes, funções, variáveis e tipos de dados. É utilizada em quase todas as fases de compilação.

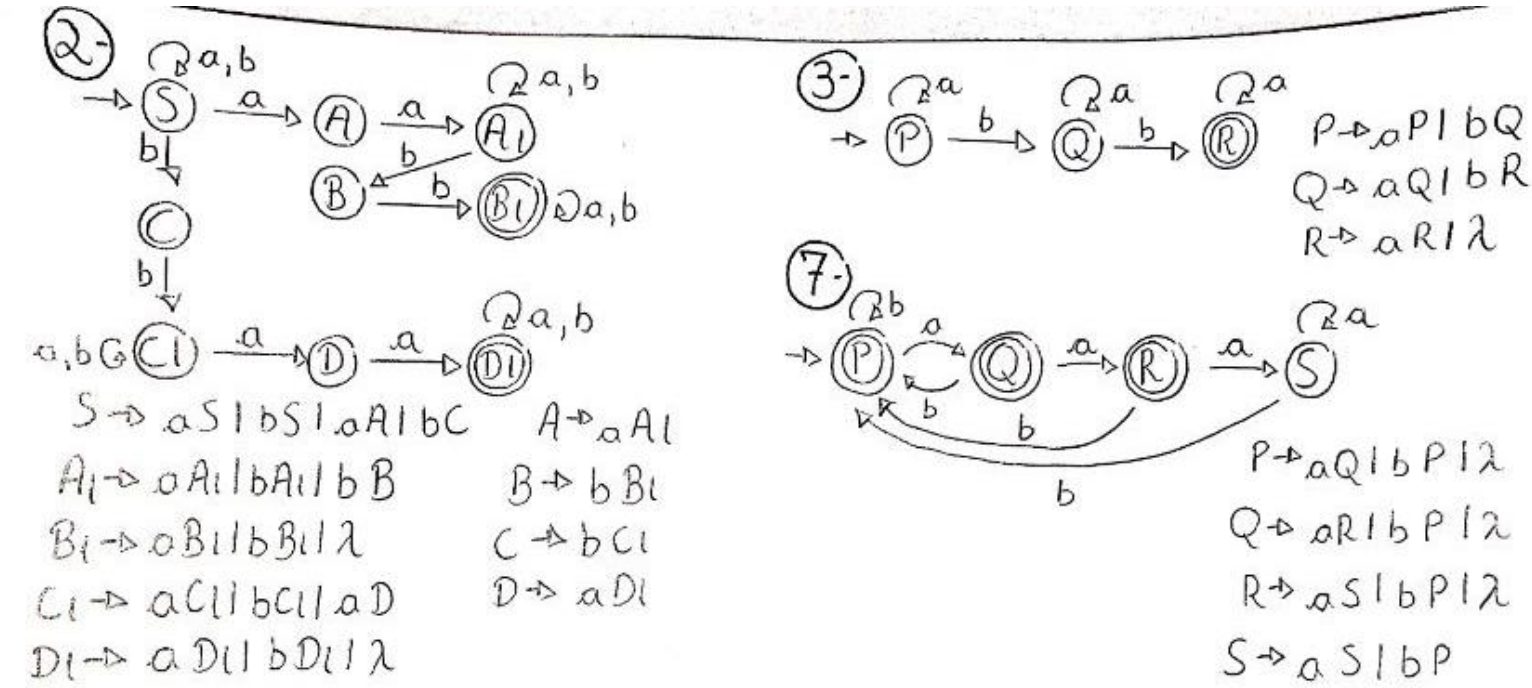
4 - Cite 2 ou mais utilizações para as técnicas aprendidas na disciplina compiladores.

- Tradução de linguagens
- Interpretação de texto e escritas (ex: html, word, etc.)
- Procura de palavras ou frases por motores de busca

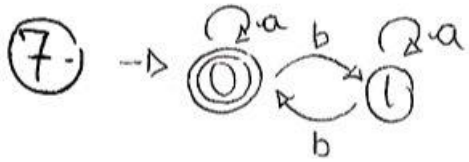
5 - Defina expressões regulares para:

1. Strings sobre {a,b} contendo aa ou bb b* aa (b|a)* | a* bb (b|a)
2. Strings sobre {a,b} contendo aa e bb (a|b)* aa (a|b)* bb (a|b)* | (a|b)* bb (a|b)* aa (a|b)*
3. Strings sobre {a,b} contendo exatamente dois b's a* b a* b a*
4. Strings sobre {a,b} contendo pelo menos dois b's (a|b)* b (a|b)* b (a|b)*
5. Strings sobre {a,b} com número par de b's λ | a* bb (a|bb)*
6. Strings sobre {a,b} que começam com ba, contém aa terminam com ab ba b* aa (a|b)* ab
7. Strings sobre {a,b} que não terminam com aaa λ | b* a b*
8. Strings sobre {a,b,c} que não contém bc λ | (c|a)*
9. Strings sobre {a,b} que não contém aa ((b* a b+ a b*) (b+))*

6 - Escreva gramáticas livres de contexto para gerar as linguagens definidas nos itens 2, 3 e 7 da questão anterior.

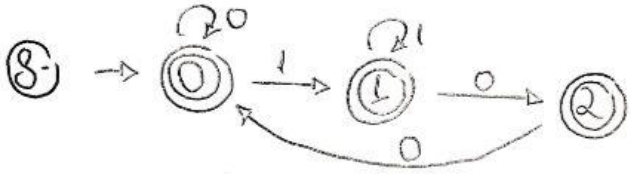


7 - Construa um autômato finito determinístico e um programa em C, para reconhecer as strings da linguagem do item 5.5.



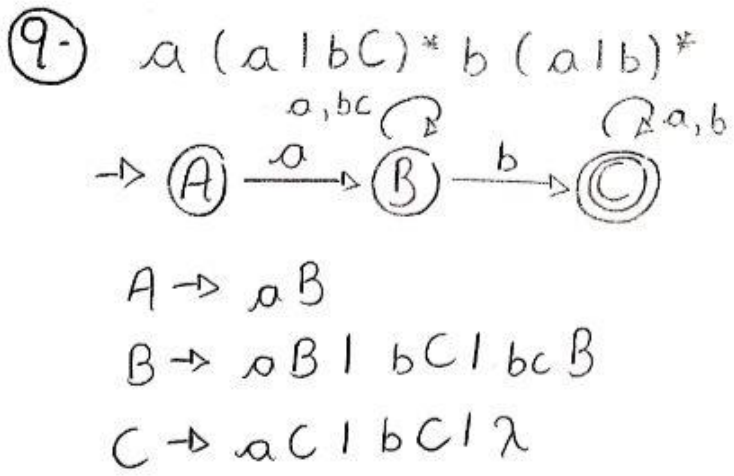

```
public class AF_sete {
public static void main(String a[]) {
   int p=a;
   int state=0;
   while (p<a.length()) {
      switch(state) {
        case 0:
           switch(a.charAt(p)) {
              case 'a':
                 state=0; break;
              case 'b':
                 state=1; break;
           }
           break;
        case 1:
           switch(a.charAt(p)) {
              case 'a':
                 state=1; break;
              case 'b':
                 state=0; break;
           }
           break;
      }
      p++;
   }
   if (state==0)
      System.out.println(”Aceita");
   else
      System.out.println("Nao aceita");
} }
```

8 - Construa um autômato finito determinístico e um programa em C, que reconheça sentenças em {0,1}*, as quais não contenham sequências do tipo 101.




```
public class AF_oito {
public static void main(String a[]) {
   int p=0;
   int state=0;
   while (p<a.length()) {
      switch(state) {
        case 0:
           switch(a.charAt(p)) {
              case '0':
                 state=0; break;
              case '1':
                 state=1; break;
           }
           break;
        case 1:
           switch(a.charAt(p)) {
              case '0':
                 state=2; break;
              case '1':
                 state=1; break;
           }
           break;
        case 2:
           switch(a.charAt(p)) {
              case '0':
                 state=0; break;
           }
           break;
      }
      p++;
   }
   if (state==0 || state == 1 || state ==2)
      System.out.println(”Aceita");
} }
```

9 - Construa um autômato finito e a gramática regular equivalente à seguinte expressão regular: a (a | bc)* b (a | b)*



10- A partir do alfabeto {[0-9] , . }. Construa uma definição regular e um autômato para reconhecer um número IP “extenso” x.x.x.x sendo que “x” pode ser qualquer número entre 0 e 999.
Ex: 127.0.0.0 , 229.120.10.1, 999.128.888.128, etc
Sugestão de definições regulares auxiliares:
d->[0-9]
e->[1-9]

[0-999] [.] [0-999] [.] [0-999] [.] [0-999]

11- Construa uma definição regular para a seguinte linguagem: valores monetários em Reais. Possuem exatamente duas casas decimais depois da vírgula e usam ponto como separador de milhar.
Exemplos: { R$2,35 ; R$1.546,98 ; R$1,00 ; R$10.000.000.000,00 }

R$[1-9]+ [,][0-9][0-9]

12- Construa uma definição regular para produzir a seguinte linguagem: todas as datas dos anos 2017, 2018 e 2019 no formato DD/MM/AAAA. Obs: nenhum destes anos é bissexto.
Ex: 28/02/2017, 31/12/2018, 06/09/2019, etc

Dias: (0[1-9]|[1-2][0-9]|3[0-1])

Mês: (0[1-9]|1[0-2])

Ano: (201[7-9])

(0[1-9]|[1-2][0-9]|3[0-1])(0[1-9]|1[0-2])(201[7-9]) *OBS: o único problema dessa resolução é que não valida fevereiro*

(?:(?:(?:[01][1-9]|2[1-8])(?:0[1-9]|1[0-2])|(?:29|30)(?:0[13-9]|1[0-2])|31(?:0[13578]|1[02]))[1-9]\d{3}|2902(?:[1-9]\d(?:0[48]|[2468][048]|[13579][26])|(?:[2468][048]|[13579][26])00)) *OBS: essa resolução valida fevereiro, mas não há limitação quanto ao ano*