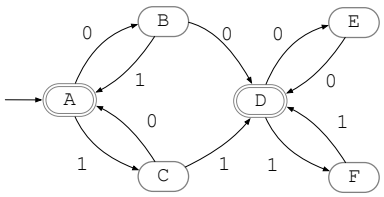
Questão 01) a)
Questão 02) e)
Questão 03) b)
Questão 04) a)
Questão 05) c)
Questão 06) e)
Questão 07) c)
Questão 08) b)
Questão 09) a)
Questão 10) e)

Questão 11)

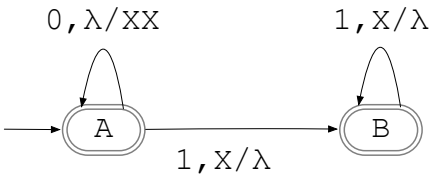
a)



b)

$A \rightarrow 0B \mid 1D$
$B \rightarrow \lambda \mid 1B$
$C \rightarrow 0B \mid 1C$
$D \rightarrow \lambda \mid 0E$
$E \rightarrow 0E \mid 1D$

c)

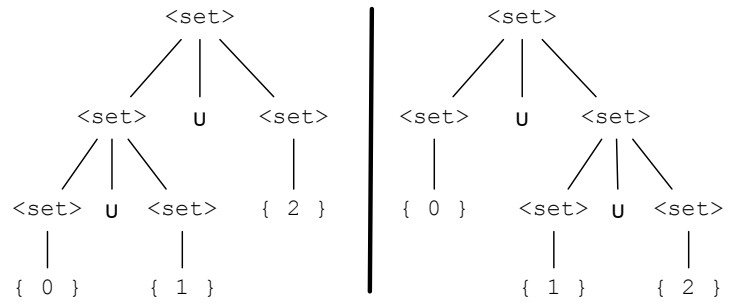


d)

$P \rightarrow 0P0 \mid 1P1 \mid 0X1 \mid 1X0$
$X \rightarrow \lambda \mid 0X \mid 1X$

Questão 12)

a) Para uma mesma sentença, pode se gerar duas árvores de derivação. Um exemplo de sentença pode ser visto a seguir, que permite provar que a gramática é ambígua: $\{0\} \cup \{1\} \cup \{2\}$



b) Um exemplo de gramática não ambígua seria:

```
<set> ::= <value>
                | ¬ <set>
                | <set> ∩ <value>
                | <set> ∪ <value>
<value> ::= { } | { <number> }
```

Questão 13)

a) Na assinatura da função não consta a anotação `throws`.

b) Uma possível implementação seria:

```
class SaqueException extends Exception {
    public SaqueException(String msg) {
        super(msg);
  }
}
```

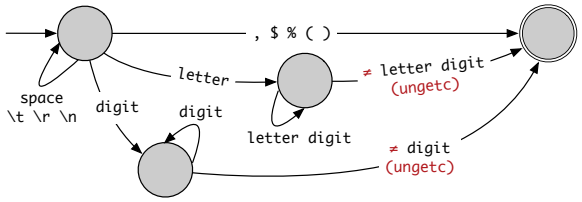
c) Uma possível reimplementação seria:

```
public void sacar(double valor) throws SaqueException {
     if (valor <= 0.0)
          throw new SaqueException("Valor inválido");
     else if (this.saldo < valor)
          throw new SaqueException("Saldo insuficiente");

     this.saldo -= valor;
}
```

Questão 14)

a) Um exemplo de diagrama para a análise léxica seria:



b) Um exemplo de gramática livre de contexto não ambígua seria:

```
<program>     ::= { <instruction> }
<instruction> ::= <opcode> [ <operand> [ ',' <operand> ] ]
<operand>     ::= <constant> | <reg> | <access>
<constant>    ::= '$' <integer>
<reg>         ::= '%' <register>
<access>      ::= [ <integer> ] '(' <reg> ')'
```

Questão 15)

a) Simplificação de expressões constantes (*constant folding*) e movimentação de código invariante de laço (*loop invariant code motion*).

b) Um exemplo de aplicação dessas transformações seria:

```
  $t0 <- pop
  $t1 <- const 0
  $t2 <- const 1
  push $t0
  call len
  $t3 <- pop
  $t4 <- const 5
%loop:
  jgt $t2, $t3, %end
  $t5 <- load $t2($t0)
  $t6 <- add $t4, $t5
  $t1 <- add $t1, $t6
  $t2 <- inc $t2
  jmp %loop
%end:
  push $t1
  ret
```