Proposta de Solução dos Exercícios do Capítulo 3

Exercício 3.1. Qual a importância do estudo da Máquina de Turing na Ciência da Computação?

O modelo da Máquina de Turing é importante para a Ciência da Computação porque através dele é possível determinar quais as funções que são computáveis e quais não são. Isto é, através de um conceito bastante simples (no qual é possível provar teoremas de forma facilitada) define-se a classe das funções calculáveis. Assim, se uma função pode ser calculada, há um modelo de Turing ou equivalente para tal função.

Exercício 3.2. Faça um quadro comparativo entre os modelos Máquina de Turing, Máquina de Post e Autômato com Duas Pilhas, destacando suas características e seus principais aspectos funcionais.

	Máquina de Turing	***Máquina de Post***	***Aut. Finito 2 Pilhas***
Memória de Trabalho	♦ *fita subdividida em células, onde cada uma contém um símbolo da palavra* ♦ *infinita a direita e finita a esquerda*	♦*variável X, do tipo fila,com comprimento igual ao da palavra de entrada* ♦ *X não tem tamanho fixo*	♦ *variável Yi do tipo pilha sem tamanho fixo* ♦ *o tamanho é igual ao da palavra de entrada.*
Entrada	♦ *a própria fita*	♦ *variável X*	♦ *variável X como a da MP*
Saída	♦ *a própria fita*	♦ *variável X*	♦ *variável Yi*
Programa	*f(estado corrente, símbolo lido) = (novo estado, símbolo a ser gravado, sentido de movimento da cabeça)*	*seqüência finita de comandos:* - *partida* - *parada* - *teste* - *atribuição*	*seqüência finita de comandos:* - *partida* - *parada* - *desvio* - *empilha* - *desempilha*

Exercício 3.3. Sobre não-determinismo:

a) O que é e quais as suas principais características?

O não determinismo é uma generalização do modelo das máquinas determinísticas. No caso da Máquina de Turing ele permite que após a leitura de um símbolo, exista mais de uma alternativa possível. Cada alternativa é percorrida de forma totalmente independente. Isto significa que alterações de conteúdo na fita realizada em um caminho não modificam o conteúdo da mesma nos demais caminhos alternativos. A mesma idéia é válida para a variável X da Máquina de Post ou para as pilhas do Autômato com Pilhas. Geralmente a facilidade do não-determinismo é interpretada como segue para uma máquina de estados: a máquina, ao processar uma entrada, tem como resultado um conjunto de novos estados. Ou seja, assume um conjunto de estados alternativos, como se houvesse um multiplicação da unidade de controle, uma para cada alternativa, processando independentemente, se compartilhar recursos com os demais. Assim o processamento de um caminho não influi no estado geral e nem no símbolo lido dos demais caminhos alternativos. Para uma máquina M não-determinística, uma palavra w pertence a ACEITA(M)

se existe pelo menos um caminho alternativo que aceita a palavra. Caso contrário, se todas as alternativas rejeitam a entrada, então w pertence a REJEITA(M). Se nenhum caminho aceita a palavra e pelo menos um fica em loop, w pertence a LOOP(M).

b) Qual o seu efeito, em termos de poder computacional, nas máquinas apresentadas?

Apesar de ser aparentemente mais poderoso, o não-determinismo não acrescenta poder computacional a um modelo. Na melhor das hipóteses o que pode acontecer é chegar num estado de aceita com um número menor de passos, mas isto não implica que dada uma função programa para máquina não-determinística, esta não possa ser simulada por uma máquina determinística.

Exercício 3.4. Sobre a Hipótese de Church:

a) Por que não é demonstrável?

A Hipótese de Church diz que:

"A capacidade de computação representada pela Máquina de Turing é o limite máximo que pode ser atingido por qualquer dispositivo de computação". *Em outras palavras, afirma que qualquer outra forma de expressar algoritmos terá, no máximo, a mesma capacidade computacional da Máquina de Turing. Como a noção de algoritmo ou função computável é intuitiva, a Hipótese de Church não é demonstrável. O fato da noção de algoritmo ser intuitiva é importante pois não se pode provar matematicamente algo baseado nessa noção intuitiva.*

b) Qual seu significado e importância na Teoria da Computação?

A Máquina de Turing é um modelo matemático de computador que expressa a solução de problemas. Na Teoria da Computação é importante pois define a classe de funções computáveis e não computáveis. Assim, todos os algoritmos, para os quais existe uma máquina de Turing, são efetivamente implementáveis (solucionáveis), uma vez que, segundo Church, essa máquina é o dispositivo de computação mais geral existente, não havendo, portanto, qualquer problema solucionável que não possa ser nela implementada.

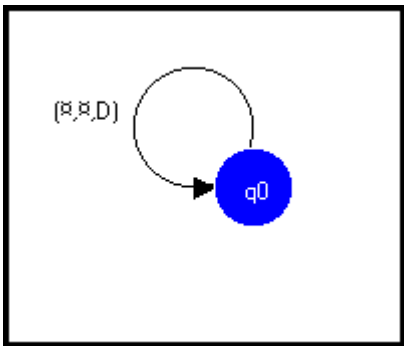
Exercício 3.5. Desenvolva Máquinas de Turing, determinísticas ou não, que aceitam as linguagens:

a) $L_1=\varnothing$

Seja uma Máquina de Turing $M=(\Sigma, Q, \Pi, q_0, F, V, \beta, \neg)$, *onde:*

$\Sigma = \{a, b\}$
$Q = \{q_0\}$
q_0 = *estado inicial*
$F = \{\ \}$
$V = \{\neg, \beta\}$
β = *símbolo branco*
$\neg$ = *símbolo de início da fita*
Π = *função programa definida a seguir:*

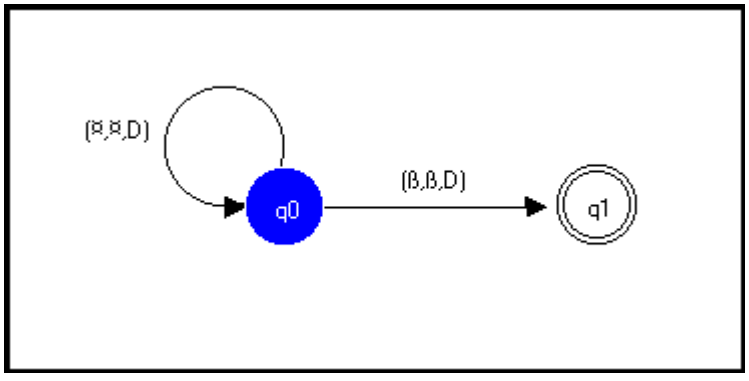
	×	a	b	B
q0	(q0,×,D)			



b) $L_2=\{\varepsilon\}$

Seja uma Máquina de Turing $M=(\Sigma, Q, \Pi, q_0, F, V, \beta, \neg)$, onde:
$\Sigma = \{a, b\}$
$Q = \{q_0\}$
q_0 = estado inicial
$F = \{q_1\}$
$V = \{\neg, \beta\}$
β = símbolo branco
$\neg$ = símbolo de início da fita
Π = função programa definida a seguir:

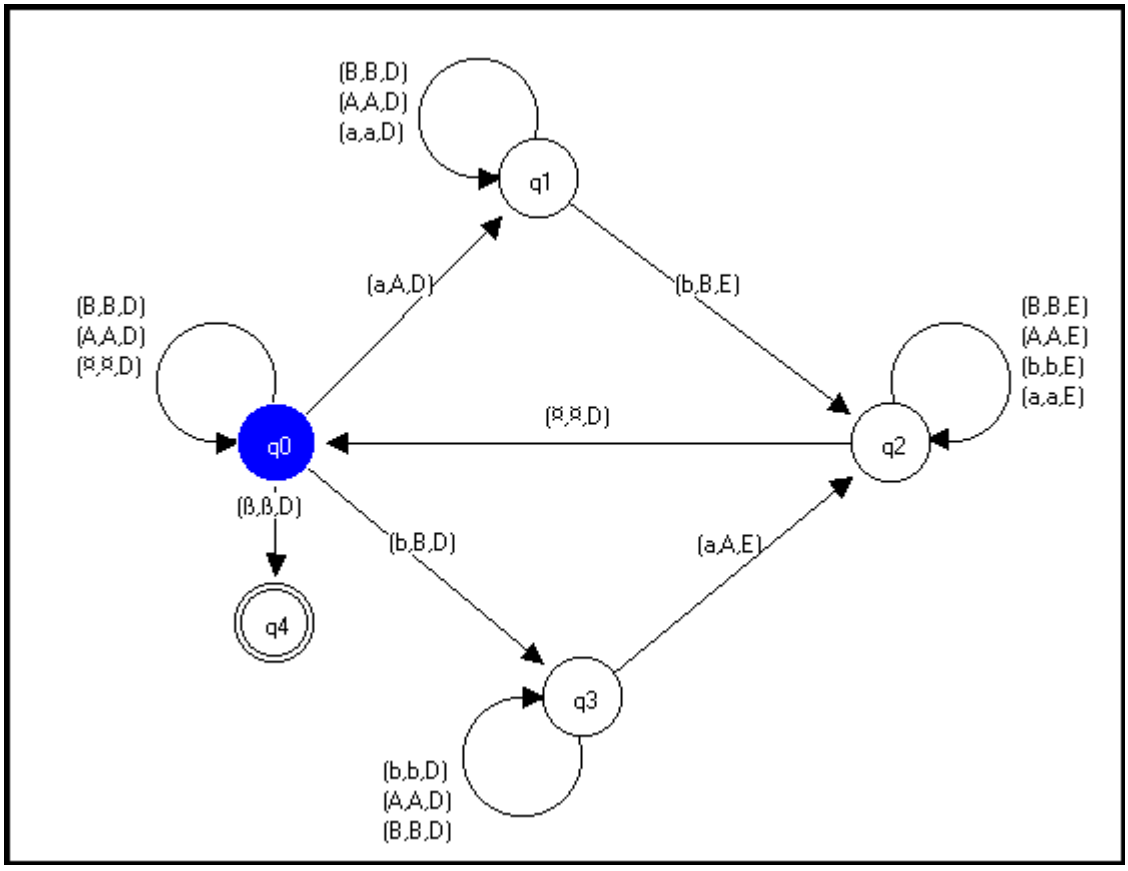
	×	a	b	B
q0	(q0,×,D)			(q1,B,D)
q1				



c) L_3={w | w tem o mesmo número de símbolos de a e b}

Seja uma Máquina de Turing M=(∑, Q, Π, q_0, F, V, β, ¬), onde:
∑ = {a, b}
Q = {q_0,q_1,q_2,q_3,q_4}
q_0 = estado inicial
F = {q_4}
V = {¬ , β , A , B}
β = símbolo branco
¬ = símbolo de início da fita
Π = função programa definida a seguir:

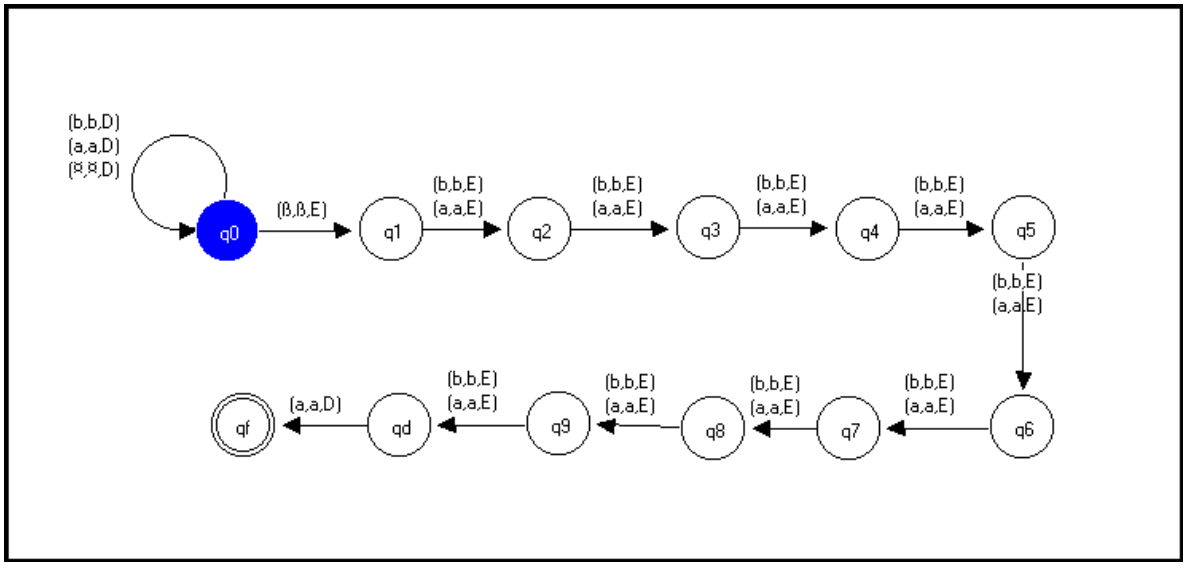
	¤	a	b	A	B	ß
q0	(q0,¤,D)	(q1,A,D)	(q3,B,D)	(q0,A,D)	(q0,B,D)	(q4,ß,D)
q4						
q1		(q1,a,D)	(q2,B,E)	(q1,A,D)	(q1,B,D)	
q2	(q0,¤,D)	(q2,a,E)	(q2,b,E)	(q2,A,E)	(q2,B,E)	
q3		(q2,A,E)	(q3,b,D)	(q3,A,D)	(q3,B,D)	



d) L_4={ w | o décimo símbolo da direita para esquerda é a}

Seja uma Máquina de Turing M=(∑, Q, Π, q_0, F, V, β, ¬), onde:
∑ = {a, b}
Q = {q_0,q_1,q_2,q_3,q_4,q_5,q_6,q_7,q_8,q_9,q_d, q_f}
q_0 = estado inicial
F = {q_f}
V = {¬, β}
β = símbolo branco
¬ = símbolo de início da fita
Π = função programa definida a seguir:

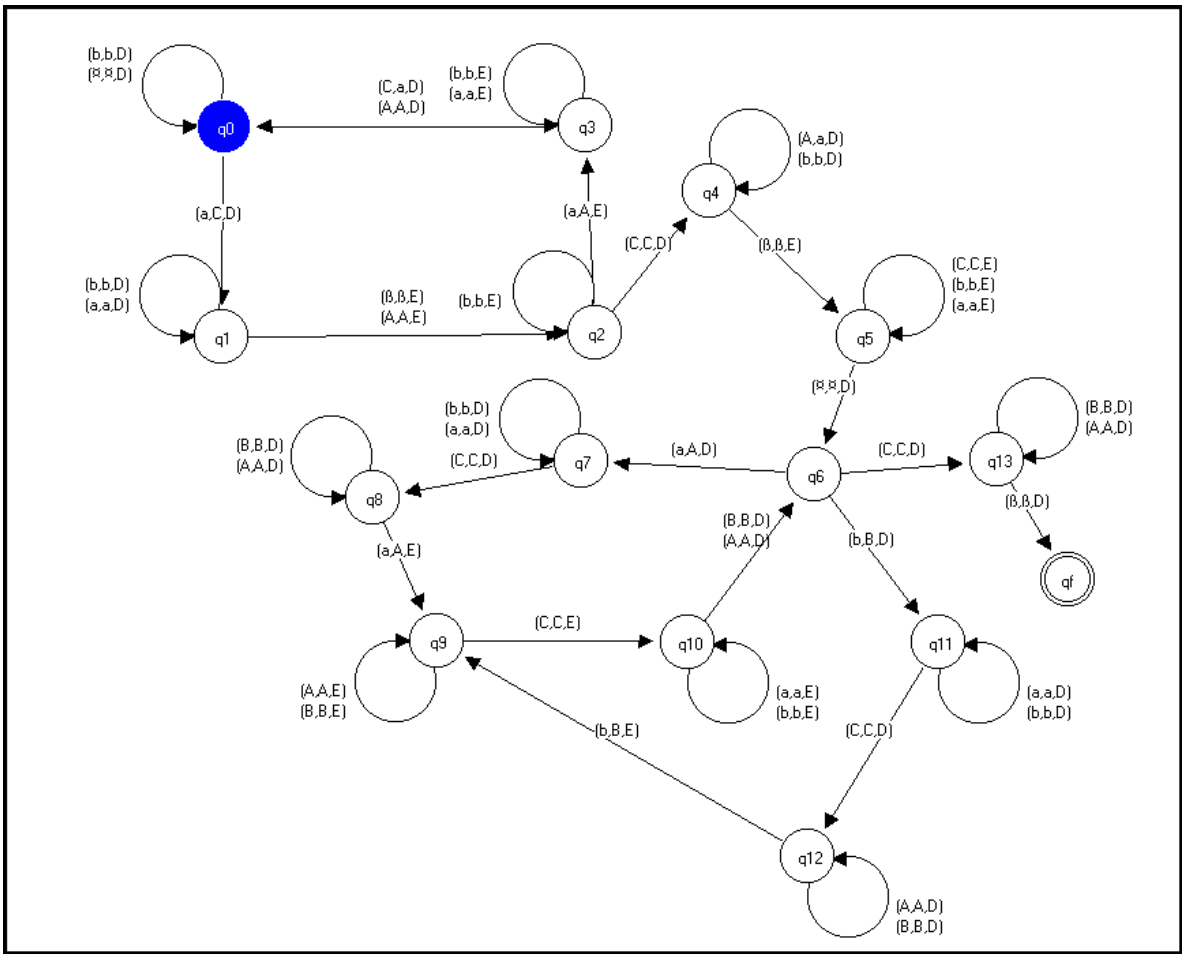
	¤	a	b	1	ß
q0	(q0,¤,D)	(q0,a,D)	(q0,b,D)		(q1,ß,E)
qf					
q1		(q2,a,E)	(q2,b,E)		
q2		(q3,a,E)	(q3,b,E)		
q3		(q4,a,E)	(q4,b,E)		
q4		(q5,a,E)	(q5,b,E)		
q5		(q6,a,E)	(q6,b,E)		
q6		(q7,a,E)	(q7,b,E)		
q7		(q8,a,E)	(q8,b,E)		
q8		(q9,a,E)	(q9,b,E)		
q9		(qd,a,E)	(qd,b,E)		
qd		(qf,a,D)			



e) **L_5={waw | w é palavra de {a,b}* }**

Seja uma Máquina de Turing M=(Σ, Q, Π, q_0, F, V, β, $\neg$), onde:
Σ = {a, b}
Q = {q_0,q_1,q_2,q_3,q_4,q_5,q_6,q_7,q_8,q_9,q_{10},q_{11},q_{12},q_{13},qf}
q_0 = estado inicial
F = {q_f}
V = {$\neg$, β, A, B, C}
β = símbolo branco
$\neg$ = símbolo de início da fita
Π = função programa definida a seguir:

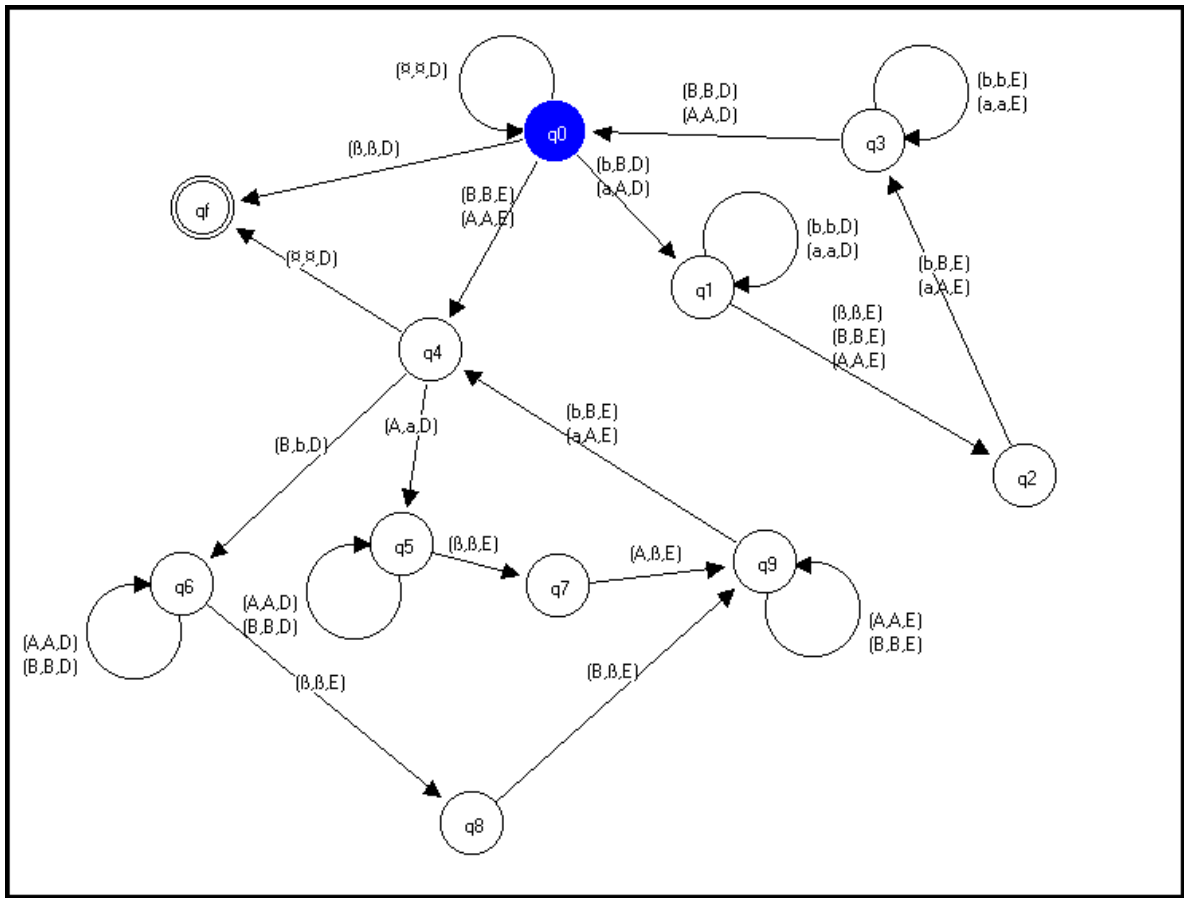
	¤	a	b	A	B	C	ß
q0	(q0,¤,D)	(q1,C,D)	(q0,b,D)				
qf							
q1		(q1,a,D)	(q1,b,D)	(q2,A,E)			(q2,ß,E)
q2		(q3,A,E)	(q2,b,E)			(q4,C,D)	
q3		(q3,a,E)	(q3,b,E)	(q0,A,D)		(q0,a,D)	
q4			(q4,b,D)	(q4,a,D)			(q5,ß,E)
q5	(q6,¤,D)	(q5,a,E)	(q5,b,E)			(q5,C,E)	
q6		(q7,A,D)	(q11,B,D)			(q13,C,D)	
q7		(q7,a,D)	(q7,b,D)			(q8,C,D)	
q8		(q9,A,E)		(q8,A,D)	(q8,B,D)		
q9				(q9,A,E)	(q9,B,E)	(q10,C,E)	
q10		(q10,a,E)	(q10,b,E)	(q6,A,D)	(q6,B,D)		
q11		(q11,a,D)	(q11,b,D)			(q12,C,D)	
q12			(q9,B,E)	(q12,A,D)	(q12,B,D)		
q13				(q13,A,D)	(q13,B,D)		(qf,ß,D)



f) **L_6={ww | w é palavra de {a,b}* }**

Seja uma Máquina de Turing M=(Σ, Q , Π, q_0, F , V , β , $\neg$), onde:
Σ = {a, b}
Q = {q_0 , q_1 , q_2 , q_3 , q_4 , q_5 , q_6 , q_7 , q_8 , q_9 , qf}
q_0 = *estado inicial*
F = {q_f}
V = {$\neg$, β , A , B}
β = *símbolo branco*
$\neg$ = *símbolo de início da fita*
Π = *função programa definida a seguir*:

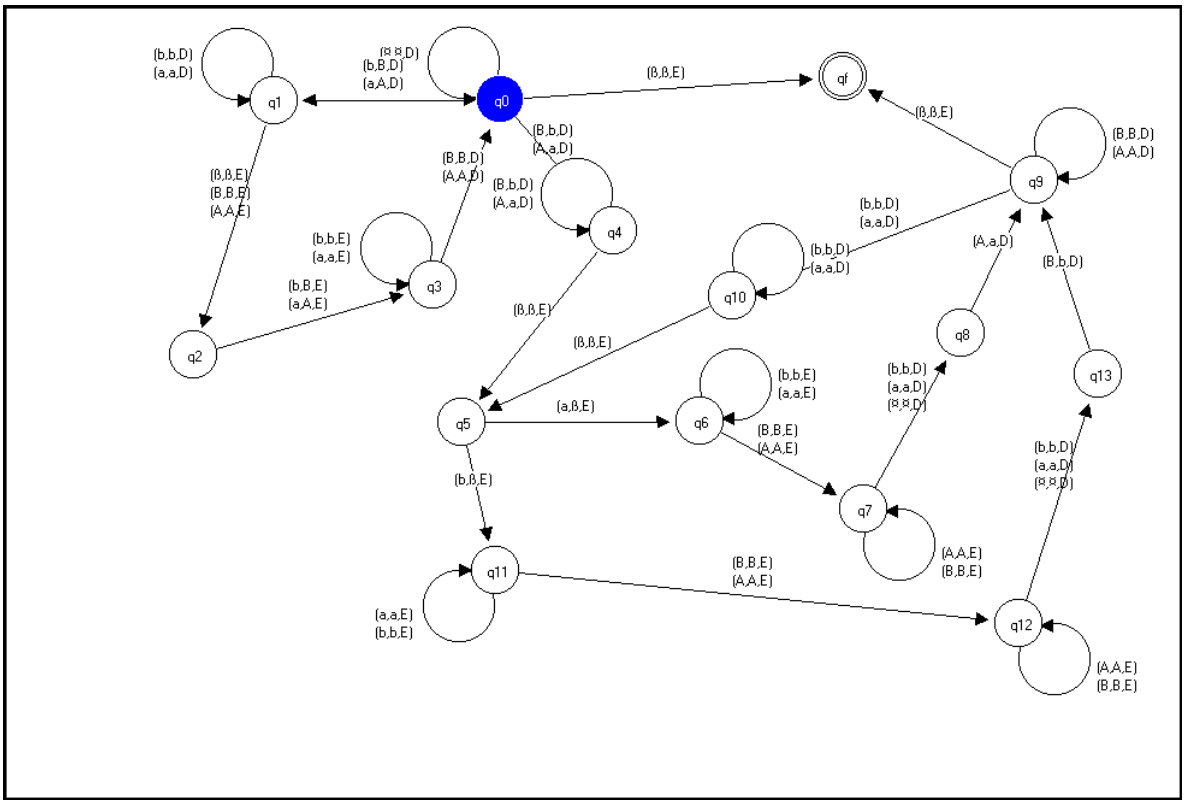
	¤	a	b	A	B	ß
q0	(q0,¤,D)	(q1,A,D)	(q1,B,D)	(q4,A,E)	(q4,B,E)	(qf,ß,D)
qf						
q1		(q1,a,D)	(q1,b,D)	(q2,A,E)	(q2,B,E)	(q2,ß,E)
q2		(q3,A,E)	(q3,B,E)			
q3		(q3,a,E)	(q3,b,E)	(q0,A,D)	(q0,B,D)	
q4	(qf,¤,D)			(q5,a,D)	(q6,b,D)	
q5				(q5,A,D)	(q5,B,D)	(q7,ß,E)
q6				(q6,A,D)	(q6,B,D)	(q8,ß,E)
q7				(q9,ß,E)		
q8					(q9,ß,E)	
q9		(q4,A,E)	(q4,B,E)	(q9,A,E)	(q9,B,E)	



g) **L_7={ ww^r | w é palavra de {a,b}* }**

Seja uma Máquina de Turing M=($\sum$, Q, Π, q_0, F, V, β, $\neg$), onde:
$\sum$ = {a , b}
Q = {q_0 , q_1 , q_2 , q_3 , q_4 , q_5 , q_6 , q_7 , q_8 , q_9 , q_{10} , q_{11} , q_{12} , q_{13} , qf}
q_0 = estado inicial
F = {q_f}
V = {$\neg$, β , A , B}
β = símbolo branco
$\neg$ = símbolo de início da fita
Π = função programa definida a seguir:

	¤	a	b	A	B	ß
q0	(q0,¤,D)	(q1,A,D)	(q1,B,D)	(q4,a,D)	(q4,b,D)	(qf,ß,E)
qf						
q1		(q1,a,D)	(q1,b,D)	(q2,A,E)	(q2,B,E)	(q2,ß,E)
q2		(q3,A,E)	(q3,B,E)			
q3		(q3,a,E)	(q3,b,E)	(q0,A,D)	(q0,B,D)	
q4				(q4,a,D)	(q4,b,D)	(q5,ß,E)
q5		(q6,ß,E)	(q11,ß,E)			
q6		(q6,a,E)	(q6,b,E)	(q7,A,E)	(q7,B,E)	
q7	(q8,¤,D)	(q8,a,D)	(q8,b,D)	(q7,A,E)	(q7,B,E)	
q8				(q9,a,D)		
q9		(q10,a,D)	(q10,b,D)	(q9,A,D)	(q9,B,D)	(qf,ß,E)
q10		(q10,a,D)	(q10,b,D)			(q5,ß,E)
q11		(q11,a,E)	(q11,b,E)	(q12,A,E)	(q12,B,E)	
q12	(q13,¤,D)	(q13,a,D)	(q13,b,D)	(q12,A,E)	(q12,B,E)	
q13					(q9,b,D)	



h) L8 = {www | w é palavra de Σ^*}

M = (Σ, Q, ∏, q0, F, V, ß, ¬)
onde:
Σ = {a, b}
Q = {q0, q1, q2, q3, q4, q5, q6, q7, q8, q9, q10, q11, q12, q13, q14, qf}
F = {qf}
V = {0, 1, A, B}
∏ = função programa definida no arquivo ex_3_5_H.mte

i) L9 = {w | w = $a^1 b^2 a^3 b^4 \ldots a^{n-1} b^n$ e n é um múmero natural par}

M = (Σ, Q, ∏, q0, F, V, ß, ¬)
onde:
Σ = {a, b}
Q = {q0, q1, q2, q3, q4, q5, q6, q7, q8, q9, q10, q11, q12, q13, q14, q15, q16, q17, qf}
F = {qf}
V = {A, B}
∏ = função programa definida no arquivo ex_3_5_i.mte

j) L10 = {w | w = $a^n b^n$ ou $b^n a^n$}

M = (Σ, Q, ∏, q0, F, V, ß, ¬)
onde:
Σ = {a, b}
Q = {q0, q1, q2, q3, q4, q5, q6, q7, q8, q9, q10, q11, q12, qf}
F = {qf}

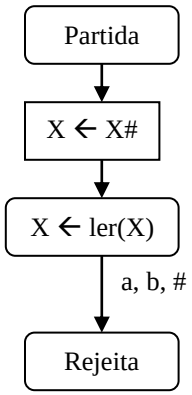
V = {A, B}
∏ = função programa definida no arquivo ex_3_5_j.mte

k) L11 = {w | w = $a^i b^j c^k$, onde i=j ou j=k}

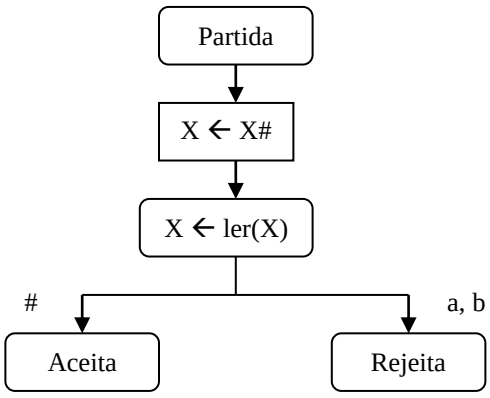
M = (Σ, Q, ∏, q0, F, V, ß, ¬)
onde:
Σ = {a, b}
Q = {q0, q1, q2, q3, q4, q5, q6, q7, q8, q9, q10, q11, q12, q13, q14, qf}
F = {qf}
V = {A, B}
∏ = função programa definida no arquivo ex_3_5_k.mte

Exercício 3.6. Desenvolva Máquinas de Post, determinísticas ou não, que aceitem as linguagens dadas no Exercício 3.5.

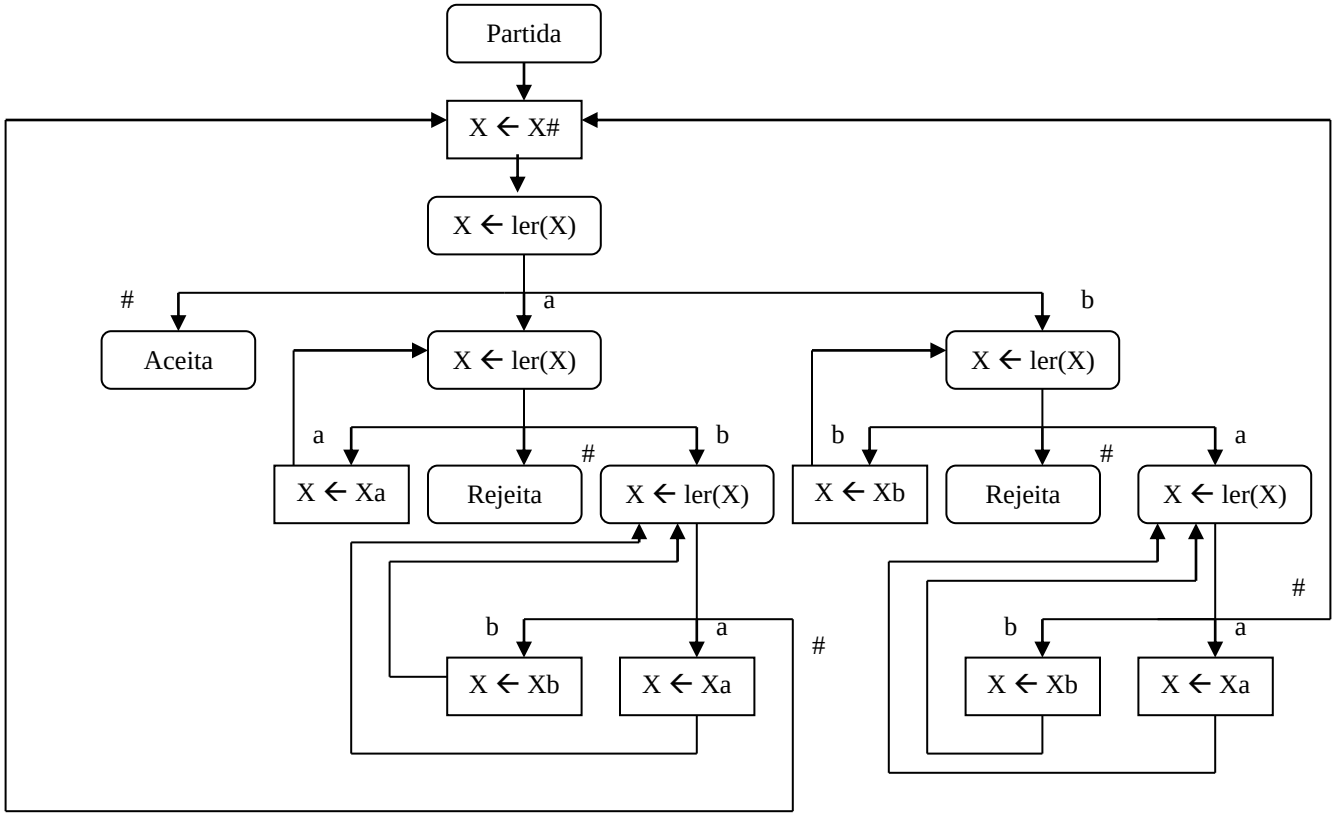
a) L1 = ∅



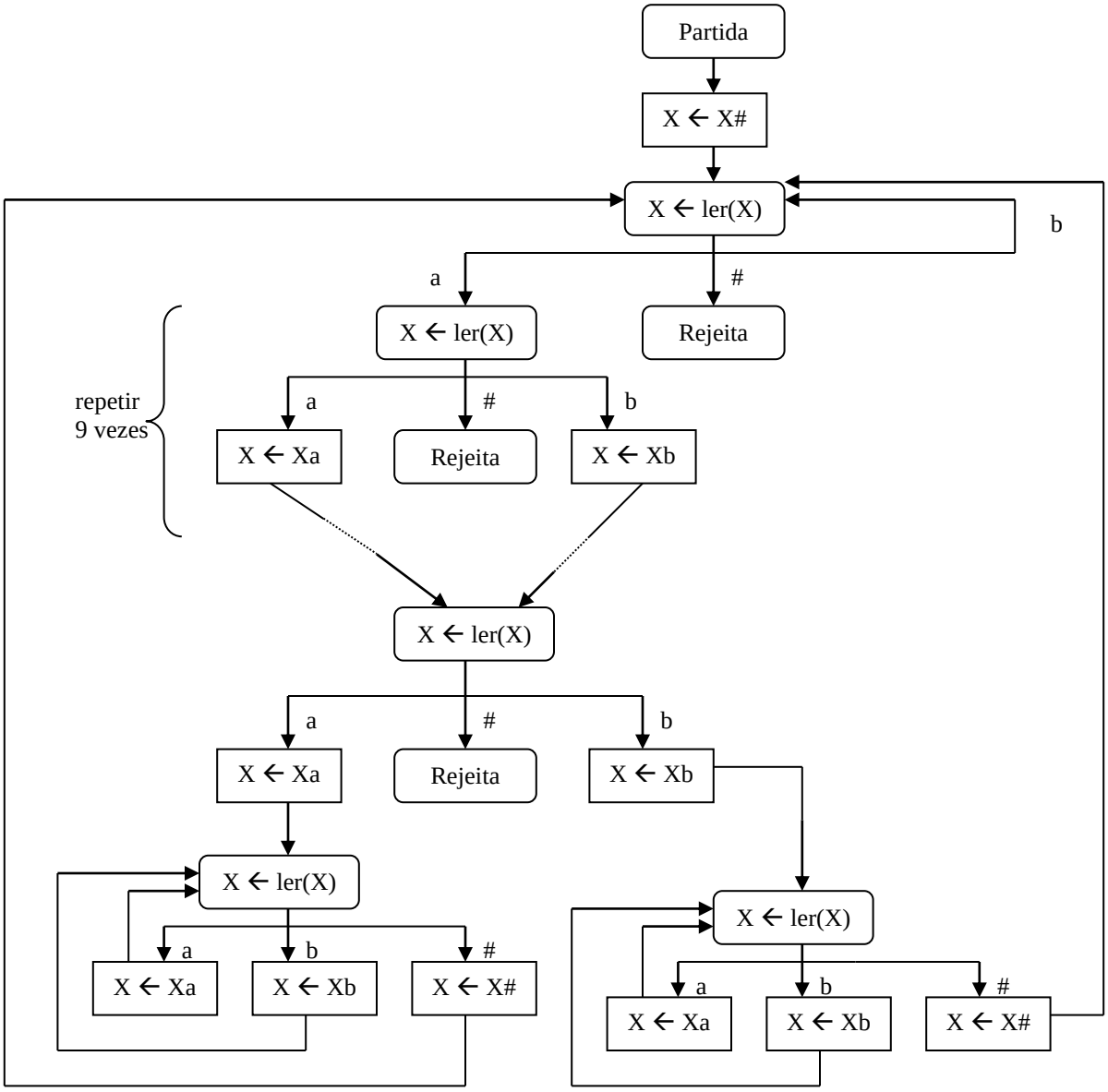
b) L2 = {ε}



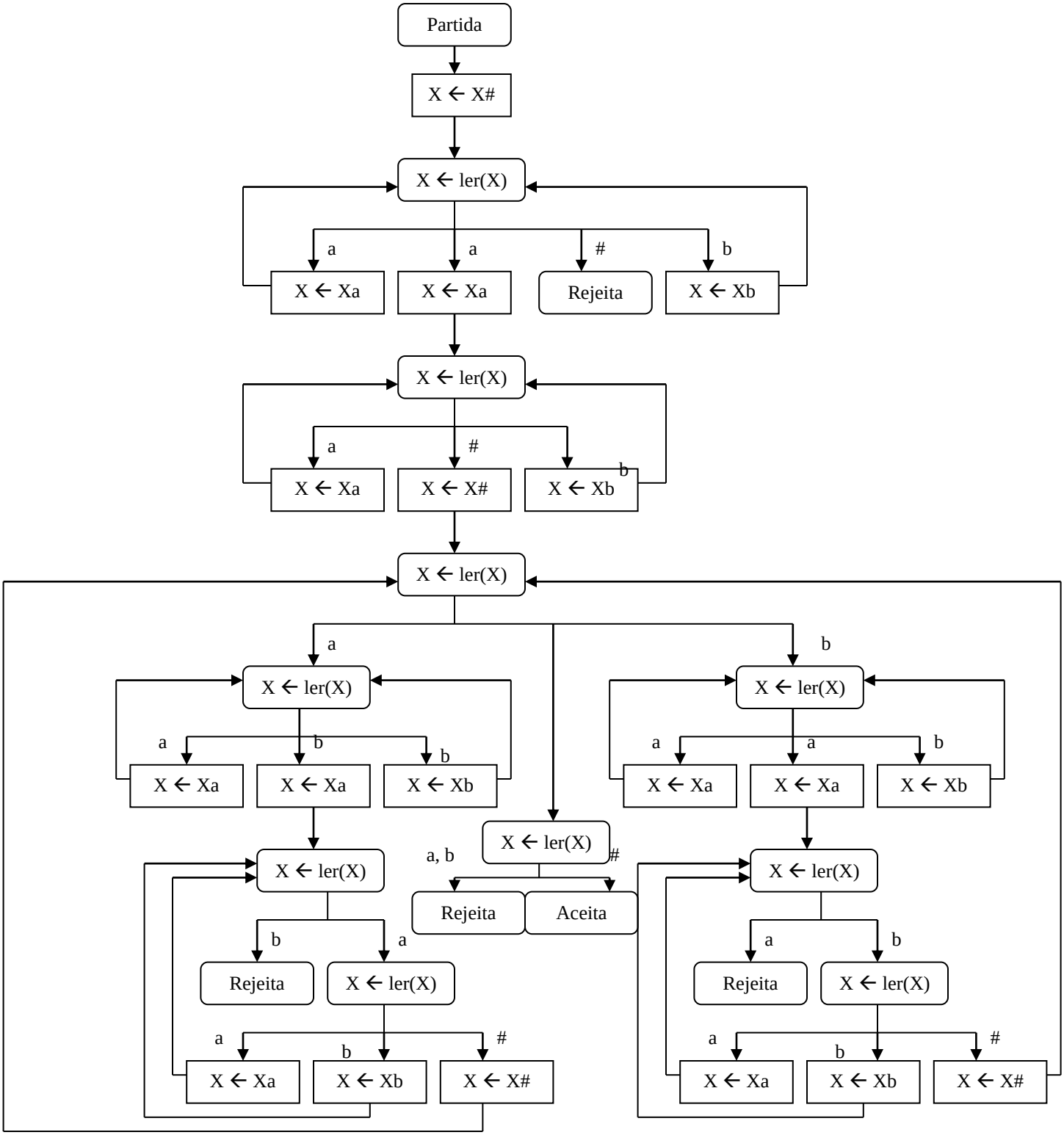
c) **L3 = {w | w tem o mesmo número de símbolos “a” e “b”}**



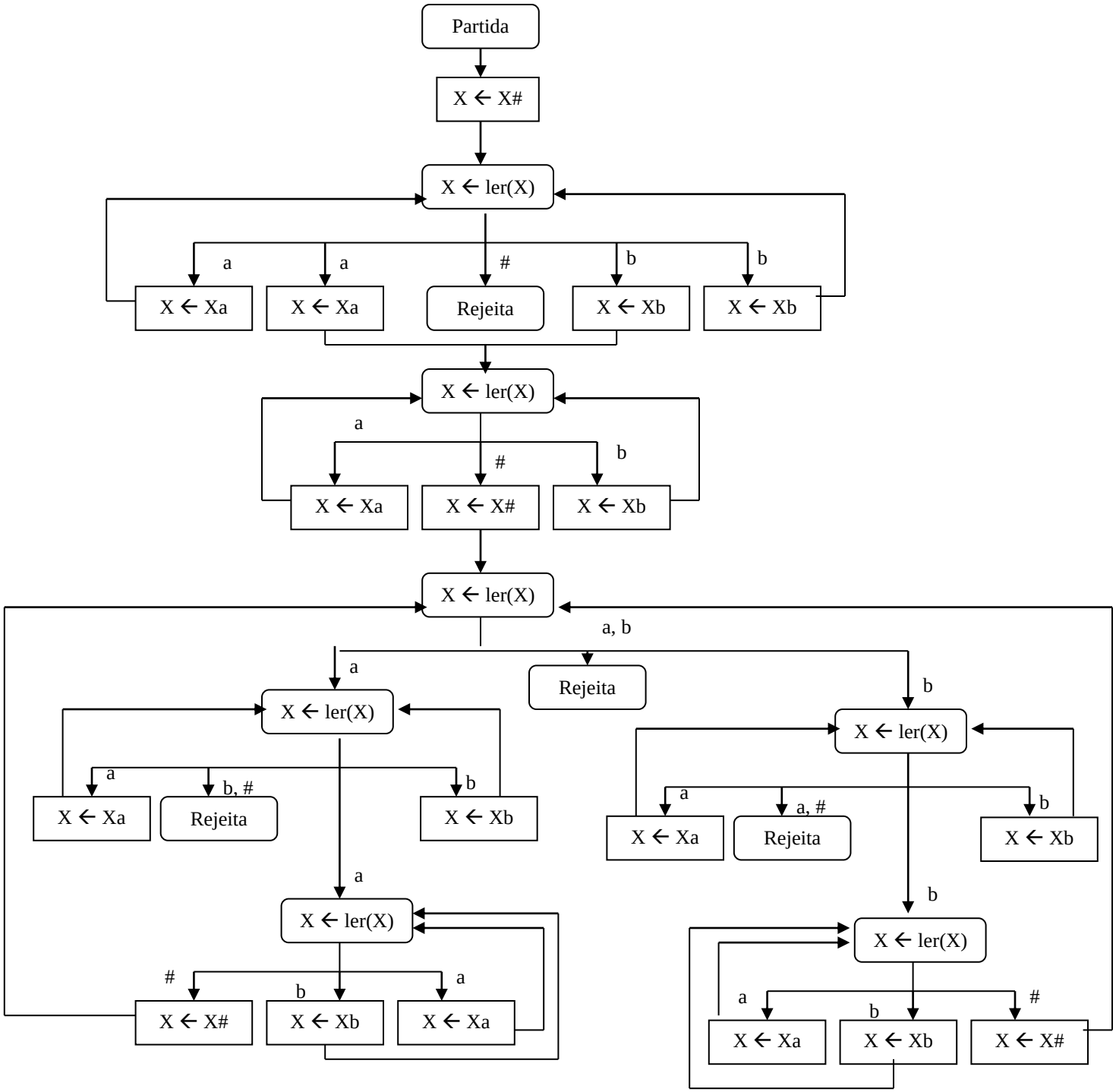
d) **L4 = {w | o 10^0 símbolo da direita para a esquerda é "a" }**



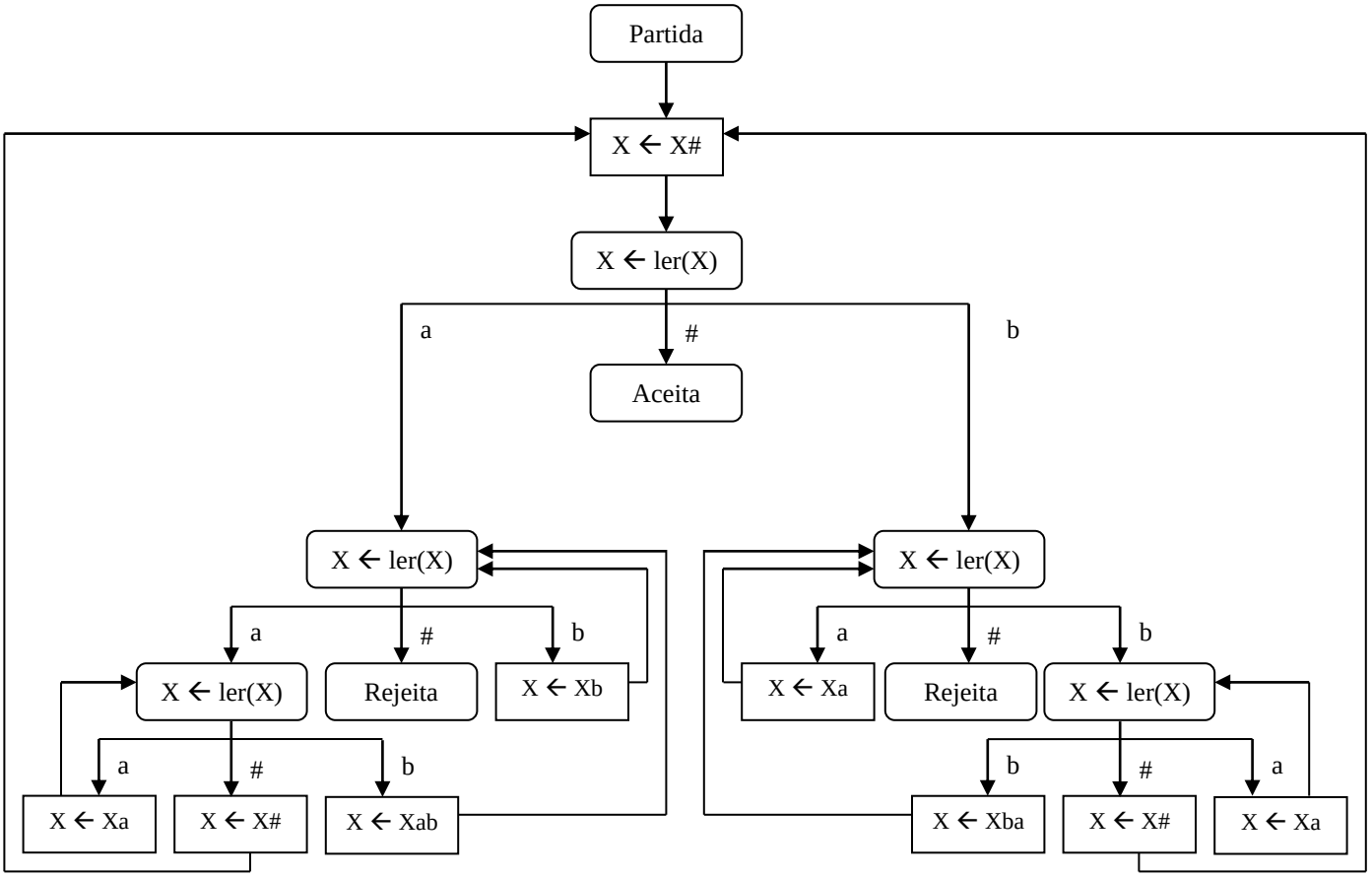
e) **L5 = {waw | w é palavra de Σ^*}**



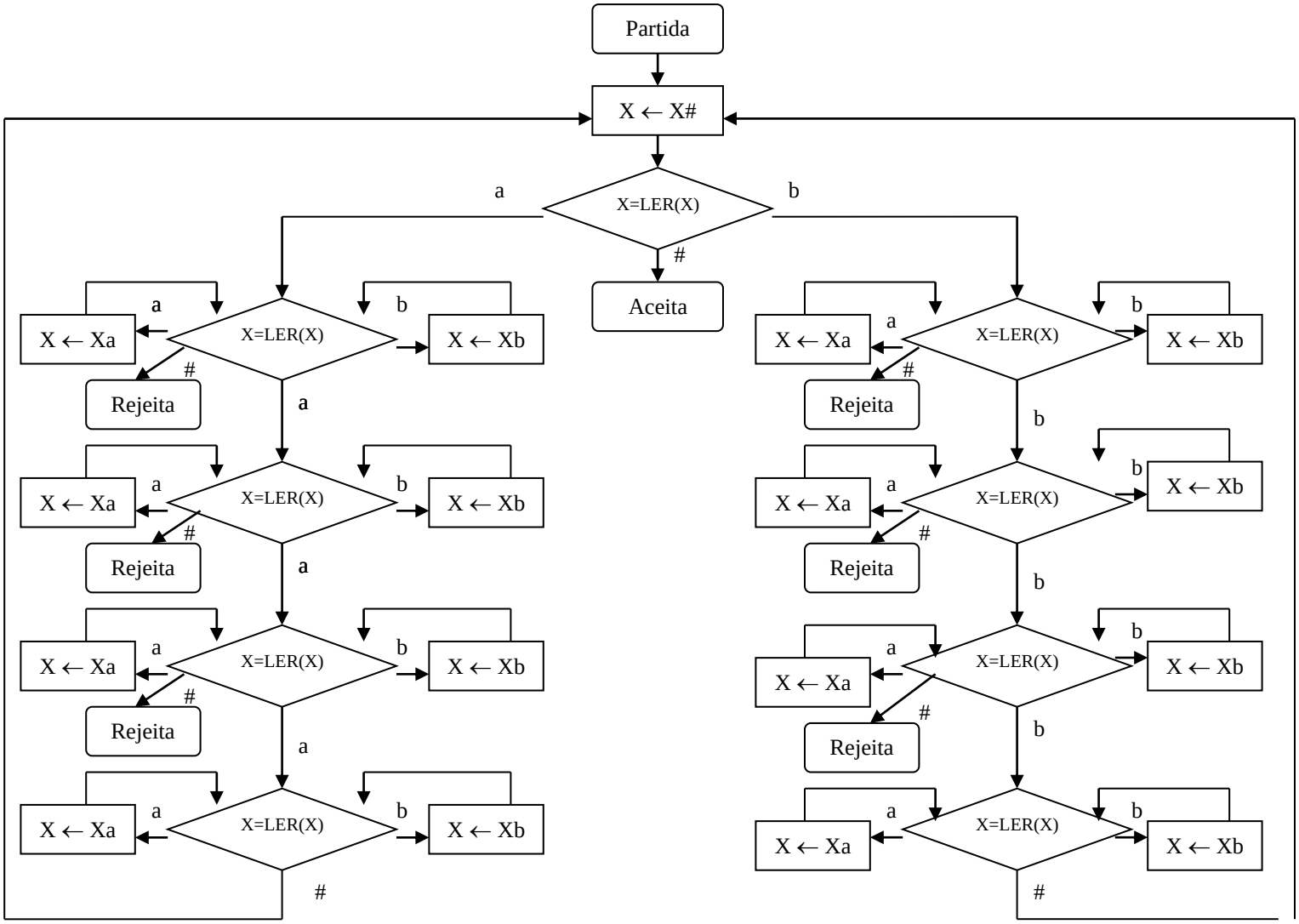
f) **L6 = { ww | w é palavra de Σ^*}**



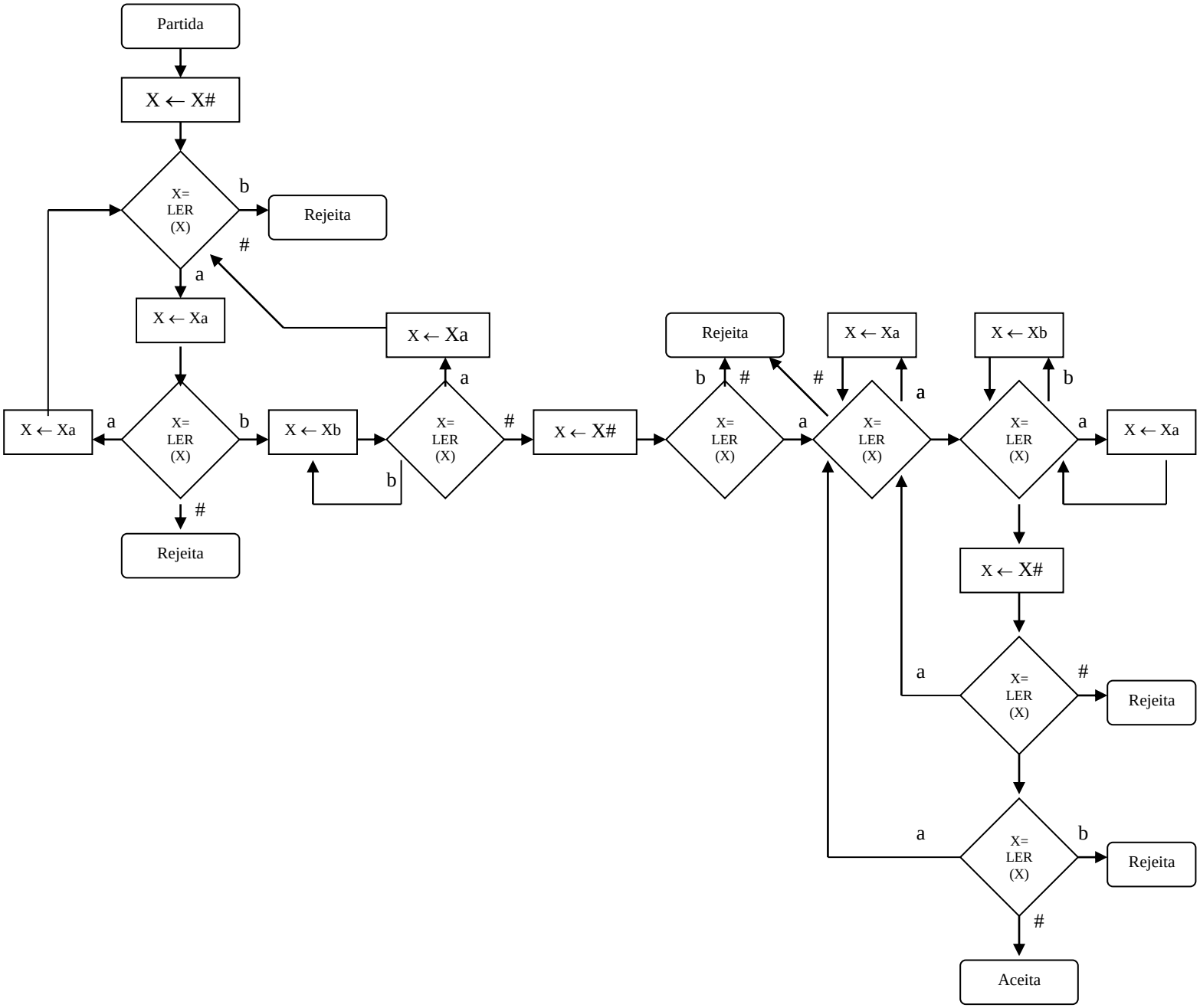
g) **L7 = { ww^r | w é palavra de Σ^*}**



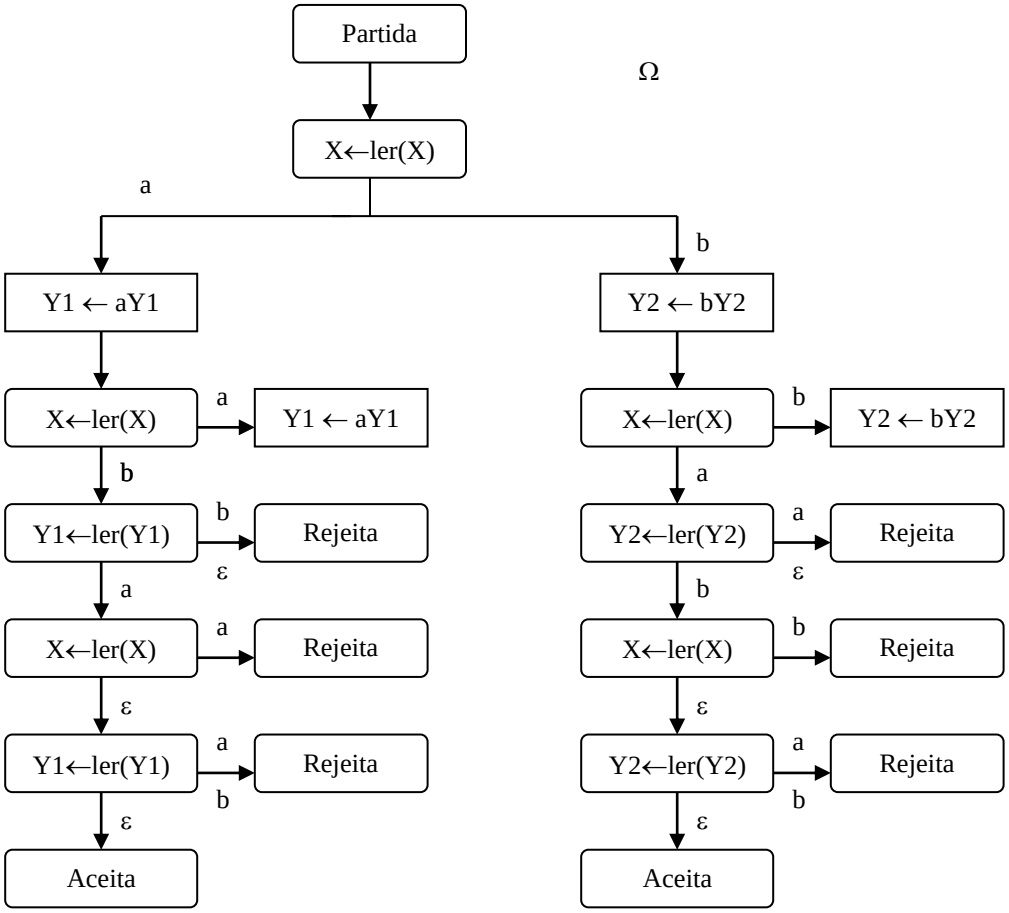
h) **L8 = {www | w é palavra de Σ^*}**



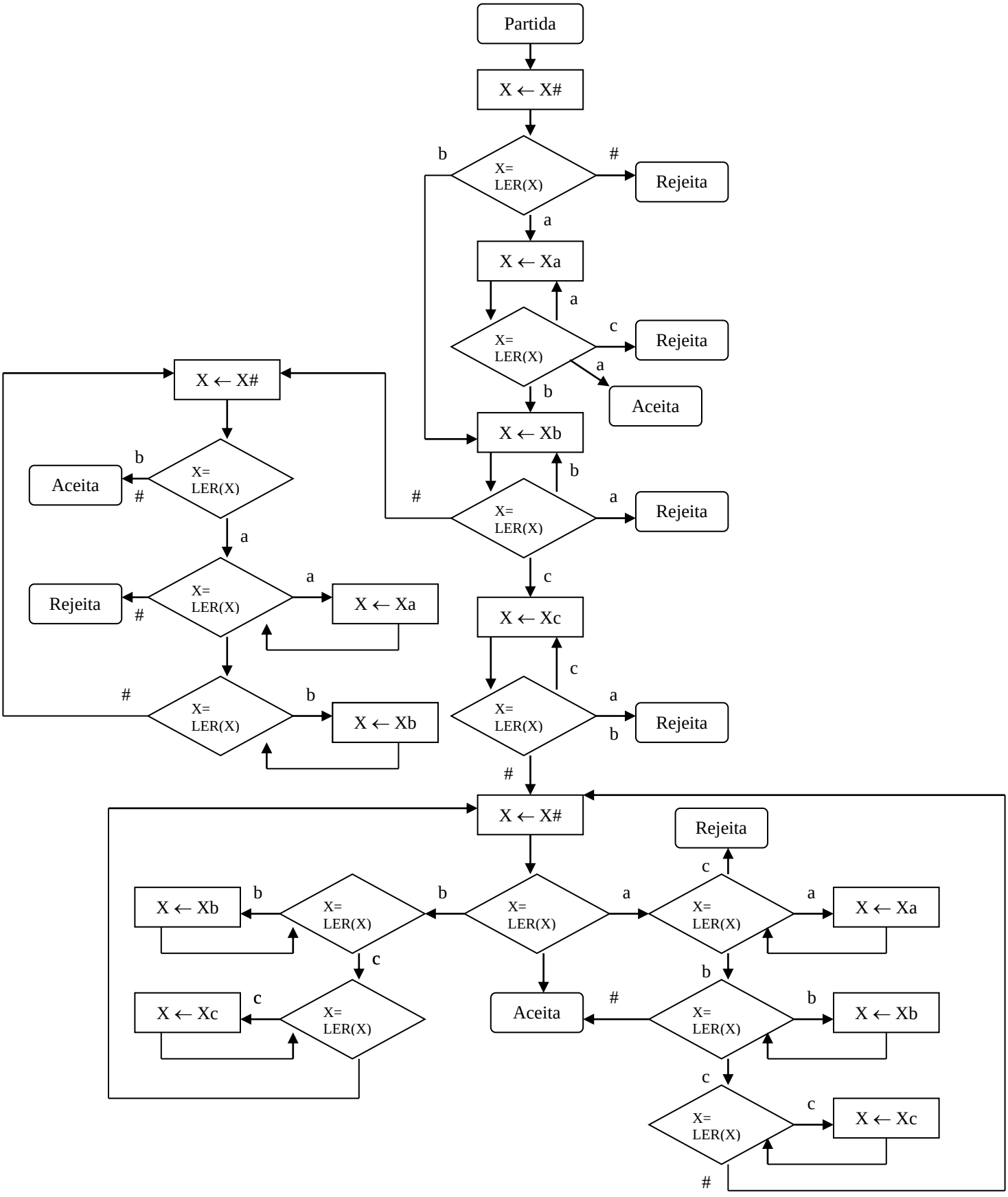
i) **L9 = {w | w = $a^1b^2a^3b^4...a^{n-1}b^n$ e n é número natural par}**



j) **L10 = {w | w = $a^n b^n$ ou w = $b^n a^n$ }**

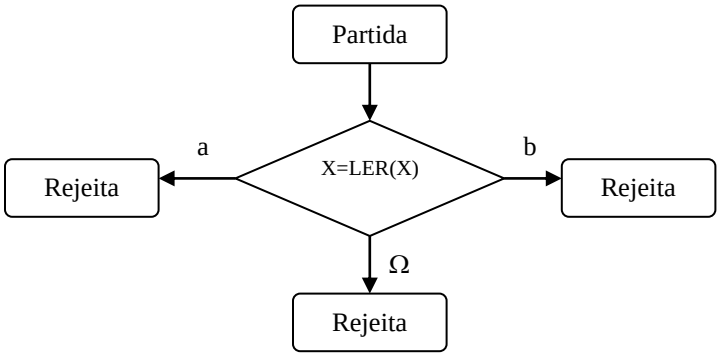


k) $L_{11} = \{ W \mid W = a^i b^j c^k$, **onde ou i=j ou j=k** $\}$

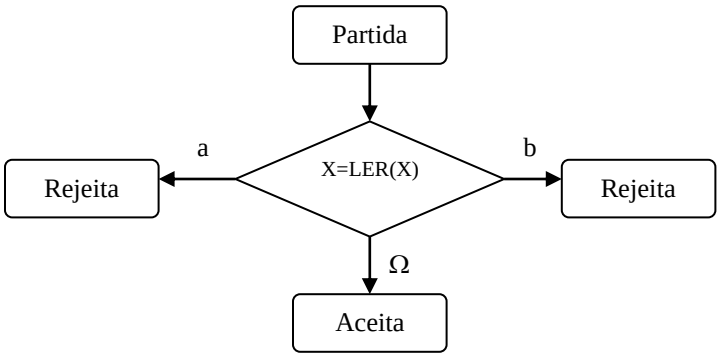


Exercício 3.7. Desenvolva Autômatos com Duas Pilhas ou Máquinas com Pilhas, determinísticas ou não, que aceitem as linguagens dadas no Exercício 3.5.

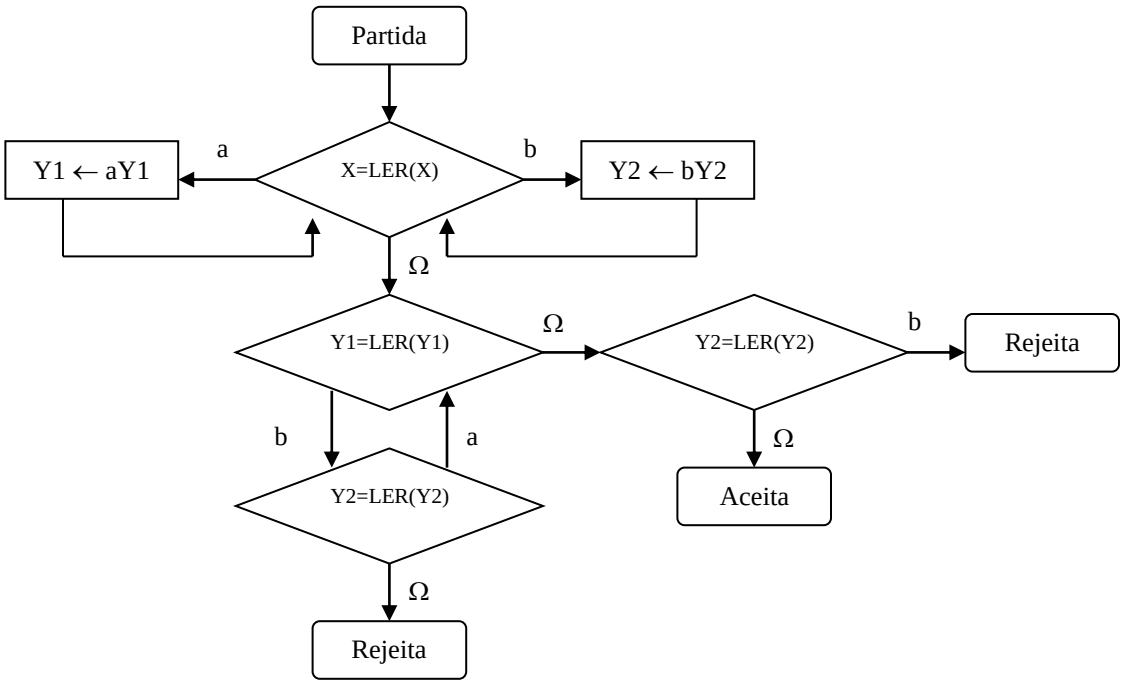
a) **L1 = ∅**



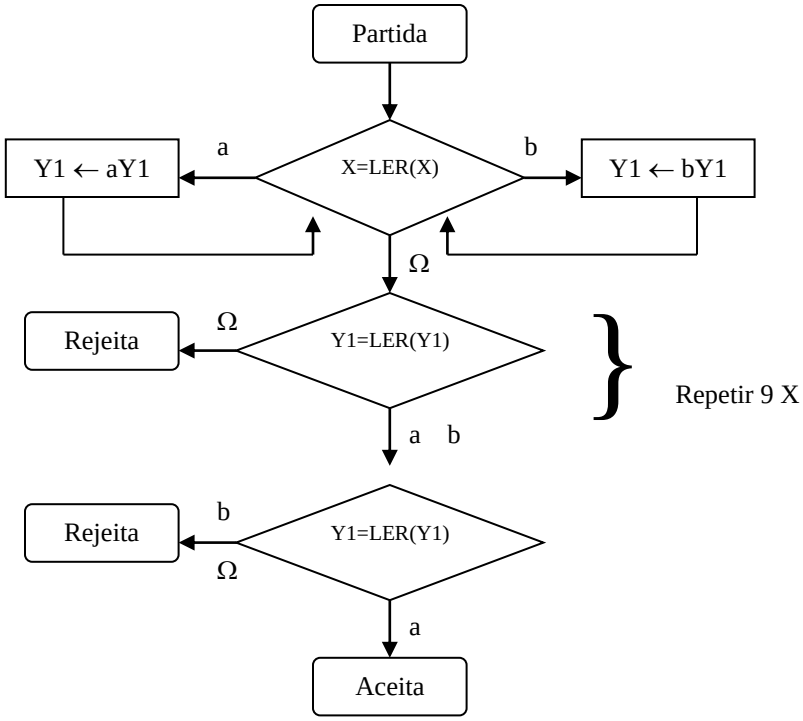
b) **L2 = {ε}**



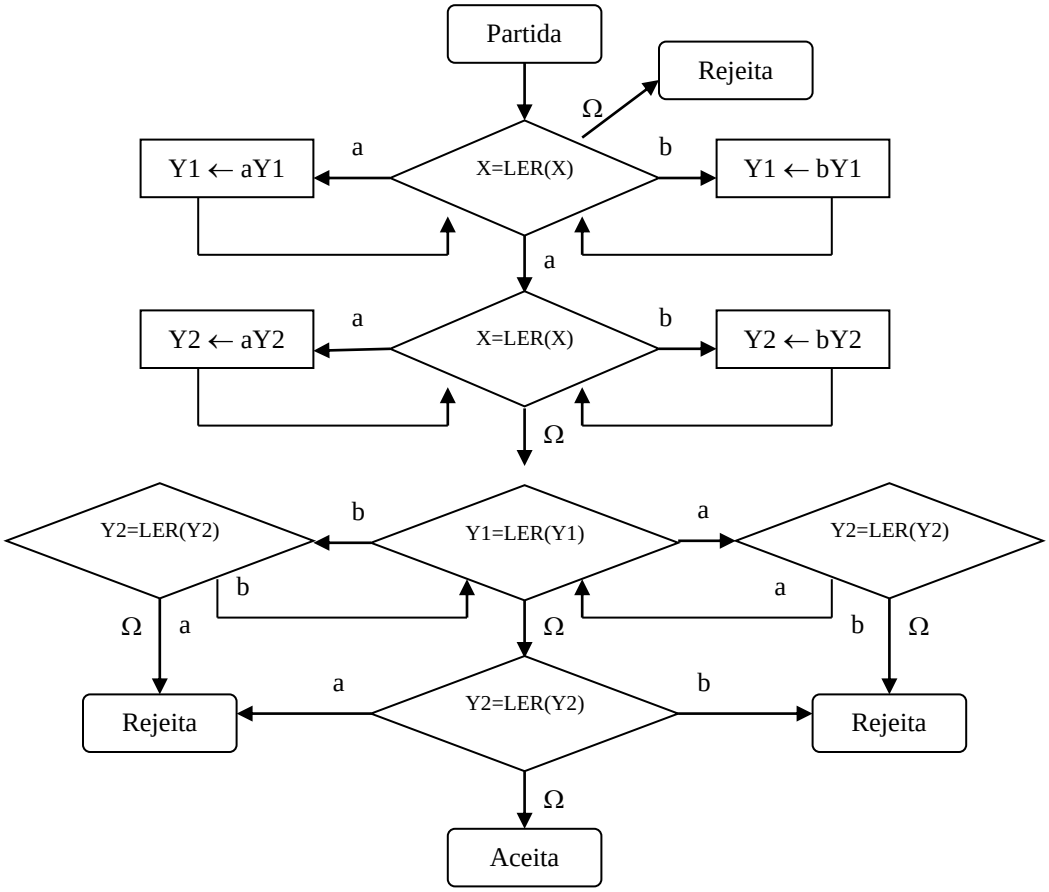
c) **L3 = {w | w tem o mesmo número de símbolos "a" e "b"}**



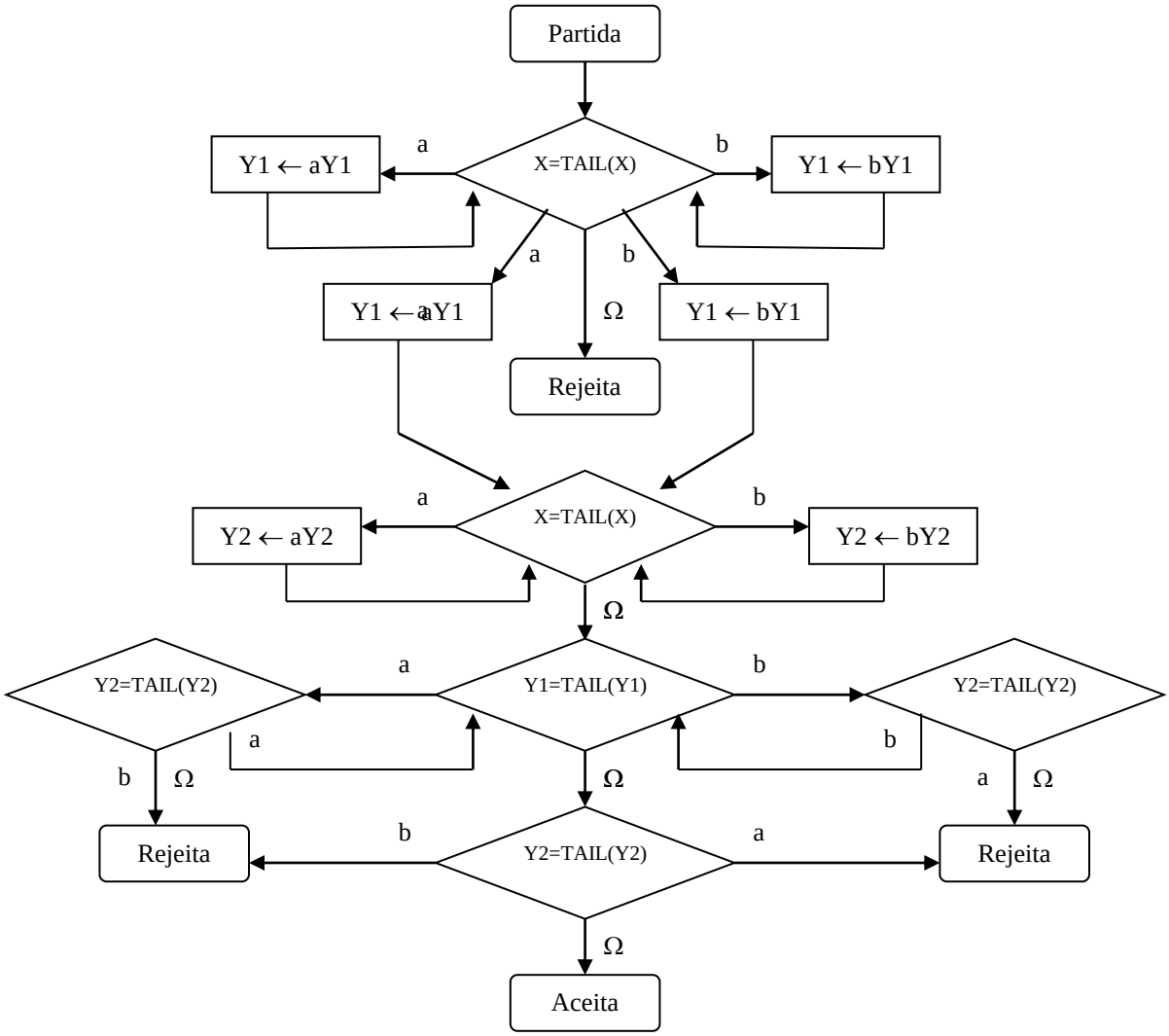
d) **L4 = {w | o 10^0 símbolo da direita para a esquerda é "a" }**



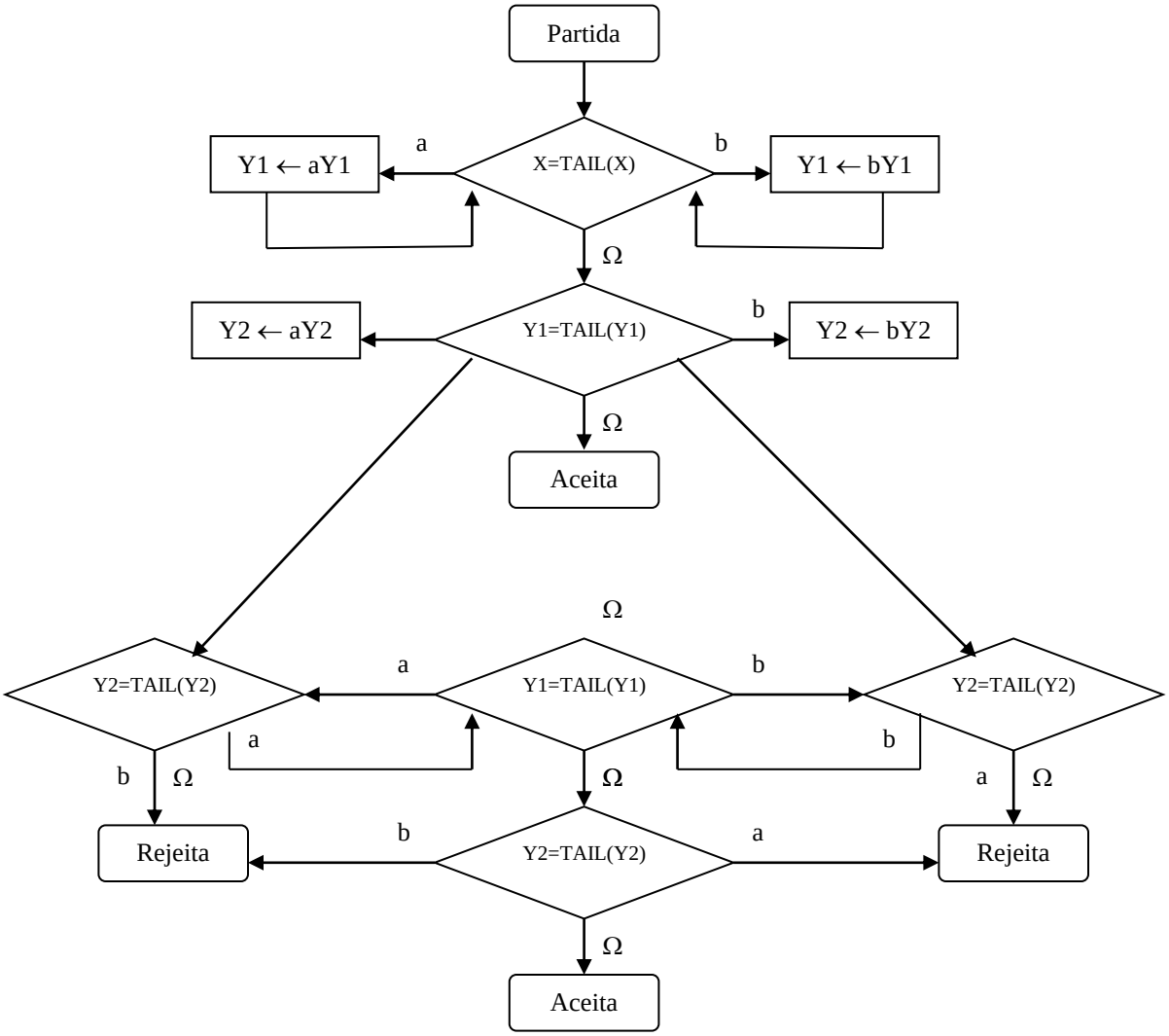
e) **L5 = {waw | w é palavra de Σ^*}**



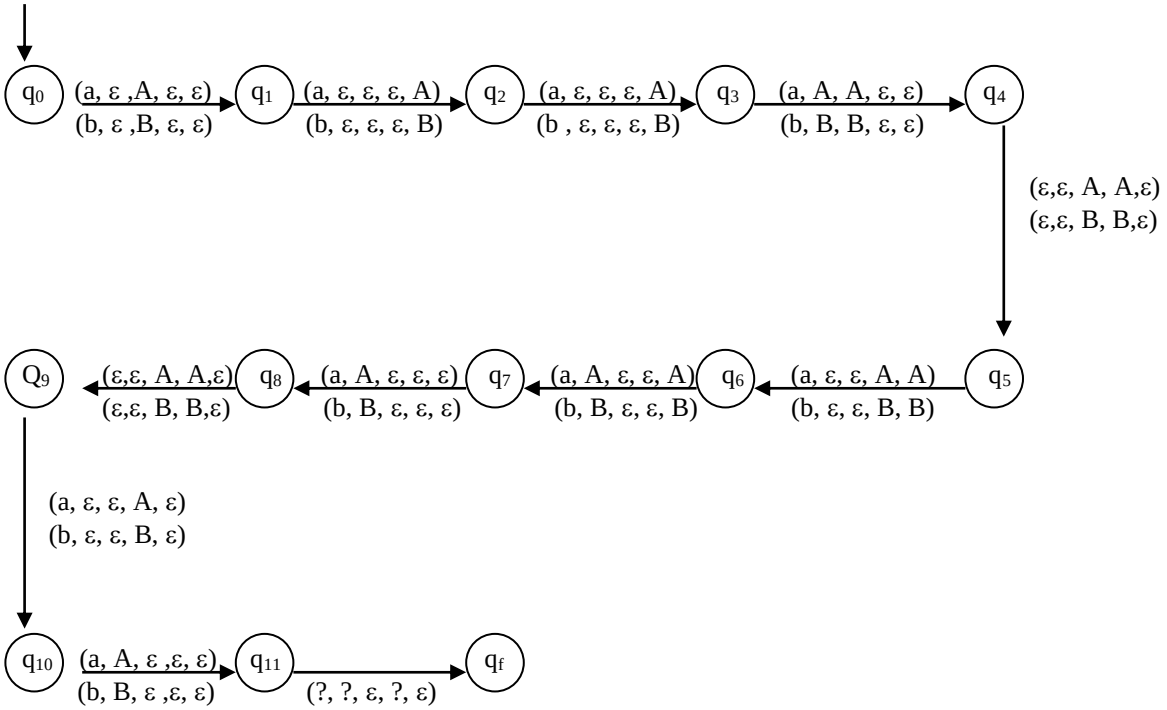
f) **L6 = { ww | w é palavra de Σ^*}**



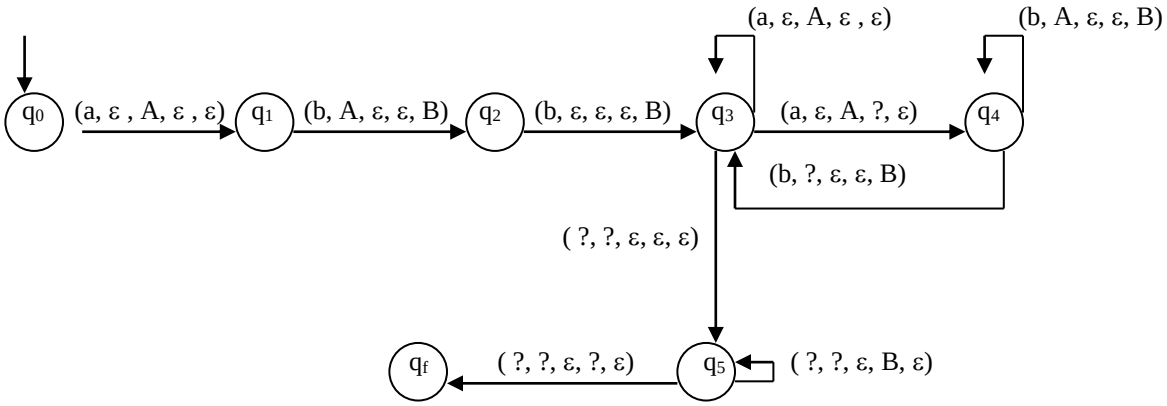
g) **L7 = { ww^r | w é palavra de Σ^*}**



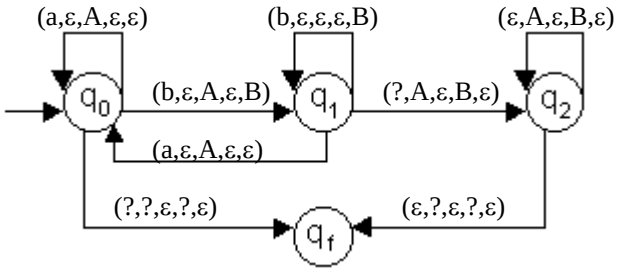
h) **L8 = {www | w é palavra de Σ^*}**



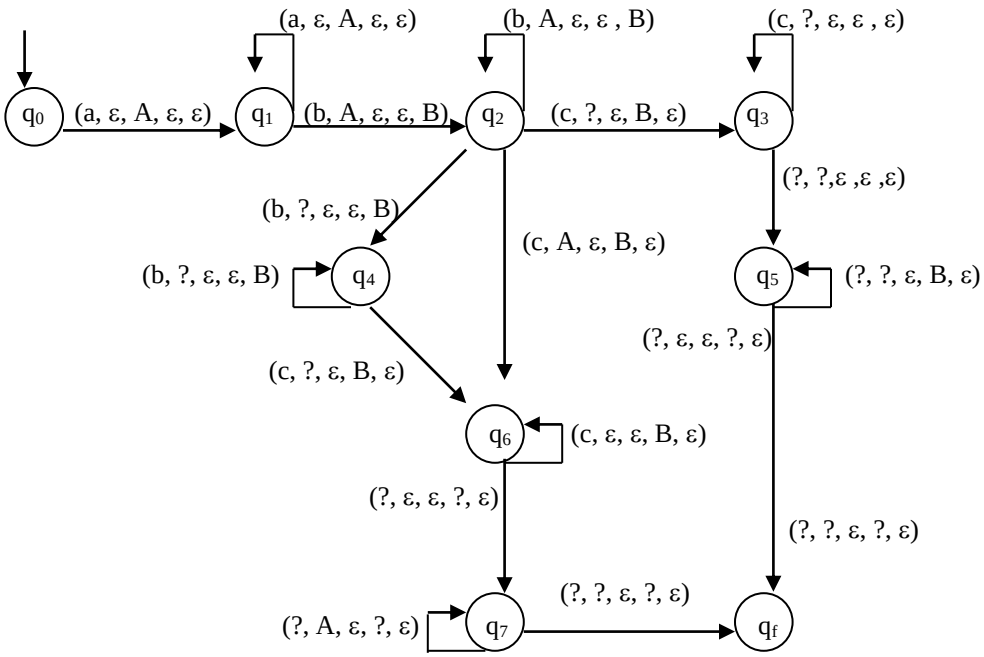
i) **L9 = {w | w = $a^1b^2a^3b^4...a^{n-1}b^n$ e n é número natural par}**



j) **L10 = {w | w = a^nb^n ou w = b^na^n }**



k) **L10 = {w | w = $a^n b^n$ ou w = $b^n a^n$ }**



Exercício 3.8. Seja a expressão booleana (EB) definida indutivamente como segue:

i) **v e f são EB;**

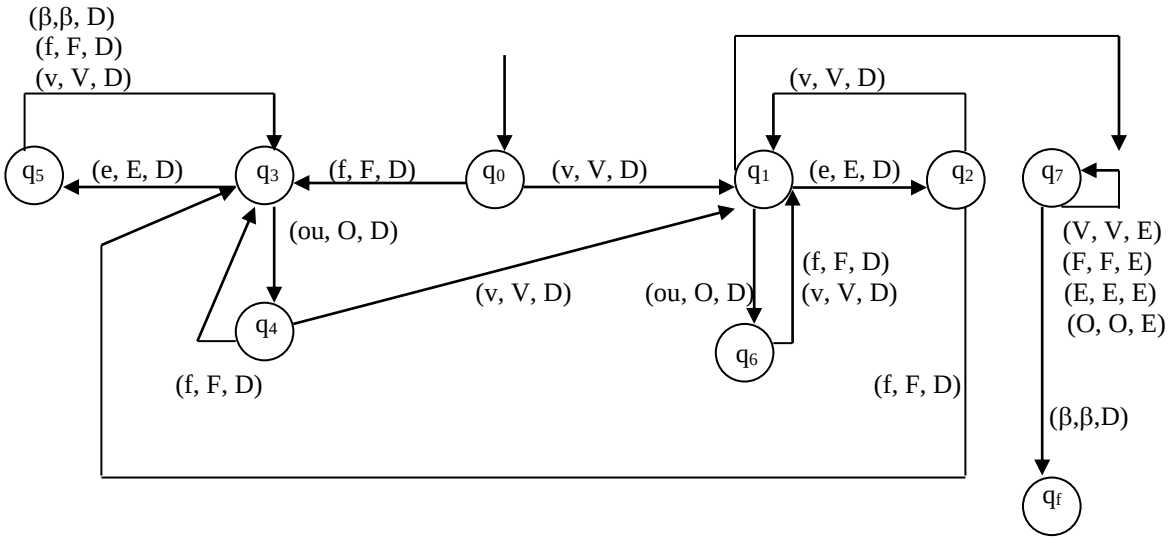
ii) **Se p e q são EB, então p e q e p ou q também são EB.**

Assuma que o conetivo e tem prioridade sobre o ou:

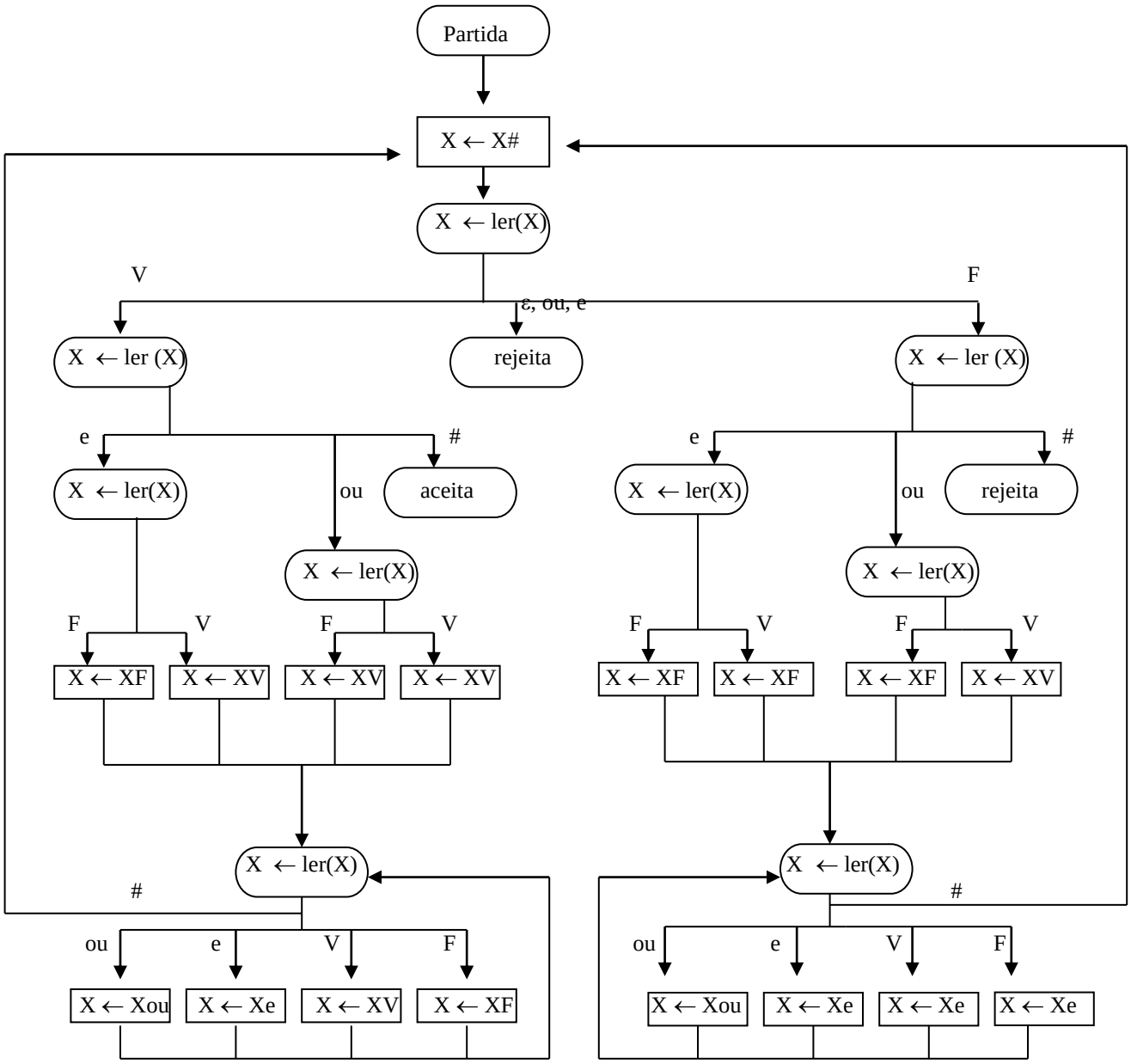
a) **Construa uma Máquina de Turing sobre ∑ = { v, f, e, ou } tal que:**

ACEITA(M) = { EB | EB é v }
REJEITA(M) = { EB | EB é f }
LOOP(M) = { W | W não é EB}

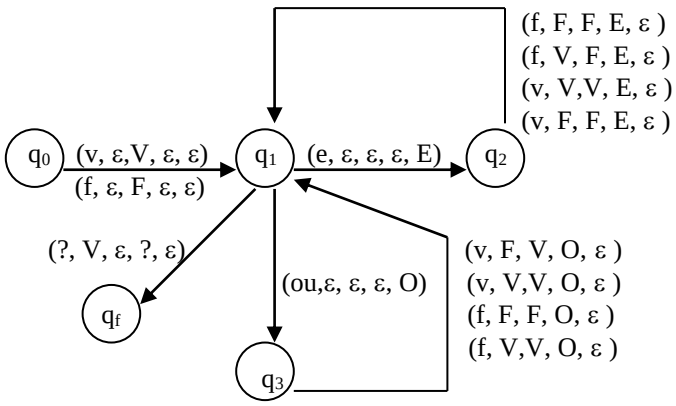
Arquivo Exerc38_a.gte



b) Construa uma Máquina de Post sobre ∑ conforme definido no item a)



c) Construa uma Máquina de Turing não-determinística ou Autômato com Duas Pilhas não-determinístico, sobre ∑, conforme definido no item a)



Exercício 3.9. Na demonstração do Teorema 3.18, é sugerido o desenvolvimento de um diagrama de fluxos de Máquina de Post que realize rotação no conteúdo de X, fazendo com que o último símbolo de X passe a ser o primeiro, ou seja, se o conteúdo da variável X é $a_1a_2...a_{n-1}\ a_n$ passa a ser $a_na_1a_2...a_{n-1}$. Desenvolva uma Máquina de Post que processe esta rotação.

Exercício 3.10 Prove que qualquer Máquina de Turing pode ser simulada por uma Máquina de Turing com somente dois estados.

Sugestão: modifique o alfabeto auxiliar, introduzindo novo símbolos.

Qualquer Máquina de Turing pode ser simulada por uma Máquina de Turing com somente dois estados ampliando o alfabeto tanto quanto necessário, um estado será o final e outro o inicial, ficando a máquina no estado inicial durante toda computação. Os símbolos do alfabeto auxiliar são utilizados para representar os demais estados.

Exercício 3.11. Prove que o poder computacional da Classe das Máquinas com n Pilhas (n>2) é equivalente ao da Classe das Máquinas com Duas Pilhas.

Para provar que a Classe das Máquinas com n pilhas (n>2) é equivalente à Classe das Máquinas com Duas Pilhas, precisamos como são simuladas as n pilhas em apenas duas pilhas.

Precisamos representar n ponteiros (cabeças das pilhas) que executam a leitura e gravação em cada uma das n pilhas. Para isso, utilizaremos um alfabeto auxiliar que contenha caracteres especiais para simular estes ponteiros. Ou seja, o caractere especial terá a função de "dividir" a pilha em tantas pilhas quando forem necessárias, e as n pilhas estarão representadas em apenas duas.

Exercício 3.12. Desenvolva um programa em computador que simule qualquer Máquina de Turing. A entrada para o simulador deve ser a função programa e a saída o estado final da máquina simulada, o conteúdo da fita e o número de movimentos da cabeça da fita.

Exercício 3.13. Como é possível ampliar o exercício anterior, introduzindo a facilidade de não-determinismo, usando uma linguagem de programação não-concorrente?

OBS: Na resolução dos exercícios 3.14 a 3.17 foram usadas as seguintes macros, desenvolvidas ao longo do capítulo 3:

Macro	*Definida em*	*Descrição*
A := 0	*Exemplo 3.3 (p. 73)*	*Atribuição do valor zero a um registrador*
A := A + B	*Exemplo 3.5 (p. 74)*	*Adição de dois registradores*
A := A + B usando C	*Exemplo 3.6 (p. 74)*	*Adição de dois registradores, preservando o conteúdo*
A := B usando C	*Exemplo 3.7 (p. 74)*	*Atribuição do conteúdo de um registrador, preservando o conteúdo*

Será usado também o seguinte teste, aqui definido: ***se B = C*** *(testa se o conteúdo de dois registradores é igual)*

```
X := B;
Y := C;
até X = 0 ou Y = 0
faça (X := X – 1;
      Y := Y – 1)
se X = 0
então se Y = 0
        então verdadeiro
        senão falso
senão falso
```

Exercício 3.14. Mostre como as instruções abaixo podem ser construídas como macros em Norma:

a) r1: se A < 2 então vá_para r2 senão vá_para r3

```
se A = 0
então r2
senão   (A := A – 1;
         se A = 0
         então r2
         senão r3)
```

b) r1: se div(A, B) então vá_para r2 senão vá_para r3

```
C := A usando D;
até B = 0
faça ( até (B = 0 ou C = 0)
       faça (B := B – 1;
             C := C – 1;)
       C := A usando D)
se C = 0
então  r2
senão  r3
```

Exercício 3.15. Desenvolva os programas, em Norma, que realizam as operações e testes abaixo:

a) A := B – C

```
A := 0;
até  C = 0
faça      ( B := B – 1;
              C := C – 1 );
até B = 0
faça      ( B := B – 1;
              A := A + 1 );
```

b) **A := B / C**

OBS: Ao final da execução, o quociente estará armazenado em A e o resto em F

```
A := 0;
D := C usando E;
até B = 0
faça (até B = 0 ou D = 0 )
       faça ( B := B – 1;
              D := D – 1 ) )
       se D = 0
       então  ( A := A + 1;
                D := C usando E ) )
F : = C usando E;
até D = 0
faça ( F := F – 1;
        D := D – 1 ) )
```

c) **Divisão inteira de B por C**

OBS: Ao final da execução, o quociente estará armazenado no registrador A

```
A := 0;
D := C usando E;
até B = 0
faça (até B = 0 ou D = 0 )
       faça ( B := B – 1;
              D := D – 1 ) )
       se D = 0
       então  ( A := A + 1;
                D := C usando E ) )
```

d) **Fatorial (A!)**

```
E := 0;
C := 0;
até A = 0
faça ( E := E + 1;
        C := C + 1;
       A : = A – 1);
até C = 0
faça ( C := C – 1;
       A : = A + 1);
       E := E – 1;
se A = 0
então A := 1
```

```
senão     (até E = 0
           faça ( C := E usando D;
                  até C = 0
                  faça ( A := A + A usando D;
                         C := C – 1)
                         E := E – 1 ));
```

e) Potenciação (A := A^N)

```
se N = 0
então A := 1
senão (N := N – 1
       até N = 0
       faça ( C := A usando B;
              até C = 0
              faça ( A := A + A usando B;
                     C := C – 1)
              N := N – 1))
```

f) Se A < B

```
até ( A = 0 ou B = 0)
faça (A := A – 1;
      B := B – 1)
se B = 0
então falso
senão verdadeiro
```

g) Se A <= B

```
até ( A = 0 ou B = 0)
faça (A := A – 1;
      B := B – 1)
se A = 0
então verdadeiro
senão falso
```

h) Se A <= 0

OBS: foi usada a representação de inteiros em dois registradores, sendo que A1 armazena o sinal e A2 armazena a magnitude.

```
se A2 = 0
então verdadeiro
senão  se A1 = 0
       então falso
       senão verdadeiro
```

Exercício 3.16. Desenvolva os programas, em Norma, que realizam as operações abaixo nos inteiros (utilize a representação sinal-magnitude introduzida no Exemplo 3.11):

a) A := B + C

```
se B1 = C1
então (A1 := B1 usando D;
       A2 := B2 + C2 usando D)
senão (A2 := B2 usando D;
       até A2 = 0 ou C2 = 0
       faça (          A2 := A2 – 1;
             C2 := C2 – 1)
       se C2 = 0
       então se A2 = 0
             então A1 = 0
             senão A1 := B1 usando D
       senão (até C2 = 0
              faça ( A2 := A2 + 1;
                     C2 := C2 – 1)
               A1 := C1 usando D))
```

b) A := B x C

```
se B1 = 0
então  se C1 = 0
       então A1 := 0
       senão A1 := 1
senão  se C1 = 0
       então A1 := 1
       senão A1 := 0;
A2 := 0;
até C2 = 0
faça (A2 := A2 + B2 usando D;
      C2 := C2 – 1)
```

c) A := B – C

```
até ( B2 = 0 ou C2 = 0 )
faça ( B2 := B2 – 1;
       C2 := C2 – 1 )
se B2 = 0
então  se C2 = 0
       então  A1 := 0
       senão  ( até C2 = 0
                faça ( B2 := B2 + 1;
                       C2 := C2 – 1 )
                A1 := C1 )
senão  A1 := B1;
A2 := B2;
```

d) A := B / C

OBS: Ao final da execução, o quociente estará armazenado em A e o resto em F

```
se B1 = 0
então  se C1 = 0
       então A1 := 0
       senão A1 := 1
A2 := 0;
```

```
D := C2 usando E;
até B2 = 0
faça  (até B2 = 0 ou D = 0
        faça ( B2 := B2 – 1;
                D := D – 1 )
        se D = 0
        então ( A2 := A2 + 1;
                 D := C2 usando E ) )
F : = C2 usando E;
até D = 0
faça ( F := F – 1;
        D := D – 1 ) )
```

Exercício 3.17. Desenvolva os programas, em Norma, que realizam as operações abaixo nos números racionais (utilize a representação de racionais introduzida no Exemplo 3.12):

a) A := B + C

```
A1 := 0;
D := C2;
até D = 0
faça ( A1 := A1 + B1 usando E;
        D := D – 1)
D := C1;
até D = 0
faça ( A1 := A1 + B2 usando E;
        D := D – 1)
A2 := 0;
D := C2;
até D = 0
faça (A2 := A2 + B2 usando E;
        D := D – 1)
```

b) A := B x C

```
A1 := 0;
até C1 = 0
faça ( A1 := A1 + B1 usando E;
        C1 := C1 – 1)
A2 := 0;
até C2 = 0
faça (A2 := A2 + B2 usando E;
        C2 := C2 – 1)
```

c) A := B – C

```
A1 := 0;
D := C2;
até D = 0
faça ( A1 := A1 + B1 usando E;
        D := D – 1)
D := C1;
até D = 0
faça ( A1 := A1 – B2 usando E;
```

D := D – 1)
A := 0;
D := C2;
até D = 0
faça (A2 := A2 + B2 usando E;
D := D – 1)

d) A := B / C

A1 := 0;
até C2 = 0
faça (A1 := A1 + B1 usando E;
C2 := C2 – 1)
A2 := 0;
até C1 = 0
faça (A2 := A2 + B2 usando E;
C1 := C1 – 1)

Exercício 3.18. Seja a *Máquina NormaNeg*, a qual é, em todos os aspectos, idêntica a Norma, exceto pelo fato de poder armazenar números negativos (inteiros) em seus registradores. Prove que, para cada fluxograma NormaNeg, pode-se definir um fluxograma Norma equivalente sem o uso de registradores extras.

***Sugestão*: utiliza a representação de inteiros baseada na função codificação e desenvolva os programas para as instruções de Norma que simulam as de NormaNeg.**

Para se obter um fluxograma Norma equivalente a cada fluxograma NormaNeg sem o uso de registradores extras, pode-se utilizar a codificação de n-uplas naturais para armazenar os números negativos em Norma.

Utiliza-se os expoentes dos números primos 2 e 3 para expressar o sinal e a magnitude, respectivamente, dos números inteiros.

Assim, representamos um número inteiro m *pelo par ordenado* (s, |m|), *onde*

- |m| *é a magnitude (valor absoluto)*
- s *é o sinal, sendo que:* s=*1, se* m<*0*
  s=*0, se* m≥*0*

A codificação do número inteiro m *é da seguinte forma:* $2^s \bullet 3^m$ = *número natural*

Exercício 3.19. Qual a diferença fundamental entre as Classes das Linguagens recursivas e das Linguagens Enumeráveis Recursivamente? Qual a importância de se distinguir estas duas classes?

A Classe das Linguagens Enumeráveis Recursivamente define a classe de todas as linguagens que podem ser reconhecidas mecanicamente e a Classe das Linguagens Recursivas define a classe de todas as linguagens que podem ser reconhecidas mecanicamente e para as quais existe sempre um reconhecedor que sempre para, para qualquer entrada, reconhecendo ou rejeitando. Estas classes de linguagens podem ser comparadas com as funções parciais e totais.

Exercício 3.20. Demonstre que a Classe das Linguagens Recursivas é fechada para as operações de união, intersecção e diferença. Demonstre inicialmente para a operação sobre duas linguagens recursivas. Após, amplie a demonstração para n linguagens (demonstre por indução em n).

Seja L uma linguagem pertencente à Classe das Linguagens Recursivas, ou seja, existe uma Máquina de Turing M tal que:

ACEITA(M) = L
REJEITA(M) = Σ^* *- L*
LOOP(M) = $\varnothing$

Vamos mostrar que a Classe das Linguagens Recursivas é fechada para as operações de união, intersecçao *e* diferença.

 Inicialmente, vamos demonstrar para a operação sobre duas linguagens recursivas. Sejam L_1 *e* L_2 *tais linguagens.*

- **UNIÃO**

$ACEITA(L_1) \cup ACEITA(L_2) = L_1 \cup L_2 = ACEITA(L_1 \cup L_2)$
$REJEITA(L_1) \cup REJEITA(L_2) = (\Sigma^* - L_1) \cup (\Sigma^* - L_2) = \Sigma^* - (L_1 \cup L_2) = REJEITA\ (L_1 \cup L_2)$
$LOOP(L_1) = \varnothing$ *e* $LOOP(L_2) = \varnothing$, *então*
$LOOP(L_1) \cup LOOP(L_2) = \varnothing$

- **INTERSECÇÃO**

$ACEITA(L_1) \cap ACEITA(L_2) = L_1 \cap L_2 = ACEITA(L_1 \cap L_2)$
$REJEITA(L_1) \cap REJEITA(L_2) = (\Sigma^* - L_1) \cap (\Sigma^* - L_2) = REJEITA\ (L_1 \cap L_2)$
$LOOP(L_1) = \varnothing$ *e* $LOOP(L_2) = \varnothing$, *então*
$LOOP(L_1) \cap LOOP(L_2) = \varnothing$

- **DIFERENÇA**

$ACEITA(L_1) - ACEITA(L_2) = L_1 - L_2 = ACEITA(L_1 - L_2)$
$REJEITA(L_1) - REJEITA(L_2) = (\Sigma^* - L_1) - (\Sigma^* - L_2) = \Sigma^* - (L_1 - L_2) = REJEITA\ (L_1 - L_2)$
$LOOP(L_1) = \varnothing$ *e* $LOOP(L_2) = \varnothing$, *então*
$LOOP(L_1) - LOOP(L_2) = \varnothing$

 Vamos, agora, ampliar a demonstração para n linguagens recursivas. Utilizaremos o Método de demonstração por Indução. Sejam $L_1, ..., L_n$ *tais linguagens.*

Base de indução

n = 2: $ACEITA(L_1) \cup ACEITA(L_2) = ACEITA(L_1 \cup L_2)$
$REJEITA(L_1) \cup REJEITA(L_2) = REJEITA\ (L_1 \cup L_2)$
$LOOP(L_1) \cup LOOP(L_2) = \varnothing$

Hipótese de indução

n = k: $ACEITA(L_1) \cup ... \cup ACEITA(L_K) = ACEITA(L_1 \cup ... \cup L_K)$
$REJEITA(L_1) \cup ... \cup REJEITA(L_K) = REJEITA\ (L_1 \cup ... \cup L_K)$
$LOOP(L_1) \cup ... \cup LOOP(L_K) = \varnothing$

Passagem de indução

n = k+1:

$(ACEITA(L_1) \cup ... \cup ACEITA(L_K)) \cup ACEITA(L_{K+1}) \overset{\text{hipótese de indução}}{=} ACEITA\ (L_1 \cup ... \cup L_K \cup L_{K+1})$

$(REJEITA(L_1) \cup ... \cup REJEITA(L_K)) \cup REJEITA(L_{K+1}) \overset{\text{hipótese de indução}}{=} REJEITA\ (L_1 \cup ... \cup L_K \cup L_{K+1})$

$(LOOP(L_1) \cup ... \cup LOOP(L_K)) \cup L_{K+1} = \varnothing$

A demonstração para as operações de intersecção *e* diferença *são feitas de maneira análoga.*

Exercício 3.21. Dê a Máquina de Post MP, sobre o alfabeto {a, b}, tal que:
ACEITA(MP) = {w | w tem o mesmo número de símbolos a e b}
REJEITA(MP) = {w | W cuja diferença entre o número de símbolos a e b é 1}
LOOP(MP) = {a, b}* - (ACEITA(MP) ∪ REJEITA(MP))

Por exemplo:
ab ∈ ACEITA(MP)
aba ∈ REJEITA(MP)

aaba ∈ LOOP(MP)

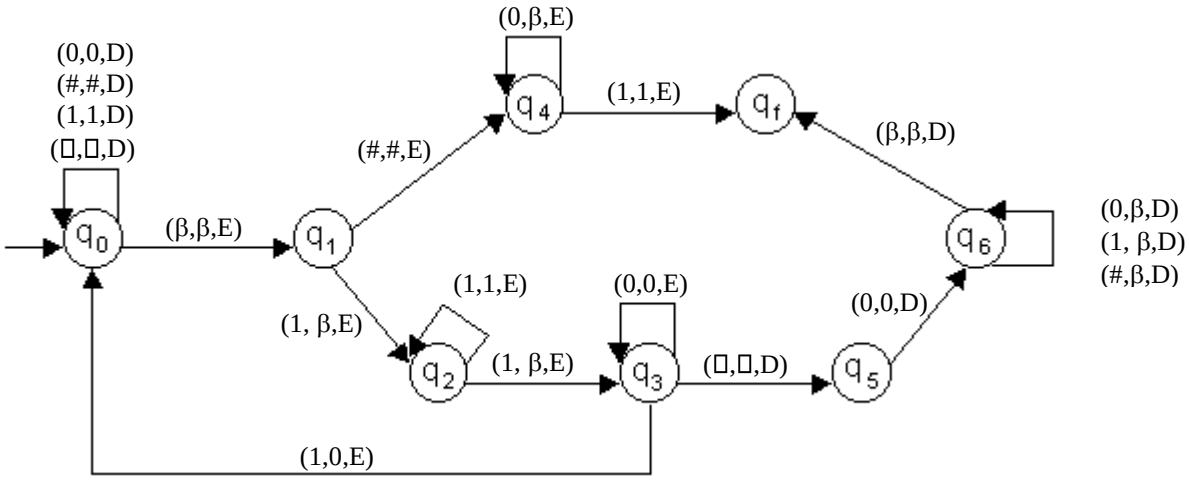
Exercício 3.22. Desenvolva Máquinas de Turing que processem as funções:

a) Subtração, definida por:

m – n, se m > n
zero, caso contrário

Arquivos: exerc22a.gte e Exercicio 22a.mte

Seja M = ({0 ,1 ,#} , {q_0 , q_1 , q_2 , q_3 , q_4 , q_5 , q_6} , Π , q_0 , {q_6} , {∅} , β , □)



b) Fatorial de n, para n ∈ N

Exercício 3.23. Codifique o fluxograma ilustrado na Figura 3.38 como um número natural, usando a codificação de programas introduzida no Exemplo 3.2.

Instruções Rotuladas do programa:
1: faça F vá para 2
2: se T então vá para 1 senão vá para 0

As instruções rotuladas correspondem às seguintes quádruplas:
1: (0, 1, 2, 2)
2: (1, 1, 1, 0)

que correspondem aos seguintes números naturais:
1: $2^0.3^1.5^2.7^2 = 3675$
2: $2^1.3^1.5^1.7^0 = 30$

Dessa forma, a codificação do programa é:
$P = (2^{3675}) \bullet (3^{30})$

Exercício 3.24. Faça uma comparação entre o poder computacional das classes de máquinas de Turing, de Post e dos Autômatos com Duas Pilhas e, então, responda:

a) Dê suas principais características;

	MEMÓRIA DE TRABALHO	***ENTRADA***	***SAÍDA***	***PROGRAMA***
MÁQUINA DE TURING	*Fita finita à esquerda e infinita à direita, dividida em células, onde cada uma armazena um símbolo*	*A própria fita, onde a palavra de entrada ocupa as células mais à esquerda*	*A própria fita*	*Programa ou Função de Transição define o estado da máquina e comanda as leituras, gravações e o sentido de movimento da cabeça*
MÁQUINA DE POST	*Variável X, que é uma memória do tipo fila, sem tamanho ou limites fixos*	*Variável X*	*Variável X*	*Seqüência finita de instruções, representado como um diagrama de fluxos, onde cada vértice é uma instrução (partida, parada, desvio ou atribuição)*
AUTÔMATO COM DUAS PILHAS	*Pilhas (variáveis Y), que não possuem tamanho máximo e podem ser usadas para leitura e gravação*	*Fita (variável X)*	*Variáveis Y*	*Comanda a leitura da fita, leitura e gravação das pilhas e define o estado da máquina*

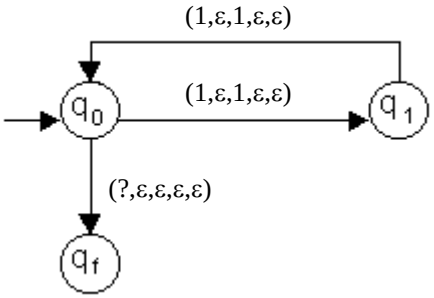
b) Quais classes de máquinas são equivalentes? Não é necessário demonstrar. Apenas justifique sua resposta.

Essas três classes de máquinas são Máquinas Universais e, portanto, são equivalentes. Não há programa que possa ser computado em uma que não seja computado nas outras. Uma máquina simula a outra. Logo, a Máquina de Turing, a máquina de Post e o Autômato de Duas pilhas possuem o mesmo poder computacional.

c) Diga se o programa de cada classe de máquina é parcial ou total. Justifique sua resposta.

Para todas as classes de máquinas acima o programa é parcial, pois em todas é perfeitamente possível ficar em loop infinito.

Exercício 3.25. Elabore um Autômato com Duas Pilhas sobre o alfabeto de entrada {1}. Suponha que as palavras de entrada são números naturais representados em unário, como no Exemplo 3.20, onde, por exemplo, 3 é denotado por 111. O autômato deve aceitar os naturais pares (e rejeitar os ímpares).



Obs.: não é necessária a utilização das duas pilhas.

Exercício 3.26. Relativamente à Hipótese de Church, justifique:

a) Quais as implicações da Hipótese de Church?

Com a Hipótese de Church, podemos definir as classes das funções que são computáveis e não-computavéis. Ela afirma que a Máquina de Turing representa a capacidade máxima de computação que pode ser atingido por qualquer dispositivo de computação. Ou seja, toda função computável é resolvida na Máquina de Turing e, toda função que não pode ser resolvida na Máquina de Turing, não é computável.

b) Por que ela é chamada de Hipótese de Church ao invés de Teorema de Church?

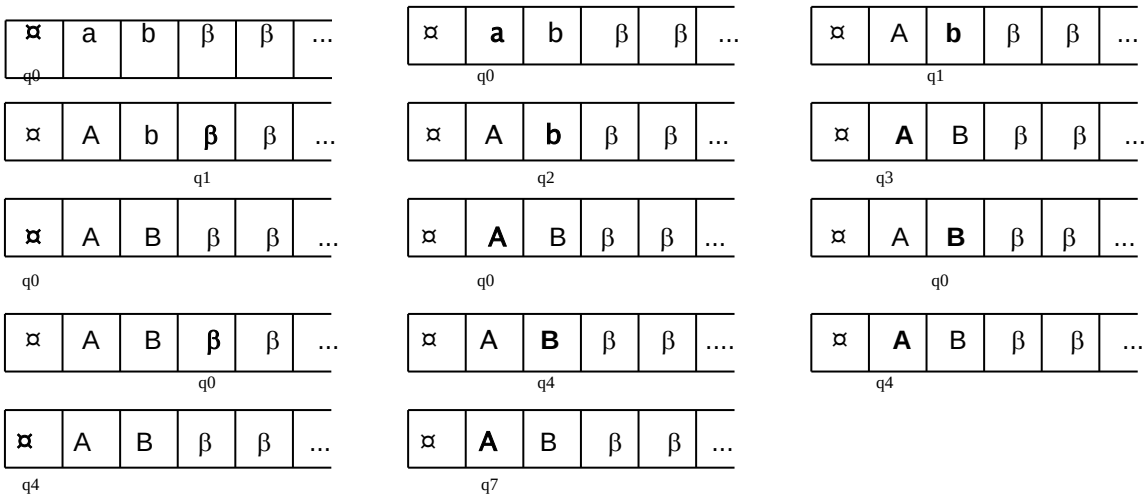
Hipótese de Church não pode ser chamada de Teorema de Church porque não pode ser demonstrada, uma vez que é baseada em uma noção intuitiva de função computável, e sabemos que todo teorema deve ser demonstrado.

Exercício 3.27. Considere a Máquina de Turing cuja função programa Π é como na Figura 3.39.

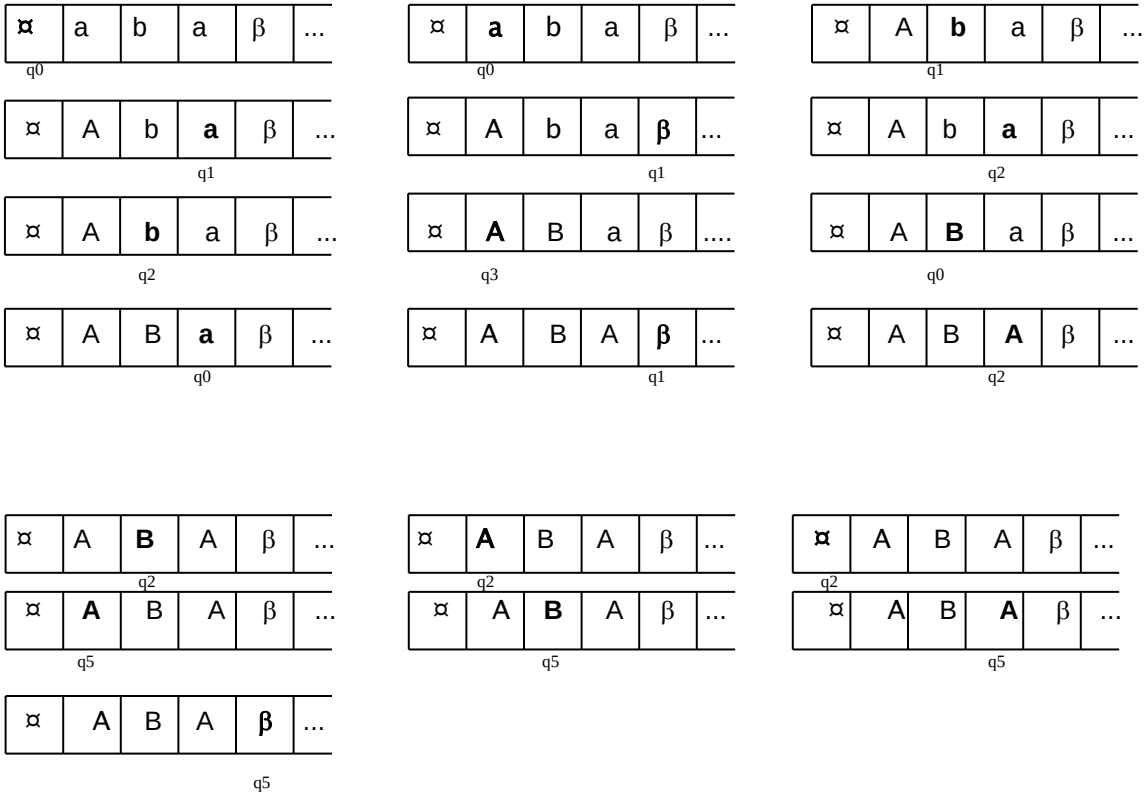
a) Verifique qual o estado final após a computação para as seguintes palavras:

ab
aba
aaba

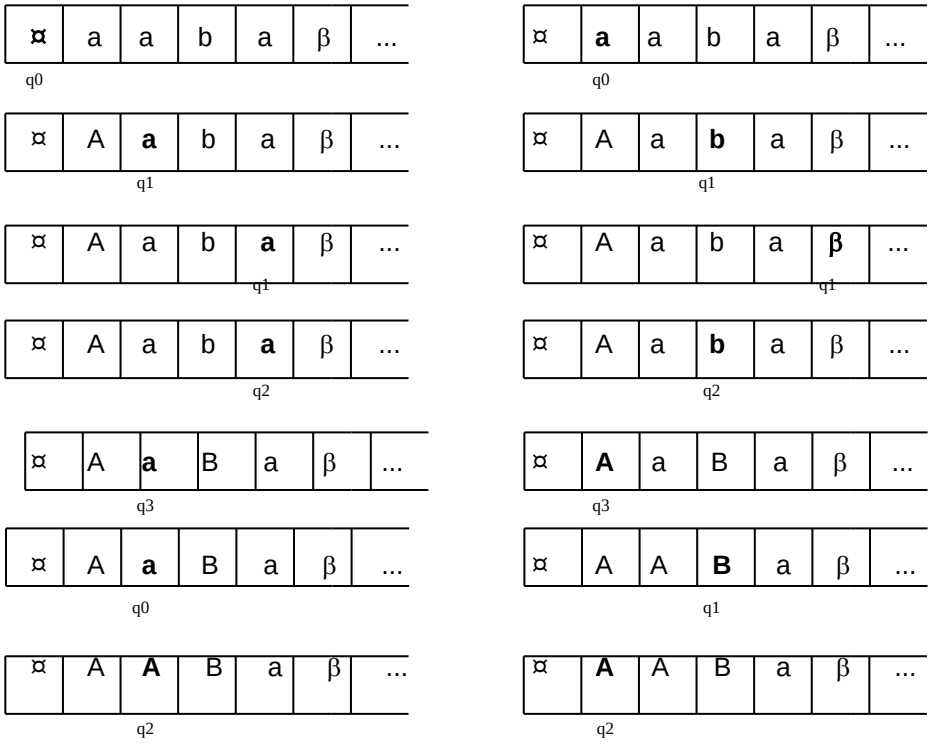
Para a palavra **ab**, *o estado final é* ***aceita****. Verifique a computação:*

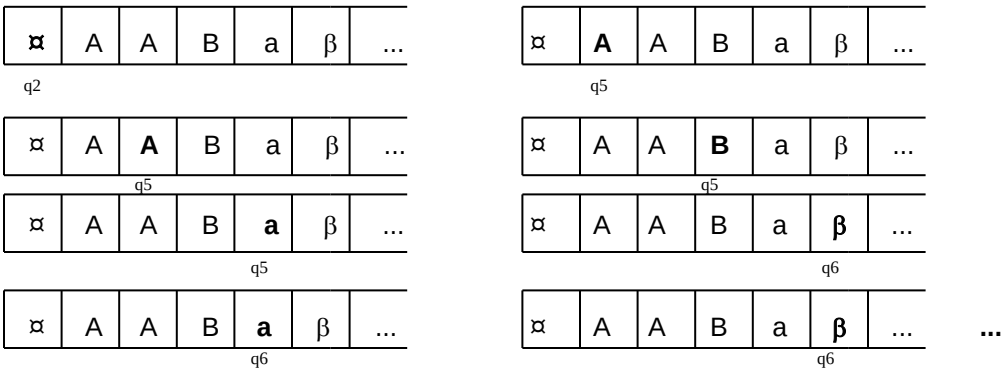


Para a palavra **aba**, *o estado final é* ***rejeita****. Verifique a computação:*



Para a palavra **aaba**, *o estado final é* ***loop***. *Verifique a computação:*





b) A linguagem aceita é enumerável recursivamente ou recursiva?

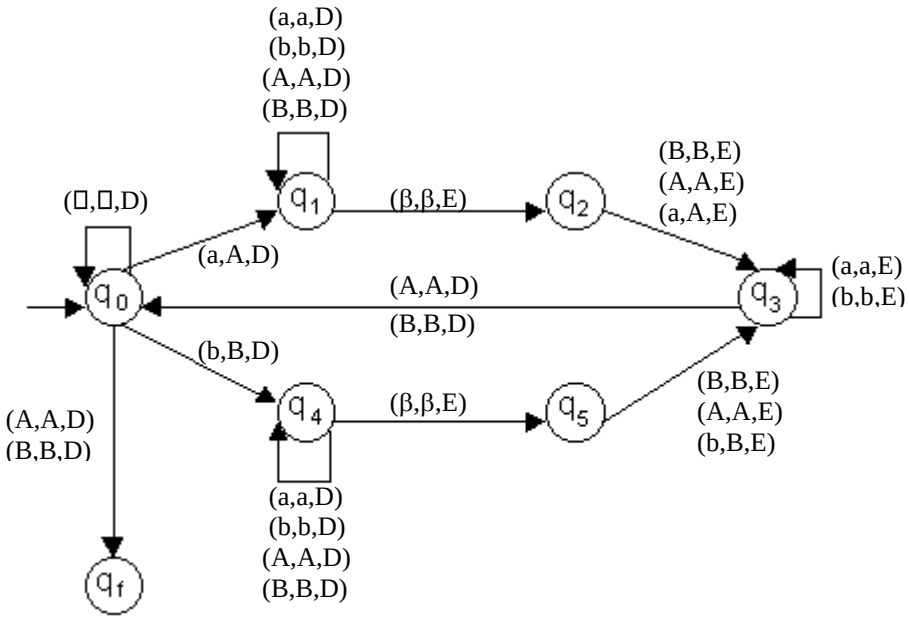
A linguagem pertence à classe das Linguagens Enumeráveis Recursivamente, pois existe uma palavra w que não pertence à linguagem tal que a máquina permanece em loop infinito, processando indefinidamente.

Exercício 3.28. Elabore uma Máquina de Turing MT_Palíndroma (determinística ou não) que sempre para para qualquer entrada e que reconhece todas as palíndromas (palavras que possuem a mesma leitura da esquerda para a direita e vice-versa) sobre o alfabeto {a, b}. Por exemplo:

aba, abba, babab ∈ ACEITA(MT_Palídroma)

abab ∈ REJEITA(MT_Palídroma)

Arquivo: exer28.gte



Exercício 3.29. Elabore uma Máquina de Turing M sobre o alfabeto {a, b} tal que:

ACEITA(M) = {w | w inicia com a e, após cada a, existe pelo menos um b}

LOOP(M) = {w | w ∉ ACEITA(M) e existe pelo menos um b entre dois a}

REJEITA(M) = {a, b}* - (ACEITA(M) ∪ LOOP(M))

Por exemplo:

ab, abbab ∈ ACEITA(MP)

b, aba, abbaba ∈ REJEITA(M)

Exercício 3.30. Elabore uma Máquina de Post P sobre o alfabeto {a, b} tal que:

ACEITA(P) = $\{a^m b^n \mid m > n \geq 0\}$

REJEITA(P) = {a, b}* - ACEITA(P)

Exercício 3.31. Codifique os programas da Figura 3.40 usando a codificação de programas monolíticos.

As Instruções Rotuladas do programa Monolítico M_1 correspondem às seguintes quádruplas:
1: (0, 1,2, 2)
2: (1, 1, 3, 5)
3: (0, 2, 4, 4)
4: (1, 1, 1, 0)
5: (0, 1, 6, 6)
6: (1, 1, 7, 2)
7: (0, 2, 8, 8)
8: (1, 1, 6, 0)

que correspondem aos seguintes números naturais:
1: $2^0.3^1.5^2.7^2 = 3675$
2: $2^1.3^1.5^3.7^5 = 12605250$
3: $2^0.3^2.5^4.7^4 = 13505625$
4: $2^1.3^1.5^1.7^0 = 30$
5: $2^0.3^1.5^6.7^6 = 5514796875$
6: $2^1.3^1.5^7.7^2 = 7350$
7: $2^0.3^2.5^8.7^8 = 20266878520000$
8: $2^1.3^1.5^6.7^0 = 93750$

Dessa forma, a codificação do programa P_1 é:
$P_1 = 2^{3675} \bullet 3^{12605250} \bullet 5^{13505625} \bullet 7^{30} \bullet 11^{5514796875} \bullet 13^{7350} \bullet 17^{20266878520000} \bullet 19^{93750}$

As Instruções Rotuladas do programa Monolítico M_2 correspondem às seguintes quádruplas:
1: (0, 1, 2, 2)
2: (1, 1, 3, 1)
3: (0, 2, 4, 4)
4: (1, 1, 1, 0)

que correspondem aos seguintes números naturais:
1: $2^0.3^1.5^2.7^2 = 3675$
2: $2^1.3^1.5^3.7^1 = 5250$
3: $2^0.3^2.5^4.7^4 = 13505625$
4: $2^1.3^1.5^1.7^0 = 30$

Dessa forma, a codificação do programa P_2 é:
$P_1 = 2^{3675} \bullet 3^{5250} \bullet 5^{13505625} \bullet 7^{30}$

Exercício 3.32. Como, sem perda de generalidade, pode-se supor que a função programa de uma Máquina de Turing seja total?

Pode-se supor que é total porque basta definir um estado final de rejeição, para explicitar a condição de parada com rejeição de entrada. Neste caso a função programa é total pois, para toda a entrada que era indefinida agora é definida e vai para um estado "lixo" não-final.

Exercício 3.33. Modifique a prova do teorema 3.15 – Máquina de Turing ≤ Máquina Norma de tal forma que Norma simule as condições ACEITA e REJEITA de parada da Máquina de Turing.

Na definição de simulação da Máquina de Turing por Máquina Norma, no Teorema 3.15, para qualquer estado final $q_f \in F$ *corresponde uma instrução rotulada* f *que é rótulo final de P.*

A fita é simulada por um arranjo unidimensional X.

Podemos alterar esta definição, de forma que a primeira posição do arranjo indique a situação (ACEITA ou REJEITA) ao final da computação e as demais posições indiquem o conteúdo da fita. Pode-se convencionar, por exemplo, que a uma situação ACEITA, corresponde o valor 0 (zero) na primeira posição do arranjo e a uma situação REJEITA corresponde o valor 1 (um) na primeira posição do arranjo.

Dessa forma, para qualquer situação de ACEITA em Turing, o programa P desvia para um rótulo **a** *que atribui 0 à primeira posição do arranjo X e desvia para a instrução rotulada f.*

Da mesma forma, para qualquer situação de REJEITA em Turing, o programa P desvia para um rótulo **r** *que atribui 1 à primeira posição do arranjo X e desvia para a instrução rotulada f.*