Aula Prolog – Autômato com Pilha

Um autômato com pilha é definido por um 6-upla (Q, A, P, f, q0, F), tal que:

Q é um conjunto de estados .

A é o alfabeto de entrada.

P é o alfabeto da pilha.

f: Q x A x P → Q x P

Descrição da função de transição:
Q é o estado inicial da transição
A é o símbolo da fita que tem que dar match
P é o símblo que deve ser desempilhado
Q é o estado final da transição
P é o símbolo que deve ser empilhado

q0 é o estado inicial.

F é o conjunto de estados finais.

Vamos construir autômato com pilha para reconhecer a linguagem:

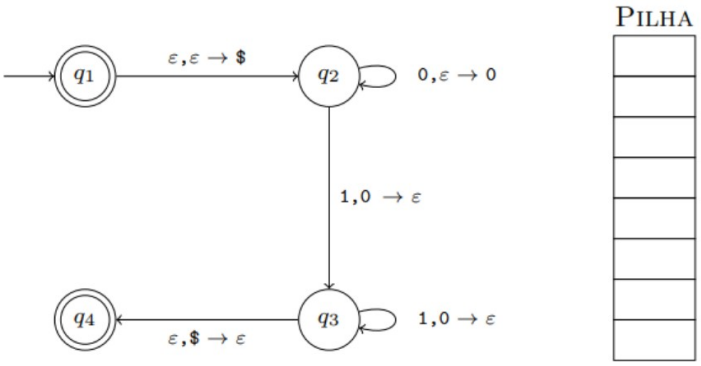
$L = \{ 0^n 1^n \}$

A ideia é que:

1- A cada símbolo 0 que for lido, devemos empilhar um 0.
2- Assim que o símbolo 1 começar a ser lido, devemos desempilhar os zeros.
3- Se a fita de entrada estiver vazia e a pilha estiver vazia, a entrada deve ser aceita.
4- Caso contrário se:
    Ainda há zeros na pilha e a fita estiver vazia, ou
    Acabaram os zeros na pilha e ainda existe uns na pilha, ou
    Foi encontrado um zero na entrada após um 1, então devemos rejeitar.

Q = {q1, q2, q3, q4}
A = {0, 1}
P = {0, $}
f: q1, e, e → q2, $
f: q2, 0, e → q2, 0
f: q2, 1, 0 → q3, e
f: q3, 1, 0 → q3, e
f: q3, e, $ → q4, e
F = {q1, q4}

$$L = \{0^n 1^n\}.$$



Como representar uma função de transição na nossa base de conhecimento?
Termos:
estado inicial da transição
fita
desepilhamento
empilhamento
estado final da transição

```
q(1, 'e', 'e', '$', 2).
q(2, '0', 'e', '0', 2).
q(2, '1', '0', 'e', 3).
q(3, '1', '0', 'e', 3).
q(3, 'e', '$', 'e', 4).
```

Quantos termos deve ter o nosso predicado reconhecedor?

```
teste(X) :- string_chars(X, Fita),
            inicio(No),
            reconhecedor(No, Fita, []), !.
```

Qual a condiçãode parada do nosso reconhecedor?

```
reconhecedor(No, [], []) :- final(No), !.
```

Mas mesmo a fita de entrada estando vazia, eu posso empilha e desempilhar?

```
reconhecedor(De,[], Pilha) :-   q(De, e, L, E, Para),
                                atualiza_pilha(Pilha, L, E, Nova_Pilha),
                                reconhecedor(Para, [], Nova_Pilha).
```

Como eu atualizo a pilha?

```
atualiza_pilha(Pilha, L, D, Nova_Pilha) :- atualiza_leitura(Pilha, L, P1),
                                           atualiza_escrita(P1, D, Nova_Pilha).

atualiza_leitura([L | Pilha], L, Pilha).
atualiza_leitura(Pilha, e, Pilha).


atualiza_escrita(Pilha, E, [E | Pilha]) :- E \== e, !.
atualiza_escrita(Pilha, e, Pilha).
```

Qual mudança nós podemos fazer para elimirar a comparação E == e?

```
atualiza_pilha(Pilha, L, D, Nova_Pilha) :- atualiza_leitura(Pilha, L, P1),
                                           atualiza_escrita(P1, D, Nova_Pilha).

atualiza_leitura([L | Pilha], L, Pilha).
atualiza_leitura(Pilha, e, Pilha).


atualiza_escrita(Pilha, e, Pilha) :- !.
atualiza_escrita(Pilha, E, [E | Pilha]).
```

Qual a regra recursiva do reconhecedor para movimento vazio?

```
reconhecedor(De,Fita, Pilha) :- q(De, e, L, E, Para),
                                atualiza_pilha(Pilha, L, E, Nova_Pilha),
                                reconhecedor(Para, Fita, Nova_Pilha).
```

Qual a regra recursiva do reconhecedor leitura da fita?

```
reconhecedor(De,Fita, Pilha) :- q(De, X, L, E, Para),
                                X \== e,
                                caminha(X, Fita, Nova_Fita),
                                atualiza_pilha(Pilha, L, E, Nova_Pilha),
                                reconhecedor(Para, Nova_Fita, Nova_Pilha).
```

Como atualizamos a fita?

```
caminha(H,[H | T],T).
```

Código completo

```
q(1, 'e', 'e', '$', 2).
q(2, '0', 'e', '0', 2).
q(2, '1', '0', 'e', 3).
q(3, '1', '0', 'e', 3).
q(3, 'e', '$', 'e', 4).

inicio(1).
final(4).

teste(X) :- string_chars(X, Fita),
        inicio(No),
        reconhecedor(No, Fita, []), !.

reconhecedor(No, [], []) :- final(No), !.

reconhecedor(De,[], Pilha) :-   q(De, e, L, E, Para),
                                atualiza_pilha(Pilha, L, E, Nova_Pilha),
                        reconhecedor(Para, [], Nova_Pilha).

reconhecedor(De,Fita, Pilha) :- q(De, X, L, E, Para),
                                X \== e,
                        caminha(X, Fita, Nova_Fita),
                                atualiza_pilha(Pilha, L, E, Nova_Pilha),
                        reconhecedor(Para, Nova_Fita, Nova_Pilha).

reconhecedor(De,Fita, Pilha) :- q(De, e, L, E, Para),
                              atualiza_pilha(Pilha, L, E, Nova_Pilha),
                        reconhecedor(Para, Fita, Nova_Pilha).

caminha(H,[H | T],T).


atualiza_pilha(Pilha, L, D, Nova_Pilha) :- atualiza_leitura(Pilha, L, P1),
                                           atualiza_escrita(P1, D, Nova_Pilha).

atualiza_leitura([L | Pilha], L, Pilha).
atualiza_leitura(Pilha, e, Pilha).


atualiza_escrita(Pilha, E, [E | Pilha]) :- E \== e, !.
atualiza_escrita(Pilha, e, Pilha).
```