Trabalho 2: Autômatos com pilha

Laura Cruz Quispe
Jarlinton Moreno Zea
SCC5832: Teoria da Computação
Prof. Diego Raphael Amancio

September 25, 2017

1 Definição

Um ACP é uma sétupla $\mathrm{M} = (\mathrm{Q}, \sigma, \gamma, \delta, \mathrm{q}_0, \mathrm{z}_0, \mathrm{F})$ onde:

- Q é um conjunto finito de estados.
- Σ é um alfabeto finito chamado **alfabeto de entrada**.
- Γ é um alfabeto finito chamado **alfabeto da pilha**.
- δ é um mapeamento de:

  $$\mathrm{Q} \times (\Sigma \cup \lambda) \times \Gamma \Rightarrow \mathrm{Q} \times \Gamma^*$$

  Onde Γ^* é o topo da pilha e λ é a cadeia vazia.
- $\mathrm{q}_0 \in Q$ é o estado inicial. A máquina começa nele.
- $\mathrm{z}_0 \in \Gamma$ é o símbolo inicial da pilha. Aparece inicialmente na pilha.
- F é o conjunto de estados finais $\mathrm{F} \subseteq \mathrm{Q}$

1.1 Interpretação de transições (δ)

Na função de transições se tem dos tipos de movimento:

1. $$\delta(\mathrm{q}, \mathrm{a}, \mathrm{z}) \subseteq \{(\mathrm{p}_1, \gamma_1), (\mathrm{p}_2, \gamma_2), \cdots (\mathrm{p}_m, \gamma_m)\}$$

   Onde $q, p_i \in Q$, $a \in \Sigma$, $z \in \Gamma^*$.
   ACP no estado q, com símbolo de entrada a e z no topo da pilha pode, para qualquer i, mudar para o estado p_i, substituir z por γ_i (substitui por uma cadeia de símbolos da pilha) e avançar a cabeça de leitura.

2. $$\delta(\mathrm{q}, \lambda, \mathrm{z}) \subseteq \{(\mathrm{p}_1, \gamma_1), (\mathrm{p}_2, \gamma_2), \cdots (\mathrm{p}_m, \gamma_m)\}$$

   ACP no estado q, com z no topo da pilha pode, para qualquer i, mudar para o estado p_i, substituir z por γ_i. Não espera símbolo de entrada e não avança a cabeça de leitura.

No caso da substituição:

- Se $\gamma_i = \lambda$ então há desempilhamento.
- Se $\gamma_i = z$ então a pilha fica inalterada
- Se $\gamma_i = yz$ então y é empilhado.

2 Implementação

Após de apresentar una definição de autômato em pilha, segundo descrevemos o processo de implementação. Este trabalho é utilizado como linguagem de programação PYTHON. Primeiro, temos uma leitura de arquivo de entrada chamado ”in_pila.txt´´. Este arquivo define a estrutura de autômato para isso é chamado a classe ”ReaderAutomaton´´ e como resultado é definido cada variável do autômato $(\mathrm{Q}, \sigma, \gamma, \delta, \mathrm{q_0}, \mathrm{z_0}, \mathrm{F})$
Segundo, uma vez construído nosso autômato por la classe ”StackAutomaton´´ e chamado o método ”evaluate´´ que é apresentado como o Algoritmo 1 para Autômato de Pila determinístico e o Algoritmo 2 para Autômato de Pila não Determinístico que recebe os casos de prova(cadeias). Neste método, nós usamos uma estrutura de mapeamento que representa as transições, e uma pilha, e dois conjuntos para guardar os estados atuais e os estados novos depois da uma transição processado uma letra da cadeia. A estrutura de mapeamento é para facilitar o aceso para as regras de transições construido no processo de ”mapper_transition´´. O método ”evaluate´´ retorna a palavra ”aceita´´ o ”rejeita´´ como a cadeia avaliada.
Em quanto a eficiência da solução em termos de espaço e tempo, nosso algoritmo tem $\mathrm{O} = n * m$, para ambos casos. Onde n é o comprimento da cadeia de entrada e m é o comprimento de possíveis estados para cada letra da cadeia de entrada como máximo m é o total de número de transições definidas para o autômato com pilha.

Data: $\mathrm{M} = (\mathrm{Q}, \Sigma, \Gamma, \delta, \mathrm{q_0}, \mathrm{z_0}, \mathrm{F})$, cadeia de entrada
Result: (Aceita/Rejeita) cadeia
Inicialização;
PUSH $\mathrm{z_0}$ na pilha ;
DEFINE conjunto atual de estados A;
ADD $\mathrm{q_0}$ no conjunto A;
ADD λ no final da cadeia;
for *cada simbolo a na cadeia* **do**
  Supondo $X \in \Gamma$ esta no tope da pilha;
  POP X;
  DEFINE conjunto de novos estados A';
  for *cada q_i no conjunto A* **do**
    if *há uma regra de transição em δ tal que $\delta(q_i, a, X) = (q, \Gamma^*)$* **then**
      if *Γ^* é distinto de vazio λ* **then**
        PUSH Γ^* na pilha;
      end
      ADD q no conjunto A';
    else
      PUSH X;
      Não há uma transição para simbolo a no estado q_i, e termina os bucles.
    end
  end
  UPDATE conjunto atual de estados A com os novos estados A'
end
if *Se pilha esta vazia* **then**
  Aceita cadeia.
else
  Se no rejeita cadeia.
end
;

Algorithm 1: Algoritmo Proposto: Automata na pilha determinístico

2.1 Provas

O lenguagem utilizada que prova o Autômato de Pila Determinístico é: wcw^* descrito no arquivo 'apd_1.txt', e o lenguagem regular utilizada que prova o Autômato Não Determinístico é $a^n b^n$

Data: $\mathrm{M} = (\mathrm{Q}, \Sigma, \Gamma, \delta, \mathrm{q}_0, \mathrm{z}_0, \mathrm{F})$, cadeia de entrada
Result: (Aceita/Rejeita) cadeia
Inicialização;
PUSH z_0 na pilha ;
DEFINE conjunto atual de estados A;
ADD q_0 no conjunto A;
ADD λ no final da cadeia;
for *cada simbolo a na cadeia* **do**
    Supondo $X \in \Gamma$ esta no tope da pilha;
    POP X;
    DEFINE conjunto de novos estados A';
    for *cada q_i no conjunto A* **do**
        if *há uma regra de transição em δ tal que* $\delta(q_i, a, X) = (q, \Gamma^*)$ **then**
            if Γ^* *é distinto de vazio* λ **then**
                PUSH Γ^* na pilha;
            end
            ADD q no conjunto A';
        else
            LOOK transições vazias, atualiza novos estados e volta a lé o mesmo simbolo a.
             PUSH X;
            Não há uma transição para simbolo a no estado q_i, e termina os bucles.
        end
    end
    UPDATE conjunto atual de estados A com os novos estados A'
end
if *Se pilha esta vazia* **then**
    Aceita cadeia.
else
    Se no rejeita cadeia.
end
;

Algorithm 2: Algoritmo Proposto: Automata na pilha não determinístico

descrito no arquivo 'apnd_1.txt'. Para testar o código de APD: ´´python automata_pila.py" e APND: ´´python automata_pila_non.py", ambos carregam por defeto os arquivos das provas.

3 Conclusões

Nossa proposta esta baseado num algoritmo iterativo, tem complexidade $\mathrm{O} = n * m$ e $\mathrm{T} = n(3 + 4 * m)$. Mas para um trabalho futuro pode ser avaliado com um algoritmo recursivo utilizando estrutura de arvores.