Minimização de um Autômato Finito

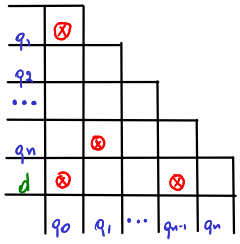
↳ objetivo: gerar um AF equivalente com o menor número de estados possível.

↳ Pré-requisitos:

* AF deve ser determinístico
* não pode ter estados inacessíveis (não atingíveis a partir do estado inicial).
* ter uma função programa total

⚠ caso não satisfaça deve-se gerar um AFD equivalente, eliminar estados inacessíveis e, se necessário, introduzir um estado "morto" para tornar a função programa total.

↳ Algoritmo de minimização:



* Passo 1: construir a tabela
* Passo 2: marcar na tabela os pares obviamente distintos (⊗), $\{q_i, q_j\}$ onde um é final e o outro não é final
* Passo 3: para cada par $\{q_i, q_j\}$ não marcado e para cada $a \in \Sigma$:
  ↳ calcule $\delta(q_i, a) = p_i$ e $\delta(q_j, a) = p_j$
  ↳ Se $p_i = p_j$ não marcar, q_i é equivalente a q_j para o símbolo a
  ↳ Se $p_i \neq p_j$ e o par $\{p_i, p_j\}$ é marcado, então $\{q_i, q_j\}$ deve ser marcado com X.
  ↳ Se $p_i \neq p_j$ e o par $\{p_i, p_j\}$ não está marcado, adicione $\{q_i, q_j\}$ a uma lista de espera associado a $\{p_i, p_j\}$.
* Passo 4: agrupar estados equivalentes, todos os pares não marcados são estados equivalentes e podem ser unidos em um único estado.
* Passo 5: para cada grupo de estados equivalentes, crie um novo estado único.

Propriedades das Linguagens Regulares

↳ Lema do Bombeamento: é uma proposição útil para o estudo das propriedades das LR. Afirma que se uma linguagem L é regular, existe um número n (o número de estados do AFD que a aceita) tal que, para qualquer palavra w em L com comprimento $|w| \geq n$, w pode ser dividida em três subpalavras $w = uvz$. As condições são:

↳ $|uv| \leq n$
↳ $|v| \geq 1$ v não é palavra vazia
↳ para todo $i \geq 0$ $uv^i z$ também pertence a L

↳ Teorema: Se L é LR então: existe n tal que:

↳ para qq $w \in L$ onde $|w| \geq n$, w pode ser definida como $w = uvz$, onde $|uv| \leq n$ e $|v| \geq 1$, para todo $i \geq 0$, $uv^i z \in L$.

Verificação se uma LR é vazio, finito ou infinito

↳ Teorema: se L é uma LR aceita por um AF M com n estados, então L é:

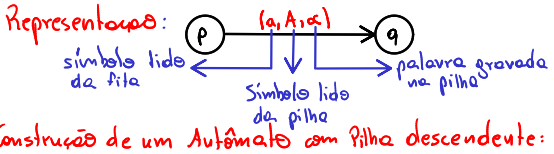
↳ vazio: sse M não aceita qualquer w tq $|w| < n$
↳ finito: sse M não aceita w tq $n \leq |w| < 2n$
↳ infinito: sse M aceita w tq $n \leq |w| < 2n$

Autômato com Pilha

↳ Estrutura e componentes: 6-upla $M = (\Sigma, Q, \delta, q_0, F, V)$

* Σ, Q, q_0, F igual à um AFD.
* $\delta: Q \times (\Sigma \cup \{\varepsilon, ?\}) \times (V \cup \{\varepsilon, ?\}) \to 2^{Q \times V^*}$, função parcial
  ↳ ?: indica que toda a palavra de entrada foi lido e pilha vazia.
  ↳ ε: movimento vazio e nenhuma gravação é realizada na pilha.
* V: alfabeto da pilha.

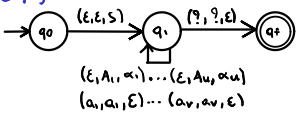
↳ Representação:



↳ Construção de um Autômato com Pilha descendente:

↳ Seja $G_1 = (V, T, P, S)$, GLC e sem recursão à esquerda
↳ $M = (T, \{q_0, q_1, q_f\}, \delta, q_0, \{q_f\}, V \cup T)$ onde

* $\delta(q_0, \varepsilon, \varepsilon) = \{(q_1, S)\}$
* $\delta(q_1, \varepsilon, A) = \{(q_1, \alpha) \mid A \to \alpha \in P\}$
* $\delta(q_1, a, a) = \{(q_1, \varepsilon)\}$
* $\delta(q_1, ?, ?) = \{(q_f, \varepsilon)\}$



↳ Eliminando recursividade à esquerda

↳ Para: $A \to A\alpha_1 \mid A\alpha_2 \mid \ldots \mid A\alpha_n \mid \beta_1 \mid \beta_2 \mid \ldots \mid \beta_m$, onde as partes do tipo $A\alpha_i$ são recursivas à esquerda e β_i são produções não recursivas.
↳ Faz-se:

$A \to \beta_1 A' \mid \beta_2 A' \mid \ldots \mid \beta_m A'$
$A' \to \alpha_1 A' \mid \alpha_2 A' \mid \ldots \mid \alpha_n A' \mid \varepsilon$

Forma Normal de Greibach (FNG)

↳ Todas as produções tem a forma: $A \to a\alpha$ onde:

* A = variável (não terminal)
* a = símbolo terminal
* α = (zero ou mais) variáveis $\Rightarrow \alpha \in V^*$

↳ Não permite:

* $A \to \varepsilon$ (vazias)
* $A \to B$ (unitárias)
* $A \to aBb$

Forma normal de Chomsky (FNC): Uma gramática $G = (V, T, P, S)$ está na FNC se todas as regras de produções em P forem de um dos dois tipos:

I) $A \to BC$, onde A, B e C são variáveis, e nem B nem C podem ser o símbolo vazio.
II) $A \to a$, onde A é uma variável e $a \in T$

obs: $S \to \varepsilon$, apenas se $\varepsilon \in L(G_1)$

↳ Simplificações combinadas

↳ Sequência Recomendada:
I) Exclusão produções vazias
II) Exclusão forma $A \to B$
III) Exclusão símbolos inúteis

Algoritmo Cocke-Younger-Kasami (CYK)

1. Inicialização ($s=1$)
   - Para cada posição r de 1 à n:
     - Coloque em $V[r][1]$ todos os $A \mid A \to a_r \in P$
2. Construção da tabela de $s=2$ até n:
   - Para cada tamanho s (2 a n):
     - Para cada início r (1 até $n-s+1$):
       - Para cada k de 1 até $s-1$
         - Para cada produção $A \to BC$:
           - Se $B \in V[r][k]$ e $C \in V[r+k][s-k]$, então $A \in V[r][s]$.
3. Aceitação, w é aceito se o símbolo inicial S estiver em $V[1][n]$.

Simplificação de Gramáticas

↳ Tipos de simplificação:

I) Símbolos inúteis: variáveis ou terminais não usados na geração de palavras de terminais.

↳ Algoritmo: GLC $G=(V,T,P,S)$

↳ Etapa 1: Remover variáveis que não geram terminais

- Passo 1: Crie um conjunto $V_1 = \emptyset$
- Passo 2: Repita até V_1 não mudar:
  - Para cada produção $A \to \alpha$, se todos os símbolos de α estão em $T \cup V_1$, adicione A a V_1.
- Passo 3: Elimine de P todos os produções com variáveis fora de V_1.

Resultado: $G_1 = (V_1, T, P_1, S)$

↳ Etapa 2: Remover símbolos inalcançáveis

- Passo 1: Crie o conjunto $V_2 = \{S\}$ e $T_2 = \emptyset$
- Passo 2: Repita até parar de crescer
  - Para cada produção $A \to \alpha$, com $A \in V_2$
    - Se α contém variável β, adicione β em V_2
    - Se α contém terminal a, adicione a em T_2
- Passo 3: elimine de P_1, todas as produções que usam símbolos fora de $V_2 \cup T_2$
- Resultado: $G_2 = (V_2, T_2, P_2, S)$

II) Exclusão de Produções Vazias: objetivo eliminar todas as produções do tipo $A \to \varepsilon$ (exceto $S \to \varepsilon$, se necessário)

↳ Algoritmo: GLC $G=(V,T,P,S)$

↳ Etapa 1: encontrar variáveis que geram ε.

- Passo 1: inicializa $V_\varepsilon = \{A \mid A \to \varepsilon\}$;
- Passo 2: enquanto estiver crescendo:
  - adicione X se existe produção $X \to X_1 X_2 \dots X_n$ tal que $X_i \in V_\varepsilon$ (ou seja, X pode gerar indiretamente ε)

↳ Etapa 2: Criar novas produções sem usar $A \to \varepsilon$

- Passo 1: copiar todas as produções não vazias $P_1 = \{A \to \alpha \mid \alpha \neq \varepsilon \text{ e } \alpha \notin V_\varepsilon\}$
- Passo 2: para cada produção que contém variáveis em V_ε crie versões alternativas com essas variáveis omitidas

Exemplo: se $A \to BCD$ e $C \in V_\varepsilon$, então adiciono $A \to BD$.

Repito até P_1 não crescer mais.

↳ Etapa 3: Verificar se ε deve ser mantido, se $\varepsilon \in L(G_1)$ então adicione $S \to \varepsilon$.

III) Eliminar produções $A \to B$: B não acrescenta nada além de redirecionar.

↳ Algoritmo: GLC $G_1 = (V,T,P,S)$

↳ Passo 1: construir fecho de cada variável

- Para cada $A \in V$:
  - Fecho(A) = $\{B \mid A \Rightarrow^* B\}$, usando apenas produções do tipo $X \to Y$.

↳ Passo 2: Construir as novas produções

- copia todas as produções $A \to \alpha$, onde α não é só uma variável (ou seja, $\alpha \in V$)
  - Para cada $B \in$ Fecho(A):
    - Se $B \to \alpha \in P$ e $\alpha \notin V$ (α não é produção unitário) adiciono $A \to \alpha$

Algoritmo de Early

↳ Pseudo-Algoritmo: $D_0 = \emptyset$

para toda produção $S \to \alpha \in P$:
faça $D_0 = D_0 \cup \{S \to .\alpha \,/\, 0\}$
para toda produção $A \to .B\beta / 0 \in D_0$:
  faça para toda produção $B \to .\phi \in P$:
    faça $D_0 = D_0 \cup \{B \to .\phi / 0\}$
até não ocorrer mais inclusões

} D_0

↳ Condição de aceitação: a palavra de entrada w é aceito se uma produção $S \to \alpha . \,/\, 0$ (indicando que a derivação partiu do símbolo inicial 'S', foi incluído em D_0, e todo o lado direito da produção foi analisado com sucesso) pertencer ao último conjunto de reduções, D_n.

$n = |w|$
para r variando de 1 até n
para $D_r = \emptyset$:
  para toda $A \to \alpha . a_r \beta / s \in D_{r-1}$: // gera símbolo a_r
    faça $D_r = D_r \cup \{A \to \alpha a_r . \beta / s\}$
  repito
    para toda $A \to \alpha . B\beta / s \in D_r$:
      faça para toda $B \to \phi \in P$:
        faça $D_r = D_r \cup \{B \to .\phi / r\}$
    para toda $A \to \alpha . / s$ de D_r
      faça para toda $B \to \beta . A\phi / k \in D_s$
        faça $D_r = D_r \cup \{B \to \beta A . \phi / k\}$
  até não ocorrer mais inclusões

} D_1 a D_n