

Linguagens Formais e Autómatos

Vasco Pedro

Departamento de Informática

Universidade de Évora

2008/2009

Alfabeto, Palavra

alfabeto – conjunto finito de **símbolos** (Σ, T)
(elementos a, b, c, d, e)

Exemplos:

- $\{a, b, c, \dots, x, y, z\}$
- $\{0, 1, \dots, 9, +, -, \div, \times, (,)\}$
- $\{\text{InsereCartão}, 0, 1, \dots, 9, \text{Confirmar}, \text{Corrigir}, \text{Anular}, \dots\}$

palavra sobre o alfabeto Σ – sequência finita de símbolos de Σ (p, q, u, v, w, x, y, z)

λ – palavra **vazia** (também ϵ e ε)

Linguagem

Σ^* – conjunto de **todas** as palavras sobre Σ

Definição recursiva:

(base) $\lambda \in \Sigma^*$

(passo recursivo) se $w \in \Sigma^*$ e $a \in \Sigma$, então
 $wa \in \Sigma^*$

(fecho) $w \in \Sigma^*$ somente se pode ser gerada por um número finito de aplicações do passo recursivo a partir de λ

linguagem sobre o alfabeto Σ – conjunto de palavras sobre Σ ($L \subseteq \Sigma^*$)

Operações sobre Palavras

$|w|$ – **comprimento** da palavra w

a **concatenação** de duas palavras $u, v \in \Sigma^*$, escrita $u.v$ ou uv , é uma operação binária em Σ^* definida como:

1. se $|v| = 0$, então $v = \lambda$ e $u.v = u$
2. se $|v| = n > 0$, então $v = wa$, para alguma palavra w com $|w| = n - 1$ e algum $a \in \Sigma$, e $u.v = (u.w)a$

a **inversão** de $u \in \Sigma^*$, escrita u^R ou u^{-1} , é uma operação unária em Σ^* definida como:

1. se $|u| = 0$, então $u = \lambda$ e $u^R = \lambda$
2. se $|u| = n > 0$, então $u = wa$, para alguma palavra w com $|w| = n - 1$ e algum $a \in \Sigma$, e $u^R = a.w^R$

Subpalavra

u é **subpalavra** de v se existem x, y t.q.

$$v = xuy$$

Prefixo

- se $x = \lambda$ então u é **prefixo** de v

Sufixo

- se $y = \lambda$ então u é **sufixo** de v

$(u, v, x, y \in \Sigma^*)$

Caracterização Finita de Linguagens

- definição recursiva
- através de operações sobre conjuntos

- **concatenação de linguagens**

se X e Y forem linguagens

$$XY = X \cdot Y = \{xy \mid x \in X \text{ e } y \in Y\}$$

- exemplo

$$\begin{aligned} \{1, 2, 3\} \cdot \{1, 00, \lambda\} = \\ \{ 11, 21, 31, \\ 100, 200, 300, \\ 1, 2, 3 \} \end{aligned}$$

Estrela de Kleene

- seja X um conjunto

$$X^* = \bigcup_{n \geq 0} X^n \qquad X^+ = \bigcup_{n > 0} X^n$$

em alternativa, $X^+ = XX^*$

- também conhecido como operador de **fecho** ou de **iteração**
- exemplo
 - linguagem dos números naturais sem zeros à esquerda

$$\{0\} \cup \{1, 2, \dots, 9\} \{0, 1, \dots, 9\}^*$$

Conjuntos Regulares

- os **conjuntos regulares** sobre o alfabeto Σ são definidos como

(base) \emptyset , $\{\lambda\}$ e $\{a\}$, para todo $a \in \Sigma$, são conjuntos regulares sobre Σ

(passo recursivo) sejam X e Y conjuntos regulares sobre Σ ; os conjuntos

$$\begin{array}{c} X \cup Y \\ XY \\ X^* \end{array}$$

são conjuntos regulares sobre Σ

(fecho) X é um conjunto regular sobre Σ somente se puder ser construído através de um número finito de aplicações do passo recursivo a partir dos elementos base

Expressões Regulares (1)

- as **expressões regulares** sobre o alfabeto Σ são definidas como

(base) \emptyset, λ e a , para todo $a \in \Sigma$, são expressões regulares sobre Σ

(passo recursivo) sejam u e v expressões regulares sobre Σ ; as expressões

$$\begin{aligned} &(u \cup v) \\ &(uv) \\ &(u^*) \end{aligned}$$

são expressões regulares sobre Σ

(fecho) u é uma expressão regular sobre Σ somente se puder ser construída através de um número finito de aplicações do passo recursivo a partir dos elementos base

Expressões Regulares (2)

Linguagem Representada

$$L(\emptyset) = \emptyset$$

$$L(\lambda) = \{\lambda\}$$

$$L(a) = \{a\} \quad (a \in \Sigma)$$

$$L(u \cup v) = L(u) \cup L(v)$$

$$L(uv) = L(u)L(v)$$

$$L(u^*) = L(u)^*$$

- duas expressões regulares são **equivalentes** se representam a mesma linguagem

Expressões Regulares (3)

Propriedades

$$\emptyset u = u \emptyset = \emptyset$$

$$\lambda u = u \lambda = u$$

$$\emptyset^* = \lambda$$

$$\lambda^* = \lambda$$

$$u \cup v = v \cup u$$

$$u \cup \emptyset = u$$

$$u \cup u = u$$

$$u^* = (u^*)^*$$

$$u(v \cup w) = uv \cup uw$$

$$u^* = \lambda \cup uu^*$$

$$(u \cup v)w = uw \cup vw$$

$$(uv)^*u = u(vu)^*$$

$$(u \cup v)^* = (u^* \cup v)^*$$

$$= u^*(u \cup v)^* = (u \cup vu^*)^*$$

$$= (u^*v^*)^*$$

$$= (u^*v)^*u^* = u^*(vu^*)^*$$

Autómatos Finitos Deterministas

Um **autômato finito determinista (AFD)** é um tuplo $M = (Q, \Sigma, \delta, q_0, F)$ onde

- Q é um conjunto finito de **estados**;
- Σ é um conjunto finito de símbolos (**alfabeto**);
- δ é a **função de transição**, uma função total de $Q \times \Sigma$ em Q ;
- $q_0 \in Q$ é o **estado inicial** do autômato; e
- $F \subseteq Q$ é o conjunto dos **estados de aceitação**.

Configuração e Computação

Seja $M = (Q, \Sigma, \delta, q_0, F)$ um AFD.

A **configuração** de um AF é um par $[q, w] \in Q \times \Sigma^*$, onde q é o estado corrente do autômato e w é a parte da palavra ainda por processar.

A **computação** de um AFD M para a palavra $w = a_1a_2 \dots a_n \in \Sigma^*$ é a sequência de configurações

$$[s_0, a_1a_2 \dots a_n] \vdash_M [s_1, a_2 \dots a_n] \vdash_M \dots \vdash_M [s_n, \lambda]$$

com

$$s_0 = q_0 \text{ e } s_i = \delta(s_{i-1}, a_i),$$

para $i > 0$.

Linguagem Reconhecida

Seja $M = (Q, \Sigma, \delta, q_0, F)$ um AFD.

A **função de transição estendida** $\hat{\delta} : Q \times \Sigma^* \rightarrow Q$ de um AFD é definida por

$$\hat{\delta}(q, \lambda) = q$$

$$\hat{\delta}(q, a) = \delta(q, a)$$

$$\hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w), a)$$

Uma palavra w é **aceite** pelo AFD sse

$$\hat{\delta}(q_0, w) \in F$$

A **linguagem reconhecida** (ou **aceite**) por M é o conjunto das palavras aceites por M

$$L(M) = \{w \mid \hat{\delta}(q_0, w) \in F\}$$

Dois autómatos finitos são **equivalentes** se reconhecem a mesma linguagem.

Autómatos Finitos Não Deterministas (1)

Um **autômato finito não determinista** é um tuplo $M = (Q, \Sigma, \delta, q_0, F)$ onde

- Q é um conjunto finito de **estados**;
- Σ é um conjunto finito de símbolos (**alfabeto**);
- δ é a **função de transição**, uma função total de $Q \times \Sigma$ em $\mathcal{P}(Q)$;
- $q_0 \in Q$ é o **estado inicial** do autômato; e
- $F \subseteq Q$ é o conjunto dos **estados de aceitação**.

Qualquer autômato finito determinista é um autômato finito não determinista.

Autómatos Finitos Não Deterministas (2)

Seja $M = (Q, \Sigma, \delta, q_0, F)$ um autómato finito não determinista.

Uma palavra w é **aceite** por M se *existe* uma computação que termina num estado de aceitação depois de terem sido processados todos os seus símbolos

$$[q_0, w] \vdash_M^* [q_i, \lambda], \text{ onde } q_i \in F$$

A **linguagem reconhecida** por M é o conjunto das palavras aceites por M

$$L(M) = \left\{ w \mid \begin{array}{l} \text{existe uma computação} \\ [q_0, w] \vdash_M^* [q_i, \lambda] \text{ em que } q_i \in F \end{array} \right\}$$

Autómatos Finitos Não Deterministas com Transições λ

Um **autômato finito não determinista com transições λ (AFND)** é um tuplo $M = (Q, \Sigma, \delta, q_0, F)$ onde

- Q é um conjunto finito de **estados**;
- Σ é um conjunto finito de símbolos (**alfabeto**);
- δ é a **função de transição**, uma função de $Q \times (\Sigma \cup \{\lambda\})$ em $\mathcal{P}(Q)$;
- $q_0 \in Q$ é o **estado inicial** do autômato; e
- $F \subseteq Q$ é o conjunto dos **estados de aceitação**.

Eliminação do Não Determinismo

O λ -**fecho** de um estado q_i é o conjunto de todos os estados alcançáveis através de zero ou mais transições λ a partir de q_i

- $q_i \in \lambda\text{-fecho}(q_i)$
- se $q_j \in \lambda\text{-fecho}(q_i)$ e $q_k \in \delta(q_j, \lambda)$, então $q_k \in \lambda\text{-fecho}(q_i)$
- mais nenhum estado está em $\lambda\text{-fecho}(q_i)$

A **função de transição de entrada** t de um AFND M é uma função de $Q \times \Sigma$ em $\mathcal{P}(Q)$ definida por

$$t(q_i, a) = \bigcup_{q_j \in \lambda\text{-fecho}(q_i)} \lambda\text{-fecho}(\delta(q_j, a))$$

Minimização de Autómatos Finitos Deterministas

Seja $M = (Q, \Sigma, \delta, q_0, F)$ um autómato finito determinista. Dois estados q_i e q_j são **equivalentes** se

$$\hat{\delta}(q_i, u) \in F \equiv \hat{\delta}(q_j, u) \in F$$

para qualquer $u \in \Sigma^*$.

Dois estados equivalentes dizem-se **indistinguíveis**.

Cálculo dos Estados Equivalentes

Seja $M = (Q, \Sigma, \delta, q_0, F)$ um AFD.

1. Seja $P = \{Q \setminus F, F\}$ uma partição de Q .
2. Enquanto existirem

$$p, p' \in P \quad a \in \Sigma \quad q_i, q_j \in p$$

tais que $\delta(q_i, a) \in p'$ e $\delta(q_j, a) \notin p'$, fazer

$$P \leftarrow P \setminus \{p\} \cup \{q \in p \mid \delta(q, a) \in p'\} \\ \cup \{q \in p \mid \delta(q, a) \notin p'\}$$

Este algoritmo calcula a partição P de Q tal que, para quaisquer estados q_i e q_j

- se q_i e q_j pertencem ao mesmo subconjunto, q_i e q_j são equivalentes;
- se q_i e q_j pertencem a subconjuntos distintos, q_i e q_j não são equivalentes.

Construção do AFD Mínimo

1. Calcular os estados equivalentes; seja P a partição determinada.

2. Para todos os $p \in P$ e todos os $a \in \Sigma$, seja q um estado em p e seja p' o elemento de P a que $\delta(q, a)$ pertence; então

$$\delta'(p, a) = p'.$$

3. O AFD **mínimo** (ou **reduzido**) equivalente a M é

$$M' = (P, \Sigma, \delta', q'_0, F')$$

onde

- q'_0 é o elemento de P que contém q_0 ;
- $F' = \{p \in P \mid p \subseteq F\}$.

Composição de Autómatos

Seja $M = (Q, \Sigma, \delta, q_0, F)$ um AFND. Existe um AFND

$$M' = (Q \cup \{q'_0, q_f\}, \Sigma, \delta', q'_0, \{q_f\})$$

equivalente a M em que

- não há transições para o estado q'_0
- o único estado de aceitação é q_f
- não há transições a partir do estado q_f

A função de transição de M' é obtida acrescentando a δ

- $(q'_0, \lambda, \{q_0\})$
- uma transição λ de cada $q \in F$ para q_f

NB: $\{q'_0, q_f\} \cap Q = \emptyset, q'_0 \neq q_f$

Composições

Sejam M_v e M_w dois autómatos finitos nas condições do acetato anterior

$$M_v = (Q_v, \Sigma, \delta_v, q_{0_v}, \{q_{f_v}\})$$

$$M_w = (Q_w, \Sigma, \delta_w, q_{0_w}, \{q_{f_w}\})$$

Definem-se os autómatos finitos seguintes

$$M_{\cdot} = (Q_v \cup Q_w, \Sigma, \delta_{\cdot}, q_{0_v}, \{q_{f_w}\})$$

$$\text{com } \delta_{\cdot} = \delta_v \cup \delta_w \cup \{(q_{f_v}, \lambda, \{q_{0_w}\})\}$$

$$M_{\cup} = (Q_v \cup Q_w \cup \{q_0, q_f\}, \Sigma, \delta_{\cup}, q_0, \{q_f\})$$

$$\text{com } \delta_{\cup} = \delta_v \cup \delta_w \cup \{(q_0, \lambda, \{q_{0_v}, q_{0_w}\}), (q_{f_v}, \lambda, \{q_f\}), (q_{f_w}, \lambda, \{q_f\})\}$$

$$M_{*} = (Q_v \cup \{q_0, q_f\}, \Sigma, \delta_{*}, q_0, \{q_f\})$$

$$\text{com } \delta_{*} = \delta_v \cup \{(q_0, \lambda, \{q_{0_v}, q_f\}), (q_{f_v}, \lambda, \{q_{0_v}, q_f\})\}$$

NB: $\{q_0, q_f\} \cap (Q_v \cup Q_w) = \emptyset$, $q_0 \neq q_f$

Pumping Lemma

Teorema (Pumping Lemma para linguagens regulares) Seja L uma linguagem regular e seja k o número de estados de um AFD que a reconhece. Então qualquer palavra p de L , tal que $|p| \geq k$, pode ser escrita como

$$uvw, \text{ com } |uv| \leq k \text{ e } |v| > 0$$

e

$$uv^i w \in L, \text{ para todo o } i \geq 0.$$

Exemplo de Aplicação do Pumping Lemma Para Linguagens Regulares

$$L = \{a^n b^n \mid n \geq 0\}$$

Se L for uma linguagem regular, existe um AFD que a reconhece.

Sejam k o número de estados desse autômato e $p = a^k b^k$. Qualquer decomposição de p nas condições do *Pumping Lemma* será da forma

$$\begin{array}{ccc} u & v & w \\ a^j & a^l & a^{k-j-l} b^k \end{array}$$

com $j + l \leq k$ e $l > 0$.

Como

$$uv^0w = a^j (a^l)^0 a^{k-j-l} b^k = a^{k-l} b^k \notin L$$

porque $l > 0$ e $k - l \neq k$, L não é uma linguagem regular.

Gramáticas (1)

1. ⟨frase⟩ → ⟨sujeito⟩ ⟨frase-verbal⟩
2. ⟨frase⟩ → ⟨sujeito⟩ ⟨verbo⟩ ⟨compl-directo⟩
3. ⟨sujeito⟩ → ⟨subst-próprio⟩
4. → ⟨artigo⟩ ⟨subst-comum⟩
5. ⟨subst-próprio⟩ → John
6. → Jill
7. ⟨subst-comum⟩ → car
8. → hamburger
9. ⟨artigo⟩ → a
10. → the
11. ⟨frase-verbal⟩ → ⟨verbo⟩ ⟨advérbio⟩
12. → ⟨verbo⟩
13. ⟨verbo⟩ → drives
14. → eats
15. ⟨advérbio⟩ → slowly
16. → frequently

símbolos terminais: John, Jill, hamburger,
car, a, the, drives, eats, slowly, frequently

símbolos não terminais: ⟨frase⟩, ⟨sujeito⟩,
⟨frase-verbal⟩, ⟨verbo⟩, ...

Gramáticas (2)

1. ⟨frase⟩ → ⟨sujeito⟩ ⟨frase-verbal⟩
2. → ⟨sujeito⟩ ⟨verbo⟩ ⟨compl-directo⟩
3. ⟨sujeito⟩ → ⟨subst-próprio⟩
4. → ⟨artigo⟩ ⟨subst-comum⟩
5. ⟨subst-próprio⟩ → John
6. → Jill
7. ⟨subst-comum⟩ → car
8. → hamburger
9. ⟨artigo⟩ → a
10. → the
11. ⟨frase-verbal⟩ → ⟨verbo⟩ ⟨advérbio⟩
12. → ⟨verbo⟩
13. ⟨verbo⟩ → drives
14. → eats
15. ⟨advérbio⟩ → slowly
16. → frequently
17. ⟨adjectivos⟩ → ⟨adjectivo⟩ ⟨adjectivos⟩
18. → λ
19. ⟨adjectivo⟩ → big
20. → juicy
21. → brown
22. ⟨compl-directo⟩ → ⟨adjectivos⟩ ⟨subst-próprio⟩
23. → ⟨artigo⟩ ⟨adjectivos⟩
 ⟨subst-comum⟩

Gramáticas Independentes do Contexto

Uma **gramática independente do contexto (GIC)** é um tuplo $G = (V, \Sigma, P, S)$ onde

- V é o conjunto finito dos símbolos **não terminais** (A, B, C, \dots);
- Σ é o conjunto finito dos símbolos **terminais** (alfabeto);
- $P \subseteq V \times (V \cup \Sigma)^*$ é um conjunto finito de **produções**; e
- $S \in V$ é o **símbolo inicial** da gramática.

NB: $V \cap \Sigma = \emptyset$.

Derivação

Seja $G = (V, \Sigma, P, S)$ uma GIC.

Se $u, v \in (V \cup \Sigma)^*$, $A \in V$ e existe uma produção $A \rightarrow w$ em P , então uAv **deriva directamente** uww

$$uAv \Rightarrow_G uww$$

Se existem $u_0, u_1, \dots, u_n \in (V \cup \Sigma)^*$, $n \geq 0$, tais que

$$u = u_0 \Rightarrow_G u_1 \Rightarrow_G \dots \Rightarrow_G u_n = v$$

então u **deriva** v **em** n **passos**

$$u \xRightarrow{n}_G v$$

Se $u \xRightarrow{n}_G v$ para algum $n \geq 0$, u **deriva** v

$$u \xRightarrow{*}_G v$$

Linguagem Gerada

Seja $G = (V, \Sigma, P, S)$ uma GIC.

O conjunto das **palavras deriváveis a partir de** $v \in (V \cup \Sigma)^*$, $D(v)$, define-se como

$$D(v) = \{w \mid v \xRightarrow{*} w\}$$

A **linguagem gerada por** G , $L(G)$, é o conjunto das palavras sobre Σ^* deriváveis a partir de S

$$L(G) = \{w \mid w \in \Sigma^* \text{ e } S \xRightarrow{*} w\}$$

$L(G)$ é uma **linguagem independente do contexto**.

Duas gramáticas são **equivalentes** se geram a mesma linguagem.

Recursividade

Uma **produção (directamente) recursiva** tem a forma

$$A \rightarrow uAv$$

O símbolo não-terminal A é **recursivo** se

$$A \stackrel{+}{\Rightarrow} uAv$$

Uma **derivação** com a forma

$$A \Rightarrow w \stackrel{+}{\Rightarrow} uAv$$

em que A não ocorre em w , diz-se **indirectamente recursiva**.

$$(u, v, w \in (V \cup \Sigma)^*)$$

Independência das Sub-derivações

Lema Sejam $G = (V, \Sigma, P, S)$ uma GIC e $v \xRightarrow{n} w$ uma derivação em G em que v tem a forma

$$v = w_1 A_1 w_2 A_2 \dots w_k A_k w_{k+1}$$

com $w_i \in \Sigma^*$. Então existem palavras $p_i \in (V \cup \Sigma)^*$ que satisfazem

1. $A_i \xRightarrow{t_i} p_i$
2. $w = w_1 p_1 w_2 p_2 \dots w_k p_k w_{k+1}$
3. $\sum_{i=1}^k t_i = n.$

Derivação Esquerda e Direita

Numa **derivação esquerda** (\Rightarrow_L), em todos os passos é reescrito o símbolo não terminal mais à esquerda.

Numa **derivação direita** (\Rightarrow_R), em todos os passos é reescrito o símbolo não terminal mais à direita.

Teorema (existência de derivação esquerda) Seja $G = (V, \Sigma, P, S)$ uma GIC. Uma palavra $w \in \Sigma^*$ pertence a $L(G)$ sse

$$S \xRightarrow{*}_L w$$

Árvore de Derivação

Seja $G = (V, \Sigma, P, S)$ uma GIC.

A **árvore de derivação** correspondente à derivação $S \xRightarrow{*} w$ é formada de acordo com as seguintes regras:

1. A raiz da árvore é o símbolo inicial S ;
2. Se $A \rightarrow x_1x_2 \dots x_n$, com $x_i \in V \cup \Sigma$, foi a produção usada para reescrever o símbolo A , então o nó A correspondente tem filhos x_1, x_2, \dots, x_n , por esta ordem;
3. Se $A \rightarrow \lambda$ foi a produção usada para reescrever o símbolo A , então o nó A correspondente tem λ como único filho.

Uma palavra **tem** árvore de derivação A se for a concatenação (dos símbolos) das folhas desta.

Ambiguidade

Uma **gramática** G diz-se **ambígua** se alguma palavra de $L(G)$ tem, pelo menos:

- duas árvores de derivação distintas; ou
- duas derivações esquerdas distintas; ou
- duas derivações direitas distintas.

Uma **linguagem** é **inerentemente ambígua** se não existir uma gramática não ambígua que a gere.

$$\{a^i b^j c^k \mid i = j \text{ ou } j = k\}$$

Expressões Aritméticas e Ambiguidade

1ª Versão (ambígua)

$$G_{EA} = (\{E\}, \{n, +, -, \times, \div\}, P_{EA}, E)$$

com produções P_{EA} :

$$E \rightarrow E + E \mid E - E \mid E \times E \mid E \div E \mid n$$

2ª Versão — Prioridades (ambígua)

$$E \rightarrow E + E \mid E - E \mid T$$

$$T \rightarrow T \times T \mid T \div T \mid F$$

$$F \rightarrow n$$

3ª Versão — Associatividade (à esquerda)

$$E \rightarrow E + T \mid E - T \mid T$$

$$T \rightarrow T \times F \mid T \div F \mid F$$

$$F \rightarrow n$$

Gramáticas Regulares

Uma **gramática regular** é uma gramática independente do contexto em que todas as produções têm uma das formas

$$A \rightarrow a$$

$$A \rightarrow aB$$

$$A \rightarrow \lambda$$

onde $A, B \in V$ e $a \in \Sigma$.

Uma linguagem gerada por uma gramática regular é uma **linguagem regular**.

Uma gramática não regular pode gerar uma linguagem regular.

Autómatos de Pilha (1)

Autómatos de pilha = autómato finito + pilha

Um **autómato de pilha (AP)** é um tuplo $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ onde

- Q, Σ, q_0 e F são como nos autómatos finitos;
- Γ é o **alfabeto da pilha**, um conjunto finito de símbolos (A, B, C, \dots); e
- δ é a **função de transição** do autómato, uma função de $Q \times (\Sigma \cup \{\lambda\}) \times (\Gamma \cup \{\lambda\})$ em $\mathcal{P}(Q \times (\Gamma \cup \{\lambda\}))$.

$\alpha, \beta, \gamma, \dots$ denotam palavras sobre Γ

Autómatos de Pilha (2)

Uma **configuração** de um autómato de pilha é um triplo $[q, w, \alpha] \in Q \times \Sigma^* \times \Gamma^*$

Transições:

- $[q', \lambda] \in \delta(q, a, \lambda)$

$$[q, aw, \alpha] \vdash [q', w, \alpha]$$

- $[q', \lambda] \in \delta(q, a, A)$

$$[q, aw, A\alpha] \vdash [q', w, \alpha]$$

- $[q', B] \in \delta(q, a, \lambda)$

$$[q, aw, \alpha] \vdash [q', w, B\alpha]$$

- $[q', B] \in \delta(q, a, A)$

$$[q, aw, A\alpha] \vdash [q', w, B\alpha]$$

Configuração inicial: $[q_0, w, \lambda]$

Autómatos de Pilha (3)

Uma palavra $w \in \Sigma^*$ é **aceite** pelo autómato de pilha M se existe uma computação

$$[q_0, w, \lambda] \vdash_M^* [q_f, \lambda, \lambda]$$

com $q_f \in F$ (**critério de aceitação por estado de aceitação e pilha vazia**).

A linguagem **reconhecida** pelo autómato de pilha M é o conjunto de todas as palavras aceites por M .

Um autómato de pilha é **determinista** se, qualquer que seja a combinação de estado, símbolo de entrada e topo da pilha, existe no máximo uma transição aplicável.

Variantes

Um autómato de pilha **atómico** é um autómato de pilha que só tem transições das formas

$$[q_j, \lambda] \in \delta(q_i, a, \lambda)$$

$$[q_j, \lambda] \in \delta(q_i, \lambda, A)$$

$$[q_j, A] \in \delta(q_i, \lambda, \lambda)$$

Um autómato de pilha **estendido** pode conter transições em que são empilhados mais do que um símbolo, como

$$[q_j, BCD] \in \delta(q_i, u, A)$$

Uma linguagem reconhecida por um AP estendido é também reconhecida por um AP. Uma linguagem reconhecida por um AP é também reconhecida por um AP atómico.

Pumping Lemma (2)

Teorema (Pumping Lemma para linguagens independentes do contexto) Seja L uma linguagem independente do contexto. Então existe um k tal que para qualquer palavra p de L , com $|p| \geq k$, existe uma decomposição da forma

$$u v w x y, \text{ com } |vwx| \leq k \text{ e } |v| + |x| > 0$$

tal que

$$u v^i w x^i y \in L, \text{ para todo o } i \geq 0.$$

Hierarquia de Chomsky

Seja $G = (V, \Sigma, P, S)$ uma gramática.

G é uma gramática

- **sem restrições** (ou **tipo 0**) se todas as suas produções tiverem a forma

$$u \rightarrow v$$

com $u \in (V \cup \Sigma)^+$ e $v \in (V \cup \Sigma)^*$;

- **dependente do contexto** (ou **tipo 1**) se todas as suas produções tiverem a forma

$$u \rightarrow v$$

com $u, v \in (V \cup \Sigma)^+$ e $|u| \leq |v|$;

- **independente do contexto** (ou **tipo 2**);
ou

- **regular** (ou **tipo 3**).

Grafo de uma Gramática

Seja $G = (V, \Sigma, P, S)$ uma GIC.

O **grafo esquerdo da gramática** G é o grafo orientado etiquetado $g(G) = (N, P, A)$ onde

$$N = \{w \in (V \cup \Sigma)^* \mid S \xRightarrow{*}_L w\}$$

$$A = \{[v, w, r] \in N \times N \times P \mid v \Rightarrow_L w \text{ por aplicação da produção } r\}$$

O grafo de uma gramática não ambígua é uma árvore.

Análise Sintáctica

Sentido

- **descendente** (parte de S)
- **ascendente** (parte da palavra)

Estratégia

- em **largura**
- em **profundidade**

Se $w = uAv$, $u \in \Sigma^*$ e $A \in V$, u é o **prefixo terminal** de w

Análise Sintáctica

Descendente em Largura

entrada: GIC $G = (V, \Sigma, P, S)$ e $p \in \Sigma^*$

cria T com raiz S % árvore de pesquisa

$Q \leftarrow \{S\}$ % fila

repete

$q \leftarrow \text{remove}(Q)$ % $q = uAv$, $u \in \Sigma^*$, $A \in V$

$i \leftarrow 0$

$done \leftarrow false$

repete

se não há uma produção para A com número maior que i **então**

$done \leftarrow true$

senão

seja $A \rightarrow w$ a primeira produção para A com número $j > i$

se $uww \notin \Sigma^*$ e o prefixo terminal de uww é um prefixo de p **então**

$\text{insere}(uww, Q)$

acrescenta o nó uww a T

$i \leftarrow j$

até $done$ **ou** $p = uww$

até vazia(Q) **ou** $p = uww$

se $p = uww$ **então** ACEITA **senão** REJEITA

Análise Sintáctica

Descendente em Profundidade

entrada: GIC $G = (V, \Sigma, P, S)$ e $p \in \Sigma^*$

$S \leftarrow \{[S, 0]\}$ % pilha

repete

$[q, i] \leftarrow \text{desempilha}(S)$

$\text{inviável} \leftarrow \text{false}$

repete

 seja $q = uAv$, com $u \in \Sigma^*$ e $A \in V$

se u não é prefixo de p **então**

$\text{inviável} \leftarrow \text{true}$

se não há uma produção para A com número maior que i **então**

$\text{inviável} \leftarrow \text{true}$

se não inviável **então**

 seja $A \rightarrow w$ a primeira produção para

A com número $j > i$

 empilha($[q, j]$, S)

$q \leftarrow uwv$

$i \leftarrow 0$

até inviável **ou** $q \in \Sigma^*$

até $q = p$ **ou** vazia(S)

se $q = p$ **então** ACEITA **senão** REJEITA

Análise Sintáctica

Ascendente em Largura

entrada: GIC $G = (V, \Sigma, P, S)$ e $p \in \Sigma^*$

cria T com raiz p % árvore de pesquisa
 $Q \leftarrow \{p\}$ % fila

repete

$q \leftarrow \text{remove}(Q)$

para cada produção $A \rightarrow w \in P$

 % TRANSFERÊNCIA(S)

para cada decomposição $u w v$ de q ,

 com $v \in \Sigma^*$

 insere(uAv , Q) % REDUÇÃO

 acrescenta o nó uAv aos filhos

 de q em T

até $q = S$ **ou** vazia(Q)

se $q = S$ **então** ACEITA **senão** REJEITA

Análise Sintáctica

Ascendente em Profundidade

entrada: GIC $G = (V, \Sigma, P, S)$, com S não recursivo, e $p \in \Sigma^*$

$S \leftarrow \{[\lambda, 0, p]\}$ % pilha

repete

$[u, i, v] \leftarrow \text{desempilha}(S)$

$\text{inviável} \leftarrow \text{false}$

repete

seja $j > i$ o nº da 1ª produção da forma

· $A \rightarrow w$ com $u = qw$ e $A \neq S$, ou

· $S \rightarrow w$ com $u = w$ e $v = \lambda$

se existe tal j **então**

$\text{empilha}([u, j, v], S)$

$u \leftarrow qA$ % REDUÇÃO

$i \leftarrow 0$

se não existe tal j **e** $v \neq \lambda$ **então**

$\text{TRANSFERÊNCIA}(u, v)$

$i \leftarrow 0$

se não existe tal j **e** $v = \lambda$ **então**

$\text{inviável} \leftarrow \text{true}$

até $u = S$ **ou** inviável

até $u = S$ **ou** vazia(S)

se vazia(S) **então** REJEITA **senão** ACEITA

Transformação de Gramáticas (1)

Símbolo inicial não recursivo

Qualquer que seja a gramática independente do contexto $G = (V, \Sigma, P, S)$, existe uma gramática independente do contexto equivalente $G' = (V', \Sigma, P', S')$ onde o símbolo inicial é não recursivo.

- Se o símbolo inicial de G é não recursivo

$$G' = G$$

- Se o símbolo inicial de G é recursivo

$$G' = (V \cup \{S'\}, \Sigma, P \cup \{S' \rightarrow S\}, S')$$

$$(S' \notin V)$$

1. Tornar o Símbolo Inicial Não Recursivo

Gramática original:

$$G = (\{L, M, N, O\}, \{a, b, c, d\}, P, L)$$

$$\begin{aligned} P : L &\rightarrow Mb \mid aLb \mid \lambda \\ M &\rightarrow Lb \mid MLN \mid \lambda \\ N &\rightarrow NaN \mid NbO \\ O &\rightarrow cO \mid \lambda \end{aligned}$$

Gramática equivalente com símbolo inicial não recursivo:

$$G' = (\{L', L, M, N, O\}, \{a, b, c, d\}, P', L')$$

$$\begin{aligned} P' : L' &\rightarrow L \\ L &\rightarrow Mb \mid aLb \mid \lambda \\ M &\rightarrow Lb \mid MLN \mid \lambda \\ N &\rightarrow NaN \mid NbO \\ O &\rightarrow cO \mid \lambda \end{aligned}$$

Transformação de Gramáticas (2)

Seja $G = (V, \Sigma, P, S)$ uma GIC.

O conjunto dos **símbolos que geram** λ é

$$\Lambda = \{A \in V \mid A \xRightarrow{*} \lambda\}$$

Uma gramática **não contraível** não contém símbolos que geram λ .

Numa gramática **essencialmente não contraível** só o símbolo inicial pode gerar λ .

Introdução de produções

Se $A \xRightarrow{*}_G u$, então $G' = (V, \Sigma, P \cup \{A \rightarrow u\}, S)$ é equivalente a G .

Eliminação das Produções- λ

Seja $G = (V, \Sigma, P, S)$ uma GIC com S não recursivo.

A gramática $G_L = (V, \Sigma, P_L, S)$, equivalente a G , é uma **gramática essencialmente não contraível** cujas produções P_L são:

1. Todas as produções de G que não são produções- λ ;
2. Todas as produções que se obtêm eliminando um ou mais símbolos de Λ do corpo de uma produção de G , desde que o corpo resultante tenha pelo menos um símbolo; e
3. A produção $S \rightarrow \lambda$ sse $S \in \Lambda$.

2. Eliminar Produções- λ

$$G' = (\{L', L, M, N, O\}, \{a, b, c, d\}, P', L')$$

$$P' : L' \rightarrow L$$

$$L \rightarrow Mb \mid aLb \mid \lambda$$

$$M \rightarrow Lb \mid MLN \mid \lambda$$

$$N \rightarrow NaN \mid NbO$$

$$O \rightarrow cO \mid \lambda$$

Símbolos que geram λ :

$$\Lambda = \{L', L, M, O\}$$

Gramática equivalente (essencialmente) não contraível:

$$G_L = (\{L', L, M, N, O\}, \{a, b, c, d\}, P_L, L')$$

$$P_L : L' \rightarrow L \mid \lambda$$

$$L \rightarrow Mb \mid aLb \mid b \mid ab$$

$$M \rightarrow Lb \mid MLN \mid b \mid LN \mid MN \mid N$$

$$N \rightarrow NaN \mid NbO \mid Nb$$

$$O \rightarrow cO \mid c$$

Eliminação das Produções Unitárias ($A \rightarrow B$)

Seja $G = (V, \Sigma, P, S)$ uma GIC essencialmente não contraível.

Para cada $A \in V$, seja $\text{CHAIN}(A)$ o conjunto

$$\{B \in V \mid A \xRightarrow{*}_G B\}$$

A gramática $G_C = (V, \Sigma, P_C, S)$ é uma gramática equivalente a G onde P_C consiste nas produções $A \rightarrow w$ que satisfazem, para algum $B \in V$:

1. $B \in \text{CHAIN}(A)$
2. $B \rightarrow w \in P$
3. $w \notin V$

3. Eliminar as Produções Unitárias

| | CHAIN |
|------|-------------|
| L' | $\{L', L\}$ |
| L | $\{L\}$ |
| M | $\{M, N\}$ |
| N | $\{N\}$ |
| O | $\{O\}$ |

Gramática sem produções unitárias, equivalente a G_L :

$$G_C = (\{L', L, M, N, O\}, \{a, b, c, d\}, P_C, L')$$

$$P_C : L' \rightarrow \lambda \mid Mb \mid aLb \mid b \mid ab$$

$$L \rightarrow Mb \mid aLb \mid b \mid ab$$

$$M \rightarrow Lb \mid MLN \mid b \mid LN \mid MN \mid NaN \mid NbO \mid Nb$$

$$N \rightarrow NaN \mid NbO \mid Nb$$

$$O \rightarrow cO \mid c$$

Símbolos Inúteis

Um símbolo $x \in V \cup \Sigma$ é **útil** se existe uma derivação

$$S \xRightarrow{*} u x v \xRightarrow{*} w$$

onde $u, v \in (V \cup \Sigma)^*$ e $w \in \Sigma^*$.

Um símbolo que não é útil é **inútil**.

Um símbolo não terminal A é **produtivo** se $A \xRightarrow{*} w$, com $w \in \Sigma^*$.

Um símbolo não terminal que não é produtivo é **improdutivo**.

Um símbolo não terminal A é **acessível** se $S \xRightarrow{*} u A v$, com $u, v \in (V \cup \Sigma)^*$.

Um símbolo não terminal que não é acessível é **inacessível**.

Um símbolo é útil se for produtivo e acessível.

4. Eliminar os Símbolos Inúteis

1. PRODUTIVOS = $\{L', L, M, O\}$

Produções sem símbolos improdutivos:

$$\begin{aligned}L' &\rightarrow \lambda \mid Mb \mid aLb \mid b \mid ab \\L &\rightarrow Mb \mid aLb \mid b \mid ab \\M &\rightarrow Lb \mid b \\O &\rightarrow cO \mid c\end{aligned}$$

2. ACESSÍVEIS = $\{L', L, M\} \cup \{a, b\}$

Gramática sem símbolos inúteis (improdutivos ou inacessíveis), equivalente a G_C :

$$\begin{aligned}G_U &= (\{L', L, M\}, \{a, b\}, P_U, L') \\P_U : L' &\rightarrow \lambda \mid Mb \mid aLb \mid b \mid ab \\L &\rightarrow Mb \mid aLb \mid b \mid ab \\M &\rightarrow Lb \mid b\end{aligned}$$

Forma Normal de Chomsky

Uma GIC $G = (V, \Sigma, P, S)$ está na **forma normal de Chomsky** se todas as suas produções têm uma das formas

- $A \rightarrow BC$
- $A \rightarrow a$
- $S \rightarrow \lambda$

onde $a \in \Sigma$ e $B, C \in V - \{S\}$.

5. Construir a Forma Normal de Chomsky

- $L' \rightarrow \lambda \mid MB \mid ALB \mid b \mid AB$
 $B \rightarrow b$
 $A \rightarrow a$
 $L \rightarrow MB \mid ALB \mid b \mid AB$
 $M \rightarrow LB \mid b$

Gramática na Forma Normal de Chomsky, equivalente a G_U :

$$G_{NC} = (\{L', L, M, A, B, X\}, \{a, b\}, P_{NC}, L')$$

$$P_{NC} : L' \rightarrow \lambda \mid MB \mid AX \mid b \mid AB$$
$$X \rightarrow LB$$
$$B \rightarrow b$$
$$A \rightarrow a$$
$$L \rightarrow MB \mid AX \mid b \mid AB$$
$$M \rightarrow LB \mid b$$

Forma Normal de Greibach

Uma GIC $G = (V, \Sigma, P, S)$ está na **forma normal de Greibach** se todas as suas produções têm uma das formas

- $A \rightarrow aA_1A_2 \dots A_n$
- $A \rightarrow a$
- $S \rightarrow \lambda$

onde $a \in \Sigma$ e $A_i \in V - \{S\}$, para $i = 1, 2, \dots, n$.

6. Construir a Forma Normal de Greibach (1)

1. Ordem dos não terminais: $L' X B A L M$
2. Todas as produções da forma $A_1 \rightarrow A_2 w$ satisfazem $A_1 < A_2$.

$$L' \rightarrow \lambda \mid MB \mid AX \mid b \mid AB$$

$$X \rightarrow LB$$

$$B \rightarrow b$$

$$A \rightarrow a$$

$$L \rightarrow MB \mid aX \mid b \mid aB$$

$$(M \rightarrow MBB \mid aXB \mid bB \mid aBB \mid b)$$

$$M \rightarrow aXBZ \mid bBZ \mid aBBZ \mid bZ \mid$$

$$aXB \mid bB \mid aBB \mid b$$

$$Z \rightarrow BBZ \mid BB$$

$(A_1, A_2, \in V \text{ e } w \in V^*)$

6. Construir a Forma Normal de Greibach (2)

Gramática na Forma Normal de Greibach, equivalente a G_{NC} :

$$G_G = (\{L', L, M, A, B, X, Z\}, \{a, b\}, P_G, L')$$

$$P_G : L' \rightarrow \lambda \mid aXBZB \mid bBZB \mid aBBZB \mid bZB \mid \\ aXBB \mid bBB \mid aBBB \mid bB \mid aX \mid b \mid \\ aB$$

$$X \rightarrow aXBZBB \mid bBZBB \mid aBBZBB \mid \\ bZBB \mid aXBBB \mid bBBB \mid aBBBB \mid \\ bBB \mid aXB \mid bB \mid aBB$$

$$B \rightarrow b$$

$$A \rightarrow a$$

$$L \rightarrow aXBZB \mid bBZB \mid aBBZB \mid bZB \mid \\ aXBB \mid bBB \mid aBBB \mid bB \mid aX \mid b \mid \\ aB$$

$$M \rightarrow aXBZ \mid bBZ \mid aBBZ \mid bZ \mid \\ aXB \mid bB \mid aBB \mid b$$

$$Z \rightarrow bBZ \mid bB$$

Eliminação da Recursividade Directa à Esquerda

Se A é um símbolo não terminal com pelo menos uma produção da forma

$$A \rightarrow Au$$

substituem-se as produções

$$A \rightarrow Au_1 \mid Au_2 \mid \dots \mid Au_j \mid v_1 \mid v_2 \mid \dots \mid v_k$$

onde o primeiro símbolo dos v_i não é A , pelas produções

$$A \rightarrow v_1Z \mid v_2Z \mid \dots \mid v_kZ \mid v_1 \mid v_2 \mid \dots \mid v_k$$

$$Z \rightarrow u_1Z \mid u_2Z \mid \dots \mid u_jZ \mid u_1 \mid u_2 \mid \dots \mid u_j$$

onde Z é um novo símbolo (não terminal).

$$(u_i, v_i \in (V \cup \Sigma)^*)$$

Gramáticas $LL(k)$

Subclasse das gramáticas independentes do contexto que admite análise sintáctica (descendente) determinista, com k símbolos de avanço.

Gramáticas $LL(1)$

Subclasse das gramáticas independentes do contexto que admite análise sintáctica (descendente) determinista, com 1 símbolo de avanço.

Gramáticas LL(1)

A GIC $G = (V, \Sigma, P, S)$, com **terminador** $\#$, é **LL(1)** se quando existem duas derivações esquerdas

$$S \xRightarrow{*} u_1Av_1 \Rightarrow u_1xv_1 \xRightarrow{*} u_1aw_1$$

$$S \xRightarrow{*} u_2Av_2 \Rightarrow u_2yv_2 \xRightarrow{*} u_2aw_2$$

onde $u_i, w_i \in \Sigma^*$ e $a \in \Sigma$, então $x = y$.

Teorema Uma gramática LL(k), para algum $k > 0$, é não ambígua.

Teorema Se algum símbolo não terminal de G é recursivo à esquerda, então G não é LL(k), para qualquer $k > 0$.

Factorização à Esquerda

Seja $G = (V, \Sigma, P, S)$ uma gramática independente do contexto.

Se algum $A \in V$ tiver produções

$$A \rightarrow uv_1 \mid uv_2 \mid \dots \mid uv_n$$

com $u \in (V \cup \Sigma)^+$, a gramática G' , obtida acrescentando o novo símbolo não terminal A' e substituindo estas produções por

$$A \rightarrow uA'$$

e

$$A' \rightarrow v_1 \mid v_2 \mid \dots \mid v_n$$

é equivalente a G .

Primeiros e Seguintes

Primeiros

Os **primeiros** de $u \in (V \cup \Sigma)^*$ são os símbolos do alfabeto que podem aparecer na primeira posição de uma palavra derivada a partir de u .

$$\text{PRIMEIROS}(u) = \{a \mid u \xrightarrow{*} ax \in \Sigma^*\}$$

Seguintes

Os **seguintes** de $A \in V$ são os símbolos do alfabeto que podem aparecer a seguir a A nalguma derivação.

$$\text{SEGUINTE}(A) = \{a \mid S \xrightarrow{*} uAv \text{ e } a \in \text{PRIMEIROS}(v)\}$$

$$(a \in \Sigma, x \in \Sigma^* \text{ e } u, v \in (V \cup \Sigma)^*)$$

Símbolos Directores

O conjunto dos símbolos **directores** da produção $A \rightarrow w \in P$ é

$$\text{DIR}(A \rightarrow w) = \begin{cases} \text{PRIMEIROS}(w) & \text{se } w \not\stackrel{*}{\rightarrow} \lambda \\ \text{PRIMEIROS}(w) \cup \text{SEGUINTE}(A) & \text{se } w \stackrel{*}{\rightarrow} \lambda \end{cases}$$

Teorema Se para todo o $A \in V$, para quaisquer produções distintas $A \rightarrow w$ e $A \rightarrow v \in P$

$$\text{DIR}(A \rightarrow w) \cap \text{DIR}(A \rightarrow v) = \emptyset$$

então a gramática é LL(1).

Cálculo dos Primeiros (1)

Construção do **grafo dos primeiros**:

- Os vértices do grafo são os elementos de V e de Σ ;
- Para cada produção $A \rightarrow u_1 u_2 \dots u_n$, $u_i \in V \cup \Sigma$
 - Acrescenta-se um arco de A para u_1 ;
 - Se $u_1 \in \Lambda$, acrescenta-se também um arco de A para u_2 ;
 - Se $u_1, u_2 \in \Lambda$, acrescenta-se também um arco de A para u_3 , e assim sucessivamente.

O grafo dos primeiros contém um caminho de $A \in V$ para $a \in \Sigma$ sse $a \in \text{PRIMEIROS}(A)$.

Cálculo dos Primeiros (2)

Define-se indutivamente $\text{PRIMEIROS}(w)$, $w \in (V \cup \Sigma)^*$, como

$$\text{PRIMEIROS}(\lambda) = \emptyset$$

$$\text{PRIMEIROS}(a) = \{a\} \quad a \in \Sigma$$

$$\text{PRIMEIROS}(A) = (\text{no grafo}) \quad A \in V$$

$$\begin{aligned} \text{PRIMEIROS}(uv) &= \\ &= \begin{cases} \text{PRIMEIROS}(u) & \text{se } u \not\stackrel{*}{\Rightarrow} \lambda \\ \text{PRIMEIROS}(u) \cup \text{PRIMEIROS}(v) & \text{se } u \stackrel{*}{\Rightarrow} \lambda \end{cases} \end{aligned}$$

Cálculo dos Seguintes

Construção do **grafo dos seguintes**:

- Os vértices do grafo são os elementos de V e de Σ ;
- Para cada produção $A \rightarrow uBv$, $B \in V$ e $u, v \in (V \cup \Sigma)^*$
 - Acrescenta-se um arco de B para cada a pertencente a $\text{PRIMEIROS}(v)$;
 - Se $v \xRightarrow{*} \lambda$, acrescenta-se um arco de B para A .

O grafo dos seguintes contém um caminho de $A \in V$ para $a \in \Sigma$ sse $a \in \text{SEGUINTE}(A)$.

Analizador Sintático

Descendente Recursivo (1)

```
proc E()  
  se símbolo-de-avanço ∈ {a, (} então  
    %  $E \rightarrow TX$   
    T()  
    X()  
  senão  
    erro()  
  
proc T()  
  se símbolo-de-avanço ∈ {a} então  
    %  $T \rightarrow a$   
    consome(a)  
  senão se símbolo-de-avanço ∈ {(} então  
    %  $T \rightarrow (E)$   
    consome(  
    E()  
    consome()  
  senão  
    erro()
```

Analizador Sintático Descendente Recursivo (2)

```
proc X()  
  se símbolo-de-avanço ∈ {+} então  
    %  $X \rightarrow Z$   
    Z()  
  senão se símbolo-de-avanço ∈ {),#} então  
    %  $X \rightarrow \lambda$   
  senão  
    erro()  
  
proc consome(símbolo)  
  se símbolo-de-avanço = símbolo então  
    se símbolo-de-avanço ≠ # então  
      símbolo-de-avanço ← próximo-símbolo()  
  senão  
    erro()
```

Gramáticas LR(0)

Seja $G = (V, \Sigma, P, S)$ uma GIC, com S não recursivo e terminador $\#$.

uw é um **contexto-LR(0)** de $A \rightarrow w \in P$ se existe uma derivação direita

$$S \xRightarrow{*}_R uAv \Rightarrow_R uwv$$

A um prefixo de um contexto-LR(0) chama-se **prefixo viável**.

Exemplo

$$G_{LR_0} = (\{S, X, Y\}, \{a, b, \#\}, P_{LR_0}, S)$$

$$P_{LR_0}: S \rightarrow X\# \quad X \rightarrow XY \mid \lambda \quad Y \rightarrow aYa \mid b$$

| | Contextos-LR(0) |
|-------------------------|--------------------------------|
| $S \rightarrow X\#$ | $X\#$ |
| $X \rightarrow XY$ | XY |
| $X \rightarrow \lambda$ | λ |
| $Y \rightarrow aYa$ | $XaYa, XaaYa, \dots = Xa^*aYa$ |
| $Y \rightarrow b$ | $Xb, Xab, Xaab, \dots = Xa^*b$ |

Item LR(0)

Os itens LR(0) de G são:

- $A \rightarrow u.v$, se $A \rightarrow uv \in P$;
- $A \rightarrow .$, se $A \rightarrow \lambda \in P$.

Um **item completo** é um item LR(0) em que o ponto está o mais à direita possível.

Um item $A \rightarrow u.v$ é **válido** para o prefixo viável xu se xuv é um contexto-LR(0).

Exemplo

Os itens LR(0) de G_{LR_0} são:

$S \rightarrow .X\#$ $S \rightarrow X.\#$ $S \rightarrow X\#.$

$X \rightarrow .XY$ $X \rightarrow X.Y$ $X \rightarrow XY.$

$X \rightarrow .$

$Y \rightarrow .aYa$ $Y \rightarrow a.Ya$ $Y \rightarrow aY.a$ $Y \rightarrow aYa.$

$Y \rightarrow .b$ $Y \rightarrow b.$

Fecho de um Conjunto de Itens LR(0)

O **fecho** de um conjunto I de itens LR(0) define-se recursivamente como:

- $I \subseteq \text{fecho}(I)$;
- se $A \rightarrow u.Bv \in \text{fecho}(I)$, com $B \in V$, então $B \rightarrow .w \in \text{fecho}(I)$ para todas as produções $B \rightarrow w$;
- nada mais pertence a $\text{fecho}(I)$.

Exemplo

$$\text{fecho}(\{X \rightarrow X.Y\}) = \\ \{X \rightarrow X.Y, Y \rightarrow .aYa, Y \rightarrow .b\}$$

Autómato Finito dos Itens LR(0) Válidos

Seja $G = (V, \Sigma, P, S)$ uma gramática independente do contexto.

O **autómato dos itens válidos** de G , que reconhece os prefixos viáveis de G , é o autómato finito determinista

$$M = (Q, V \cup \Sigma, \delta, q_0, Q \setminus \{\emptyset\})$$

onde

- $q_0 = \text{fecho}(\{S \rightarrow \cdot w \mid S \rightarrow w \in P\})$
- para todo o $q \in Q$ e todo o $x \in V \cup \Sigma$, $\delta(q, x) \in Q$, com

$$\delta(q, x) = \text{fecho}(\{A \rightarrow ux \cdot v \mid A \rightarrow u \cdot xv \in q\})$$

Condições LR(0)

Uma gramática independente do contexto é LR(0) se o seu autómato dos itens válidos satisfaz as seguintes condições:

- Nenhum estado contém dois itens completos;
- Se um estado contém um item completo, todos os outros itens desse estado têm o ponto imediatamente à esquerda de um símbolo não terminal da gramática.

Analizador Sintáctico

LR(0)

entrada: GIC LR(0) $G = (V, \Sigma, P, S)$,
AFD dos itens válidos de G
 $M = (Q, V \cup \Sigma, \delta, q_0, F)$ e
 $p \in \Sigma^*$

$u \leftarrow \lambda$

$v \leftarrow p$

$erro \leftarrow false$

repete

$q \leftarrow \hat{\delta}(q_0, u)$

se q contém $A \rightarrow w.$, sendo $u = xw$ **então**

$u \leftarrow xA$ % REDUÇÃO

senão se q contém $A \rightarrow y.z$, com $z \neq \lambda$,

e $v \neq \lambda$ **então**

TRANSFERÊNCIA(u, v)

senão

$erro \leftarrow true$ % REJEIÇÃO

até $u = S$ **ou** $erro$

se $u = S$ **então** ACEITA **senão** REJEITA

Tabela de Análise Sintáctica LR(0)

Uma linha por estado do AFD dos itens válidos, excepto para o estado \emptyset .

Uma coluna por cada símbolo de $(V \cup \Sigma) \setminus \{S\}$, cujo conteúdo corresponde à função de transição do autómato.

Uma coluna ACÇÃO que, na linha q_i contém:

- ACEITA se q_i contém um item completo de uma produção de S ;
- TRANSF se q_i contém um item $A \rightarrow u \cdot av$, com $a \in \Sigma$;
- $A \rightarrow w$, indicando uma REDUÇÃO, se q_i contém o item completo $A \rightarrow w \cdot$, $A \neq S$.

As posições vazias da tabela indicam a REJEIÇÃO da palavra.

AP Reconhecedor LR(0)

Dada uma gramática LR(0) $G = (V, \Sigma, P, S)$ e o seu AFD dos itens válidos $M = (Q, V \cup \Sigma, \delta, q_0, Q \setminus \{\emptyset\})$, pode-se construir o autómato de pilha estendido que reconhece a linguagem gerada por G

$$R = (\{q_I, q\}, \Sigma, V \cup \Sigma \cup Q \setminus \{\emptyset\}, \delta', q_I, \{q\})$$

com

- $[q, q_0] \in \delta'(q_I, \lambda, \lambda)$;
- $[q, \lambda] \in \delta'(q, \lambda, q_i a_n \dots q_{j_2} a_2 q_{j_1} a_1 q_0)$ para todo o $q_i \in Q$ que contém um item completo $S \rightarrow a_1 a_2 \dots a_n \cdot$, onde

$$[q_0, a_1 a_2 \dots a_n] \vdash_M [q_{j_1}, a_2 \dots a_n] \vdash_M^* [q_i, \lambda];$$

- $[q, q_j A q_{j_0}] \in \delta'(q, \lambda, q_i a_n \dots q_{j_2} a_2 q_{j_1} a_1 q_{j_0})$ para todo o $q_i \in Q$ que contém um item completo $A \rightarrow a_1 a_2 \dots a_n \cdot$, $A \neq S$, onde

$$q_j = \delta(q_{j_0}, A) \text{ e}$$

$$[q_{j_0}, a_1 a_2 \dots a_n] \vdash_M [q_{j_1}, a_2 \dots a_n] \vdash_M^* [q_i, \lambda];$$

- $[q, q_j a q_i] \in \delta'(q, a, q_i)$ se $\delta(q_i, a) = q_j$, $a \in \Sigma$.

AP LR(0) para G_{LR_0}

Inicialização do AP

$$(q_I, \lambda) \xrightarrow{\lambda} (q, \mathbf{0})$$

Aceitação

$$(q, \mathbf{2\#1X0}) \xrightarrow{\lambda} (q, \lambda)$$

Redução

$$(q, \mathbf{0}) \xrightarrow{\lambda} (q, \mathbf{1X0})$$

$$(q, \mathbf{3Y1X0}) \xrightarrow{\lambda} (q, \mathbf{1X0})$$

$$(q, \mathbf{5b1}) \xrightarrow{\lambda} (q, \mathbf{3Y1})$$

$$(q, \mathbf{5b4}) \xrightarrow{\lambda} (q, \mathbf{6Y4})$$

$$(q, \mathbf{7a6Y4a1}) \xrightarrow{\lambda} (q, \mathbf{3Y1})$$

$$(q, \mathbf{7a6Y4a4}) \xrightarrow{\lambda} (q, \mathbf{6Y4})$$

Transferência

$$(q, \mathbf{1}) \xrightarrow{a} (q, \mathbf{4a1})$$

$$(q, \mathbf{1}) \xrightarrow{b} (q, \mathbf{5b1})$$

$$(q, \mathbf{1}) \xrightarrow{\#} (q, \mathbf{2\#1})$$

$$(q, \mathbf{4}) \xrightarrow{a} (q, \mathbf{4a4})$$

$$(q, \mathbf{4}) \xrightarrow{b} (q, \mathbf{5b4})$$

$$(q, \mathbf{6}) \xrightarrow{a} (q, \mathbf{7a6})$$

(De acordo com o AFD dos itens LR(0) válidos obtido na aula.)

Item LR(1)

Seja $G = (V, \Sigma, P, S)$ uma gramática independente do contexto.

Os **itens LR(1)** de G têm a forma

$$A \rightarrow u.v, L$$

onde

- $A \rightarrow u.v$ é um item LR(0), o **núcleo**, e
- $L \subseteq \Sigma \cup \{\#\}$ é o conjunto de **símbolos de avanço**.

Um item $A \rightarrow u.v, L$ é **válido** para xu se, para todo o $a \in L$, existe uma derivação

$$S \xRightarrow{*}_R xAy$$

com $a \in \text{PRIMEIROS}(y\#)$.

Fecho LR(1)

O **fecho** de um conjunto I de itens LR(1) define-se recursivamente como:

- $I \subseteq \text{fecho}_1(I)$;
- se $A \rightarrow u.Bv, L \in \text{fecho}_1(I)$, com $B \in V$, então $B \rightarrow .w, K \in \text{fecho}_1(I)$ para todas as produções $B \rightarrow w$, com

$$K = \begin{cases} \text{PRIMEIROS}(v) & \text{se } v \not\stackrel{*}{\Rightarrow} \lambda \\ \text{PRIMEIROS}(v) \cup L & \text{se } v \stackrel{*}{\Rightarrow} \lambda \end{cases}$$

- nada mais pertence a $\text{fecho}_1(I)$.

Exemplo de Fecho LR(1)

$$G_{LR_1} = (\{S, A\}, \{a, b\}, P_{LR_1}, S)$$

$$P_{LR_1}: \begin{array}{l} S \rightarrow AbA \\ A \rightarrow Aa \mid \lambda \end{array}$$

$$\begin{aligned} \text{fecho}_1(\{ S \rightarrow Ab\underline{.}A, \{\#\} \}) &= \\ &\{ S \rightarrow Ab\underline{.}A, \{\#\} \} \\ &\cup \{ A \rightarrow \underline{.}Aa, \{\#\}, A \rightarrow \underline{.}, \{\#\} \} \\ &\cup \{ A \rightarrow \underline{.}Aa, \{a\}, A \rightarrow \underline{.}, \{a\} \} = \\ &\{ S \rightarrow Ab\underline{.}A, \{\#\}, \\ &A \rightarrow \underline{.}Aa, \{a, \#\}, \\ &A \rightarrow \underline{.}, \{a, \#\} \} \end{aligned}$$

Autómato Finito dos Itens LR(1) Válidos

Seja $G = (V, \Sigma, P, S)$ uma gramática independente do contexto e seja $G' = (V \cup \{S'\}, \Sigma, P \cup \{S' \rightarrow S\}, S')$.

O **autómato dos itens válidos** de G' é o autómato finito determinista

$$M = (Q, V \cup \Sigma, \delta, q_0, Q \setminus \{\emptyset\})$$

onde

- $q_0 = \text{fecho}_1(\{S' \rightarrow \cdot S, \{\#\}\})$
- para todo o $q \in Q$ e todo o $x \in V \cup \Sigma$, $\delta(q, x) \in Q$, com

$$\delta(q, x) =$$

$$\text{fecho}_1(\{A \rightarrow ux \cdot v, L \mid A \rightarrow u \cdot xv, L \in q\})$$

Condições LR(1)

Uma gramática independente do contexto é LR(1) se o seu autómato dos itens válidos satisfaz as seguintes condições:

- Se um estado contém um item completo $A \rightarrow w., L$ e um item $B \rightarrow u.av, K$, então $a \notin L$;
- Se um estado contém dois itens completos $A \rightarrow w., L$ e $B \rightarrow u., K$, então $L \cap K = \emptyset$.

Tabela de Análise Sintáctica LR(1)

Uma linha por estado do AFD dos itens válidos, excepto para o estado \emptyset .

Uma coluna por cada símbolo de $(V \cup \Sigma) \setminus \{S\}$, cujo conteúdo corresponde à função de transição do autómato.

Uma coluna por cada símbolo $a \in \Sigma \cup \{\#\}$ que, na linha q_i contém a acção:

- ACEITA se q_i contém um item completo de uma produção de S e $a = \#$;
- TRANSF se q_i contém um item $A \rightarrow u \cdot av, L$;
- $A \rightarrow w$, indicando uma REDUÇÃO, se q_i contém o item completo $A \rightarrow w \cdot, L$, $A \neq S$ e $a \in L$.

As posições vazias da tabela indicam a REJEIÇÃO da palavra.

Uma Tabela de Análise Sintáctica LR(1)

$$G_{LR_1} = (\{S, A\}, \{a, b\}, P_{LR_1}, S)$$

$$P_{LR_1}: \begin{aligned} S &\rightarrow AbA \\ A &\rightarrow Aa \mid \lambda \end{aligned}$$

| | <i>S</i> | <i>A</i> | <i>a</i> | <i>b</i> | <i>a</i> | <i>b</i> | # |
|---|----------|----------|----------|----------|-------------------------|-------------------------|-------------------------|
| 0 | 1 | 2 | | | $A \rightarrow \lambda$ | $A \rightarrow \lambda$ | |
| 1 | | | | | | | ACEITA |
| 2 | | | 4 | 3 | TRANSF | TRANSF | |
| 3 | | 5 | | | $A \rightarrow \lambda$ | | $A \rightarrow \lambda$ |
| 4 | | | | | $A \rightarrow Aa$ | $A \rightarrow Aa$ | |
| 5 | | | 6 | | TRANSF | | $S \rightarrow AbA$ |
| 6 | | | | | $A \rightarrow Aa$ | | $A \rightarrow Aa$ |

(De acordo com o AFD dos itens LR(1) válidos obtido na aula.)

AP Reconhecedor LR(1)

O autómato de pilha que reconhece a linguagem gerada por uma gramática LR(1) $G = (V, \Sigma, P, S)$ com AFD dos itens válidos $M = (Q, V \cup \Sigma, \delta, q_0, Q \setminus \{\emptyset\})$, é o autómato de pilha estendido

$$R = (Q_R, \Sigma \cup \{\#\}, V \cup \Sigma \cup Q \setminus \{\emptyset\}, \delta_R, q_I, F_R)$$

com

- $Q_R = \{q_I, q\} \cup \{q_a \mid a \in \Sigma \cup \{\#\}\}$
- $F_R = \{q_\#\}$

Função de Transição do AP LR(1)

$$[q, q_0] \in \delta_R(q_I, \lambda, \lambda).$$

$$[q_a, \lambda] \in \delta_R(q, a, \lambda) \text{ para todo o } a \in \Sigma \cup \{\#\}.$$

$[q_{\#}, \lambda] \in \delta_R(q_{\#}, \lambda, q_i a_n \dots q_{j_2} a_2 q_{j_1} a_1 q_0)$ para todo o $q_i \in Q$ que contém um item completo $S \rightarrow a_1 a_2 \dots a_n \cdot, L, \# \in L$ e

$$\begin{aligned} [q_0, a_1 a_2 \dots a_n] \vdash_M [q_{j_1}, a_2 \dots a_n] \vdash_M \dots \\ \vdash_M [q_{j_{n-1}}, a_n] \vdash_M [q_i, \lambda]. \end{aligned}$$

$[q_a, q_j A q_{j_0}] \in \delta_R(q_a, \lambda, q_i a_n \dots q_{j_2} a_2 q_{j_1} a_1 q_{j_0})$ para todo o $q_i \in Q$ que contém um item completo $A \rightarrow a_1 a_2 \dots a_n \cdot, L, A \neq S, a \in L,$

$$q_j = \delta(q_{j_0}, A) \text{ e}$$

$$\begin{aligned} [q_{j_0}, a_1 a_2 \dots a_n] \vdash_M [q_{j_1}, a_2 \dots a_n] \vdash_M \dots \\ \vdash_M [q_{j_{n-1}}, a_n] \vdash_M [q_i, \lambda]. \end{aligned}$$

$[q, q_j a q_i] \in \delta_R(q_a, \lambda, q_i)$ para todo o $q_i \in Q$ que contém um item $A \rightarrow u \cdot a v, L, a \in \Sigma$ e $q_j = \delta(q_i, a).$

AP LR(1) para G_{LR_1}

| | |
|--|------------------------------|
| $(q_I, \lambda) \xrightarrow{\lambda} (q, \mathbf{0})$ | Inicialização do AP |
| $\left. \begin{array}{l} (q, \lambda) \xrightarrow{a} (q_a, \lambda) \\ (q, \lambda) \xrightarrow{b} (q_b, \lambda) \\ (q, \lambda) \xrightarrow{\#} (q_{\#}, \lambda) \end{array} \right\}$ | Leitura do símbolo de avanço |
| $(q_{\#}, \mathbf{1S0}) \xrightarrow{\lambda} (q_{\#}, \lambda)$ | Aceitação |
| $\left. \begin{array}{l} (q_a, \mathbf{0}) \xrightarrow{\lambda} (q_a, \mathbf{2A0}) \\ (q_b, \mathbf{0}) \xrightarrow{\lambda} (q_b, \mathbf{2A0}) \\ (q_a, \mathbf{3}) \xrightarrow{\lambda} (q_a, \mathbf{5A3}) \\ (q_{\#}, \mathbf{3}) \xrightarrow{\lambda} (q_{\#}, \mathbf{5A3}) \\ (q_a, \mathbf{4a2A0}) \xrightarrow{\lambda} (q_a, \mathbf{2A0}) \\ (q_b, \mathbf{4a2A0}) \xrightarrow{\lambda} (q_b, \mathbf{2A0}) \\ (q_{\#}, \mathbf{5A3b2A0}) \xrightarrow{\lambda} (q_{\#}, \mathbf{1S0}) \\ (q_a, \mathbf{6a5A3}) \xrightarrow{\lambda} (q_a, \mathbf{5A3}) \\ (q_{\#}, \mathbf{6a5A3}) \xrightarrow{\lambda} (q_{\#}, \mathbf{5A3}) \end{array} \right\}$ | Redução |
| $\left. \begin{array}{l} (q_a, \mathbf{2}) \xrightarrow{\lambda} (q, \mathbf{4a2}) \\ (q_b, \mathbf{2}) \xrightarrow{\lambda} (q, \mathbf{3b2}) \\ (q_a, \mathbf{5}) \xrightarrow{\lambda} (q, \mathbf{6a5}) \end{array} \right\}$ | Transferência |

(De acordo com o AFD dos itens LR(1) válidos obtido na aula.)

Autômato Amalgamado

Seja $M = (Q, \Sigma, \delta, q_0, F)$ o autômato dos itens LR(1) válidos de uma gramática.

O **autômato amalgamado** M_A é o autômato que resulta de fundir num só os estados de M com o mesmo núcleo LR(0).

Seja $Q_i = \{q_{i_1}, q_{i_2}, \dots, q_{i_m}\}$ um conjunto de estados de M com o mesmo núcleo. O estado \widehat{Q}_i é o resultado da **fusão dos estados** de Q_i e contém os itens $A \rightarrow u.v, L_{i_1} \cup L_{i_2} \cup \dots \cup L_{i_m}$ tais que $A \rightarrow u.v, L_{i_j}$ é um item de q_{i_j} .

Seja $\{Q_1, Q_2, \dots, Q_n\}$ uma partição de Q tal que todos os estados de Q_i têm o mesmo núcleo e os núcleos dos estados de Q_i e de Q_j são diferentes, se $i \neq j$. Então

$$M_A = (Q_A, \Sigma, \delta_A, \{\widehat{q_0}\}, Q_A \setminus \{\{\widehat{\emptyset}\}\})$$

com

- $Q_A = \{\widehat{Q}_1, \widehat{Q}_2, \dots, \widehat{Q}_n\}$;
- $\delta_A(\widehat{Q}_i, a) = \widehat{Q}_j$ se $\delta(q, a) \in Q_j$ para $q \in Q_i$.

LALR(1)

Uma gramática independente do contexto é LALR(1) se o seu autómato amalgamado satisfaz as condições LR(1).

LR(0) (bis)

Uma gramática independente do contexto é LR(0) se o seu autómato dos itens LR(1) válidos, considerando somente os núcleos dos estados, satisfaz as condições LR(0).

A Linguagem WHILE

Átomos (conjunto finito)

$$A = \{\text{nil}, \text{while}, :=, \text{quote}, \text{var}, \dots\}$$

Valores D_A (elementos d, e, f, \dots)

- $A \subseteq D_A$
- se $d, e \in D_A$, então $(d.e) \in D_A$
- D_A é o menor conjunto que satisfaz os pontos anteriores.

Variáveis Vars (conjunto infinito, X, Y, \dots)

Expressões

$$\begin{array}{l} E \rightarrow X \quad \text{(variável)} \\ \quad | \quad d \quad \text{(valor)} \\ \quad | \quad =? \ E \ E \\ \quad | \quad \text{cons } E \ E \ | \ \text{hd } E \ | \ \text{tl } E \end{array}$$

Instruções

$$C \rightarrow X := E \ | \ C; C \ | \ \text{while } E \ \text{do } C$$

Programas

$$\text{read } X; C; \text{write } Y$$

Açúcar Sintático para a Linguagem WHILE (1)

Booleanos

`false` \equiv `nil`

`true` \equiv `(nil.nil)`

`if E then C1 else C2` \equiv

`Z := E;`

`W := true;`

`while Z do`

`{ Z := false; W := false; C1 };`

`while W do { W := false; C2 };`

(onde Z e W são variáveis que não ocorrem
no resto do programa)

`skip` \equiv `X := X`

Açúcar Sintático para a Linguagem WHILE (2)

Listas

`nil` é a lista vazia

$(e) \equiv (e . \text{nil})$

$(e_1 e_2 \dots e_n) \equiv$
 $(e_1 . (e_2 . \dots (e_n . \text{nil}) \dots))$

Naturais

$0 \equiv \text{nil}$

$1 \equiv (\text{nil} . \text{nil})$

$n \equiv (\text{nil} . n - 1)$

Açúcar Sintáctico para a Linguagem WHILE (3)

Macros

Se p é o programa

```
read  $X_p$ ;  
 $C_p$ ;  
write  $Y_p$ 
```

então a instrução

```
 $W := p e$ ;
```

é equivalente a

```
 $X_p := e$ ;  
 $C_p$ ;  
 $W := Y_p$ ;
```

Representação Interna de um Programa WHILE

O programa WHILE

```
read X;
Y := nil;
while X do
  Y := cons (hd X) Y;
  X := tl X;
write Y
```

é representado internamente pela lista

```
(
  (var 1)
  (; (:= (var 2) (quote nil))
    (while (var 1)
      (; (:= (var 2)
              (cons (hd (var 1)) (var 2)))
        (:= (var 1) (tl (var 1))))))
  (var 2)
)
```

Problema de Decisão

Um **problema de decisão** é um problema cujas instâncias têm resposta 'sim' ou 'não'.

Exemplos

- x é um quadrado perfeito?

Instâncias:

0 é um quadrado perfeito?

1 é um quadrado perfeito?

2 é um quadrado perfeito?

...

- A palavra w pertence à linguagem L ?
- O programa p termina se corre com dados d ?
- A máquina de Turing M pára quando o conteúdo inicial da fita é w ?

Solução de um Problema de Decisão

A **solução** de um problema de decisão é um *procedimento efectivo (algoritmo)* que permite calcular a resposta para todas as instâncias do problema.

Um **algoritmo** deve ser

- **completo**: produz uma resposta para todas as instâncias de um problema;
- **executável mecanicamente**: consiste num número finito de passos, que podem ser executados 'sem pensar';
- **determinista**: produz sempre a mesma resposta para a mesma instância do problema.

Um problema de decisão sem solução diz-se **indecidível**.

Tese de Church-Turing

Existe um procedimento efectivo que é solução de um problema de decisão se e só se existe uma máquina de Turing que pára sempre e que resolve todas as instâncias do problema.

Formalismos Equivalentes

Máquinas de Turing

Cálculo- λ

Funções recursivas

Sistemas de Post

URM (Unlimited Register Machine)

Linguagem WHILE

Problema da Terminação (*Halting Problem*) (1)

Enunciado: O programa p termina quando corre com dados d ?

Seja *termina* a função

$$termina(p, d) = \begin{cases} \text{true} & \text{se } p \text{ termina com} \\ & \text{dados } d \\ \text{false} & \text{se } p \text{ não termina} \\ & \text{com dados } d \end{cases}$$

e seja t o programa que implementa a função *termina*: quando corrido com dados $(p.d)$, o resultado de t é

- **true** se o programa p termina quando corre com dados d ;
- **false** no caso contrário.

Problema da Terminação (*Halting Problem*) (2)

Seja t' o programa que, quando corrido com dados p , tem o seguinte comportamento

- se o resultado de $t(p.p)$ é **true**, t' não termina;
- se o resultado de $t(p.p)$ é **false**, o resultado de $t'(p)$ é **true**.

Qual o resultado de $t'(t')$?

- Se $t'(t')$ termina, então o resultado de $t(t'.t')$ é **true** e $t'(t')$ não termina;
- Se $t'(t')$ não termina, então o resultado de $t(t'.t')$ é **false** e o resultado de $t'(t')$ é **true**.

Há uma contradição em ambos os casos!

Problema da Terminação (*Halting Problem*) (3)

O programa t não existe.

O problema da terminação é **indecidível**

A função *termina* é **não computável**.

Redução de Problemas

O problema A pode ser **reduzido** ao problema B se qualquer instância de A puder ser expressa como uma instância de B cuja resposta é a resposta à instância de A .

Se A pode ser reduzido a B e se A é um problema indecidível, então B também é indecidível.

Exemplo

O problema da terminação pode ser reduzido ao problema de saber se o programa p termina quando corre com dados `nil`.

Exemplo de Redução (1)

Seja N o problema de decisão: o programa p termina quando corrido com dados `nil`?

Seja p_N o programa que implementa a solução de N .

Sejam p um programa e d dados para p :

```
read  $X_p$ ;  
 $C_p$ ;  
write  $Y_p$ 
```

Seja p' o programa:

```
read  $X_p$ ;  
 $X_p := d$ ;  
 $C_p$ ;  
write  $Y_p$ 
```

e seja s o programa que constrói p' a partir de $(p.d)$.

Exemplo de Redução (2)

O comportamento de p' , quando corrido com quaisquer dados, é o comportamento de p quando corrido com dados d .

Seja t o programa:

```
read PD;           (PD contém o par (p.d))
P' := s PD;        (transforma p)
R := pN P';       (pN corre com dados p')
write R
```

Dados p e d , t constrói p' e calcula $p_N(p')$.

O resultado de $p_N(p')$ é **true** se $p'(nil)$ termina e **false** caso contrário.

Como $p'(nil)$ tem o comportamento de $p(d)$, o programa t determina se p termina quando corrido com dados d .

Exemplo de Redução (3)

O programa t implementa uma solução para o problema da terminação.

Mas o problema da terminação é indecidível e o programa t não existe.

Como existe uma redução do problema da terminação ao problema N — o programa s pode ser construído e as restantes construções usadas na construção de t são possíveis —, a premissa errada é a existência do programa p_N .

Logo, o programa p_N não existe e o problema N também é indecidível.

Teorema de Rice

Qualquer propriedade extensional não-trivial de programas é indecidível.

Uma propriedade é **extensional** se diz respeito à função que o programa calcula.

Uma propriedade é **não-trivial** se é satisfeita por pelo menos um programa, mas não por todos.

Exemplos

- O programa termina quando corre com dados `nil`.
- O conjunto $\{d \mid p(d) \text{ termina}\}$ é finito.
- O programa implementa uma função total.

Problemas Indecidíveis

- A GIC G é ambígua?
- As GIC G_1 e G_2 são equivalentes?
- A intersecção das linguagens geradas pelas GIC G_1 e G_2 é não vazia?
- O programa p reconhece a linguagem vazia?
- A linguagem reconhecida pelo programa p é regular?
- A linguagem reconhecida pelo programa p é Σ^* ?