Linguagens Formais e Autómatos

Vasco Pedro

Departamento de Informática
Universidade de Évora

2012/2013

Alfabeto, palavra

Alfabeto – conjunto finito de símbolos (Σ, T) (elementos representados por a, b, c, d)

Exemplos

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

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

λ – palavra vazia (também ϵ e ε)

Construção de palavras, comprimento

Se w é uma palavra sobre Σ e $a \in \Sigma$, então wa é uma palavra sobre Σ

$|w|$ – comprimento da palavra w (número de símbolos que a constituem):

1. $|\lambda| = 0$
2. $|va| = |v| + 1$ (v é uma palavra sobre Σ e $a \in \Sigma$)

Todas as palavras

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

Operações sobre palavras (1)

Concatenação, potência

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 $u.v = (u.w)a$

Potências de uma palavra

Seja u uma palavra sobre Σ^*

$$u^0 = \lambda$$

$$u^1 = u$$

$$uu = u^2$$

$$u^3 = u^2u = uuu$$

$$u^n = u^{n-1}u = u \ldots u \quad (u \text{ concatenada } n \text{ vezes})$$

Operações sobre palavras (2)

Inversão

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, prefixo e sufixo

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

$$v = xuy$$

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

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

Linguagem

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

Propriedades das operações sobre palavras

Concatenação

elemento neutro	$\lambda.w = w.\lambda = w$
associatividade	$(uv)w = u(vw)$
não comutatividade	e.g., $aa.bb \neq bb.aa$
comprimento	$\|uv\| = \|u\| + \|v\|$

Inversão

da concatenação	$(uv)^R = v^R u^R$

Ordens

Sejam $\Sigma = \{a_1, a_2, \ldots, a_n\}$ um alfabeto com $a_1 < a_2 < \ldots < a_n$ e $u, v, x, y \in \Sigma^*$

Ordem lexicográfica ou do dicionário ($<$)

$u < v$ se

- u é um prefixo próprio de v

  ou

- $u = x a_i y, v = x a_j z$ e $a_i < a_j$

Ordem mista ($<_M$)

$u <_M v$ se

- $|u| < |v|$

  ou

- $|u| = |v|$ e $u < v$

Com igualdade: $u \leq v$ se $u < v$ ou $u = v$, e o mesmo para $\leq_M$

Caracterização finita de linguagens

- Através de uma definição recursiva

- Recorrendo a operações sobre conjuntos
  - união, intersecção, complemento, ...
  - concatenação de conjuntos:

    se X e Y forem conjuntos de palavras (i.e., linguagens)

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

Exemplo

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

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 conhecida como operador de fecho ou de iteração

Exemplo

Linguagem dos números naturais sem zeros à esquerda

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

Propriedades do fecho

Sejam X e Y conjuntos

- $X \subseteq X^*$
- $\emptyset^* = \{\lambda\}$
- se $X \subseteq Y$, então $X^* \subseteq Y^*$
- se $X \neq \emptyset$, então X^* é infinito

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

$$X \cup Y$$

$$XY$$

$$X^*$$

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 da base

Expressões regulares

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

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

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

Linguagem regular

A linguagem representada por uma expressão regular é:

$$
\begin{aligned}
L(\emptyset) &= \emptyset \\
L(\lambda) &= \{\lambda\} \\
L(a) &= \{a\} \qquad (a \in \Sigma) \\
L(u \cup v) &= L(u) \cup L(v) \\
L(uv) &= L(u)L(v) \\
L(u^*) &= L(u)^*
\end{aligned}
$$

Duas expressões regulares são equivalentes se representam a mesma linguagem

Uma linguagem representada por uma expressão regular é uma linguagem regular

Propriedades das expressões regulares (1)

$$\emptyset u = u\emptyset = \emptyset \qquad \lambda u = u\lambda = u$$

$$(uv)w = u(vw)$$

$$u \cup v = v \cup u \qquad u \cup \emptyset = u$$

$$(u \cup v) \cup w = u \cup (v \cup w) \qquad u \cup u = u$$

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

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

$$\emptyset^* = \lambda \qquad \lambda^* = \lambda$$

$$u^* = (u^*)^* \qquad u^* = \lambda \cup uu^*$$

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

Propriedades das expressões regulares (2)

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

Simplificação de expressões regulares (1)

Exemplo

$$0^*(1 \cup (0^*1^*)^*)00^*(10^*)^*0$$

$$= \quad 0^*(1 \cup (0 \cup 1)^*)00^*(10^*)^*0$$

$$= \quad 0^*(0 \cup 1)^*00^*(10^*)^*0$$

$$= \quad (0 \cup 1)^*00^*(10^*)^*0$$

$$= \quad (0 \cup 1)^*0(0 \cup 1)^*0$$

Justificação

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

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

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

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

Já está numa forma aceitável, mas pode-se continuar

Simplificação de expressões regulares (2)

Exemplo (continuando...)

$$
\begin{aligned}
&= (0 \cup 1)^* 0 (0 \cup 1)^* 0 \\
&= (1 \cup 0)^* 0 (0 \cup 1)^* 0 \\
&= 1^* (01^*)^* 0 (0 \cup 1)^* 0 \\
&= 1^* 0 (1^* 0)^* (0 \cup 1)^* 0 \\
&= 1^* 0 (1^* 0)^* (1 \cup 0)^* 0 \\
&= 1^* 0 (1^* 0)^* (1^* 0)^* 1^* 0 \\
&= 1^* 0 (1^* 0)^* 1^* 0 \\
&= 1^* 0 (1 \cup 0)^* 0
\end{aligned}
$$

Justificação

$u \cup v = v \cup u$

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

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

$u \cup v = v \cup u$

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

$u^* u^* = u^*$ (porquê?)

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

Autómatos finitos deterministas

Um autómato 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 autómato

$F \subseteq Q$ 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 autómato e w é a parte da palavra ainda por processar

A computação de um AFD M para a palavra $w = a_1 a_2 \ldots a_n \in \Sigma^*$ é a sequência de configurações

$$[s_0, a_1 a_2 \ldots a_n] \vdash_M [s_1, a_2 \ldots a_n] \vdash_M \cdots \vdash_M [s_n, \lambda]$$

com

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

para $i > 0$

Função de transição estendida

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

$$\begin{aligned} \hat{\delta}(q, \lambda) &= q \\ \hat{\delta}(q, a) &= \delta(q, a) \\ \hat{\delta}(q, wa) &= \delta(\hat{\delta}(q, w), a) \end{aligned}$$

Para um AFD,

$$\hat{\delta}(q_0, w) = q \text{ sse } [q_0, w] \vdash_M^* [q, \lambda]$$

Linguagem reconhecida

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 autómato 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 autómato

$F \subseteq Q$ é o conjunto dos **estados de aceitação**

Qualquer autómato finito determinista é um autómato 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 símbolos de w

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

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

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

Autómatos finitos não deterministas com transições λ

Um autómato 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 autómato

$F \subseteq Q$ 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

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

Autómato finito determinista equivalente

O AFD equivalente ao AFND $M = (Q, \Sigma, \delta, q_0, F)$ é o autómato

$$M' = (Q', \Sigma, \delta', q_0', F')$$

tal que

$$q_0' = \lambda\text{-fecho}(q_0)$$

$$\delta'(q, a) = \bigcup_{s \in q} t(s, a)$$

- $q_0' \in Q'$
- se $q \in Q'$ então $\delta'(q, a) \in Q'$, para todo o $a \in \Sigma$
- mais nenhum estado está em Q'

$$F' = \{q \in Q' \mid q \cap F \neq \emptyset\}$$

Minimização de autómatos finitos deterministas

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

Dois estados q_i e q_j de M são equivalentes sse

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

para qualquer $u \in \Sigma^*$

Dois estados equivalentes dizem-se indistinguíveis

Observação

Se $q_i \in F$ e $q_j \in Q \setminus F$ então q_i e q_j não são equivalentes (porquê?)

Cálculo dos estados equivalentes

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

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

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 (1)

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: $\{q_0', q_f\} \cap Q = \emptyset$, $q_0' \neq q_f$

Composição de autómatos (2)

Sejam M_A e M_B dois autómatos finitos nas condições anteriores

$$M_A = (Q_A, \Sigma, \delta_A, q_{0_A}, \{q_{f_A}\}) \qquad M_B = (Q_B, \Sigma, \delta_B, q_{0_B}, \{q_{f_B}\})$$

Definem-se os autómatos finitos seguintes

1.
$$M_. = (Q_A \cup Q_B, \Sigma, \delta_., q_{0_A}, \{q_{f_B}\})$$

onde
$$\delta_. = \delta_A \cup \delta_B \cup \{(q_{f_A}, \lambda, \{q_{0_B}\})\}$$

com
$$L(M_.) = L(M_A)L(M_B)$$

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

Composição de autómatos (3)

2.

$$M_{\cup} = (Q_A \cup Q_B \cup \{q_0, q_f\}, \Sigma, \delta_{\cup}, q_0, \{q_f\})$$

onde

$$\delta_{\cup} = \delta_A \cup \delta_B \cup \{ (q_0, \lambda, \{q_{0_A}, q_{0_B}), (q_{f_A}, \lambda, \{q_f\}), (q_{f_B}, \lambda, \{q_f\}) \}$$

com

$$L(M_{\cup}) = L(M_A) \cup L(M_B)$$

3.

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

onde

$$\delta_* = \delta_A \cup \{ (q_0, \lambda, \{q_{0_A}, q_f\}), (q_{f_A}, \lambda, \{q_{0_A}, q_f\}) \}$$

com

$$L(M_*) = L(M_A)^*$$

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

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

Aplicação do *Pumping Lemma* para linguagens regulares

Exemplo

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

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

Sejam k o número de estados desse autómato e $p = a^k b^k$

Qualquer decomposição de p nas condições do *Pumping Lemma* será da forma

u	v	w
a^j	a^m	$a^{k-j-m} b^k$

com $j + m \leq k$ e $m > 0$

Para $i = 0$ temos

$$uv^0w = a^j (a^m)^0 a^{k-j-m} b^k = a^{k-m} b^k$$

Sendo $m > 0$, então $k - m \neq k$. Como todas as palavras de L têm igual número de as e de bs, $a^{k-m} b^k \notin L$ e 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

Gramáticas (2)

Geração por reescrita

$\langle$frase$\rangle$
$\Rightarrow \langle$sujeito$\rangle$ $\langle$frase-verbal$\rangle$
$\Rightarrow \langle$artigo$\rangle$ $\langle$subst-comum$\rangle$ $\langle$frase-verbal$\rangle$
$\Rightarrow \langle$artigo$\rangle$ hamburger $\langle$frase-verbal$\rangle$
$\Rightarrow \langle$artigo$\rangle$ hamburger $\langle$verbo$\rangle$ $\langle$advérbio$\rangle$
$\Rightarrow$ the hamburger $\langle$verbo$\rangle$ $\langle$advérbio$\rangle$
$\Rightarrow$ the hamburger $\langle$verbo$\rangle$ slowly
$\Rightarrow$ the hamburger drives slowly

$\langle$frase$\rangle \rightarrow \langle$sujeito$\rangle$ $\langle$frase-verbal$\rangle$
$\langle$sujeito$\rangle \rightarrow \langle$artigo$\rangle$ $\langle$subst-comum$\rangle$
$\langle$subst-comum$\rangle \rightarrow$ hamburger
$\langle$frase-verbal$\rangle \rightarrow \langle$verbo$\rangle$ $\langle$advérbio$\rangle$
$\langle$artigo$\rangle \rightarrow$ the
$\langle$advérbio$\rangle \rightarrow$ slowly
$\langle$verbo$\rangle \rightarrow$ drives

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

Símbolos não terminais: $\langle$frase$\rangle$, $\langle$frase-verbal$\rangle$, $\langle$sujeito$\rangle$, $\langle$verbo$\rangle$, ...

Gramáticas (3)

1. $\langle \text{frase} \rangle \rightarrow \langle \text{sujeito} \rangle \ \langle \text{frase-verbal} \rangle$
2. $\rightarrow \langle \text{sujeito} \rangle \ \langle \text{verbo} \rangle \ \langle \text{compl-directo} \rangle$

$\vdots$

17. $\langle \text{adjectivos} \rangle \rightarrow \langle \text{adjectivo} \rangle \ \langle \text{adjectivos} \rangle$
18. $\rightarrow \lambda$
19. $\langle \text{adjectivo} \rangle \rightarrow$ big
20. $\rightarrow$ juicy
21. $\rightarrow$ brown
22. $\langle \text{compl-directo} \rangle \rightarrow \langle \text{adjectivos} \rangle \ \langle \text{subst-próprio} \rangle$
23. $\rightarrow \langle \text{artigo} \rangle \ \langle \text{adjectivos} \rangle \ \langle \text{subst-comum} \rangle$

Gramáticas independentes do contexto

Definição

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

V é o conjunto finito dos símbolos **não terminais** $(A, B, C, \ldots)$

Σ é o conjunto finito dos símbolos **terminais** (alfabeto)

$P \subseteq V \times (V \cup \Sigma)^*$ é um conjunto finito de **produções**

$S \in V$ é o **símbolo inicial** da gramática

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

Derivação

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

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

$$uAv \Rightarrow_G uwv$$

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

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

então u deriva v em n passos

$$u \Rightarrow_G^n v$$

Se $u \Rightarrow_G^n v$ para algum $n \geq 0$, u deriva v

$$u \Rightarrow_G^* v$$

Linguagem gerada

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

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

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

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

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

$L(G)$ 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

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

Uma derivação com a forma

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

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 $v \Rightarrow^n w$ uma derivação em G em que v tem a forma

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

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

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

Derivação esquerda e direita

Existência

Numa derivação esquerda ($\Rightarrow_{\mathrm{L}}$), em todos os passos é reescrito o símbolo não terminal mais à esquerda

Numa derivação direita ($\Rightarrow_{\mathrm{R}}$), 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

$$S \Rightarrow_{\mathrm{L}}^{*} w$$

É, igualmente, garantida a existência de derivação direita

Árvore de derivação

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

A árvore de derivação correspondente à derivação $S \Rightarrow^* w$ é formada de acordo com as seguintes regras:

1. A raiz da árvore é o símbolo inicial S
2. Se $A \rightarrow x_1 x_2 \ldots x_n$, 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, $\ldots$, 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 w tem árvore de derivação T (e T é uma árvore de derivação de w) se w for a concatenação das folhas de T

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

$$G_{\text{EA}} = (\{E\}, \{n, +, -, \times, \div\}, P_{\text{EA}}, E) \quad \text{com produções } P_{\text{EA}}:$$

1ª versão (ambígua)

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

2ª versão — Prioridades (ambígua)

$$E \to E + E \mid E - E \mid T$$

$$T \to T \times T \mid T \div T \mid F$$

$$F \to n$$

3ª versão — Associatividade (à esquerda)

$$E \to E + T \mid E - T \mid T$$

$$T \to T \times F \mid T \div F \mid F$$

$$F \to n$$

Gramáticas regulares

Uma gramática regular é uma GIC (V, Σ, P, S) em que todas as produções têm uma das formas

$$A \to a$$

$$A \to aB$$

$$A \to \lambda$$

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

A linguagem gerada por uma gramática regular é uma linguagem regular

Uma gramática não regular pode gerar uma linguagem regular

Autómatos de pilha (1)

Autómato 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, ...)
- δ é a **função de transição** do autómato, uma função de
  $Q \times (\Sigma \cup \{\lambda\}) \times (\Gamma \cup \{\lambda\})$ em $\mathcal{P}(Q \times (\Gamma \cup \{\lambda\}))$

α, β, γ, ... denotam palavras sobre Γ

Autómatos de pilha (2)

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

Transições (com símbolo)

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

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

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

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

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

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

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

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

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

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

Propriedades

Qualquer linguagem reconhecida por um AP é também reconhecida por um AP atómico

Qualquer linguagem reconhecida por um AP estendido é também reconhecida por um AP

Outro *Pumping Lemma*

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

Uma gramática $G = (V, \Sigma, P, S)$ é de um dos seguintes tipos

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

$$u \to v$$

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

$$u \to v$$

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

- independente do contexto (ou tipo 2)
- regular (ou tipo 3)

Análise sintáctica

Sentido

- Descendente (parte do símbolo inicial)
- 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

Grafo de uma gramática

O grafo (esquerdo) da GIC $G = (V, \Sigma, P, S)$ é o grafo orientado etiquetado $g(G) = (N, P, A)$ onde

$$N = \{w \in (V \cup \Sigma)^* \mid S \Rightarrow_{\mathrm{L}}^* w\}$$

e

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

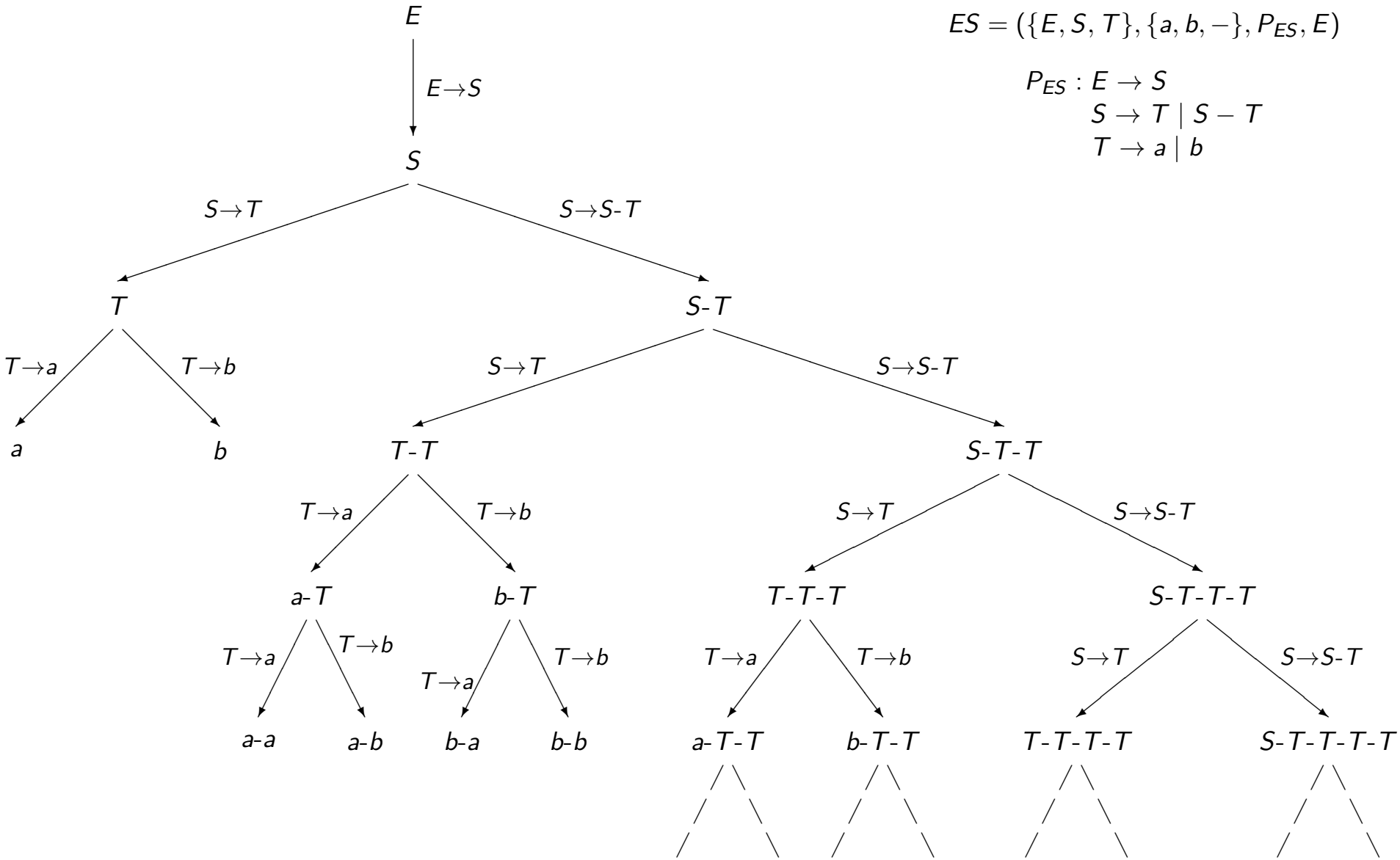
O grafo direito de G define-se de modo análogo

Um grafo (esquerdo ou direito) de uma gramática não ambígua é uma árvore

Grafo (esquerdo) de uma gramática

$ES = (\{E, S, T\}, \{a, b, -\}, P_{ES}, E)$

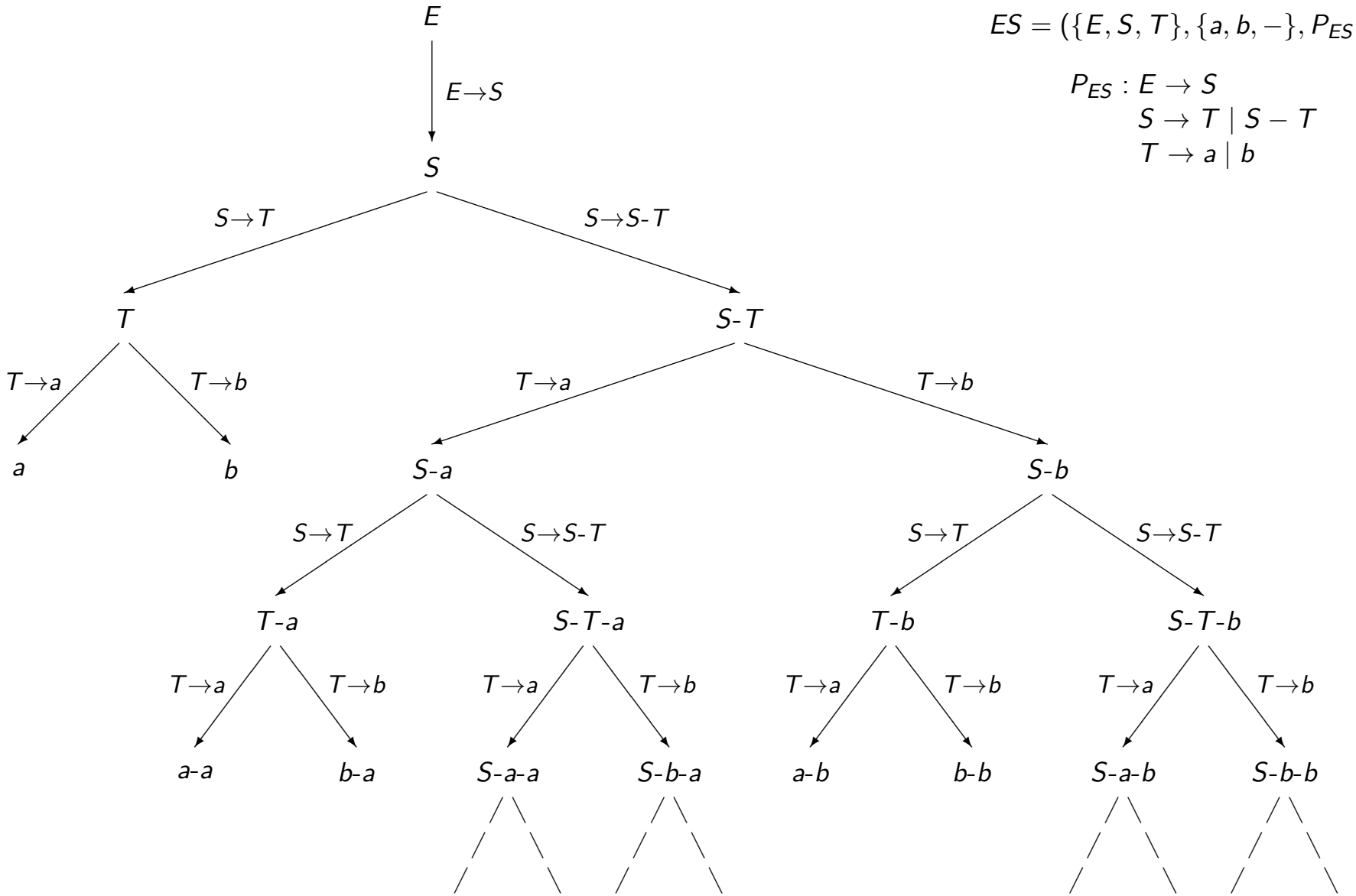
$P_{ES} : E \to S$
$S \to T \mid S - T$
$T \to a \mid b$



Grafo direito de uma gramática

$$ES = (\{E, S, T\}, \{a, b, -\}, P_{ES}, E)$$

$$P_{ES} : E \to S$$
$$S \to T \mid S - T$$
$$T \to a \mid b$$

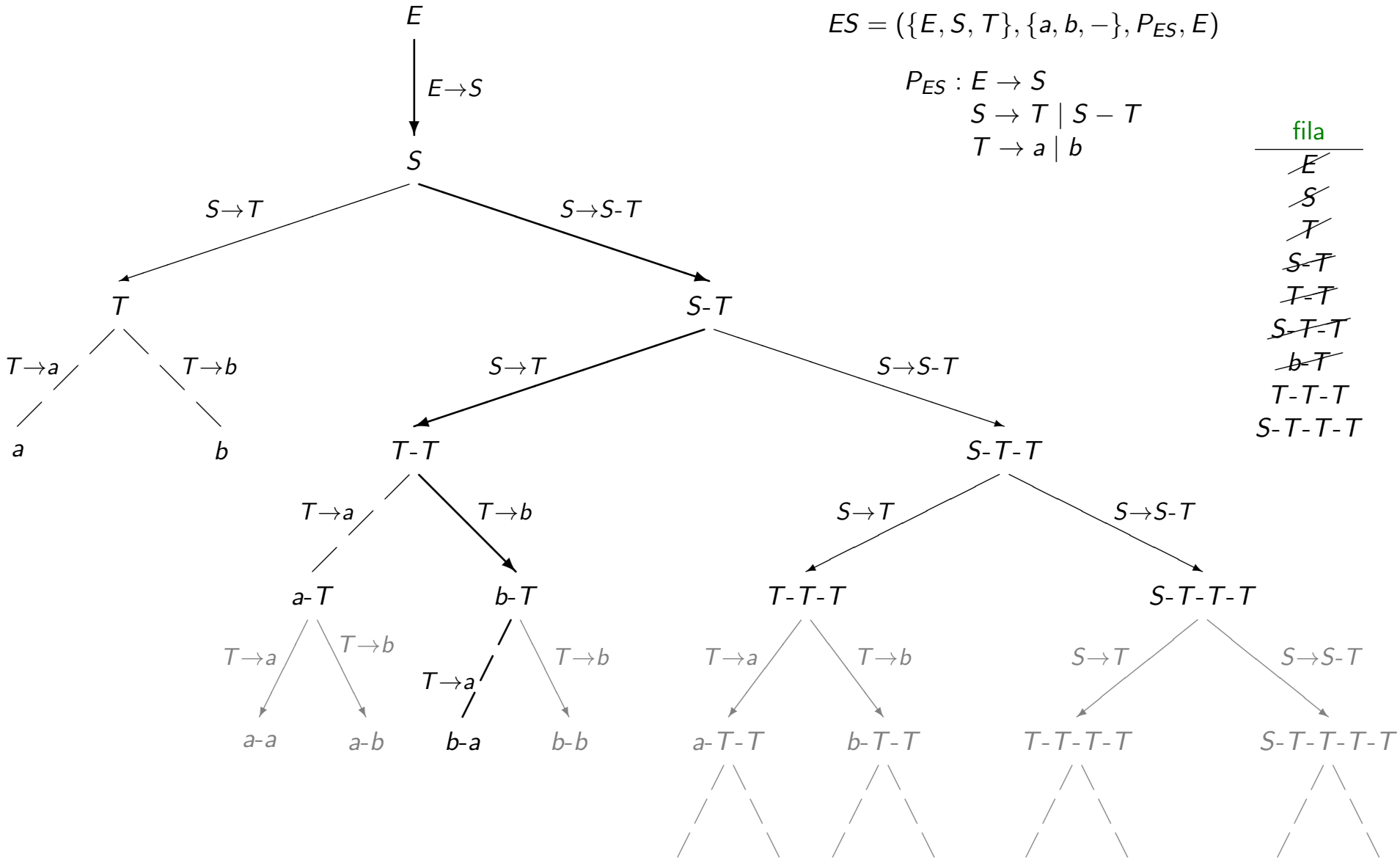


Algoritmo de análise sintáctica descendente em largura

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

cria T com raiz S % árvore de pesquisa
$Q \leftarrow \{S\}$ % fila
repete
  $q \leftarrow$ 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 $uwv \notin \Sigma^*$ e o prefixo terminal de uwv é um prefixo de p **então**
        insere(uwv, Q)
        acrescenta o nó uvw a T como filho de q
      $i \leftarrow j$
  até *done* **ou** $p = uwv$
até vazia(Q) **ou** $p = uwv$
se $p = uwv$ **então** ACEITA **senão** REJEITA

Análise sintáctica descendente em largura para $b - a$



$ES = (\{E, S, T\}, \{a, b, -\}, P_{ES}, E)$

$P_{ES} : E \to S$
$S \to T \mid S - T$
$T \to a \mid b$

fila

~~E~~
~~S~~
~~T~~
~~$S\text{-}T$~~
~~$T\text{-}T$~~
~~$S\text{-}T\text{-}T$~~
~~$b\text{-}T$~~
$T\text{-}T\text{-}T$
$S\text{-}T\text{-}T\text{-}T$

Algoritmo de análise sintáctica descendente em profundidade

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

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

repete

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

  inviável $\leftarrow$ *false*

  repete

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

    se u não é prefixo de p **então**

      inviável $\leftarrow$ *true*

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

      inviável $\leftarrow$ *true*

    se não *inviável* **então**

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

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

      $q \leftarrow uwv$

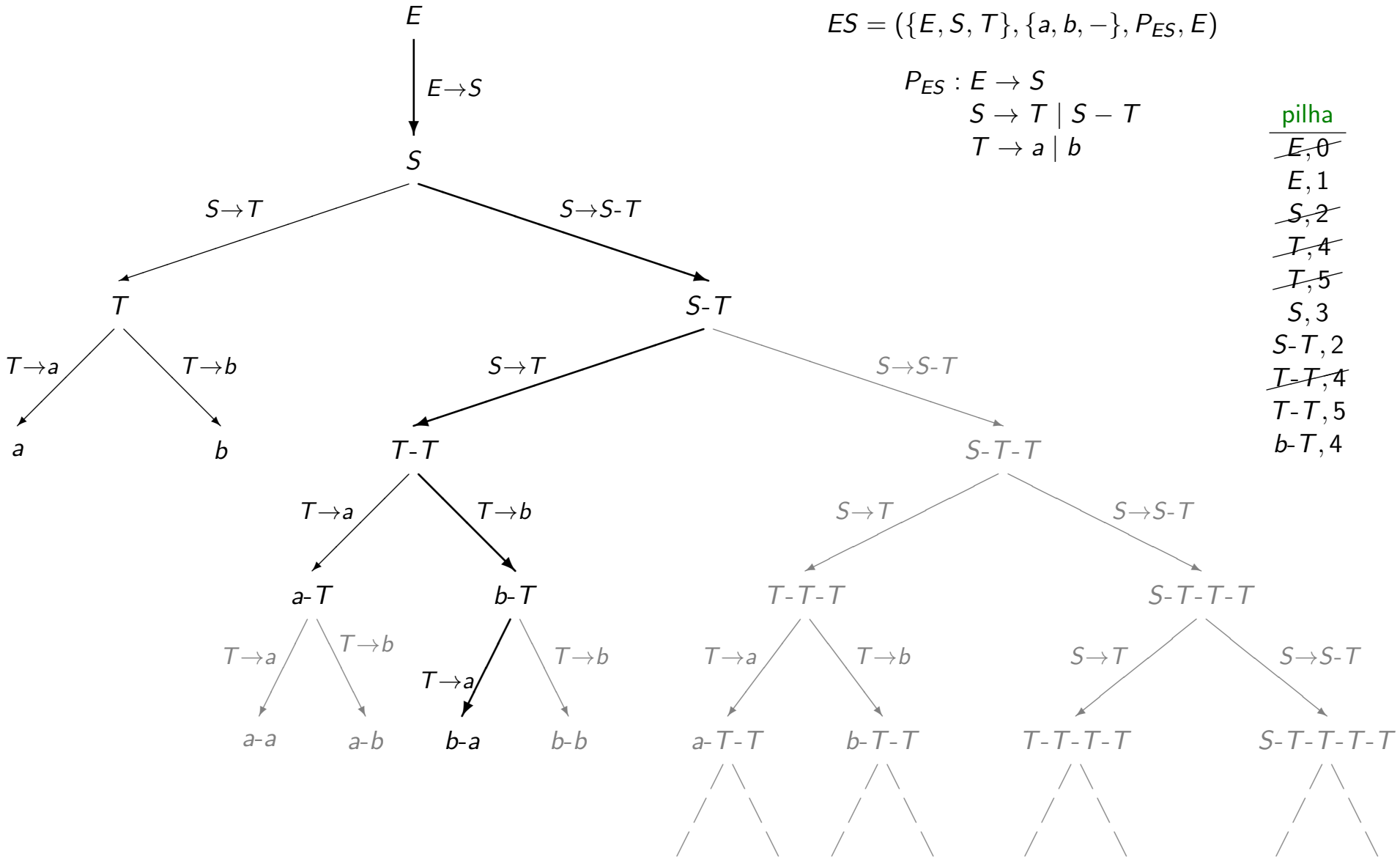
      $i \leftarrow 0$

  até *inviável* **ou** $q \in \Sigma^*$

até $q = p$ **ou** vazia(S)

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

Análise sintáctica descendente em profundidade para $b - a$



$ES = (\{E, S, T\}, \{a, b, -\}, P_{ES}, E)$

$P_{ES} : E \to S$
$S \to T \mid S - T$
$T \to a \mid b$

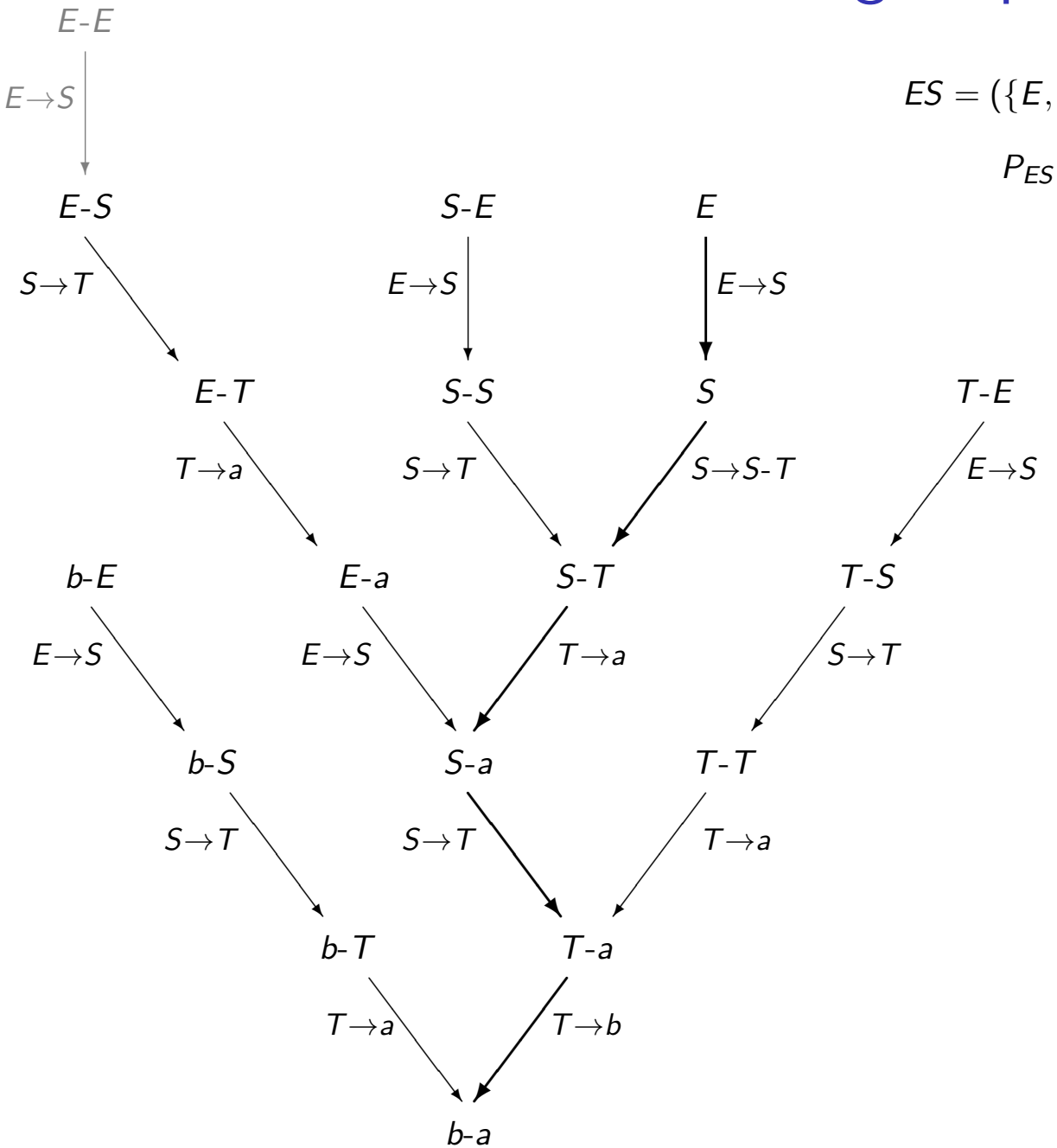
pilha
~~$E, 0$~~
$E, 1$
~~$S, 2$~~
~~$T, 4$~~
~~$T, 5$~~
$S, 3$
$S\text{-}T, 2$
~~$T\text{-}T, 4$~~
$T\text{-}T, 5$
$b\text{-}T, 4$

Algoritmo de 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 ← {p}                        % fila
repete
   q ← remove(Q)
   para cada produção A → w ∈ P
      % TRANSFERÊNCIA(S)
      para cada decomposição uwv de q, com v ∈ Σ*
         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 largura para $b - a$



$$ES = (\{E, S, T\}, \{a, b, -\}, P_{ES}, E)$$

$$P_{ES} : E \to S$$
$$S \to T \mid S - T$$
$$T \to a \mid b$$

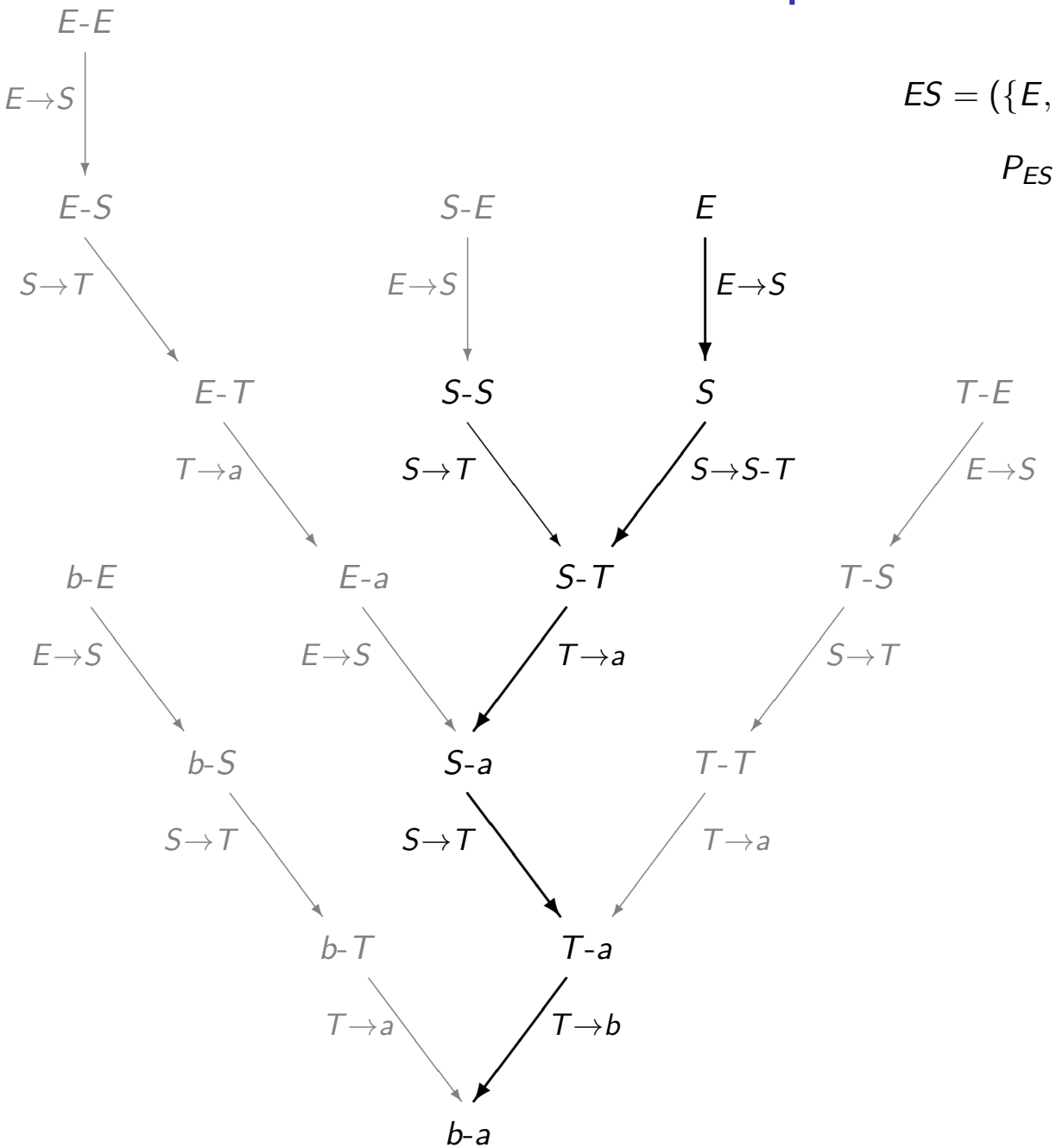
fila
~~$b - a$~~
~~$b - T$~~
~~$T - a$~~
~~$b - S$~~
~~$S - a$~~
~~$T - T$~~
~~$b - E$~~
~~$E - a$~~
~~$S - T$~~
~~$T - S$~~
~~$E - T$~~
~~$S - S$~~
~~S~~
$T - E$
$E - S$
$S - E$

Algoritmo de 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
  $[u, i, v] \leftarrow$ desempilha(S)
  inviável $\leftarrow$ *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**
      empilha($[u, j, v]$, S)
      $u \leftarrow qA$ % REDUÇÃO
      $i \leftarrow 0$
    se não existe tal j **e** $v \neq \lambda$ **então**
      TRANSFERÊNCIA(u, v)
      $i \leftarrow 0$
    se não existe tal j **e** $v = \lambda$ **então** *inviável* $\leftarrow$ *true*
  até $u = S$ **ou** *inviável*
até $u = S$ **ou** vazia(S)
se vazia(S) **então** REJEITA **senão** ACEITA

Análise sintáctica ascendente em profundidade para $b - a$



$$ES = (\{E, S, T\}, \{a, b, -\}, P_{ES}, E)$$

$$P_{ES} : E \to S$$
$$S \to T \mid S - T$$
$$T \to a \mid b$$

pilha
~~$\lambda, 0, b - a$~~
$b, 5, -a$
$T, 2, -a$
$S - a, 4, \lambda$
~~$S - T, 2, \lambda$~~
$S - T, 3, \lambda$
$S, 1, \lambda$