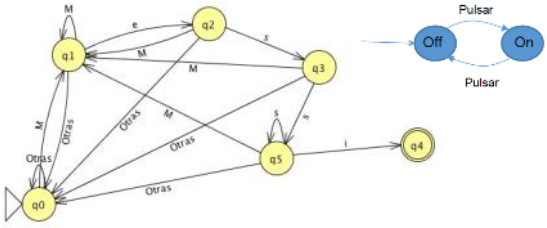
HISTORIA

Pazo

luns, 25 de setembro de 2023 10:09

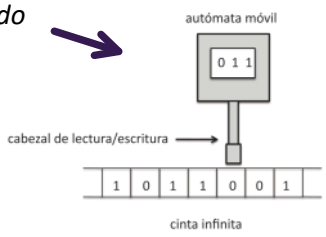
Exemplos de autómatas sinxelos



- **Teoría de autómatas:** estudo de "máquinas" ou dispositivos abstractos con capacidade de computación.

- **Turing (1936):** estudo dunha máquina abstracta
  - Determinar a fronteira entre o que se pode e o que non se pode facer cun computador.
  - Máquina de Turing:

    Dispositivo que manipula símbolos sobre unha tira de cinta dacordo cunha táboa de regras, capaz de implementar calquera problema matemático expresado como un algoritmo.



- **1940, 1950:** autómatas finitos (máis sinxelos ca a máquina de Turing).

- **Noam Chomsky (finais dos 50):** estudo das gramáticas formais.

- **Cook (1971):** separa os problemas que se poden resolver eficientemente dos que non (intratables).

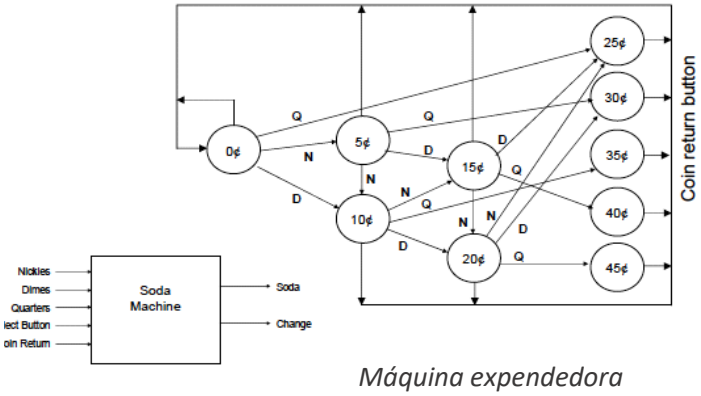
> *"O problema de satiisfacibilidade booleana (SAT) é NP-completo"*

UTILIDADE: ***autómatas*** *[de número de estados]* ***finito(s)***

- Software para o deseño e verificación do comportamento de **circuítos dixitais**.
- **Analizador léxico** dun compilador.
- Software para **explorar textos** buscando a aparición de certos **patróns**.
- Software para comprobar o **funcionamento dun sistema** cun **número finito de estados diferentes** (protocolos de comunicación, de intercambio seguro de información, ...).
- **Gramáticas**
  - Descrición de analizadores sintácticos (*parser*).
- **Expresións regulares**
  - Especificación de patróns de cadeas.
  - Deseño do software de verificación do formato de texto en formularios web.
- **Autómatas e complexidade**
  - Que pode facer unha computadora? Decidibilidade e computabilidade.
  - Que pode facer unha computadora *eficientemente*? Complexidade.

Current state	Nickle	Dime	Quarter	Select button	Coin return button
0¢	5¢ [1]	10¢ [1]	25¢ [1]	0¢	0¢ [4]
5¢	10¢ [1]	15¢ [1]	30¢ [1]	5¢	0¢ [4]
10¢	15¢ [1]	20¢ [1]	35¢ [1]	10¢	0¢ [4]
15¢	20¢ [1]	25¢ [1]	40¢ [1]	15¢	0¢ [4]
20¢	25¢ [1]	30¢ [1]	45¢ [1]	20¢	0¢ [4]
25¢	25¢ [2]	25¢ [2]	25¢ [2]	0¢ [3]	0¢ [4]
30¢	30¢ [2]	30¢ [2]	30¢ [2]	0¢ [3]	0¢ [4]
35¢	35¢ [2]	35¢ [2]	35¢ [2]	0¢ [3]	0¢ [4]
40¢	40¢ [2]	40¢ [2]	40¢ [2]	0¢ [3]	0¢ [4]
45¢	45¢ [2]	45¢ [2]	45¢ [2]	0¢ [3]	0¢ [4]

[1] – Accepts inserted money
[2] – Any money inserted is returned
[3] – Soda delivered and change returned
[4] – All inserted money is returned



Máquina expendedora

TEORÍA DE CONXUNTOS

luns, 25 de setembro de 2023 10:09

- $x \in X$, $x \notin X$
- $X = \{1, 2, 3\}$, $X = \{x \mid x \in N \text{ e } x \leq 3\}$
- $X \subseteq N$
- $X \cup Y = \{z \mid z \in X \text{ ou } z \in Y\}$
- $X \cap Y = \{z \mid z \in X \text{ e } z \in Y\}$
- $X - Y = \{z \mid z \in X \text{ e } z \notin Y\}$

- Complemento de X respecto a U: $\bar{X} = U - X$
- Leis de DeMorgan: $\overline{(X \cup Y)} = \bar{X} \cap \bar{Y}$, $\overline{(X \cap Y)} = \bar{X} \cup \bar{Y}$
- Cardinalidade: tamaño dun conxunto, $card(X)$
  - Finito
  - Infinito
    - contable ou numerable: correspondencia un a un cos números naturais
    - incontable

ALFABETOS E PALABRAS

- **Alfabeto:** conxunto finito non baleiro de símbolos, $\sum$.
- **Cadea ou palabra:** secuencia finita de símbolos pertencentes a un alfabeto.
  - Cadea baleira: λ, ε
  - Lonxitude da cadea: $|w|$
- **Potencias dun alfabeto:** conxunto de todas as cadeas dunha certa lonxitude que se poden formar cun alfabeto.
  - $\sum^k$
  - Clausura positiva: unión de todas as potencias do alfabeto ata o infinito. $\sum^+ = \sum^1 \cup \sum^2 \cup \sum^3 \cup \cdots$
  - Cierre ou clausura: unión da clausura positiva e a cadea baleira. $\sum^* = \sum^+ \cup \{\lambda\}$

OPERACIÓNS CON PALABRAS

- **Concatenación de palabras:** sexan *x* e *y* dúas cadeas, *xy* é a súa concatenación.
- **Potencia:** a potencia i-ésima dunha palabra *x*, x^i, fórmase pola concatenación *i* veces de *x*.
- **Reflexión:** se a palabra *x* está formada polos símbolos $A_1 \ldots A_n$, entón a palabra inversa de *x*, x^{-1}, fórmase invertindo a orda dos símbolos na palabra. $x^{-1} = A_n \ldots A_1$.

LINGUAXES

Dado un alfabeto $\sum$, calquera $L \subseteq \sum^*$ vai ser unha linguaxe de $\sum$.
O alfabeto sobre o que se defina a linguaxe debe ser finito, aínda que a linguaxe pode ter un número infinito de cadeas.

Exemplos:
- Linguaxe baleira: ∅
- Linguaxe que contén únicamente a cadea baleira: {λ}
- L = {w | w contén o mesmo número de 0 ca de 1}
- $L = \{0^n 1^n \mid n \geq 1\}$
- $L = \{a^i b^j c^k \mid j = i+k \text{ e } i, j, k > 0\}$

Un autómata de numero de estados finito ten, por definición, un número finito de estados; pero pode recoñecer un número infinito de cadeas dunha linguaxe.
p.e.: calquera cadea en binario que teña un 1 no medio, independentemente do número de 0 que veñan antes e 0 e 1 que vaian despois

Pazo

OPERACIÓNS CON LINGUAXES

martes, 26 de setembro de 2023 11:25

- **Unión:** $L_1 \cup L_2$, vai conter todas as palabras que pertenzan a calquera dos dous.
- **Intersección:** $L_1 \cap L_2$, vai conter todas as palabras que pertenzan a ambos.
- **Resta:** $L_1 - L_2$, vai conter todas as palabras que pertenzan a L_1 e non a L_2.
- **Concatenación:** $L_1 \cdot L_2$, vai conter todas as palabras que se poidan formar pola concatenación dunha palabra de L_1 e outra de L_2.
- **Potencia:** a potencia i-ésima dunha linguaxe é a concatenación *i* veces da linguaxe consigo mesma.
- **Reflexión:** L^{-}, está formada pola aplicación da reflexión a cada unha das palabras da linguaxe.

GRAMÁTICAS E AUTÓMATAS

Unha **gramática** establece a estrutura dunha linguaxe, é dicir, as sentencias que o forman; proporcionando as formas válidas nas que se poden combinar os símbolos do alfabeto.

Noam Chomsky

(Filadelfia, 1928)
Lingüista, filósofo, politólogo e activista estadounidense.

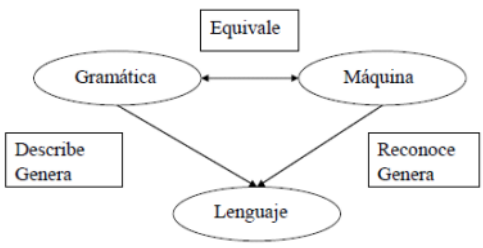
CLASIFICACIÓN DAS GRAMÁTICAS

- ***G0 ou Tipo 0:*** gramáticas sen restricións.
- ***G1 ou Tipo 1:*** gramáticas sensibles ao contexto.
- ***G2 ou Tipo 2:*** gramáticas independentes do contexto.
- ***G3 ou Tipo 3:*** gramáticas regulares

$$G3 \subseteq G2 \subseteq G1 \subseteq G0$$

Cada gramática é capaz de xerar un **tipo de linguaxe**.
A linguaxe L será de tipo "i" (i=0 1, 2, 3) se existe unha gramática de tipo "i" capaz de xerala ou describila.

Unha **máquina abstracta** ou **autómata** é un dispositivo teórico capaz de recibir e transmitir información. Produce unha cadea de símbolos á saída en función da cadea de símbolos que se presente á súa entrada e dos estados internos polos que transita.



Gramática	Lenguaje	Máquina / Compeljidad
Tipo 0: sin restricciones	Recursivamente enumerable	Máquina de Turing (MT) / Indecidible
Tipo 1: sensible al contexto	Sensible al contexto	Autómata Linealmente Acotado (ALA) / Exponencial
Tipo 2: contexto libre o independiente del contexto	Contexto libre o independiente del contexto	Autómata con Pila (AP) / Polinómica
Tipo 3: regular	Regular	Autómata Finito (AF) / Lineal

20

Pazo

INTRODUCIÓN

martes, 26 de setembro de 2023 12:33

Os autómatas [de número de estados] finito(s) son máquinas secuenciais (Mealy ou Moore) que recoñecen as linguaxes regulares. Clasifícanse:

- **Deterministas (AFD):** o autómata non pode estar en máis dun estado simultáneamente.
- **Non deterministas (AFN):** pode estar en varios estados ao mesmo tempo.

O *non determinismo* non engade ningunha linguaxe aos xa definidos polos AFD; só aumenta a eficiencia na descrición dunha aplicación.

AUTÓMATAS FINITOS DETERMINISTAS (AFD)

- **Determinista:** para cada entrada existe un único estado ao que o autómata pode chegar partindo do actual.

Un AFD consta de:

A = (Q, S, δ, q_0, F)

- Un conxunto finito de **estados**, Q.
- Un conxunto finito de **símbolos de entrada**, S.
- Unha **función de transición** (δ) que, dados un estado e unha entrada, devolve un estado; $\delta(q, a) = p$.
- Un **estado inicial** (un dos estados de Q), q_0.
- Un conxunto de **estados finais** ou de aceptación (subconxunto de Q), F.

FUNCIONAMENTO DUN AFD

A **linguaxe dun AFD** é o conxunto das cadeas que acepta.

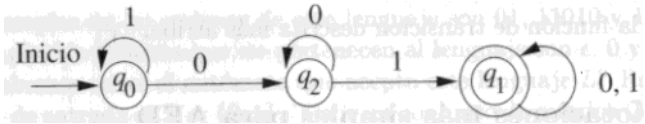
p.e: acepta todas as cadeas de 0 e 1 que conteñen a secuencia 01 nalgún lugar.

A = ($\{q_0, q_1, q_2\}$, {0, 1}, δ, q_0, $\{q_1\}$)

DIAGRAMA DE TRANSICIÓNS

É un **grafo** coas seguintes características:

- Un nodo por cada estado de Q.
- Un arco de q a p etiquetado con a para cada $\delta(q, a) = p$.
- Unha frecha dirixida ao estado inicial.
- Os estados finais están marcados por un dobre círculo.



TÁBOA DE TRANSICIÓNS

É a representación tabular da función δ. As filas representan os estados e as columnas as entradas. O estado inicial márcase cunha frecha e os finais con asteriscos (*).

	0	1
->q_0	q_2	q_0
*q_1	q_1	q_1
q_2	q_2	q_1

Pazo

EXTENSIÓN A CADEAS

xoves, 28 de setembro de 2023 11:12

- Función de transición: δ
- **Función de transición estendida**: $\hat{\delta}$
  - Dado un estado *q* e unha cadea *w*, devolve un estado *p*.
- Definición por indución da función de transición estendida.
  - Base: $\hat{\delta}(q,\lambda) = q$, $\hat{\delta}(q,w) = \delta(\hat{\delta}(q,x),a)$
  - Paso indutivo: $w = xa$
- Linguaxe dun AFD: linguaxe regular.

$$L(A) = \{w \mid \hat{\delta}(q_0,w) \text{ pertenece a } F\}$$

AUTÓMATAS FINITOS NON DETERMINISTAS (AFN)

- **Non determinista:** ten a capacidade de estar en varios estados simultáneamente.

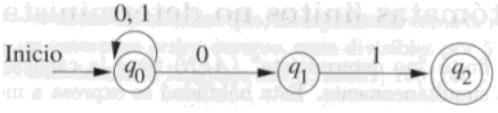
Un AFN:

A = $(Q, \sum, \delta, q_0, F)$

- A **función de transición** (δ) devolve un conxunto de estados; non un só.
- Acepta as linguaxes regulares, igual ca os AFD.
- Son máis compactas e doadas de deseñar ca os AFD.
- Sempre é posible a conversión dun AFN nun AFD.

Por exemplo: AFN que acepta as cadeas rematadas en 01.

A = $(\{q_0, q_1, q_2\}, \{0, 1\}, \delta, q_0, \{q_2\})$



FUNCIÓN DE TRANSICIÓN EXTENDIDA

- Base: $\hat{\delta}(q,\lambda) = \{q\}$
- Paso indutivo:

$$\left.\begin{array}{l} w = xa \\ \hat{\delta}(q,x) = \{p_1, p_2, \ldots, p_k\} \\ \bigcup_{i=1}^{k} \delta(p_i,a) = \{r_1, r_2, \ldots, r_m\} \end{array}\right\} \hat{\delta}(q,w) = \{r_1, r_2, \ldots, r_m\}$$

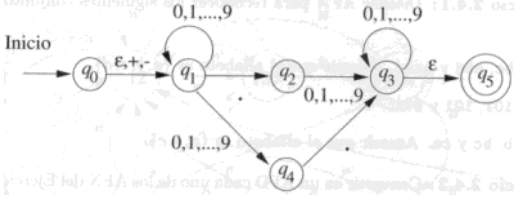
- Linguaxe dun AFN: $L(A) = \{w \mid \hat{\delta}(q_0,w) \cap F \neq \emptyset\}$

	ε	+, -	.	0, 1, ..., 9
->q_0	{q_1}	{q_1}	∅	∅
q_1	∅	∅	{q_2}	{q_1, q_4}
q_2	∅	∅	∅	{q_3}
q_3	{q_5}	∅	∅	{q_3}
q_4	∅	∅	{q_3}	∅
*q_5	∅	∅	∅	∅

AFN CON TRANSICIÓNS ε (AFN-ε)

Un AFN con transicións ε proporciona "facilidades de programación", mais non expande a clase de linguaxes que aceptan os autómatas formais.

- A **función de transición** (δ) é agora unha función dun estado de Q e un elemento de $\sum \cup \{\varepsilon\}$.
- O símbolo ε non pode formar parte do alfabeto.



Por exemplo: AFN-ε que recoñece decimais.

Pazo

CLAUSURAS RESPECTO DE ε

luns, 9 de outubro de 2023 11:30

Clausura do estado *q* respecto de ε, *CLAUSε*(q):

- **Base**: o estado *q* está en *CLAUSε*(q).
- **Paso indutivo**: se δ é a función de transición do AFN-ε, e o estado *p* está en *CLAUSε*(q), entón *CLAUSε*(q) contén todos os estados de δ(*p*, ε); é dicir: a clausura dun estado é o propio estado máis todos aqueles aos que se poida chegar sen introducir ningunha palabra (ε).

TRANSICIÓNS E LINGUAXES EXTENDIDAS

Definición recursiva de $\hat{\delta}$:

- Base: $\hat{\delta}(q,\varepsilon) = CLAUS_\varepsilon(q)$
- Paso indutivo: sexa $w = xa$, onde *a* é un elemento de $\sum$
  - Sexa $\hat{\delta}(q,x) = \{p_1, p_2, \dots, p_k\}$
  - Sexa $\bigcup_{i=1}^{k} \delta(p_i, a) = \{r_1, r_2, \dots, r_m\}$

  $\rightarrow \hat{\delta}(q,w) = \bigcup_{j=1}^{m} CLAUS_\varepsilon(r_j)$
- Linguaxe dun AFD-ε, E=(Q, $\sum$, δ, q_0, *F*): $L(E) = \{w \mid \hat{\delta}(q_0, w) \cap F \neq \emptyset\}$

EXEMPLO NA BUSCA DE TEXTO

"Dado un conxunto de palabras, atopar todos os documentos que conteñen unha ou máis delas."
Características que fan apropiado o uso de autómatas nunha aplicación:

- O repositorio a analizar cambia rápidamente (*noticias do día*, *robot de compras*, ...)
- Os documentos non poden ser catalogados: *Amazon (páxinas xeradas a partir de consultas)*.

AFN PARA BUSCA DE TEXTO

- Estado inicial con transición a si mesmo para cada símbolo de entrada (conxetura).
- Para cada palabra clave a_1, a_2, ..., a_k hai k estados q_1, q_2, ..., q_k.
- O estado q_k indicará que a palabra a_1, a_2, ..., a_k foi aceptada.

EQUIVALENCIA ENTRE AFD E AFN

Toda linguaxe descrita por un AFN *de n* estados pode ser descrito tamén por un AFD que teña, como máximo, 2^n estados.

Para **demostrar** que un AFD pode facer o mesmo ca un AFN constrúense todos os subconxuntos do conxunto de estados do AFN.

CONSTRUCIÓN DE SUBCONXUNTOS

Teorema: Dado o AFN N=(Q_N, $\sum$, δ_N, q_0, F_N), obter o AFD D=(Q_D, $\sum$, δ_D, {q_0}, F_D) tal que L(D)=L(N).

- Estado inicial: conxunto co estado inicial de N.
- Q_D: conxunto de subconxuntos de Q_N. Se Q_N ten *n* estados, Q_D terá 2^n. Elimínanse os estados non accesibles.
- F_D: conxunto de subcconxuntos S de Q_N tales que $S \cap F_N \neq \emptyset$.
- Para cada $S \subseteq Q_N$ e cada símbolo de entrada *a*:

$$\delta_D(S,a) = \bigcup_{p \text{ en } S} \delta_N(p,a)$$

Teorema: Unha linguaxe *L* é aceptada por algún AFN se e só se *L* é aceptada por algún AFD.

Pazo

- **Base:** o conxunto dun elemento que contén o estado inicial de N é accesible.
- **Paso indutivo:** determinamos que o conxunto *S* de estados é accesible; así que para cada símbolo de entrada *a* calcúlase o conxunto de estados $\delta_D(S, a)$, que tamén van ser accesibles.

ELIMINACIÓN DE TRANSICIÓNS ε

- Q_D é o conxunto de subconxuntos de Q_E.
- $q_D = CLAUS\varepsilon(q_0)$
- $F_D = \{S \mid S \text{ pertence a } Q_D \text{ e } S \cap F_N \neq \emptyset\}$
- $\delta_D(S, a)$: $S = \{p_1, p_2, \dots, p_k\}$

$$\bigcup_{i=1}^{k} \delta(p_i, a) = \{r_1, r_2, \dots, r_m\}$$

$$\delta_D(S, a) = \bigcup_{j=1}^{m} CLAUS \in (r_j)$$

EQUIVALENCIA DE ESTADOS

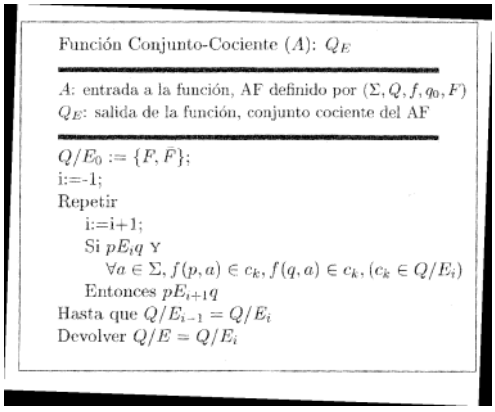
- Dous estados *p* e *q* dun AFD son equivalentes se, para toda cadea de entrada w, $\hat{\delta}(p, w)$ é un estado de aceptación se e só se $\hat{\delta}(q, w)$ é un estado de aceptación.

Teorema: A equivalencia de estados é transitiva. É dicir, que se para un AFD dous estados *p* e *q* son equivalentes e *q* e *r* tamén, entón *p* e *r* son equivalentes.

Teorema: Se para cada estado *q* dun AFD se crea un bloque que contén a *q* e a todos os estados equivalentes a *q*, os bloques de estados formarán unha partición do conxunto de estados.

- Cada estado estará nun único bloque.
- Todos os membros dun bloque son equivalentes.
- Dous estados de bloques diferentes non poden ser equivalentes.

Conxunto cociente, Q/E: partición do conxunto de estados en clases. Cada clase contén estados equivalentes entre si.



Función Conjunto-Cociente (A): Q_E

A: entrada a la función, AF definido por (Σ, Q, f, q_0, F)
Q_E: salida de la función, conjunto cociente del AF

$Q/E_0 := \{F, \bar{F}\}$;
i:=-1;
Repetir
  i:=i+1;
  Si pE_iq Y
    $\forall a \in \Sigma, f(p,a) \in c_k, f(q,a) \in c_k, (c_k \in Q/E_i)$
  Entonces $pE_{i+1}q$
Hasta que $Q/E_{i-1} = Q/E_i$
Devolver $Q/E = Q/E_i$

EQUIVALENCIA DE LINGUAXES REGULARES

Sexan L e M dúas linguaxes representadas dalgunha forma:

1) Convértense as representacións en AFD.
2) Compróbase se os estados iniciais de ambos AFD sson equivalentes.
3) Se son equivalentes, entón L=M. Se non, serán diferentes.

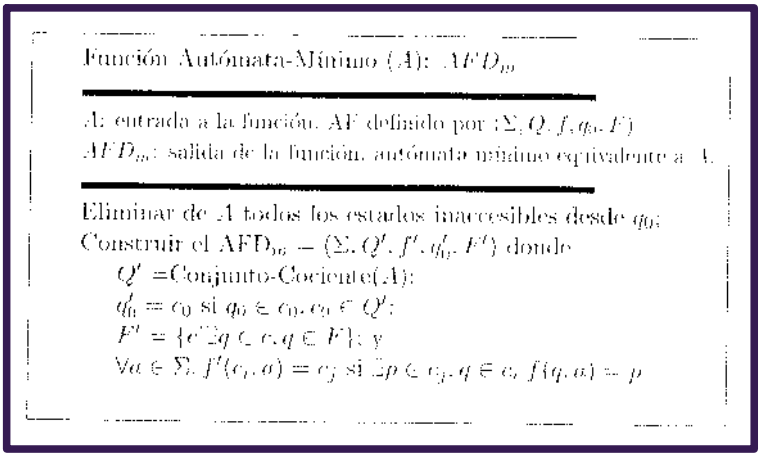
Pazo

MINIMIZACIÓN DUN AFD

luns, 9 de outubro de 2023 23:40

Para todo AFD é posible atopar un AFD equivalente con igual ou menos número de estados que acepte a mesma linguaxe. Dito AFD é único.

1) Elimínanse os estados non accesibles dende o inicial.
2) Divídese o conxunto de estados Q en bloques de estados mutuamente equivalentes.
3) Constrúese o AFD mínimo *B* equivalente a *A*, empregando os bloques de estados resultantes como estados do novo AFD.
   a. A función de transición de *B* é $\gamma(S, a) = T$, onde *S* e *T* son bloques de estados. Calquera bloque de *S* debe levar con entrada *a* a un estado de *T*.
   b. O estado inicial de *B* é o bloque [único] que contén o estado inicial de *A*.
   c. O conxunto de estados de aceptación de *B* é o conxunto de bloques que conteñen os estados de aceptación de A.



Función Autómata-Mínimo (A): AFD_m

A: entrada a la función. AF definido por (Σ, Q, f, q_0, F)
AFD_m: salida de la función, autómata mínimo equivalente a A.

Eliminar de A todos los estados inaccesibles desde q_0;
Construir el $AFD_m = (\Sigma, Q', f', q_0', F')$ donde
Q' = Conjunto-Cociente(A);
$q_0' = c_0$ si $q_0 \in c_0, c_0 \in Q'$;
$F' = \{c' | \exists q \in c, q \in F\}$; y
$\forall a \in \Sigma$, $f'(c_i, a) = c_j$ si $\exists p \in c_j, q \in c_i$ $f(q, a) = p$

Pazo

EXPRESIÓNS REGULARES (ER)

martes, 10 de outubro de 2023 10:11

As expresións regulares denotan linguaxes: 01* + 10*

Representa unha unión, non unha concatenación.

- Son a descrición alxebraica das linguaxes regulares.
- Forma declarativa de expresar as cadeas que queremos aceptar.
- Empréganse como linguaxe de entrada en moitos sistemas de proceso de cadeas:
  - *Especificación de caracteres no comando grep de UNIX.*
  - *Deseño de analizadores léxicos mediante Lex ou Flex.*
    - *Acepta ER (formas das unidades sintácticas) e produce un AFD que recoñece a unidade sintáctica que aparece a continuación na entrada.*
  - *Deseño de verificadores do formato de texto en formularios web (JavaScript, Perl, ...)*

OPERADORES DAS ER

- **Unión** de dúas linguaxes *L* e *M*, $L \cup M$: conxunto de cadeas que pertencen a *L*, a *M* ou a ambas.
- **Concatenación** de dúas linguaxes *L* e *M*, *L.M* (ou *LM*): conxunto de cadeas formadas pola concatenación dunha cadea de *L* e outra de *M*.
- **Clausura, estrela ou clausura de Kleene** dunha linguaxe *L*, *L**: conxunto de cadeas formado pola concatenación de calquera número de cadeas de *L*. Formalmente:

$$L^* = \bigcup_{i=0}^{\infty} L^i$$

  - Só existen dúas linguaxes con clausura non infinita:
    - $L = \{\emptyset\}. L^* = \{\varepsilon\}$, xa que $\emptyset^0 = \{\varepsilon\}$ e $\emptyset^i = \emptyset$ para $i > 0$
    - $L = \{\varepsilon\}.\ L^* = \{\varepsilon\}$

Orde de precedencia: () → * → . (concatenación) → ∪

ÁLXEBRA DE ER

- □ Propiedade conmutativa da unión: $L + M = M + L$
- □ Propiedade asociativa da unión: $(L + M) + N = L + (M + N)$
- □ Propiedade asociativa da concatenación: $(LM)N = L(MN)$
  - A concatenación non é conmutativa: $LM \neq ML$
- □ $\emptyset$ é o elemento identidade da unión: $\emptyset + L = L + \emptyset = L$
- □ ε é o elemento identidade da concatenación: $\varepsilon L = L\varepsilon = L$
- □ $\emptyset$ é o elemento nulo da concatenación: $\emptyset L = L\emptyset = \emptyset$
- □ Propiedade distributiva pola esquerda da concatenación respecto da unión: $L(M + N) = LM + LN$
- □ Propiedade distributiva pola dereita da concatenación respecto da unión: $(M + N)L = ML + NL$
- □ Operador idempotente: o resultado de aplicalo a dous valores iguais é o mesmo valor.
  - Propiedade de idempotencia da unión: $L + L = L$
- □ Propiedades relativas ás clausuras:
  - $(L^*)^* = L^*$
  - $\emptyset^* = \varepsilon$
  - $\varepsilon^* = \varepsilon$
  - $L^+ = LL^* = L^*L$
  - $L^* = L^+ + \varepsilon$
  - $L? = L + \varepsilon$

CONSTRUCIÓN DE ER

- **Base:**
  - As constantes ε e $\emptyset$ son ER. $L(\varepsilon) = \{\varepsilon\}$ e $L(\emptyset) = \{\emptyset\}$.
  - Se *a* é un símbolo, ***a*** é a ER da linguaxe$L(a) = \{a\}$.
- **Paso indutivo:**
  - Se *E* e *F* son ER, $E + F$ é unha ER e $L(E + F) = L(E) \cup L(F)$
  - Se *E* e *F* son ER, *EF* é unha ER e $L(EF) = L(E)L(F)$
  - Se *E* é ER, E^* é unha ER e $L(E^*) = (L(E))^*$
  - Se *E* é ER, (E) é unha ER e $L\big((E)\big) = L(E)$

Pazo

AUTÓMATAS FINITOS E ER

Pazo

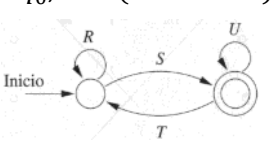
martes, 10 de outubro de 2023 11:26

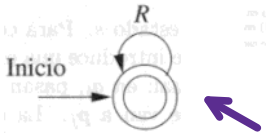
As **expresións regulares** definen as linguaxes regulares igual ca os autómatas finitos:

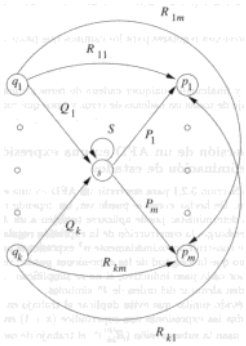
- Toda linguaxe definida por un autómata formal (definido, non definido ou con transicións ε) tamén pode definirse mediante unha ER.
- Toda linguaxe definida por unha ER pode definirse mediante un autómata formal.

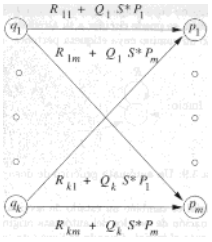
DE AUTÓMATAS FINITOS A ER:

- **Eliminación de estados:** nos arcos aparecen ER.
- Para cada estado de aceptación q, aplícase o proceso de redución. Elimínanse todos os estados excepto q e o estado inicial q_0.
- Se $q \neq q_0$, $L = (R^* + SU^*T)^*SU^*$







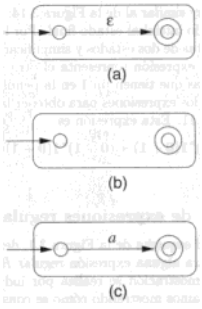


- Se o estado inicial é estado de aceptación, haberá que eliminar todos os estados agás o inicial: $L = R^*$
- A ER desexada é a unión das cadeas obtidas do autómata para cada estado de aceptación.

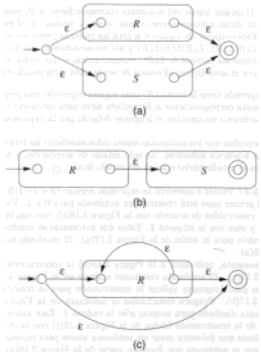
DE ER A AUTÓMATAS FINITOS:

- **Teorema**: toda linguaxe definido por unha ER pode definirse tamén mediante un AF.
- **Proba**: sexa $L = L(R)$, para ler a ER R. Demostrarase que $L = L(E)$ para algún AFN-ε E.

- **Base**:



- **Paso indutivo**:
  - $R + S$: $L(R) \cup L(S)$
  - RS: $L(R)L(S)$
  - R^*: $L(R^*)$



APLICACIÓNS DAS ER

- Busca de **patróns de texto** mediante ER que dan unha "imaxe" do patrón que se quere recoñecer.
- Aplicacións:
  - Analizadores léxicos.
  - Busca de textos.
  - Software de verificación do formato de texto en formularios web.

BUSCA DE PATRÓNS EN TEXTOS

- Busca eficiente de palabras nun gran repositorio de texto, como a Web.
- A notación das ER é valiosa para describir patróns de busca interesantes ou de texto vagamente definidos (porque é posible modificar as ER con pouco esforzo).
- Posibilidade de pasar de ER a unha implementación eficiente (autómatas).

p.e.: direcións de rúas en páxinas web Calle|c Avenida|Avda Plaza|Pza \\.) [A Z][a z]*([A Z][a z]*)* [0 9]9]++[A Z]?

PROPIEDADES DAS LINGUAXES REGULARES

xoves, 19 de outubro de 2023 10:18

Descrición das linguaxes regulares:

- ✧ AFD
- ✧ AFN
- ✧ AFN-ε
- ✧ Expresións regulares

Non todas as linguaxes son regulares: $L = \{0^n 1^n \mid n \geq 1\}$
Se fose regular, existiría un AF con k estados que a recoñecería.

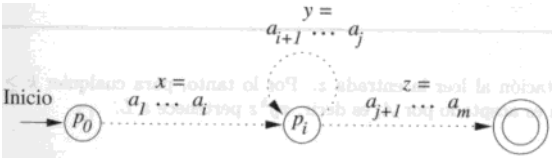
LEMA DO BOMBEO PARA LINGUAXES REGULARES

Para unha linguaxe regular [infinito], o cumplimento do **lema do bombeo (LB)** é unha condición necesaria, pero non suficiente.

Teorema: sexa *L* unha linguaxe regular, entón existe unha constante *n* (que depende de *L*) tal que, para toda cadea *w* pertencente a *L*, con $|w| \geq n$, podemos dividir *w* en tres cadeas, *w = xyz*, de modo que:

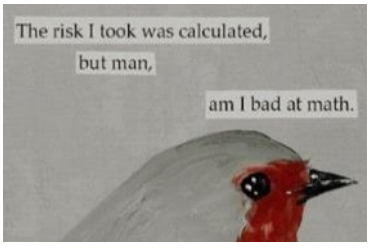
→ $y \neq \varepsilon$
→ $|xy| \leq n$
→ Para todo k ≥ 0, a cadea $xy^k z$ tamén pertence a *L*.

Sempre é posible atopar unha cadea *y* non baleira e non demasiado lonxe do comezo de *w* que se pode "bombear".



APLICACIÓN DO LEMA DO BOMBEO

1) Eliximos unha linguaxe *L* para a que tratamos de demostrar que non é regular.
2) O valor de *n* é descoñecido, polo que debemos considerar calquera posible valor.
3) Eliximos *w* (podemos empregar *n* como parámetro).
   a. Se para un *n* suficientemente grande non podemos escoller *w*, L será regular.
4) Repetir para todas as descomposicións:
   a. Escoller unha descomposición de *w* en *xyz* suxeita ás restricións::
      i. $y \neq \varepsilon$
      ii. $|xy| \leq n$
   b. Se $xy^k z$ pertence a *L* para todo valor de *k*
      i. Verifícase o LB
      ii. Non se pode afirmar que a linguaxe sexa regular
      iii. Non é necesario probar con outras descomposicións (remata o algoritmo).
5) Se 4.b non se cumpriu para ningunha descomposición, non se verifica o LB e, polo tanto, a linguaxe non é regular.



Pazo

DEFINICIÓN DE GIC

xoves, 19 de outubro de 2023 10:50

$$G = (V, T, P, S)$$

As GIC están formadas por catro compoñentes:

- O conxunto finito de **símbolos non terminais** (***V***) ou variables, que permiten representar subconxuntos da linguaxe ou estados intermedios na xeración das palabras da linguaxe.
- O alfabeto de **símbolos terminais** (***T***), que son os símbolos finais da linguaxe.
- Un conxunto finito de **producións ou regras** (***P***), que indican as transformacións posibles dende os símbolos non terminais ás palabras da linguaxe.
- O **símbolo inicial ou axioma** (***S***) da gramática (unha das variables), a partir da que se obtén calquera palabra da linguaxe.

As regras da gramática están formadas por:

- Unha variable, cabeza da produción
- O símbolo de produción →
- Unha cadea de cero ou máis símbolos terminais, que son o corpo da produción

$$B \to x \mid B \in V \; e \; x \in (V \cup T)^*$$

DISTINTAS GRAMÁTICAS

As distintas gramáticas admiten distintas formas para as producións:

- **Tipo 0:** non restrinxida ou recursivamente enumerable.

$$\begin{gathered} x \to y \\ x \in (NT/T)^+ \\ y \in (NT/T)^* \end{gathered}$$

- **Tipo 1:** sensible ao contexto.

$$\begin{gathered} \alpha \to \beta \; ; |\alpha| \leq |\beta| \\ \alpha = z_1 x \, z_2 \\ \beta = z_1 y \, z_2 \\ z_1, z_2 \in T^* \\ x \in NT \\ y \in (NT/T)^+ \end{gathered}$$

- **Tipo 2:** libre de contexto.

$$\begin{gathered} x \to y \\ x \in NT \\ y \in (NT/T)^* \end{gathered}$$

! Estamos usando NT como V

- **Tipo 3:** regular.

$$\begin{gathered} \alpha \to \beta \\ \alpha \in NT \\ \beta \in \begin{cases} aB \\ Ba \\ b \end{cases} \\ B \in NT \\ a \in T^+ \\ b \in T^* \end{gathered}$$

GRAMÁTICAS REGULARES

As **linguaxes regulares** poden asociarse a unha gramática de tipo 3 ou regular. Estas gramáticas poden ser lineares pola dereita ou lineares pola esquerda:

- Unha gramática $G = (V, T, P, S)$ é **linear pola dereita** se todas as súas producións son da forma
  - $A \to xB$
  - $A \to x$

  Onde *A* e *B* pertencen a *V* e *x* Pertence a T^*
- Unha gramática $G = (V, T, P, S)$ é **linear pola esquerda** se todas as súas producións son da forma
  - $A \to Bx$
  - $A \to x$

Pazo

DERIVACIÓNS DUNHA GRAMÁTICA

xoves, 26 de outubro de 2023 12:01

Sexa $G = (V, T, P, S)$ e $\alpha A \beta$ unha cadea de símbolos terminais e non terminais (variables), onde *A* está en *V* e α e β están en $(V \cup T)^*$.
Sexa $A \to \gamma$ unha produción de *G*.
Entón:

$$\alpha A \beta \underset{G}{\Rightarrow} \alpha \gamma \beta$$

Unha **derivación dunha sentencia ω** é a secuencia de substitucións de non terminais que, partindo do símbolo inicial *S*, produce como resultado ω.

Exemplo:

$$E \Rightarrow E * E \Rightarrow I * E \Rightarrow a * E \Rightarrow$$
$$a * (E) \Rightarrow a * (E + E) \Rightarrow a * (I + E) \Rightarrow a * (a + E) \Rightarrow$$
$$a * (a + I) \Rightarrow a * (a + I0) \Rightarrow a * (a + I00) \Rightarrow a * (a + b00)$$

LINGUAXE DUNHA GRAMÁTICA

Se $G = (V, T, P, S)$ é unha GIC, a **linguaxe de *G*** será:

$$L(G) = \{ w \; que \; están \; en \; T^* \mid S \overset{*}{\underset{G}{\Rightarrow}} w \}$$

- Se $G = (V, T, P, S)$ é unha GIC, calquera cadea $\alpha \in (V \cup T)^*$, tal que $S \overset{*}{\Rightarrow} \alpha$ é unha ***forma sentencial***.
- A linguaxe *L(G)* está formada polas formas sentenciais que están en T^* e denomínanse **sentencias**.

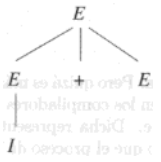
ÁRBORES DE DERIVACIÓN

Representación das **derivacións en forma de árbore**.
Sexa $G = (V, T, P, S)$. A árbore de derivación para *G* vai ter as seguintes características:

- Cada nodo interior está etiquetado cunha variable.
- Cada folla está etiquetada cunha variable, un terminal ou ε. Neste último caso, debe ser o único fillo do seu nodo proxenitor.
- Se un nodo interior está etiquetado con *A* e os seus fillos están etiquetados con $X_1, X_2 \ldots X_k$ (de esquerda a dereita), entón $A \to X_1, X_2 \ldots X_k$ é unha produción de *P*.

Exemplo:

$$E \overset{*}{\Rightarrow} I + E$$



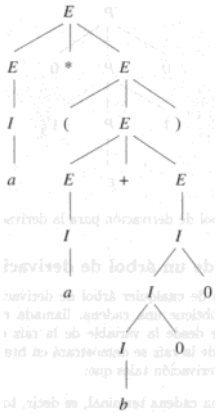
RESULTADO DUNHA ÁRBORE DE DERIVACIÓN

Concatenando as follas dunha árbore (terminais) dende a esquerda obtense a súa **cadea resultado**, que se deriva dende a raíz (símbolo inicial).

Exemplo:

$$E \overset{*}{\Rightarrow} a^*(a + b00)$$

1. $E \to I$
2. $E \to E + E$
3. $E \to E * E$
4. $E \to (E)$
5. $I \to a$
6. $I \to b$
7. $I \to Ia$
8. $I \to Ib$
9. $I \to I0$
10. $I \to I1$



APLICACIÓNS DAS GIC

- Descrición de linguaxes de programación (análise sintáctica).
- Linguaxes de marcado (HTML, XML).

Pazo

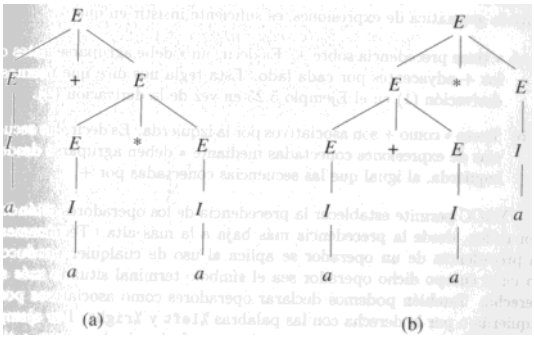
AMBIGÜIDADE

sábado, 28 de outubro de 2023 22:23

Unha GIC $G = (V, T, P, S)$ é ambigua se existe polo menos unha cadea *W* en T^* para a que podemos atopar dúas árbores de derivación distintos coa raíz etiquetada como *S* e con resultado *w*.

A existencia de derivacións diferentes para unha cadea non supón un defecto na gramática, mais *a existencia de árbores de derivación diferentes si*.

1. $E \rightarrow I$
2. $E \rightarrow E + E$
3. $E \rightarrow E * E$
4. $E \rightarrow (E)$
5. $I \rightarrow a$
6. $I \rightarrow b$
7. $I \rightarrow Ia$
8. $I \rightarrow Ib$
9. $I \rightarrow I0$
10. $I \rightarrow I1$



FORMAS NORMAIS PARA GIC

- As gramáticas en **formas normais** poden xerar todas as linguaxes independentes de contexto (LIC).
  - Forma normal de Chomsky
  - Forma normal de Greibach
- As gramáticas en formas normais reducen a complexidade para a obtención das derivacións.
- Para obter unha gramática en forma normal é necesario realizar unha serie de transformacións que non modifican a linguaxe xerada:
  - Eliminación de producións ε
  - Eliminación de producións unitarias (regras de encadeamento)
  - Eliminación de símbolos inútiles

ELIMINACIÓN DE PRODUCIÓNS ε

◇ Unha variable é anulable se $A \overset{*}{\Rightarrow} \varepsilon$

◇ Algoritmo: Sexa $G = (V, T, P, S)$ unha GIC.
Atoparemos todos o s símbolos anulables de G mediante o seguinte algoritmo:

- Base: se $A \rightarrow \varepsilon$ é unha produción de *G*, *A* é anulable
- Paso indutivo: se existe unha produción $B \rightarrow C_1 C_2 \ldots C_k$ na que as $C_{i,i=1\ldots k}$ son anulables, *B* é anulable.

Construción dunha gramática sen producións ε:

- Sexa $G = (V, T, P, S)$ unha GIC.
- Determínanse todos os símbolos anulables de *G*.
- Constrúese a gramática $G_1 = (V, T, P_1, S)$, onde P_1 se determina como segue:
  - Para cada produción $A \rightarrow X_1 X_2 \ldots X_k$ onde $k \geq 1$, supoñamos que *m* dos *k* símbolos son anulables.
  - A nova gramática vai ter 2^m versións desta produción (as X_i anulables estarán presentes ou ausentes en todas as combinacións posibles).
  - Se *m* = *k*, non se vai incluír o caso de todas as X_i ausentes.
  - As producións de *P* da forma $A \rightarrow \varepsilon$ non van estar en P_1.
  - Se ε forma parte da linguaxe, engádese a produción $S \rightarrow \varepsilon$.

Pazo

ELIMINACIÓN DE PRODUCIÓNS UNITARIAS

sábado, 28 de outubro de 2023 23:02

- ◇ **Produción unitaria**: $A \to B$, onde *A* e *B* son variables.
  (Engade pasos adicionais nas derivacións)
- ◇ **Pares unitarios**: pares de variables *A* e *B* tales que $A \overset{*}{\Rightarrow} B$, empregando unha secuencia que só fai uso de producións unitarias.
- ◇ Obtención dos pares unitarios (*A*, *B*):
  - Base: (*A*, *A*) é un par unitario para toda variable *A*.
  - Paso indutivo: sexa (*A*, *B*) un par unitario e $B \to C$ unha produción onde *C* é unha variable, entón (*A*, *C*) é un par unitario.

Construción dunha gramática sen producións unitarias:

- Sexa $G = (V, T, P, S)$ unha GIC.
- Determínanse todos os pares unitarios de *G*.
- Constrúese a gramática $G_1 = (V, T, P_1, S)$, onde P_1 se determina como segue:
  - Para cada par unitario (*A*, *B*), engadimos a P_1 todas as producións $A \to \alpha$, onde $B \to \alpha$ é unha produción non unitaria de *P*.

ELIMINACIÓN DE SÍMBOLOS INÚTILES

- ◇ Un símbolo X é **útil** para unha gramática $G = (V, T, P, S)$ se existe algunha derivación da forma $S \overset{*}{\Rightarrow} \alpha X \beta \overset{*}{\Rightarrow} w$, onde *w* está en T^*.
- ◇ *X* é **xerador** se $X \overset{*}{\Rightarrow} w$
  - Todo símbolo terminal é xerador
- ◇ *X* é **acadable** se existe unha derivación $S \overset{*}{\Rightarrow} \alpha X \beta$ para algún α e β.
- ◇ **Todo símbolo útil é xerador e acadable.**
- ◇ Eliminación de símbolos inútiles:
  - Elimínanse os símbolos non xeradores.
  - Elimínanse os símbolos non acadables.

Cálculo de símbolos xeradores:

- Sexa $G = (V, T, P, S)$ unha gramática.
- Base: todo símbolo de *T* é xerador.
- Paso indutivo: dada unha produción $A \to a$, onde todo símbolo de *a* é xerador, entón *A* é xerador (inclúese o caso a = ε.

Cálculo de símbolos acadables:

- Sexa $G = (V, T, P, S)$ unha gramática.
- Base: *S* é acadable.
- Paso indutivo: dada unha variable *A* acadable, os símbolos dos corpos das producións con *A* como cabeza serán acadables.

FORMA NORMAL DE CHOMSKY

Toda LIC non baleira ten unha gramática *G* na que todas as producións teñen unha das formas seguintes:

- $A \to BC$, onde *A*, *B* e *C* son variables.
- $A \to a$, onde *A* é unha variable e *a* un símbolo terminal.
- $S \to \varepsilon$

Dise que *G* está en **Forma Normal de Chomsky** (FNC):

→ Unha cadea de lonxitude *n* analízase en 2*n*-1 pasos.
→ A árbore de derivación é binaria e a súa profundidade máxima é *n*.
→ Úsase como algoritmo o método CYK.

Pazo

TRANSFORMAR UNHA GRAMÁTICA A FNC

domingo, 29 de outubro de 2023 12:30

Para **transformar** unha gramática a FNC:

- Non pode ter producións ε, producións unitarias nin símbolos inútiles.
- As súas producións teñen a forma $S \to \varepsilon$, $A \to a$ (xa están en FNC) ou ben un corpo de lonxitude dous ou máis. Vai haber que:
  a) Conseguir que nos corpos de lonxitude dous ou máis só aparezan variables.
  b) Descompor os corpos de lonxitude tres ou máis nunha cascada de producións con corpos onde só aparezan dúas variables.

Construción para (a):

- Para cada símbolo terminal *a* que apareza nun corpo de lonxitude dous ou máis, créase unha nova variable *A*.
- Esta variable só vai ter a produción $A \to a$.
- Substitúense as aparicións de *a* por *A* sempre que apareza nun corpo de lonxitude maior ou igual ca dous.

Construción para (b):

- Descompóñense as producións da forma $A \to B_1 B_2 \ldots B_k$ para k ≥ 3 nun grupo de producións con dúas variables en cada corpo.
- Introdúcense *k*-2 variables novas, $C_1 C_2 \ldots C_{k-2}$.
- Reemplázase a produción orixinal polas *k*-1 producións $A \to B_1 C_1$, $C_1 \to B_2 C_2$,..., $C_{k-3} \to B_{k-2} C_{k-2}$, $C_{k-2} \to B_{k-1} C_{k-1}$.

FORMA NORMAL DE GREIBACH

Toda LIC non baleira *é L*(*G*) para algunha gramática *G* na que todas as producións teñen a forma $A \to a\alpha$, onde *a* é un símbolo terminal e α unha cadea de cero ou máis variables.

O uso dunha produción introduce un símbolo terminal nunha forma sentencial:

→ Unha cadea de lonxitude *n* ten unha derivación de *n* pasos.
→ Un analizador sintáctica descendente parará a profundidade *n*.
→ Nunca vai haber recursividade pola esquerda.

Pazo

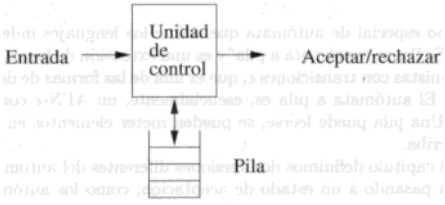
INTRODUCIÓN AOS AP

domingo, 29 de outubro de 2023 13:09

- Un **autómata con pila** (AP) é un AFN con transicións ε e cunha pila (LIFO) na que se pode almacenar unha cadea de "símbolos de pila".
- O AP pode recordar unha cantidade **infinita** de información.
- Os AP recoñecen todos os LIC e só estes (existen linguaxes que non son LIC, coma $\{0^n1^n2^n \mid n \geq 1\}$).

Funcionamento:

- Consúmese da entrada un símbolo ou ε
- Pásase a un novo estado
- Reemplazase o símbolo no alto da pila por unha cadea (podería ser ε).



DEFINICIÓN FORMAL DO AP

$$P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

- Q : Conxunto finito de estados
- Σ : Conxunto finito de símbolos de entrada
- Γ : Alfabeto de pila finito
- δ : Función de transición, $\delta(q, a, X) = (p, \gamma)$
- q_0 : Estado inicial
- Z_0 : Símbolo inicial da pila
- F : Conxunto de estados de aceptación

DESCRICIÓN INSTANTÁNEA DUN AP

(q, w, γ)

- q : estado actual
- w : entrada que falta por ler
- γ : contido da pila (cima á esquerda, fondo á dereita)

LINGUAXES ACEPTADAS POR UN AP

Hai dous tipos (equivalentes) de aceptación:

→ Aceptación por **estado final**:

Sexa $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ un AP,

$L(P) = \{w \mid (q_0, w, Z_0) \vdash^* (q, \varepsilon, \alpha)$ para algún estado *q* de *F* e calquera cadea de pila α.

→ Aceptación por **pila baleira**:

Sexa $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$ un AP,

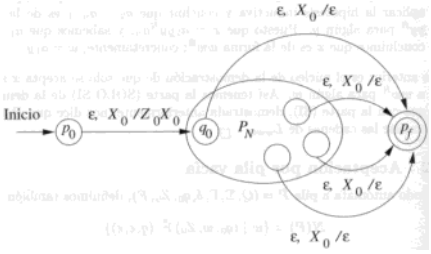
$N(P) = \{w \mid (q_0, w, Z_0) \vdash^* (q, \varepsilon, \varepsilon)$ para calquera estado *q*.

CONVERSIÓN DE BALEIRADO DE PILA A ESTADO FINAL

Teorema: Se $L = N(P_N)$ para algún AP $P_N = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$, existe un AP P_F tal que $L = L(P_F)$.

Proba: $P_F = (Q \cup \{p_0, p_F\}, \Sigma, \Gamma \cup \{X_0\}, \delta_F, p_0, X_0, \{p_F\})$, onde δ_F se define:

1. $\delta_F(p_0, \varepsilon, X_0) = \{(q_0, Z_0, X_0)\}$
2. Para todo estado *q* de *Q*, entrada *a* de Σ ou *a* = ε, e símbolos de pila γ de Γ, $\delta_F(q, a, \gamma)$ contén todos os pares de $\delta_N(q, a, \gamma)$.
3. Ademáis, $\delta_F(p_0, \varepsilon, X_0)$ contén (p_F, ε) para todo estado *q* de *Q*.



Pazo

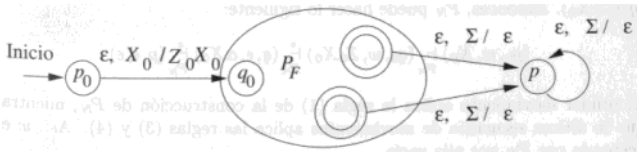
CONVERSIÓN DE BALEIRADO DE PILA A ESTADO FINAL

luns, 13 de novembro de 2023 20:23

Teorema: Sexa L a linguaxe $L(P_F)$ dalgún AP $P_F = (Q, \Sigma, \Gamma, \delta_F, q_0, Z_0, F)$, existe un AP P_N tal que $L = N(P_N)$.

Proba: $P_N = (Q \cup \{p_0, p\}, \Sigma, \Lambda = \Gamma \cup \{X_0\}, \delta_N, p_0, X_0)$, onde δ_N se define:

1. $\delta_N(p_0, \varepsilon, X_0) = \{(q_0, Z_0, X_0)\}$
2. Para todo estado *q* de *Q*, entrada *a* de Σ ou *a* = ε, e símbolos de pila γ de Γ, $\delta_N(q, a, \gamma)$ contén todos os pares de $\delta_F(q, a, \gamma)$.
3. Para todo estado de aceptación *q* en *F* e símbolos de pila γ en Λ, $\delta_N(q, \varepsilon, \gamma)$ contén (p, ε).
4. Para todos os símbolos da pila γ en Λ, $\delta_N(q, \varepsilon, \gamma) = \{(p, \varepsilon)\}$



Necesidade do novo símbolo inicial de pila: Se o APF baleira a súa pila nun estado non final, non debería recoñecer a secuencia. Se non se engadise o novo símbolo inicial de pila, o APN recoñeceríaa.

EQUIVALENCIA ENTRE AP E GIC

O obxectivo é demostrar que as tres seguintes linguaxes son todas da mesma clase:

1) Linguaxes independentes de contexto.
2) Linguaxes aceptadas por estado final por algún AP.
3) Linguaxes aceptadas por baleirado de pila por algún AP.

- A equivalencia de (2) e (3) xa se demostrou.
- Tqd que de (1) se segue (3), aínda que non que de (3) se segue (1):

CONVERSIÓN DE GRAMÁTICAS A AP

Sexa a GIC $G = (V, T, Q, S)$, o AP que acepta L(G) por pila baleira será:

- $P = (\{q\}, T, V \cup T, \delta, q, S)$
- δ defínese por:
  - Para cada variable *A*, $\delta(q, \varepsilon, A) = \{(q, \beta) | A \to \beta \text{ é } unha\ produción\ de\ P\}$
  - Para cada símbolo terminal a, $\delta(q, a, a) = (q, \varepsilon)$

AUTÓMATAS CON PILA DETERMINISTAS

Os APD aceptan un conxunto de linguaxes a medio camiño entre as linguaxes regulares e as GIC.

Os analizadores sintácticos compórtanse xeralmente como APD.

Un AP $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ é determinista se:

- $\delta(q, a, X)$ ten como máximo un elemento para calquera *q* en *Q*, *a* en Σ ou $a = \varepsilon$, e *X* en Γ.
- Se $\delta(q, a, X)$ non está baleiro para algún *a* en Σ, $\delta(q, \varepsilon, X)$ debe estar baleiro.

Pazo

LEMA DE BOMBEO PARA LIC

martes, 14 de novembro de 2023 10:41

Para un LIC, o cumprimento do **lema de bombeo** (LB) é unha condición **necesaria**, pero **non suficiente**.

Teorema: sexa *L* un LIC, existe unha constante *n* tal que se *z* é calquera cadea de *L* de lonxitude $|z| \geq n$. Podemos escribir $z = uvwxy$ coas seguintes condicións:

1) $|vwx| \leq n$
2) $vx \neq \varepsilon$
3) Para todo $k \geq 0$, $uv^k wx^k y$ está en *L*

APLICACIÓN DO LEMA DE BOMBEO

1. Eliximos *L* da que queremos demostrar que non é LIC.
2. O valor de *n* é descoñecido, polo que debemos considerar calquera posible valor.
3. Eliximos *z* (podemos empregar *n* como parámetro).
4. Repetir para todas as descomposicións:
    a. Escoller unha descomposición de *z* en *uvwxy* suxeita ás restricións:
        i. $vx \neq \varepsilon$
        ii. $|vwx| \leq n$
    b. Se $uv^k wx^k y$ pertence a *L* para todo valor de *k*:
        i. Verifícase o LB
        ii. Non se pode afirmar que a linguaxe sexa independente do contexto.
        iii. Non é necesario probar con outras descomposicións (remata o algoritmo).
5. Se 4.b non se cumpriu para ningunha descomposición, non se verifica o LB e, polo tanto, a linguaxe non é unha LIC.

Exemplos de linguaxes non IC:

$L = \{ 0^p 1^p 2^p \mid p \geq 1 \}$

Unha LIC non pode emparellar tres grupos de símbolos de acordo coa súa igualdade ou desigualdade.

$L = \{ 0^i 1^j 2^i 3^j \mid i, j \geq 1 \}$

Unha LIC non pode emparellar dous pares de números iguais de símbolos que se entrelacen.

$L = \{ ss \mid s \in (0+1)^* \}$

Unha LIC non pode emparellar dúas cadeas de lonxitude arbitraria se as cadeas se escollen dun alfabeto de máis dun símbolo.

Pazo

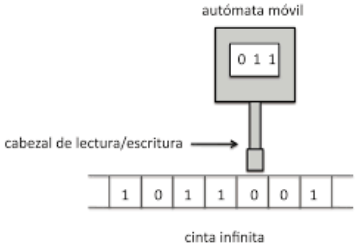
ALAN TURING: "máquinas intelixentes"

martes, 14 de novembro de 2023 11:24

" [...] unha ilimitada capacidade de memoria obtida en forma dunha cinta infinita marcada con cadrados, en cada un dos que se podería imprimir un símbolo. En calquera momento hai un símbolo na máquina; chamado símbolo lido. A máquina pode alterar o símbolo lido e o seu comportamento está en parte determinado por el, pero os símbolos noutros lugares da cinta non afectan ao comportamento da máquina. Non obstante, a cinta pode moverse cara adiante e cara atrás a través da máquina, sendo esta unha das operacións elementais da máquina. Polo tanto, **calquera símbolo na cinta pode ter finalmente unha oportunidade**."

MÁQUINA DE TURING

Unha **Máquina de Turing** (MT) é un autómata que conta cun dispositivo de almacenamento denominado **cinta**. Asociada coa cinta, existe unha cabeza de lectura/escritur



- A entrada está escrita na cinta ao principio.
- A saída escribirase na cinta durante a operación da MT.

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

→ Q: conxunto de estados.
→ Σ: alfabeto de entrada.
→ Γ: alfabeto da cinta.
→ δ: función de transición (determinista).
→ $q_0 \in Q$: estado inicial.
→ $B \in \Gamma$: espazo en branco ($B \notin \Sigma$).
→ $F \subseteq$ Q: conxunto de estados finais.

$\Sigma \subseteq \Gamma - \{B\}$: o alfabeto de entrada é un subconxunto do alfabeto da cinta sen o espazo en branco.

$\delta: Q \times \Gamma \to Q \times \Gamma \times \{I, D\}$

Accións:
- Escribir o novo símbolo
- Cambiar de estado
- Mover a cabeza de lectura/escritura (un só desprazamento -esquerda ou dereita- e despois da operación de lectura/escritura)

Descrición instantánea:
- Estado.
- Contido da cinta (brancos só incluídos se son relevantes).
- Posición da cabeza de lectura(escritura).

$$a_1 \dots a_{k-1}\ q\ a_k \dots a_n$$

A MT finaliza o procesamento cando chega a un **estado de parada**:
a) Non hai transicións definidas para esa combinación de estado e símbolo.
b) Asúmase que os estados finais non teñen transicións definidas: unha MT parará sempre que acade un estado final.

A diferencia dos AF e AP, nunha MT non é necesario ler todo o contido da cinta para aceptar.

Pazo

MÁQUINA DE TURING E LINGUAXES

mércores, 22 de novembro de 2023 19:37

$$L(M) = \{w \in \Sigma^+ : q_0 w| \, -^* x_1 q_f x_2;\ q_f \in F;\ x_1, x_2 \in \Gamma^*\}$$

- Os espazos en branco empréganse para delimitar a cadea de entrada.
  → A cadea baleira non forma parte da linguaxe para poder limitar a rexión na que se busca a entrada.

- Se $w \notin L(M)$:
  a) M para nun estado non final.
  b) M entra nun bucle infinito e non hai parada.

As MT pódense combinar

COMPUTACIÓN DE FUNCIÓNS

Unha función *f* con dominio *D* é **Turing-computable** ou computable sen máis se existe unha MT tal que:

$$q_0 w| \, -^* q_f\ f(w);\ q_f \in F \text{ para todo } w \in D$$

! Todas as funcións matemáticas comúns, non importa o complicadas que sexan, son Turing-computables.

TESE DE CHURCH-TURING

Hipótese: calquera problema de decisión resoluble pode ser transformado nun problema equivalente para unha MT.

Argumentos:
- Calquera problema que se poida resolver nunha computadora tamén se pode resolver nunha MT.
- Non se atopou ningún problema resoluble (por un algoritmo) para o que non se puidese escribir un programa para unha MT.
- Propuxéronse modelos alternativos de computación, pero ningún probou ser máis potente ca o modelo de MT.

Definición de algoritmo:
Un algoritmo para unha función $f: D \to R$ é unha MT, que dada calquera entrada $d \in D$ na súa cinta, finalmente se para coa resposta correcta $f(d) \in R$ na cinta:

$$q_0 d| \, -^* q_f\ f(d);\ q_f \in F \text{ para todo } d \in D$$

- Empregando a tese de Church-Turing, podemos substituír na definición "MT" por "programa en C", "programa en Java",... .

Pazo

OUTROS MODELOS DE MT

mércores, 22 de novembro de 2023 20:18

Todas as variacións son equivalentes á MT estándar.

MT CON OPCIÓN DE NON-MOVEMENTO

$\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{I, D, E\}$

E indica que a cabeza de lectura/escritura permanece estática (é o mesmo que mover o cabezal á esquerda e á dereita ou viceversa).

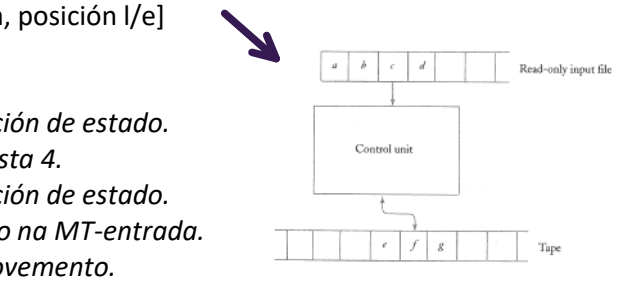
MT CON CINTA SEMIINFINITA

É unha MT cunha cinta limitada por un extremo (se se unen dúas limitadas por lados opostos obtense unha MT estándar).

MT CON CINTA DE ENTRADA

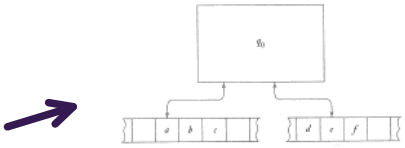
A entrada está escrita nunha cinta de só lectura (para simular a MT estándar só habería que copiar o contido da cinta de entrada na cinta normal, mais requiriría varias pistas e máis movementos).

[Cinta con 4 pistas: entrada, posición da cabeza l, cinta, posición l/e]
Posición de partida: extremo esquerdo da cinta.
Busca da posición da cabeza de lectura na pista 2
Lectura do símbolo correspondente na pista 1 e transición de estado.
Busca da posición da cabeza de lectura/escritura na pista 4.
Lectura do símbolo correspondente na pista 3 e transición de estado.
Modificación das pistas para representar o movemento na MT-entrada.
Volta á posición de partida para simular o seguinte movemento.



MT MULTICINTA

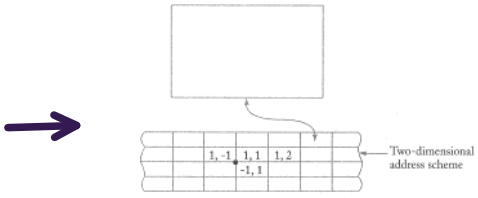
$\delta: Q \times \Gamma^n \rightarrow Q \times \Gamma^n \times \{I, D\}^n$



As pistas pares representan a posición da cabeza nas cintas e as impares o contido (os pasos para a simulación dunha MT estándar son similares aos da variación anterior).

MT MULTIDIMANSIONAL

A cinta é infinita en máis dunha dimensión.
Bidimensional: $\delta: \mathrm{Q} \times \Gamma \rightarrow \mathrm{Q} \times \Gamma \times \{\mathrm{I}, \mathrm{D}, \mathrm{AR}, \mathrm{AB}\}$



(A simulación dunha MT estándar emprega 2 cintas, unha para o contido da cinta bidimensional e outra coas direccións asociadas ao contido da pista 1.)

MT NON DETERMINISTA

$\delta: Q \times \Gamma \rightarrow 2^{Q \times \Gamma \times \{I, D\}}$

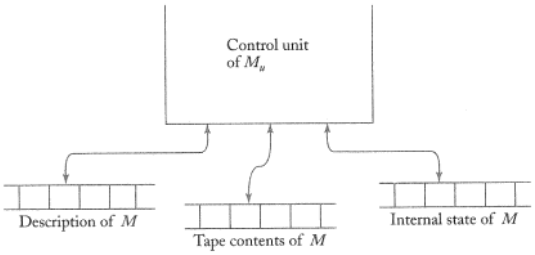
(É unha MT con 2n pistas que se pode replicar a si mesma cando sexa necesario. MTD e MTN son equivalentes)

Pazo

MT UNIVERSAL (MTU)

xoves, 23 de novembro de 2023 13:09

As MTU son reprogramables.
Dada unha descrición de calquera MT M e unha cadea q, unha MTU pode simular a computación de M para w.



Descrición:

- $Q = \{q_1, q_2, \dots, q_n\}$, sendo q_1 o estado inicial e q_n o final.
- $\Gamma = \{a_1, a_2, \dots, a_m\}$, sendo a_1 un espazo en branco.

Funcionamento:

1) Examínase o contido das cintas 2 e 3: configuración de M.
2) Consúltase a cinta 1 para determinar a transición a realizar.
3) Modifícanse as cintas 2 e 3 como resultado do movemento realizado.

> Unha MT, dado calquera programa, pode realizar as computacións especificadas e é, polo tanto, un modelo adecuado dunha computadora de propósito xeral. Calquera MT pode ser codificada con 0s e 1s.

GRAMÁTICAS E LINGUAXES SENSIBLES AO CONTEXTO

- Gramáticas Sen Restricións (GSR), $G = (NT, T, S, P)$
  - Producións: $x \to y$, $x \in (NT/T)^+$, $y \in (NT/T)^*$
  - As GSR xeran as LRE (Linguaxes Recursivamente Enumerables).

- Gramáticas Sensibles ao Contexto (GSC), $G = (NT, T, S, P)$
  - Producións: $\alpha \to \beta\,;\ |\alpha| \leq |\beta|$, $\alpha = z_1 x\, z_2$, $\beta = z_1 y\, z_2$, $z_1, z_2 \in T^*$, $x \in NT$, $y \in (NT/T)^+$
  - Unha linguaxe *L* é sensible ao contexto (LSC) se existe unha GSC tal que $L = L(G)$ ou $L = L(G) \cup \{\lambda\}$.

$LSC \subset \text{LREC}$

As LSC (que non conteñan λ) son recoñecidas polos **Autómatas Linealmente Acotados** (ALA).

AUTÓMATAS LINEALMENTE ACOTADOS (ALA)

Restricións no uso da cinta:

- □ Funcionamento tipo pila: autómata con pila.
- □ Uso dunha parte finita da cinta: autómata de estados finitos.
- □ Uso da parte da cinta ocupada pola cadea de entrada: ALA.
  - Canta maior lonxitude teña a cadea de entrada, maior é o espazo a empregar.

> Os ALA son máis potentes cós AP e menos cás MT.

Definición: Un ALA é unha **MT determinista** $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ coa restrición de que Σ debe conter dous símbolos especiais:

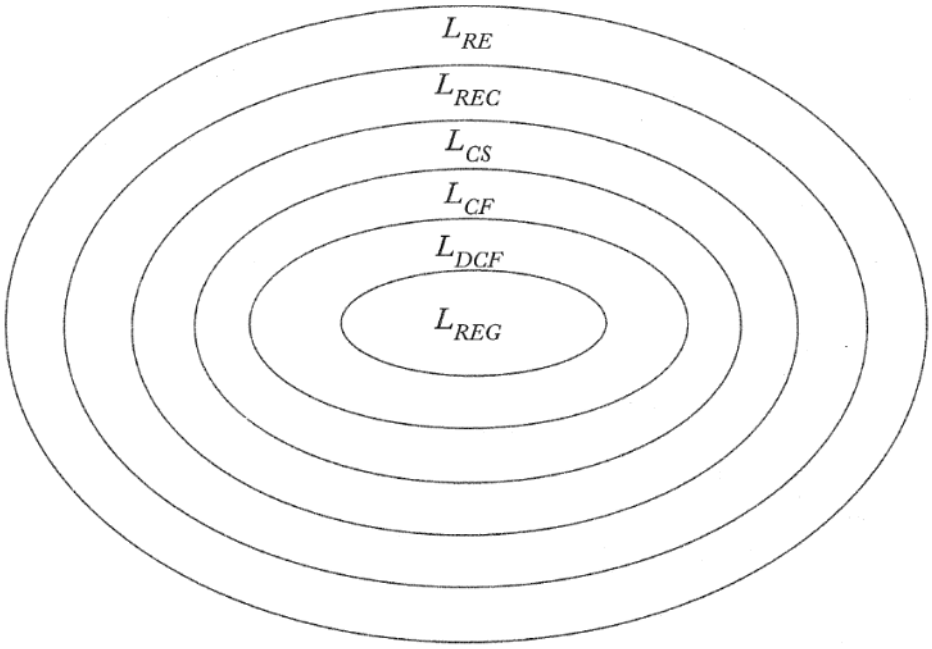
- [: marcador esquerdo, con transicións do tipo $\delta(q_i, [\,) = (q_j, [, D)$
-]: marcador dereito, con transicións do tipo $\delta(q_i,]\,) = (q_j,], I)$

$$L(M) = \{w \in \Sigma^+ :\ q_0[w] \mid -^* \left[x_1 q_f x_2\right];\ q_f \in F;\ x_1, x_2 \in \Gamma^*\}$$

Pazo

LINGUAXES FORMAIS: Xerarquía de Chomsky

venres, 24 de novembro de 2023 12:02



L_{REG}: Linguaxe Regular
L_{DCF}: Linguaxe Determinista Independente de Contexto
L_{CF}: Linguaxe Independente de Contexto
L_{CS}: Linguaxe Dependente de Contexto
L_{REC}: Linguaxe Recursiva
L_{RE}: Linguaxe Recursivamente Enumerable

Pazo

LINGUAXES RECURSIVAS E RECURSIVAMENTE ENUMERABLES

venres, 24 de novembro de 2023 12:24

Unha linguaxe *L* é **recursivamente enumerable** (LRE) se existe unha MT que acepta calquera entrada da linguaxe e se para:

→ Se $w \in L$, $q_0 w \mid -^* \ x_1 q_f x_2;\ q_f \in F$
Non obstante:

→ Se $w \notin L$, a MT parará nun estado non final -rexeitándoa- ou entrara nun bucle infinito.

Unha linguaxe *L* sobre un alfabeto Σ é **recursiva** (LRC) se exite unha MT que acepta *L* e párase con calquera cadea $w \in \Sigma^*$ -párase acéptea ou non-.

! Hai linguaxes que non son LRE. A demostración é complexa, posto que todas as linguaxes descritas de forma algorítmica son aceptados por MT (e son, polo tanto, LRE).

COMPUTABILIDADE E DECIDIBILIDADE

- Unha función *f* é **computable** nun dominio se existe unha MT que computa o valor de *f* para todos os argumentos do dominio.

- Se o resultado da computación dun problema é si/non, fálase de decidibilidade/indecibilidade. Problemas **decidibles**: existe unha MT que dá a resposta correcta (si/non) para cada argumento do dominio.

PROBLEMA DA PARADA EN MT

Sexa w_M unha cadea que describe a MT M, e sexa *w* unha cadea sobre o alfabeto de M; unha solución ao problema de parada sería unha MT H na que, para calesquera w_M e *w*, se executa a computación:

- $q_0 w_M w \mid -^* \ x_1 q_y x_2$, se M aplicada para *w* se para.
- $q_0 w_M w \mid -^* \ x_1 q_n x_2$, se M aplicada para *w* non se para.
- q_y e q_n son estados finais de H.

- Non existe ningunha MT H que se comporte como se require para o problema da parada?
  ⇒ **problema indecidible**
- Se o problema da parada fose decidible, entón todos os LRE serían LRC.

COMPLEXIDADE COMPUTACIONAL

Para determinar a **complexidade**: usaremos MT, o tamaño do problema será *n* e interésanos saber cánto aumenta o tempo a medida que aumenta *n*. Non nos interesa o tempo exacto, senón a orde de magnitude: O(...).

Se unha computación ten complexidade temporal T(n), significa que pode ser resolta en non máis de T(n) movementos dunha MT para un tamaño de problema n.

Pazo

MT E COMPLEXIDADE

venres, 24 de novembro de 2023 13:15

Dende o punto de vista da decidibilidade, todas as MT son equivalentes. Dende o punto de vista da complexidade, non.

Problema da satisfacibilidade (SAT):

- Expresións en forma normal conxuntiva: $e = t_i \wedge t_j \wedge \ldots \wedge t_k$
  - $t_i = s_m \vee s_p \vee \ldots \vee s_q$: s_l son variables ou as súas negacións.
- Dada unha expresión *e* en forma normal conxuntiva, hai algunha asignación de valores ás súas variables que faga *e* verdadeira?
- MT estándar: $O(2^n)$
- MT non determinista: $O(n)$

Unha MT acepta unha linguaxe *L* en tempo T(n) se calquera cadea de *w* de *L* de tamaño *n* ou menor é aceptada en O(T(n)) movementos.

✧ Se a MT é non determinista, para calquera cadea *w* de *L* existe polo menos unha secuencia de movementos de lonxitude O(T(|w|)) que leva á aceptación.

➢ Unha linguaxe *L* pertence á clase **TD(T(n))** se hai unha MT **multicinta determinista** que acepta *L* en tempo T(n).

➢ Unha linguaxe *L* pertence á clase **TND(T(n))** se hai unha MT **multicinta non determinista** que acepta *L* en tempo T(n).

TD(T(n)) ⊆ TND(T(n))

COMPLEXIDADES P E NP

- $P = \cup_{i \geq 1} TD(n^i)$ Linguaxes aceptadas por unha MT determinista en tempo polinómico.

- $NP = \cup_{i \geq 1} TND(n^i)$ Linguaxes aceptadas por unha MT non determinista en tempo polinómico.

$P \subseteq \text{NP}$? $P = NP$

Problemas intratables: problemas computables pero que requerirían, para entradas grandes, tal cantidade de recursos (tempo e memoria) que a súa implementación non é viable. Os problemas da clase P son tratables e, o resto, intratables.

Unha linguaxe L_1 é **reducible** en tempo polinómico a outra linguaxe L_2 se existe unha MT determinista tal que calquera cadea $w_1 \in {\Sigma_1}^+$ pode ser transformada en tempo polinómico noutra cadea $w_2 \in {\Sigma_2}^+$ de tal forma que $w_1 \in L_1 \Leftrightarrow w_2 \in L_2$.

→ Se L_1 é reducible en tempo polinómico a L_2 e $L_2 \in P$, entón $L_1 \in P$.

→ De igual forma, se $L_2 \in NP$, entón $L_1 \in NP$.

Unha linguaxe *L* é **NP-completa** se $L \in NP$ e todo $L' \in NP$ é reducible en tempo polinómico a *L*.

Pazo