MÁQUINA DE TURING UNIVERSAL

INFORMÁTICA TEÓRICA
Curso 2005-2006

MT LOCALIZADORA DE INFORMACIÓN.

La información aparece en la cinta, en registros de igual longitud, separados por marcadores, que constan de dos partes: una primera parte con el código de referencia que identifica al registro y una segunda parte con la información del registro.

La búsqueda se hace por comparación con la referencia que aparece al principio de la información, en las posiciones más a la izquierda de la cinta.

⫲0010 ǂ0000 ..... ǂ0001 ..... ǂ0010 ..... ǂ

| Ref || Ref | Inf || Ref | Inf || Ref | Inf |

Reg   Reg

Los símbolos son:

$\Sigma_E = \{0, 1, ⫲, ǂ\} = \{ \text{binario} \} \cup \{⫲, ǂ\}$

$\Sigma_A = \{A, B, \#\}$

siendo:

⫲ principio de información
ǂ separador de registros
A memoriza 0
B memoriza 1
blanco
* sin significación

Parada de la MT·localizadora de inf.
no tiene definida transición para q_9,
q_9 es un estado de la MT transcriptora

	0	1	A	B	⫲	ǂ	#	*
q_0	q_0 0 I	q_0 1 I	q_0 A I	q_0 B I	q_1 ⫲ D	q_0 ǂ I	-	-
q_1	q_2 A D	q_3 B D	q_1 A D	q_1 B D	-	q_9 ǂ D	-	-
q_2	q_2 0 D	q_2 1 D	-	-	-	q_4 ǂ D	-	-
q_3	q_3 0 D	q_3 1 D	-	-	-	q_5 ǂ D	-	-
q_4	q_0 A I	q_6 B D	q_4 A D	q_4 B D	-	q_4 ǂ D	-	-
q_5	q_6 A D	q_0 B I	q_5 A D	q_5 B D	-	q_5 ǂ D	-	-
q_6	q_6 A D	q_6 B D	-	-	-	q_7 ǂ I	-	-
q_7	q_7 0 I	q_7 1 I	q_7 A I	q_7 B I	q_8 ⫲ D	q_7 ǂ I	-	-
q_8	q_8 0 D	q_8 1 D	q_8 0 D	q_8 1 D	-	q_0 ǂ I	-	-

Parada de la MT universal
No existe la Ref buscada porque
es de una transición sin definir

[illegible]

[illegible]

[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

La memorización asociada a cada estado es:

q_0 Retrocede hasta † para buscar la referencia que comparar

q_1 Busca el primer símbolo de la ref que está sin comparar o anular y acaba si se comparó toda la Ref. parando porque transita a un estado sin definir en esta MT, q_9 que es el estado inicial de la MT transcriptora de información

q_2 | Memorizan el símbolo de la Ref que hay que comparar

q_3 | y buscan el final de la Ref

q_4 | Avanzan por las referencias no buscadas y anuladas y buscan el primer símbolo no anulado

q_5 | Si hay coincidencia, se marca el símbolo encontrado y se pasa a q_0 para continuar

q_6 Si no hubo coincidencia se continua anulando el resto del Reg por no ser el buscado

q_7 Retrocede hasta‡ dejando las referencias no válidas marcadas

q_8 Se restaura la Ref que se busca para compararla con el siguiente registro sin explorar

MT TRANSCRIPTORA DE INFORMACIÓN (o copiadora).

Copia información de un lugar a otro de la cinta.

	0	1	A	B	†	‡	#	*
q_9	q_{10} A I	q_{11} B I	q_9 A D	q_9 B D	-	q_9 ‡D	-	-
q_{10}	q_{12} 0 D	q_{12} 1 D	q_{10} A I	q_{10} B I	q_{12} † D	q_{10} ‡I	-	-
q_{11}	q_{13} 0 D	q_{13} 1 D	q_{11} A I	q_{11} B I	q_{13} † D	q_{11} ‡I	-	-
q_{12}	-	-	q_9 0 D	q_9 0 D	-	q_{14} ‡I	-	-
q_{13}	-	-	q_9 1 D	q_9 1 D	-	q_{15} ‡I	-	-

Parada de la MT transcriptora de información no tiene definida transición para q_{14} ni para q_{15} son estados de la MT simuladora

La memorización asociada a cada estado es:

q_9 Avanza por todos los reg no válidos hasta que encuentra el símbolo de la inf buscada

q_{10} | Retroceden hasta encontrar la primera posición disponible de escritura,

q_{11} | ignora el contenido de dicha posición

q_{12} | Escriben el símbolo memorizado y pasa a q_9 en busca de más información que transcribir,

q_{13} | si se detecta el final de Reg (‡) ya se ha transcrito toda la inf. y pasa a la MT simuladora

MT SIMULADORA.

El símbolo especial sin significación es:
* posición de la cabeza lectora de la MT simulada

	0	1	A	B	ǂ	‡	#	*
q_{14}	q_{16} 0 I	q_{17} 1 I	-	-	-	-	-	-
q_{15}	q_{18} 0 I	q_{19} 1 I	-	-	-	-	-	-
q_{16}	q_{16} 0 I	q_{16} 1 I	-	-	q_{16} ǂ I	-	-	q_{20} 0 D
q_{17}	q_{17} 0 I	q_{17} 1 I	-	-	q_{17} ǂ I	-	-	q_{20} 1 D
q_{18}	q_{18} 0 I	q_{18} 1 I	-	-	q_{18} ǂ I	-	-	q_{20} 0 I
q_{19}	q_{19} 0 I	q_{19} 1 I	-	-	q_{19} ǂ I	-	-	q_{20} 1 I
q_{20}	q_{21} * D	q_{22} * D	-	-	-	-	q_{21} * D	-
q_{21}	q_{21} 0 D	q_{21} 1 D	-	-	q_{21} ǂ D	q_{23} ‡ I	-	-
q_{22}	q_{22} 0 D	q_{22} 1 D	-	-	q_{22} ǂ D	q_{24} ‡ I	-	-
q_{23}	q_{25} 0 D	q_{25} 0 D	-	-	-	-	-	-
q_{24}	q_{25} 1 D	q_{25} 1 D	-	-	-	-	-	-
q_{25}	q_{26} 0 I	q_{26} 1 I	q_{25} A D	q_{25} B D	-	q_{25} ‡ D	q_{26} # I	-
q_{26}	q_0 0 D	q_0 1 D	q_{26} 0 1	q_{26} 1 I	-	q_{26} ‡ I	-	-

Parada de la MT simuladora, q_0 es un estado de la MT localizadora de información

La memorización asociada a cada estado es:

q_{14}, q_{15} | Memorizan el desplazamiento

q_{16}, q_{17}, q_{18}, q_{19} | Memorizan el símbolo 0 ó 1 para cada desplazamiento:
q_{D0}, q_{D1}, q_{I0}, q_{I1}

q_{20} Lee el siguiente caracter despues del desplazamiento y escribe un *
(si lee # expande la zona de información de la cinta de la MT simulada)

q_{21}, q_{22} | Memoriza el símbolo leido y se desplazan a la izquierda para escribirlo

q_{23}, q_{24} | Estados que escriben

q_{25} Avanza por todas las posiciones anuladas

q_{26} Restaura a numérico las posiciones anuladas quedando la cabeza lectora en la posición inicia y

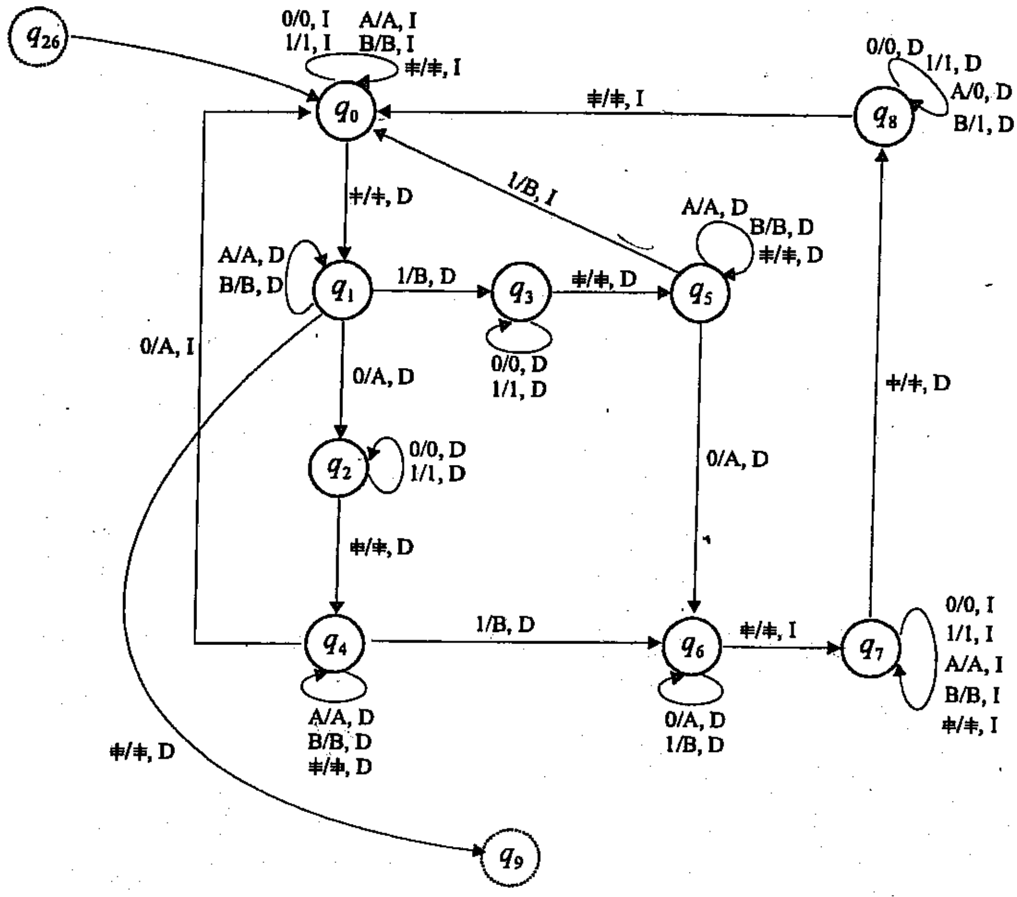


Figura 1: Módulo localizador de la MTU. Estados: del q_0 al q_8.

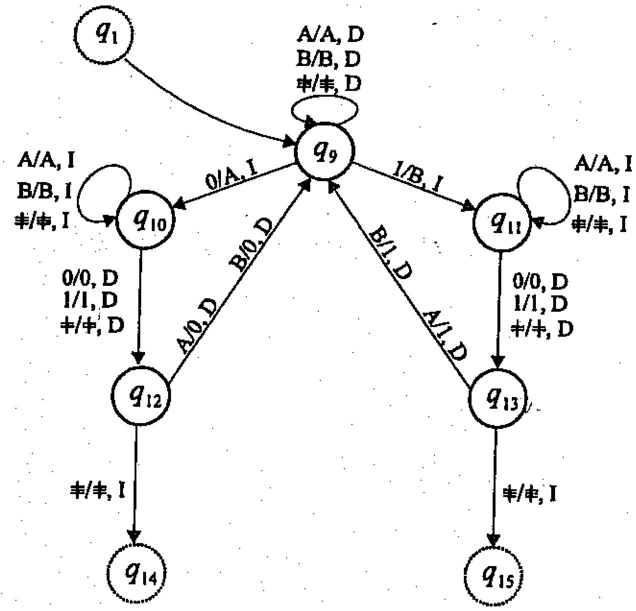


Figura 2: Módulo transcriptor de la MTU. Estados: del q_9 al q_{12} o q_{13}.

[illegible]

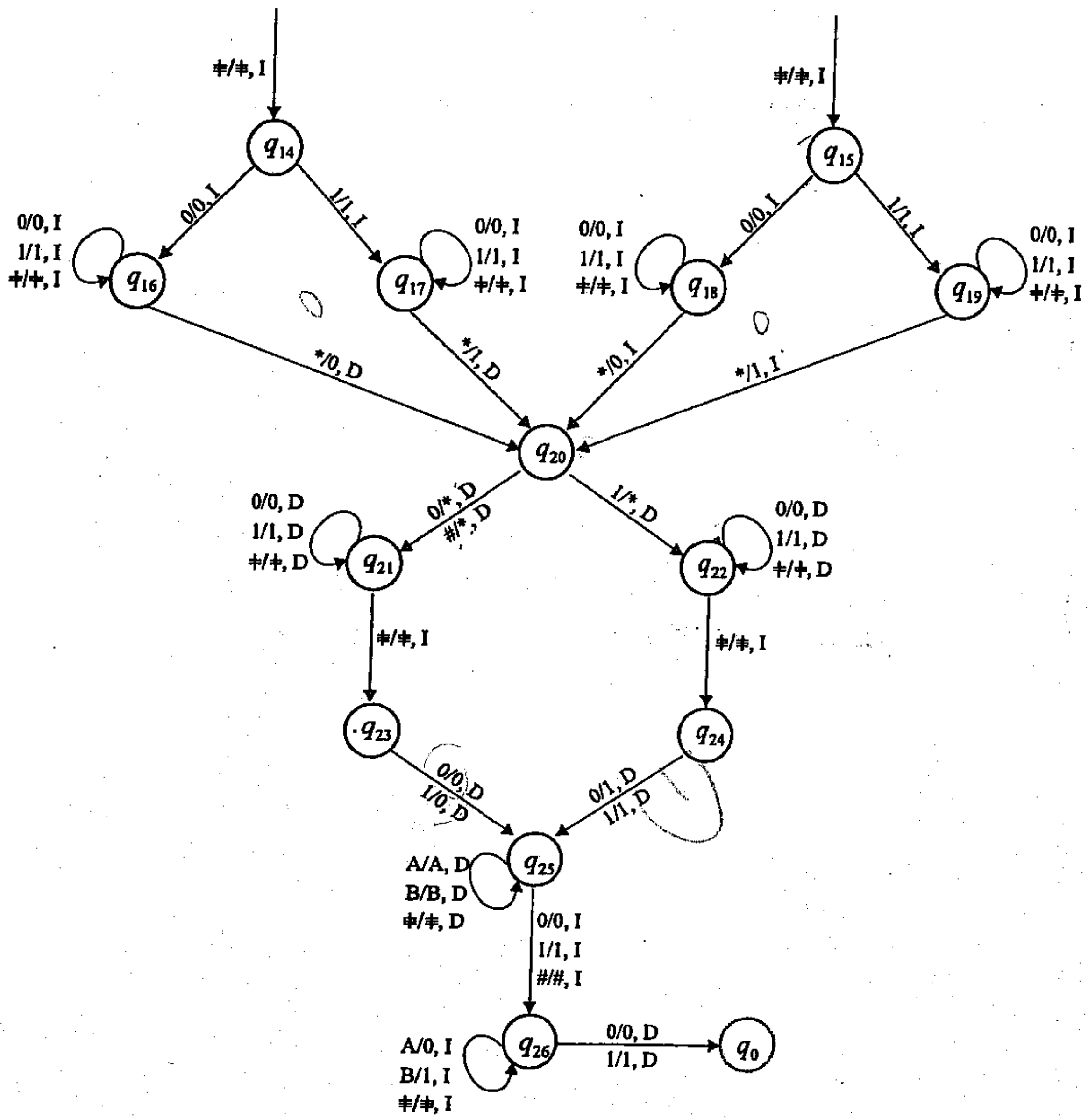
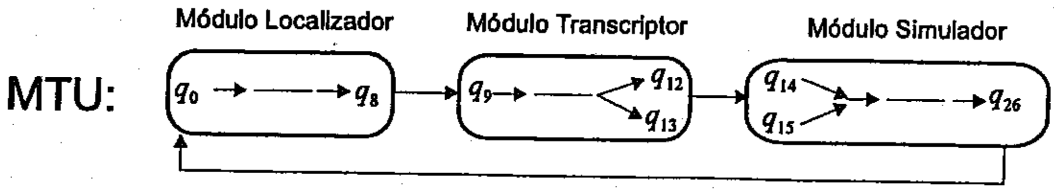


Figura 3: **Módulo simulador** de la MTU. Estados: del q_14 (o q_{15}) al q_{26}.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1	#	•	1	0	1	1	1	0	*	0	0	1	q_0	*	0	0	1	0	0	1	0	*	0	0
2	#	•	1	0	1	1	1	0	*	0	0	q_0	1	*	0	0	1	0	0	1	0	*	0	0
3	#	•	1	0	1	1	1	0	*	0	q_0	0	1	*	0	0	1	0	0	1	0	*		
4	#	•	1	0	1	1	1	0	*	q_0	0	0	1	*	0	0	1	0	0	1	0	*		
5	#	•	1	0	1	1	1	0	q_0	*	0	0	1	*	0	0	1	0	0	1	0	*		
6	#	•	1	0	1	1	1	0	*	q_1	0	0	1	*	0	0	1	0	0	1	0	*		
7	#	•	1	0	1	1	1	0	*	A	q_2	0	1	*	0	0	1	0	0	1	0	*		
8	#	•	1	0	1	1	1	0	*	A	0	q_2	1	*	0	0	1	0	0	1	0	*		
9	#	•	1	0	1	1	1	0	*	A	0	1	q_2	*	0	0	1	0	0	1	0	*		
10	#	•	1	0	1	1	1	0	*	A	0	1	*	q_4	0	0	1	0	0	1	0	*		
11	#	•	1	0	1	1	1	0	*	A	0	1	q_0	*	A	0	1	0	0	1	0	*		
12	#	•	1	0	1	1	1	0	*	A	0	q_0	1	*	A	0	1	0	0	1	0	*		
13	#	•	1	0	1	1	1	0	*	A	q_0	0	1	*	A	0	1	0	0	1	0	*		
14	#	•	1	0	1	1	1	0	*	q_0	A	0	1	*	A	0	1	0	0	1	0	*		
15	#	•	1	0	1	1	1	0	q_0	*	A	0	1	*	A	0	1	0	0	1	0	*		
16	#	•	1	0	1	1	1	0	*	q_1	A	0	1	*	A	0	1	0	0	1	0	*		
17	#	•	1	0	1	1	1	0	*	A	q_1	0	1	*	A	0	1	0	0	1	0	*		
18	#	•	1	0	1	1	1	0	*	A	A	q_2	1	*	A	0	1	0	0	1	0	*		
19	#	•	1	0	1	1	1	0	*	A	A	1	q_2	*	A	0	1	0	0	1	0	*		
20	#	•	1	0	1	1	1	0	*	A	A	1	*	q_4	A	0	1	0	0	1	0	*		
21	#	•	1	0	1	1	1	0	*	A	A	1	*	A	q_4	0	1	0	0	1	0	*		
22	#	•	1	0	1	1	1	0	*	A	A	1	*	q_0	A	A	1	0	0	1	0	*		
23	#	•	1	0	1	1	1	0	*	A	A	1	q_0	*	A	A	1	0	0	1	0	*		
24	#	•	1	0	1	1	1	0	*	A	A	q_0	1	*	A	A	1	0	0	1	0	*		
25	#	•	1	0	1	1	1	0	*	A	q_0	A	1	*	A	A	1	0	0	1	0	*		
26	#	•	1	0	1	1	1	0	*	q_0	A	A	1	*	A	A	1	0	0	1	0	*		
27	#	•	1	0	1	1	1	0	q_0	*	A	A	1	*	A	A	1	0	0	1	0	*		
28	#	•	1	0	1	1	1	0	*	q_1	A	A	1	*	A	A	1	0	0	1	0	*		
29	#	•	1	0	1	1	1	0	*	A	q_1	A	1	*	A	A	1	0	0	1	0	*		
30	#	•	1	0	1	1	1	0	*	A	A	q_1	1	*	A	A	1	0	0	1	0	*		
31	#	•	1	0	1	1	1	0	*	A	A	B	q_3	*	A	A	1	0	0	1	0	*		
32	#	•	1	0	1	1	1	0	*	A	A	B	*	q_5	A	A	1	0	0	1	0	*		
33	#	•	1	0	1	1	1	0	*	A	A	B	*	A	q_5	A	1	0	0	1	0	*		
34	#	•	1	0	1	1	1	0	*	A	A	B	*	A	A	q_5	1	0	0	1	0	*		
35	#	•	1	0	1	1	1	0	*	A	A	B	*	A	q_0	A	B	0	0	1	0	*		
36	#	•	1	0	1	1	1	0	*	A	A	B	*	q_0	A	A	B	0	0	1	0	*		
37	#	•	1	0	1	1	1	0	*	A	A	B	q_0	*	A	A	B	0	0	1	0	*		
38	#	•	1	0	1	1	1	0	*	A	A	q_0	B	*	A	A	B	0	0	1	0	*		
39	#	•	1	0	1	1	1	0	*	A	q_0	A	B	*	A	A	B	0	0	1	0	*		
40	#	•	1	0	1	1	1	0	*	q_0	A	A	B	*	A	A	B	0	0	1	0	*		
41	#	•	1	0	1	1	1	0	q_0	*	A	A	B	*	A	A	B	0	0	1	0	*		
42	#	•	1	0	1	1	1	0	*	q_1	A	A	B	*	A	A	B	0	0	1	0	*		
43	#	•	1	0	1	1	1	0	*	A	q_1	A	B	*	A	A	B	0	0	1	0	*		
44	#	•	1	0	1	1	1	0	*	A	A	q_1	B	*	A	A	B	0	0	1	0	*		
45	#	•	1	0	1	1	1	0	*	A	A	B	q_1	*	A	A	B	0	0	1	0	*		
46	#	•	1	0	1	1	1	0	*	A	A	B	*	q_9	A	A	B	0	0	1	0	*		
47	#	•	1	0	1	1	1	0	*	A	A	B	*	A	q_9	A	B	0	0	1	0	*		
48	#	•	1	0	1	1	1	0	*	A	A	B	*	A	A	q_9	B	0	0	1	0	*		
49	#	•	1	0	1	1	1	0	*	A	A	B	*	A	A	B	q_9	0	0	1	0	*		
50	#	•	1	0	1	1	1	0	*	A	A	B	*	A	A	q_{10}	B	A	0	1	0	*		
51	#	•	1	0	1	1	1	0	*	A	A	B	*	A	q_{10}	A	B	A	0	1	0	*		
52	#	•	1	0	1	1	1	0	*	A	A	B	*	q_{10}	A	A	B	A	0	1	0	*	0	0	0

26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53
0	1	1	0	*	0	1	1	0	1	1	0	*	0	1	0	1	0	0	1	*	1	0	1	1	0	0	0
0	1	1	0	*	0	1	1	0	1	1	0	*	0	1	0	1	0	0	1	*	1	0	1	1	0	0	0

1 1 0 * 0 1 1 0 1 1 0 * 0 1 0 1 0 0 1 * 1 0 1 1 0 0 0

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	
52	#	.	1	0	1	1	1	0	*	A	A	B	*	q_{10}	A	A	B	A	0	1	0	*	0	0	
53	#	.	1	0	1	1	1	0	*	A	A	B	q_{10}	*	A	A	B	A	0	1	0	*	0	0	
54	#	.	1	0	1	1	1	0	*	A	A	q_{10}	B	*	A	A	B	A	0	1	0	*	0	0	
55	#	.	1	0	1	1	1	0	*	A	q_{10}	A	B	*	A	A	B	A	0	1	0	*	0	0	
56	#	.	1	0	1	1	1	0	*	q_{10}	A	A	B	*	A	A	B	A	0	1	0	*	0	0	
57	#	.	1	0	1	1	1	0	q_{10}	*	A	A	B	*	A	A	B	A	0	1	0	*	0	0	
58	#	.	1	0	1	1	1	0	*	q_{12}	A	A	B	*	A	A	B	A	0	1	0	*	0	0	
59	#	.	1	0	1	1	1	0	*	0	q_9	A	B	*	A	A	B	A	0	1	0	*	0	0	
60	#	.	1	0	1	1	1	0	*	0	A	q_9	B	*	A	A	B	A	0	1	0	*	0	0	
61	#	.	1	0	1	1	1	0	*	0	A	B	q_9	*	A	A	B	A	0	1	0	*	0	0	
62	#	.	1	0	1	1	1	0	*	0	A	B	*	q_9	A	A	B	A	0	1	0	*	0	0	
63	#	.	1	0	1	1	1	0	*	0	A	B	*	A	q_9	A	B	A	0	1	0	*	0	0	
64	#	.	1	0	1	1	1	0	*	0	A	B	*	A	A	q_9	B	A	0	1	0	*	0	0	
65	#	.	1	0	1	1	1	0	*	0	A	B	*	A	A	B	q_9	A	0	1	0	*	0	0	
66	#	.	1	0	1	1	1	0	*	0	A	B	*	A	A	B	A	q_9	0	1	0	*	0	0	
67	#	.	1	0	1	1	1	0	*	0	A	B	*	A	A	B	q_{10}	A	A	1	0	*	0	0	
68	#	.	1	0	1	1	1	0	*	0	A	B	*	A	A	q_{10}	B	A	A	1	0	*	0	0	
69	#	.	1	0	1	1	1	0	*	0	A	B	*	A	q_{10}	A	B	A	A	1	0	*	0	0	
70	#	.	1	0	1	1	1	0	*	0	A	B	*	q_{10}	A	A	B	A	A	1	0	*	0	0	
71	#	.	1	0	1	1	1	0	*	0	A	B	q_{10}	*	A	A	B	A	A	1	0	*	0	0	
72	#	.	1	0	1	1	1	0	*	0	A	q_{10}	B	*	A	A	B	A	A	1	0	*	0	0	
73	#	.	1	0	1	1	1	0	*	0	q_{10}	A	B	*	A	A	B	A	A	1	0	*	0	0	
74	#	.	1	0	1	1	1	0	*	q_{10}	0	A	B	*	A	A	B	A	A	1	0	*	0	0	
75	#	.	1	0	1	1	1	0	*	0	q_{12}	A	B	*	A	A	B	A	A	1	0	*	0	0	
76	#	.	1	0	1	1	1	0	*	0	0	q_9	B	*	A	A	B	A	A	1	0	*	0	0	
77	#	.	1	0	1	1	1	0	*	0	0	B	q_9	*	A	A	B	A	A	1	0	*	0	0	
78	#	.	1	0	1	1	1	0	*	0	0	B	*	q_9	A	A	B	A	A	1	0	*	0	0	
79	#	.	1	0	1	1	1	0	*	0	0	B	*	A	q_9	A	B	A	A	1	0	*	0	0	
80	#	.	1	0	1	1	1	0	*	0	0	B	*	A	A	q_9	B	A	A	1	0	*	0	0	
81	#	.	1	0	1	1	1	0	*	0	0	B	*	A	A	B	q_9	A	A	1	0	*	0	0	
82	#	.	1	0	1	1	1	0	*	0	0	B	*	A	A	B	A	q_9	A	1	0	*	0	0	
83	#	.	1	0	1	1	1	0	*	0	0	B	*	A	A	B	A	A	q_9	1	0	*	0	0	
84	#	.	1	0	1	1	1	0	*	0	0	B	*	A	A	B	A	q_{11}	A	B	0	*	0	0	
85	#	.	1	0	1	1	1	0	*	0	0	B	*	A	A	B	q_{11}	A	A	B	0	*	0	0	
86	#	.	1	0	1	1	1	0	*	0	0	B	*	A	A	q_{11}	B	A	A	B	0	*	0	0	0
87	#	.	1	0	1	1	1	0	*	0	0	B	*	A	q_{11}	A	B	A	A	B	0	*	0	0	0
88	#	.	1	0	1	1	1	0	*	0	0	B	*	q_{11}	A	A	B	A	A	B	0	*	0	0	0
89	#	.	1	0	1	1	1	0	*	0	0	B	q_{11}	*	A	A	B	A	A	B	0	*	0	0	0
90	#	.	1	0	1	1	1	0	*	0	0	q_{11}	B	*	A	A	B	A	A	B	0	*	0	0	0
91	#	.	1	0	1	1	1	0	*	0	q_{11}	0	B	*	A	A	B	A	A	B	0	*	0	0	0
92	#	.	1	0	1	1	1	0	*	0	0	q_{13}	B	*	A	A	B	A	A	B	0	*	0	0	0
93	#	.	1	0	1	1	1	0	*	0	0	1	q_9	*	A	A	B	A	A	B	0	*	0	0	0
94	#	.	1	0	1	1	1	0	*	0	0	1	*	q_9	A	A	B	A	A	B	0	*	0	0	0
95	#	.	1	0	1	1	1	0	*	0	0	1	*	A	q_9	A	B	A	A	B	0	*	0	0	0
96	#	.	1	0	1	1	1	0	*	0	0	1	*	A	A	q_9	B	A	A	B	0	*	0	0	0
97	#	.	1	0	1	1	1	0	*	0	0	1	*	A	A	B	q_9	A	A	B	0	*	0	0	0
98	#	.	1	0	1	1	1	0	*	0	0	1	*	A	A	B	A	q_9	A	B	0	*	0	0	0
99	#	.	1	0	1	1	1	0	*	0	0	1	*	A	A	B	A	A	q_9	B	0	*	0	0	0
100	#	.	1	0	1	1	1	0	*	0	0	1	*	A	A	B	A	A	B	q_9	0	*	0	0	0
101	#	.	1	0	1	1	1	0	*	0	0	1	*	A	A	B	A	A	q_{10}	B	A	*	0	0	0
102	#	.	1	0	1	1	1	0	*	0	0	1	*	A	A	B	A	q_{10}	A	B	A	*	0	0	0
103	#	.	1	0	1	1	1	0	*	0	0	1	*	A	A	B	q_{10}	A	A	B	A	*	0	0	0

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	
103	#	*	1	0	1	1	1	0	*	0	0	1	*	A	A	B	q_{10}	A	A	B	A	*	0	0	
104	#	*	1	0	1	1	1	0	*	0	0	1	*	A	A	q_{10}	B	A	A	B	A	*	0	0	
105	#	*	1	0	1	1	1	0	*	0	0	1	*	A	q_{10}	A	B	A	A	B	A	*	0	0	
106	#	*	1	0	1	1	1	0	*	0	0	1	*	q_{10}	A	A	B	A	A	B	A	*	0	0	
107	#	*	1	0	1	1	1	0	*	0	0	1	q_{10}	*	A	A	B	A	A	B	A	*	0	0	
108	#	*	1	0	1	1	1	0	*	0	0	q_{10}	1	*	A	A	B	A	A	B	A	*	0	0	
109	#	*	1	0	1	1	1	0	*	0	0	1	q_{12}	*	A	A	B	A	A	B	A	*	0	0	
110	#	*	1	0	1	1	1	0	*	0	0	q_{14}	1	*	A	A	B	A	A	B	A	*	0	0	
111	#	*	1	0	1	1	1	0	*	0	q_{17}	0	1	*	A	A	B	A	A	B	A	*	0	0	
112	#	*	1	0	1	1	1	0	*	q_{17}	0	0	1	*	A	A	B	A	A	B	A	*	0	0	
113	#	*	1	0	1	1	1	0	q_{17}	*	0	0	1	*	A	A	B	A	A	B	A	*	0	0	
114	#	*	1	0	1	1	1	q_{17}	0	*	0	0	1	*	A	A	B	A	A	B	A	*	0	0	
115	#	*	1	0	1	1	q_{17}	1	0	*	0	0	1	*	A	A	B	A	A	B	A	*	0	0	
116	#	*	1	0	1	q_{17}	1	1	0	*	0	0	1	*	A	A	B	A	A	B	A	*	0	0	
117	#	*	1	0	q_{17}	1	1	1	0	*	0	0	1	*	A	A	B	A	A	B	A	*	0	0	
118	#	*	1	q_{17}	0	1	1	1	0	*	0	0	1	*	A	A	B	A	A	B	A	*	0	0	
119	#	*	q_{17}	1	0	1	1	1	0	*	0	0	1	*	A	A	B	A	A	B	A	*	0	0	
120	#	q_{17}	*	1	0	1	1	1	0	*	0	0	1	*	A	A	B	A	A	B	A	*	0	0	
121	#	1	q_{20}	1	0	1	1	1	0	*	0	0	1	*	A	A	B	A	A	B	A	*	0	0	
122	#	1	*	q_{22}	0	1	1	1	0	*	0	0	1	*	A	A	B	A	A	B	A	*	0	0	
123	#	1	*	0	q_{22}	1	1	1	0	*	0	0	1	*	A	A	B	A	A	B	A	*	0	0	
124	#	1	*	0	1	q_{22}	1	1	0	*	0	0	1	*	A	A	B	A	A	B	A	*	0	0	
125	#	1	*	0	1	1	q_{22}	1	0	*	0	0	1	*	A	A	B	A	A	B	A	*	0	0	
126	#	1	*	0	1	1	1	q_{22}	0	*	0	0	1	*	A	A	B	A	A	B	A	*	0	0	
127	#	1	*	0	1	1	1	0	q_{22}	*	0	0	1	*	A	A	B	A	A	B	A	*	0	0	0
128	#	1	*	0	1	1	1	0	*	q_{22}	0	0	1	*	A	A	B	A	A	B	A	*	0	0	0
129	#	1	*	0	1	1	1	0	*	0	q_{22}	0	1	*	A	A	B	A	A	B	A	*	0	0	0
130	#	1	*	0	1	1	1	0	*	0	0	q_{22}	1	*	A	A	B	A	A	B	A	*	0	0	0
131	#	1	*	0	1	1	1	0	*	0	0	1	q_{22}	*	A	A	B	A	A	B	A	*	0	0	0
132	#	1	*	0	1	1	1	0	*	0	0	q_{24}	1	*	A	A	B	A	A	B	A	*	0	0	0
133	#	1	*	0	1	1	1	0	*	0	0	1	q_{25}	*	A	A	B	A	A	B	A	*	0	0	0
134	#	1	*	0	1	1	1	0	*	0	0	1	*	q_{25}	A	A	B	A	A	B	A	*	0	0	0
135	#	1	*	0	1	1	1	0	*	0	0	1	*	A	q_{25}	A	B	A	A	B	A	*	0	0	0
136	#	1	*	0	1	1	1	0	*	0	0	1	*	A	A	q_{25}	B	A	A	B	A	*	0	0	0
137	#	1	*	0	1	1	1	0	*	0	0	1	*	A	A	B	q_{25}	A	A	B	A	*	0	0	0
138	#	1	*	0	1	1	1	0	*	0	0	1	*	A	A	B	A	q_{25}	A	B	A	*	0	0	0
139	#	1	*	0	1	1	1	0	*	0	0	1	*	A	A	B	A	A	q_{25}	B	A	*	0	0	0
140	#	1	*	0	1	1	1	0	*	0	0	1	*	A	A	B	A	A	B	q_{25}	A	*	0	0	0
141	#	1	*	0	1	1	1	0	*	0	0	1	*	A	A	B	A	A	B	A	q_{25}	*	0	0	0
142	#	1	*	0	1	1	1	0	*	0	0	1	*	A	A	B	A	A	B	A	*	q_{25}	0	0	0
143	#	1	*	0	1	1	1	0	*	0	0	1	*	A	A	B	A	A	B	A	q_{26}	*	0	0	0
144	#	1	*	0	1	1	1	0	*	0	0	1	*	A	A	B	A	A	B	q_{26}	A	*	0	0	0
145	#	1	*	0	1	1	1	0	*	0	0	1	*	A	A	B	A	A	q_{26}	B	0	*	0	0	0
146	#	1	*	0	1	1	1	0	*	0	0	1	*	A	A	B	A	q_{26}	A	1	0	*	0	0	0
147	#	1	*	0	1	1	1	0	*	0	0	1	*	A	A	B	q_{26}	A	0	1	0	*	0	0	0
148	#	1	*	0	1	1	1	0	*	0	0	1	*	A	A	q_{26}	B	0	0	1	0	*	0	0	0
149	#	1	*	0	1	1	1	0	*	0	0	1	*	A	q_{26}	A	1	0	0	1	0	*	0	0	0
150	#	1	*	0	1	1	1	0	*	0	0	1	*	q_{26}	A	0	1	0	0	1	0	*	0	0	0
151	#	1	*	0	1	1	1	0	*	0	0	1	q_{26}	*	0	0	1	0	0	1	0	*	0	0	0
152	#	1	*	0	1	1	1	0	*	0	0	q_{26}	1	*	0	0	1	0	0	1	0	*	0	0	0
153	#	1	*	0	1	1	1	0	*	0	0	1	q_0	*	0	0	1	0	0	1	0	*	0	0	0
154	#	1	*	0	1	1	1	0	*	0	0	q_0	1	*	0	0	1	0	0	1	0	*	0	0	0

Máquinas de Turing

En su fundamento lógico, todo ordenador digital incorpora en sí uno de estos dispositivos conceptuales ideados por el matemático inglés A. M. Turing. Las máquinas demarcan los límites de la computabilidad

John E. Hopcroft

En 1900, el preeminente matemático David Hilbert presentó al Congreso Internacional de Matemáticos, que se celebraba en París, una relación de problemas no resueltos, que supusieron para el mundo de los expertos un reto y un estímulo. El problema segundo de aquella lista consistía en demostrar que los axiomas de la aritmética ordinaria eran consistentes (coherentes) entre sí. El trabajo de Hilbert sobre la cuestión le llevó a proponer un problema más general: el *Entscheidungsproblem*, o problema de la decisión. El *Entscheidungsproblem* iba a descubrir un método general para determinar si una fórmula de la lógica formal podía o no satisfacerse, o declararse verdadera. Treinta y seis años hubieron de transcurrir hasta su resolución completa, que señaló en las matemáticas un giro tan extraordinario como inesperado. En la Universidad de Cambridge, un joven matemático, miembro del King's College, de nombre Alan Mathison Turing, había llegado a familiarizarse con el problema gracias a una serie de lecciones impartidas por M. H. A. Newman. Turing reflexionó sobre el problema durante los largos paseos que por las tardes solía dar por la campiña inglesa. Fue tras uno de esos paseos cuando le llegó la respuesta. El problema de Hilbert era imposible de resolver.

La publicación en la que Turing anunció su resultado ha tenido una significación y trascendencia que rebasan con mucho el problema a que inmediatamente se dirigía. Al atacar el problema de Hilbert, Turing se vio forzado a plantearse cómo dar al concepto de método una definición precisa. A partir de la idea intuitiva de que un método es un algoritmo –un procedimiento que puede ser ejecutado mecánicamente, sin intervención creativa alguna–, Turing hizo ver cómo esta idea puede refinarse y convertirse en un modelo detallado del proceso de computación, en el cual un algoritmo cualquiera es descompuesto en una secuencia de pasos atómicos simples. El modelo computacional resultante es una construcción lógica conocida por máquina de Turing.

La forma más sencilla de describir la máquina de Turing es mediante elementos mecánicos, tales como ruedas, cintas perforadas y un sensor capaz de desplazarse hacia adelante y atrás, a lo largo de la cinta. Si bien toda esta maquinaria no es imprescindible, pues en su nivel fundamental la máquina de Turing no es sino la concreción material de un método de razonamiento matemático, la eliminación total de la analogía mecánica podría inducir a confusiones. Tal analogía le resultaba sugerente al propio Turing, que fue uno de los pioneros en el desarrollo del ordenador digital. Además, los méritos de la máquina de Turing en tanto que instrumento conceptual son hoy proclamados por las ciencias de cómputo con, por lo menos, tanta fuerza como pueda hacerlo la lógica. En las ciencias de cómputo su importancia es fundamental: si a la máquina de Turing se le da tiempo suficiente, que será mucho, pero finito, podrá llevar a cabo cualquier cómputo que pueda realizar un moderno ordenador digital, por muy potente que sea.

Esta capacidad universal de la máquina de Turing no implica que pueda ser utilizada de modo práctico como ordenador. Los ordenadores reales pueden funcionar a velocidad enormemente mayor que la máquina de Turing, porque al diseñarlos se sacrifica deliberadamente la claridad de funcionamiento en aras de mayor eficiencia y velocidad. No obstante, en el estudio teórico de la capacidad última que para la resolución de problemas pueda tener el ordenador real, la máquina de Turing se ha hecho indispensable. La máquina de Turing ha permitido, por ejemplo, demostrar que además problema de Hilbert hay otros mu que no pueden ser resueltos, por potente y veloz que sea el orden utilizado para ello.

Funcionamiento

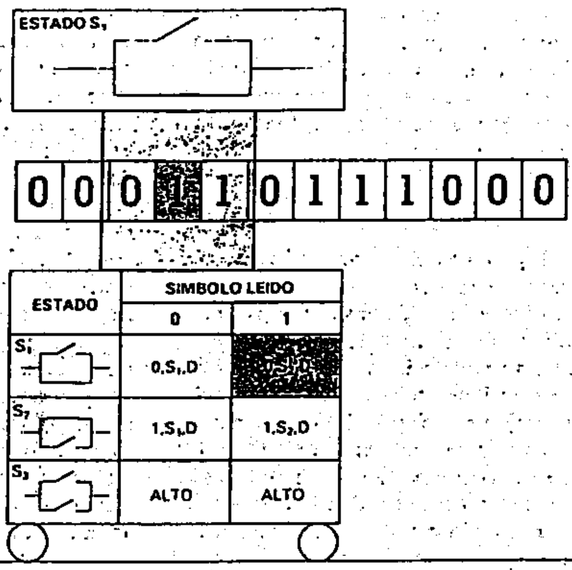
¿Qué es y cómo funciona una má na de Turing? Andrew Hodges, en reciente biografía de Turing, la cor ra a una máquina de escribir ordin: Lo mismo que la máquina de escr la de Turing dispone de un cab móvil para la impresión, que g: sobre una superficie adecuada símb individuales y discretos tomados d alfabeto finito. Para simplificar los vimientos del cabezal, se supone qu superficie de impresión es una cint: vidida en segmentos o cuadrículas cretas. Así pues, el cabezal de escri de la máquina de Turing tan sólo t que moverse en una dimensión, h la derecha o hacia la izquierda, y la: ciones de la máquina no tienen nec dad de entrar en complicaciones irrelevantes como la longitud de línea impresa o el interlineado. En c cuadro de la cinta tan sólo puede barse un símbolo, pero no hay lin ción sobre la longitud de la cinta, consiguientemente, sobre la long de la cadena de caracteres que pue quedar registrados en ella.

El cabezal móvil de la máquina Turing puede realizar, además de la presión, otras dos funciones. Al i que muchas de las máquinas de escr fabricadas durante el decenio pasa puede borrar, o suprimir, uno a u los símbolos grabados en la superf de escritura. Y a diferencia de la i quina de escribir, el cabezal puede t: bién "leer", esto es, reconocer el coi nido simbólico de cada cuadro de cinta, a razón de uno cada vez. De c forma, los símbolos de la cinta pue ingresar con carácter de datos en la i quina e influir en la determinación de su acción subsiguiente.

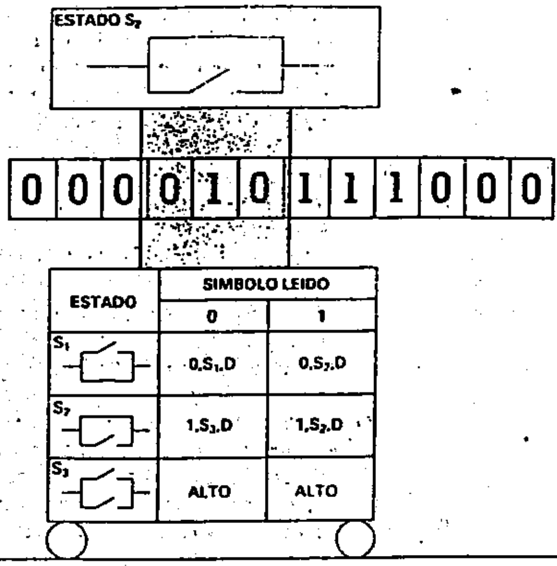
Una máquina de escribir puede adoptar en cada caso uno de varios "estados", o modos de funcionamiento en el transcurso de su actividad. En estado "minúsculas" imprime letras minúsculas o cifras, mientras que en el estado "mayúsculas" imprime éstas, o signos especiales. Análogamente, una máquina de Turing puede ir adoptando uno de entre un número finito de estados. Presumiblemente, cada estado constituye una configuración o disposición distinta de la máquina, pero puesto que la máquina de Turing es un artefacto relativamente abstracto, por lo común no se hace ningún esfuerzo para dar descripciones más concretas y mecánicas de sus estados. Será suficiente describir cada estado de la máquina a partir de los efectos que produzca sobre la actividad de la misma.

La actividad de la máquina de Turing

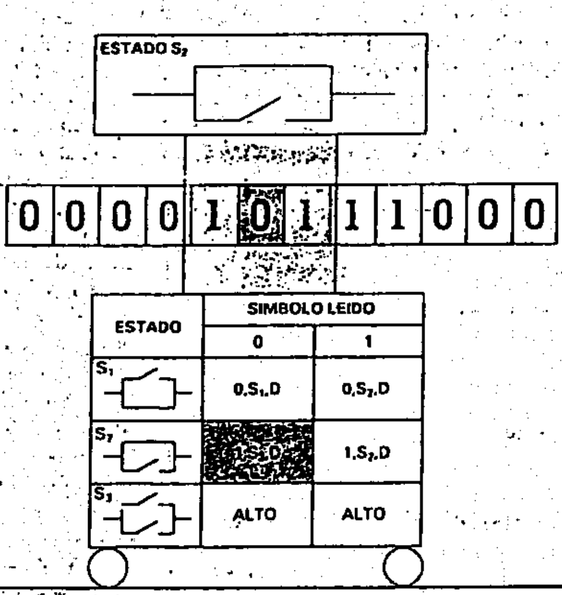
1



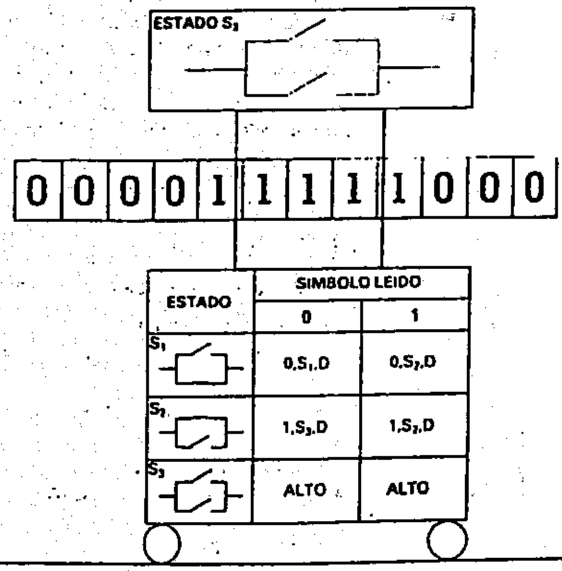
2



3



4



1. LA SUMA DE 2 Y 3 se realiza en cuatro pasos con una máquina de Turing. Cada sumando está representado en notación unaria sobre una cinta, en forma de secuencia de "unos" consecutivos, encerrados entre ceros por ambos extremos. La máquina puede inspeccionar uno por uno los contenidos de las cuadrículas de la cinta (*en color*), yendo y viniendo a izquierda o derecha de ella, en una serie de pasos discretos. La tarea de la máquina consiste en generar una hilera de cinco "unos" consecutivos y detenerse después. La tabla de instrucciones, que vemos bajo cada una de las máquinas del diagrama, es un conjunto fijo de movimientos que cubre todas las condiciones iniciales posibles y que da un procedimiento para sumar dos números cualesquiera. Obedeciendo las instrucciones, la máquina suprime el cero que separa dos hileras de unos y hace avanzar un paso hacia la derecha a la hilera de la izquierda, empalmándola así con la hilera derecha. El número de instrucciones iniciales previstas en la tabla debe ser suficiente para atender todas las contingencias que pudieran presentarse en la cinta; tal número puede incrementarse aumentando el número de estados internos, o configuraciones. Para cada posible combinación de símbolo de la cinta y de estado de la máquina, la tabla de instrucciones debe, o bien detener la máquina, o bien especificar tres variables: primero, dar el símbolo que ha de quedar en la casilla en ese momento inspeccionada; segundo, especificar el estado que la máquina ha de adoptar antes de examinar otra cuadrícula; y tercero, indicar si la máquina ha de desplazarse una cuadrícula hacia la derecha o la izquierda de la cinta.

se compone enteramente de pasos instantáneos y discretos; cada paso está determinado por dos condiciones iniciales: el estado presente de la máquina y el símbolo que está situado en el cuadro de la cinta que en ese momento se esté inspeccionando. Dado un cierto par de condiciones iniciales, la máquina recibe para su próximo paso una instrucción compuesta por tres secciones. La primera parte de la instrucción designa el símbolo que la máquina ha de depositar en el cuadro sometido a examen. Por ejemplo, si la instrucción especifica que en el cuadro ha de quedar el símbolo 1, la máquina imprimirá este símbolo si el cuadro está en blanco, dejará el símbolo intacto si hay un 1 ocupándolo y, si hubiera otro símbolo distinto de 1, lo borraría y reemplazaría por un 1.

La segunda parte de la instrucción especifica cuál será el siguiente estado de la máquina. Al igual que en la especificación del símbolo en la primera parte de la instrucción, la designación de un estado puede requerir que la máquina cambie su estado, o que conserve el que ya tiene. La tercera parte de la instrucción especifica si el cabezal impresor ha de inspeccionar en la cinta un cuadro hacia la izquierda o hacia la derecha.

Podemos abreviar la instrucción completa mencionando los tres valores de sus variables –el símbolo de la cinta, el estado de la máquina y el desplazamiento del cabezal de impresión– en un orden prefijado. Por ejemplo, si un par de condiciones iniciales han de hacer que la máquina deposite el símbolo 1 en el cuadro sometido a inspección, hacer que la máquina adopte el estado S_2 y que desplace un cuadro hacia la izquierda el cabezal de impresión, la instrucción podría abreviarse (1,S_2,I).

La mejor manera de comprender cómo funciona una máquina de Turing es tratar de construir una. En este contexto, construir una máquina de Turing significa construir una tabla de instrucciones que especifiquen la actuación de la máquina en cada posible par formado por un estado y un símbolo de la cinta. En la práctica, la construcción de la tabla conlleva además dotar a la máquina de un número suficiente de estados, que la permitan realizar la tarea que se le encomienda.

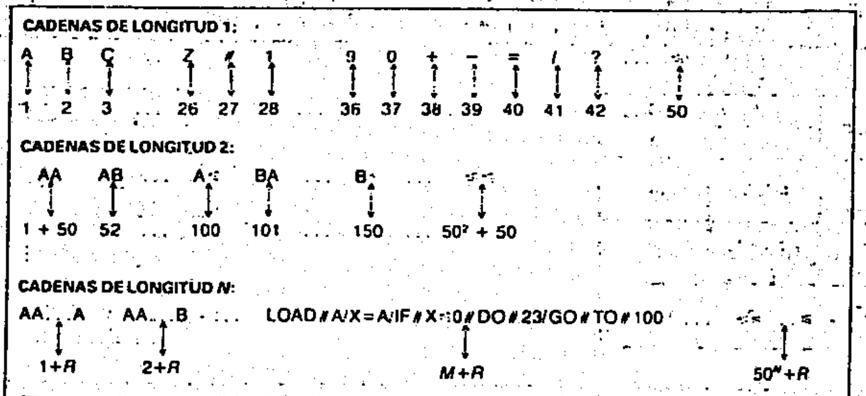
La máquina de sumar

Fijémonos en cómo podríamos proceder para diseñar una máquina que realice la suma de dos números y se detenga. Es costumbre, aunque en modo alguno es esencial, hacer que en la cinta se impriman solamente dos símbolos, por ejemplo, 0 y 1. Cualquier número N dado puede ser entonces representado en la cinta por una serie de N símbolos 1 consecutivos. Si es necesario imprimir en la cinta dos números, el M y el N, podemos representarlos por una primera serie de M "unos", seguida a la derecha de un 0, y seguida consecutivamente a su derecha de otra serie de N "unos". Supongamos que la máquina de Turing se encuentre en un estado S_1 inicial y que su cabezal esté inspeccionando el primero desde la izquierda de los 1 de la serie que expresa a M. Para construir una máquina de Turing ca de sumar, lo que se requiere es gen una tabla de instrucciones capaz de grar que la máquina imprima una s de $M + N$ "unos" consecutivos y pués se detenga.

Una forma sencilla de ejecutar éxito esta tarea es llevar un cua hacia la derecha a toda la serie "unos" correspondientes a M, que de las dos series, la situada más a la quierda. Si tal desplazamiento es rrectamente realizado, el 0 dejará separar las dos cadenas de "unos": serie única de "unos" resultante ten longitud $M + N$. A la máquina de ring no le es posible desplazar de solo golpe toda la serie de M "unos" cabezal de impresión, por ejemp avanza o retrocede a razón de un s cuadro cada vez. Una forma de con guir el resultado es crear una serie instrucciones para una máquina de T ring de tres estados. En el primer es do, el cabezal inspecciona la cinta izquierda a derecha, a razón de un cu dro por vez, hasta encontrar el 1 sit do más a la izquierda. La máquina co vierte el 1 en un 0, adopta el segun estado y prosigue su movimiento ha la derecha. En el segundo estado, cua do el cabezal encuentra un 1 en la ca lla bajo inspección, no hace nada – cambia ni el símbolo de la cinta, ni estado de la máquina– excepto avanz una casilla más hacia la derecha. I este modo, el cabezal va dejando atr los $M - 1$ "unos" que aún quedan en primera serie. Cuando finalmente ap rece un 0, las instrucciones hacen que cabezal transforme el 0 en un 1 y que máquina se detenga.

Invitamos ahora al lector para qu construya una máquina de Turir capaz de efectuar el producto de do números dados. Los convenios para entrada de los números son los mism que para la máquina sumadora: los do enteros a multiplicar vienen represe tados por dos series de "unos" consec tivos, separados por un 0. La salid una serie de "unos" cuya longitud ha d ser igual al producto de las dos prim ras series, puede quedar situada com una tercera serie, a la derecha de la dos primeras. Damos en la figura 5 u diseño capaz de funcionar. Para ser sir ceros, es preciso decir que la construc ción es delicada y requiere una "conta bilidad" algo minuciosa. También parc ce leal dar una indicación, de la que lo aficionados a rompecabezas pueder prescindir saltando el próximo pará grafo.

La multiplicación resulta mucho má



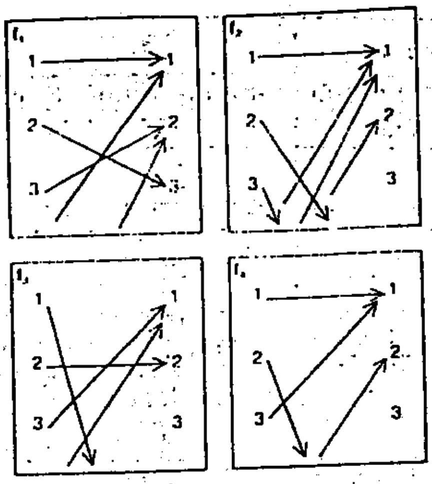
2. CADENAS DE CARACTERES de longitudes finitas cualesquiera forman un conjunto coordinable con el de los enteros positivos, siempre que cada carácter de la cadena sea elegido de entre un conjunto finito. Según definición formulada por Georg Cantor, tal correspondencia biunívoca expresa la igualdad de tamaños de ambos conjuntos, no obstante ser ambos conjuntos infinitos. Cualquier conjunto infinito coordinable con los números enteros positivos se llama "conjunto contable" o "numerable". Dado que todo posible programa de ordenador y toda posible tabla de instrucciones para máquinas de Turing pueden ser codificadas mediante cadenas finitas de símbolos, el emparejamiento muestra que los posibles programas de ordenador, o de máquinas de Turing, forman conjuntos infinitos de tipo numerable.

sencilla si primero preparamos una rutina, encargada de copiar cadenas de "unos" inmediatamente a la derecha de un punto dado. Siempre es posible copiar N "unos" utilizando una máquina de N estados; los estados pueden servir de modo efectivo como contadores de los "unos" de la serie. Sin embargo, no siendo deseable que el número de estados crezca ilimitadamente, resulta conveniente dar con un procedimiento para copiar la cadena sin contar sus elementos. Uno de los posibles es hacer que un marcador recorra la serie desde su extremo izquierdo hasta el derecho; tal marcador es un 0 que va ocupando sucesivamente el lugar de cada 1 de la serie. Para cada avance del marcador desde una cuadrícula a la siguiente se instruye al cabezal que inspeccione la serie hasta rebasar su extremo derecho, que pase por alto el cero indicador de espacio en blanco y que cambie después el siguiente 0 que encuentre, convirtiéndolo en un 1. Si el cabezal de impresión va y viene una vez, a modo de lanzadera de un telar por cada avance del marcador, quedará escrita una nueva serie de N "unos" consecutivamente a la derecha de la primera serie [*véase la figura 4*].

Construcción de programas complejos

Con algo de práctica se aprende pronto a construir máquinas de Turing capaces de ejecutar rutinas computacionales sencillas y a combinar entre sí estas máquinas a fin de realizar cálculos más complejos. Por ejemplo, los valores de una expresión polinómica pueden ser evaluados combinando rutinas de adición, copiado y multiplicación. Todavía más versátiles son rutinas breves y elementales dedicadas a manipulación de símbolos, tales como "desplazamiento del cabezal hacia la derecha, hasta que encuentre un 0" o "desplazamiento del marcador de la serie de unos situada más a la izquierda una cuadrícula hacia la derecha". Se saca provecho de variantes de estas rutinas breves tanto en la máquina copiadora como en la multiplicadora.

Si nos sentimos inclinados a construir, pongamos por caso, la máquina de Turing capaz de multiplicar, pronto comenzaremos a apreciar las dificultades que es preciso afrontar al diseñar un programa de utilidad para un ordenador. La mayoría de las máquinas de Turing pequeñas, esto es, que tan sólo admiten un reducido número de estados, son incapaces de realizar tareas



	1	2	3	4	5	6	...
f_1	$\boxed{1}$	3	2	1	7	2	
f_2	1	$\boxed{5}$	9	1	2	1	
f_3	9	2	$\boxed{1}$	6	1	6	
f_4	1	8	1	$\boxed{6}$	2	4	
f_5	1	1	2	4	$\boxed{1}$	1	
f_6	1	7	1	8	9	$\boxed{6}$	
⋮							
F	2	4	2	5	2	5	

3. EL CONJUNTO DE FUNCIONES MATEMATICAS con valores enteros que pueden definirse sobre el conjunto de los enteros positivos no es numerable; hay demasiadas de tales funciones como para emparejarlas una a una con números naturales. Supongamos que, por el contrario, todas las funciones tales pudieran ser nombradas asociándole un subíndice a cada una, con lo que podríamos enumerarlas todas en un orden fijo. Los valores que cada función asocia a cada número entero podrían ser entonces presentados en una hilera de infinitas cifras. Dado que una función está determinada por su hilera infinita de dígitos, imaginemos una función construida cambiando la primera cifra de la primera hilera, la segunda cifra de la segunda hilera de dígitos, y así sucesivamente. La hilera de dígitos correspondientes a la función así construida (*en color*) se diferencia al menos en una cifra de cualquier otra hilera de la lista original. La hipótesis según la cual todas las funciones pueden ser enumeradas en una lista engendra la contradicción de que siempre puede construirse una función nueva, no incluida en la lista primitiva.

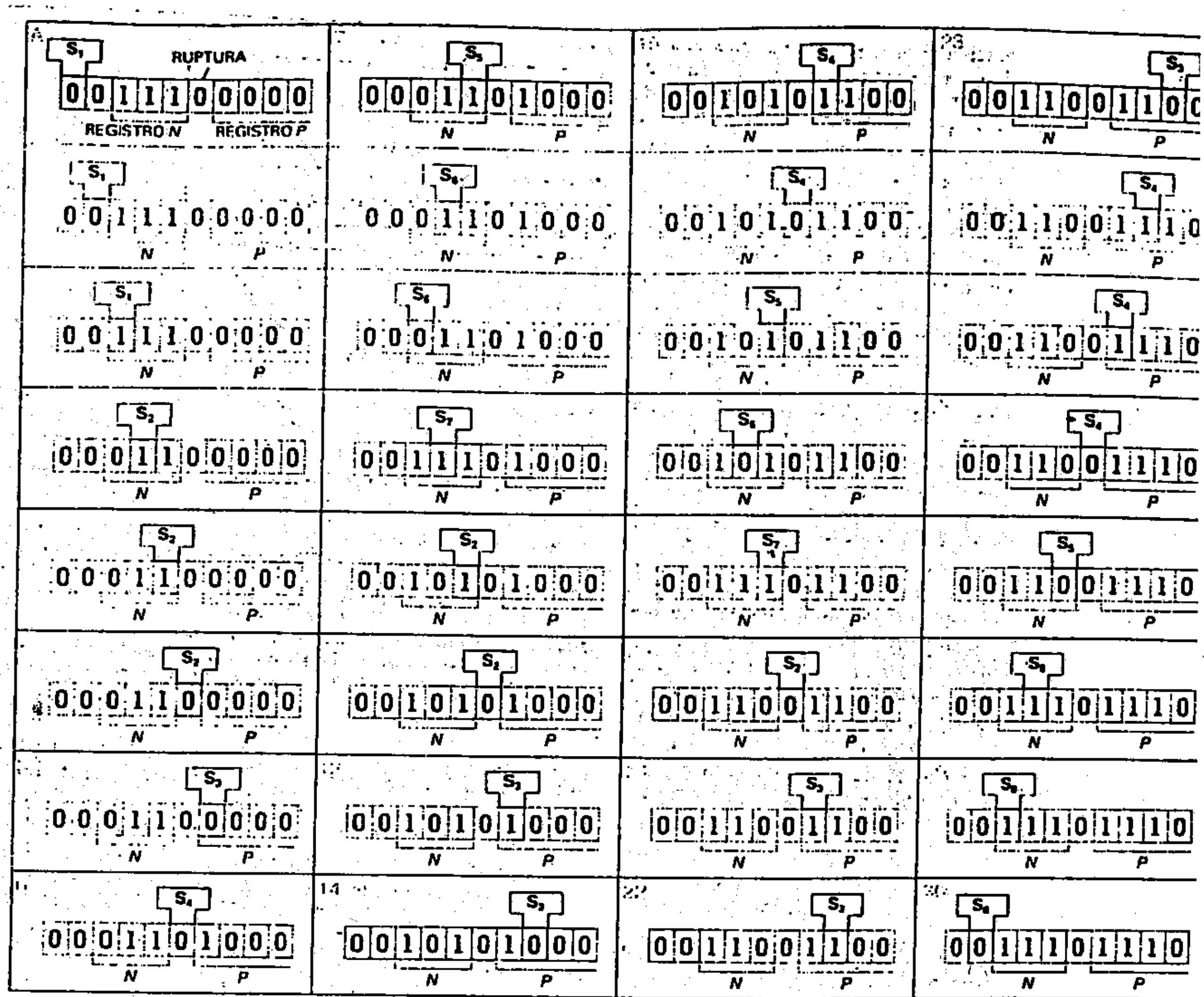
útiles, y, muchas veces, ni siquiera tareas razonables. Una buena parte de ellas quedan atrapadas en bucles infinitos, oscilando indefinidamente adelante y atrás de la cinta, sin jamás llegar a detenerse. De entre las máquinas que sí ejecutan tareas razonables debemos elegir una combinación de ellas capaces de colaborar eficientemente. La impresión inicial puede ser que la tarea más nimia puede plantear la dificultad más perversa, y que apenas si hay esperanza de computación en casos tomados de la realidad. Tales dificultades son sin duda enojosas, pero su carácter es meramente técnico; con unas cuantas rutinas bien elegidas la capacidad de resolución de problemas que adquiere la máquina de Turing se multiplica explosivamente, y no es ahora la debilidad de la máquina, sino su potencial, el que nos deja pasmados. Como Turing pudo demostrar, es posible combinar máquinas sencillas y combinar con ellas una máquina capaz de llevar a cabo cualquier tarea que admita descripción secuencial explícita.

El ordenador electrónico, que debe en parte su existencia a las máquinas conceptuales de Turing, es hoy, con toda probabilidad, la demostración más convincente de la potencia computacional de aquellas máquinas. En el curso de su trabajo, Turing señaló que cualquier máquina de Turing M puede quedar codificada en forma de sucesión de unos y ceros impresos en una cinta. La razón fundamental de que tal codificación puede realizarse es que toda máquina de Turing queda unívocamente definida por su tabla de instrucciones, tabla que ha de ser de longitud finita, ya que tanto el número de estados de la máquina como el alfabeto de símbolos a grabar son finitos.

La máquina universal de Turing

Turing demostró que la actuación de una máquina de Turing arbitraria M sobre una sucesión cualquiera X de símbolos de su cinta puede ser simulada por otra máquina, la máquina universal de Turing. Los símbolos que la máquina universal va registrando y reconociendo en su cinta están agrupados en dos secciones principales: a la izquierda está la descripción codificada de la máquina M; a la derecha está la sucesión de símbolos X que M encontraría al ir inspeccionando la cinta que le es propia. La máquina universal se construye entonces de modo que su cabezal vaya y venga entre las dos secciones, izquierda y derecha, de la cinta. Mediante un complejo sistema de marcadores, la máquina universal lleva la cuenta del estado codificado de M que se está consultando. Turing demostró que el efecto de la máquina universal sobre la sucesión de símbolos X es exactamente el que produciría M sobre la misma sucesión.

El éxito de la simulación que la má-



ESTADO	SIMBOLO DE LA CINTA: 0	COMENTARIO	EJEMPLOS	SIMBOLO DE LA CINTA: 1	COMENTARIO	EJEMPLOS
S_1	$0,S_1,D$	Se comienza con el desplazamiento hacia la derecha, hasta el límite izquierdo del registro *N*.		$0,S_2,D$	Marcar con 0 el extremo izquierdo del registro *N*.	1
S_2	$0,S_3,D$	Desplazamiento hacia la derecha, a través de la ruptura.		$1,S_2,D$	Desplazamiento hacia la derecha, a través del registro *N*, desde el marcador hasta la ruptura.	2, 3, 11
S_3	$1,S_4,I$	Copiar un 1 en el extremo derecho del registro *P*.		$1,S_3,D$	Desplazamiento hacia la derecha, a través de los "unos" del registro *P*.	13, 21, 22
S_4	$0,S_5,I$	Desplazamiento hacia la izquierda, a través de la ruptura.		$1,S_4,I$	Desplazamiento hacia la izquierda, a través del registro *P*.	15, 24, 25
S_5	$1,S_8,I$	El marcador 0 es detectado inmediatamente a la izquierda de la ruptura, lo que indica que al registro *P* se le ha añadido una copia completa del registro *N*.		$1,S_6,I$	Desplazamiento de un paso hacia la izquierda, a partir de la ruptura, entrando en el registro *N*. El estado no puede continuar en S_5. De hacerlo, el encuentro que llegaría a producirse con el 0 haría que la máquina se detuviese prematuramente.	7, 17
S_6	$1,S_7,D$	Iniciar el desplazamiento del marcador 0 un paso hacia la derecha.		$1,S_6,I$	Proseguir el desplazamiento a través del registro *N*, hasta el marcador 0.	8
S_7		No se encuentra condición; por tanto, puede insertarse una instrucción ficticia cualquiera o no insertarse ninguna.		$0,S_2,D$	Terminar el desplazamiento del marcador 0 una unidad hacia la derecha. Repetir el ciclo.	10, 19
S_8	0,–,–	La ruptura de copiado está terminada. Detención de la máquina o espera de nuevas instrucciones.		$1,S_8,I$	Desplazamiento hacia la izquierda, hasta el extremo izquierdo del registro *N*.	28, 29

4. MAQUINA COPIADORA, máquina de Turing que se utiliza como componente de otros dispositivos más complejos. Dada una cadena cualquiera de "unos" impresos en la cinta, la máquina escribe a continuación del 0 que señala el fin de la cadena una segunda serie con igual número de "unos".

quina universal hace de la máquina M es consecuencia tan sólo de que M pueda quedar exhaustivamente descrita por un número finito de símbolos. En principio, no obstante, todo ordenador digital puede ser descrito de igual manera. El ordenador tiene un número grande, aunque finito, de estados internos, y su respuesta a los datos ingresados en él queda totalmente determinada por el conjunto finito de enunciados que componen su programación. Por tanto, la descripción completa de cualquier ordenador digital puede quedar codificada en forma de sucesión de unos y ceros grabados sobre una cinta, y cualesquiera datos de entrada, quedar codificados en la misma cinta, a la derecha de la descripción del ordenador. Consultando alternativamente la descripción del ordenador y la serie de datos de entrada grabados en la cinta, la máquina universal ideada por Turing puede simular, paso por paso, la acción del ordenador sobre los datos de entrada.

Proveyéndolo de memoria suficiente, que haga el papel de cinta con la que manipular los símbolos, cualquier ordenador real puede desempeñar el papel de máquina universal de Turing. Por ejemplo, si un ordenador doméstico fuese programado para funcionar como máquina universal de Turing, y suponiendo que una descripción de un gran ordenador "de cabecera" le fuese codificada al pequeño como datos de entrada, el microordenador simularía la actuación que el ordenador grande pudiera tener sobre cualquier serie de símbolos-datos. En este sentido, todos los ordenadores digitales pueden computar exactamente la misma clase de funciones matemáticas: a saber, todas las que sean computables por algunas máquinas de Turing. La existencia de tan sólo una clase tal respalda con fuerza la definición formal de computabilidad dada por Turing: una función matemática es computable si puede ser computada por alguna máquina de Turing. Turing arguyó convincentemente que su definición equivale a cualquier interpretación razonable de la noción intuitiva de computabilidad. Tal vez convenga mencionar, no obstante, que es absurdo pedir una demostración matemática rigurosa de que una definición formal, como la de Turing, capta perfectamente una cierta noción de carácter intuitivo.

El programa de Hilbert

Para comprender por qué estaba Turing tan empeñado en definir qué es la computabilidad conviene tener alguna noción del desarrollo de la lógica matemática hasta 1936. La matemática rigurosa, en la forma en que hoy es conocida, es cosa relativamente reciente. El primer intento serio de reducción de los enunciados matemáticos a enunciados de la lógica formal fue el iniciado por Gottlob Frege en 1879, con la publicación de su *Begriffsschrift* ("La notación de los conceptos", literalmente). El problema planteado por Hilbert tenía, pues, trascendencia inmediata para los matemáticos, si el esquema de Frege podía llevarse a conclusión. En efecto, si pudiera darse con un método para determinar la veracidad o falsedad de un enunciado cualquiera de la lógica formal, y de ser correctas las ideas de Frege, con tal método se podría determinar también la veracidad o falsedad de cualquier enunciado matemático, no importa cuán complejo. Con un tal método quedarían inmediatamente liquidadas conjeturas como "el último teorema de Fermat" que durante siglos han venido resistiendo todos los embates para demostrarlas o refutarlas. Si el osado reto lanzado por Hilbert tuviera respuesta afirmativa, toda la matemática habría quedado reducida a cálculo mecánico.

Durante los primeros decenios de nuestro siglo se produjeron en lógica dos importantes desarrollos, que desorganizaron y echaron por tierra gran parte del programa de Hilbert. En 1901, Bertrand Russell descubrió en la teoría elemental de conjuntos una paradoja irrefutable, teoría elemental que desempeñaba un papel fundamental en el programa de reducción de la matemática a la lógica propuesto por Frege. Russell comunicó a Frege su descubrimiento justamente cuando el segundo volumen de la última gran obra de Frege, *Grundgesetze der Arithmetik* ("Las leyes fundamentales de la aritmética"), estaba a punto de ser publicado. Frege cerró el volumen con una desalentada nota: "Difícilmente puede un científico encontrarse nada más indeseable que ver hundirse los cimientos justamente cuando se está concluyendo el trabajo. Esta ha sido la situación en que me he encontrado a causa de una carta del Sr. Bertrand Russell, recibida cuando la obra estaba a punto de salir de imprenta". A pesar de los fallos, que Frege no pudo corregir en su obra, Russell y Alfred North Whitehead pudieron más tarde salvar el programa de Frege y eludir la paradoja en la teoría de conjuntos.

Un segundo y fundamental descubrimiento en lógica fue el realizado por Kurt Gödel, a la sazón en la Universidad de Viena. Implícitamente, el programa de Hilbert presuponía que en la lógica formal debiera existir un procedimiento para distinguir los enunciados verdaderos de los falsos: el problema consistía en dar con el método. Gödel demostró que tal presunción es injustificada. En 1931 pudo probar que cualquier sistema consistente de la lógica formal que fuera lo bastante potente como para formular en él enunciados acerca de la teoría de números (la aritmética) ha de contener enunciados verdaderos que no pueden ser demostrados. Dado que sistemas axiomáticos consistentes, tales como el ideado por Russell y Whitehead, no pueden contener todos los enunciados verdaderos de la teoría que pretenden formalizar, se dice que tales sistemas son incompletos.

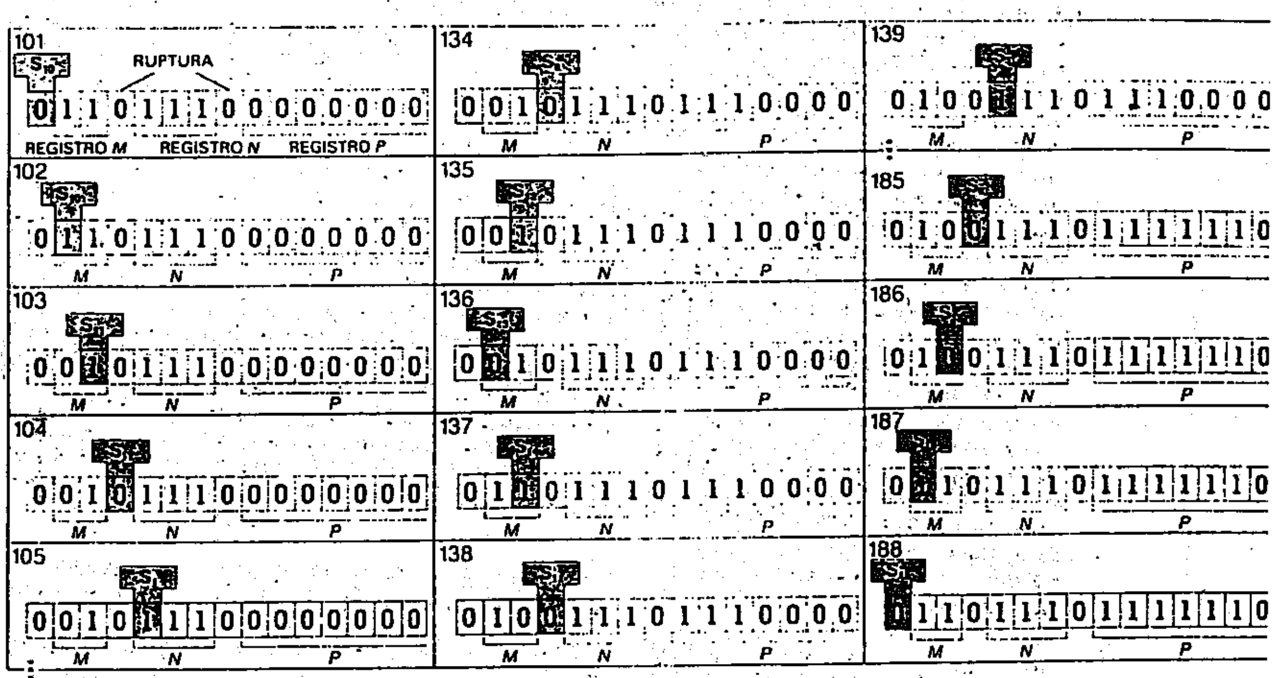
La lógica de la computabilidad

El trabajo de Gödel puso verdaderamente término al programa de Hilbert. No puede existir un método que permita decidir si un enunciado arbitrariamente dado es verdadero o falso. De haberlo, el método constituiría una demostración de todos los enunciados verdaderos, y Gödel había demostrado que en el seno de un sistema axiomático consistente, de potencia suficiente para abarcar la aritmética ordinaria, una tal demostración es imposible. La atención de los lógicos se desplazó desde la noción de verdad a la de demostrabilidad. Y en este contexto quedaba por elucidar una cuestión, análoga sencilla de la de Hilbert; a saber: ¿Existe un único método mediante el cual todos los enunciados matemáticos demostrables puedan ser demostrados a partir de un sistema de axiomas lógicos?

El más distinguido investigador de la lógica de demostrabilidad, de los años inmediatamente posteriores a la demostración de Gödel, fue Alonzo Church, de la Universidad de Princeton. Church y dos de sus discípulos, Stephen C. Kleene y J. Barkley Rosser, pusieron a punto un lenguaje formal coherente, bautizado por ellos "lambda-cálculo". Este lenguaje es útil para razonar sobre funciones matemáticas, tales como la raíz cuadrada, los logaritmos y cualesquiera otras funciones más complicadas que pudieran definirse. (Church eligió la letra griega "lambda", equivalente a la L latina, para sugerir que su sistema formal es un lenguaje). El moderno lenguaje de programación llamado Lisp (de *List* *processing*, procesamiento de listas) está inspirado en el lambda-cálculo. Kleene demostró que amplias clases de funciones matemáticas, incluidas todas las utilizadas por Gödel en su demostración, podrían ser expresadas en el lambda-cálculo.

El siguiente paso importante en esta misma línea de pensamiento fue dado por Church. Aducía que si una función matemática es computable –queriendo con ello decir que puede calcularse su valor para todo número perteneciente a su dominio de definición– entonces tal función es definible en el lambda-cálculo. El trabajo de Church mostraba que de haber cosa tal como una función matemática expresable en el lambda-cálculo, pero no computable, no habría método alguno para determinar si un enunciado matemático dado es o no demostrable, y no digamos verdadero. La última de las hipótesis aún supervivie tes del programa de Hilbert quedaría fin infirmada: en abril de 1936, Churc publicó una fórmula lógica que no computable en su sistema.

Turing, trabajando independient mente, y sin relación con Church, hab comprendido también que existía ur relación técnica entre el problema d Hilbert y la noción de función comp table. Empero, al atacar el problen



ESTADO	SIMBOLO DE LA CINTA: 0	COMENTARIO	EJEMPLOS	SIMBOLO DE LA CINTA: 1	COMENTARIO	EJEMPLOS
S_{10}	$0,S_{10},D$	Se comienza con desplazamiento hacia la derecha, hasta el límite izquierdo del registro *M*.	101	$0,S_{11},D$	Marcar con un 0 el extremo izquierdo del registro *M*.	102
S_{11}	$0,S_1,D$	Desplazamiento hacia la derecha, a través de la ruptura que separa los registros *M* y *N*. Comienza la rutina de copiado (*véase la ilustración precedente*).	104, 138	$1,S_{11},D$	Desplazamiento hacia la derecha, a través del registro *M*, desde el marcador hasta la ruptura.	103
S_8	$0,S_{12},I$	Desplazamiento hacia la izquierda, a través de la ruptura entre *M* y *N*. Fin de la rutina de copiado.	134, 185	$1,S_8,I$	Desplazamiento hacia la izquierda, hasta el extremo izquierdo del registro *N*.	No figuran
S_{12}	$1,S_{15},I$	El marcador 0 es detectado inmediatamente a la izquierda de la ruptura entre los registros *M* y *N*, lo que indica que el producto de los números almacenados en los registros *M* y *N* ha sido almacenado en el registro *P*.	186	$1,S_{13},I$	Desplazamiento izquierdo, desde la ruptura entre los registros *M* y *N*, entrando un paso en el registro *M*. El estado no puede seguir siendo S_{12}; de serlo, el encuentro con el marcador 0 haría detenerse prematuramente la máquina de Turing.	135
S_{13}	$1,S_{14},D$	Comienza el avance del marcador 0 situado en el registro *M*, un paso hacia la derecha.	136	$1,S_{13},D$	Prosigue el desplazamiento hacia la izquierda, a través del registro *M* hasta el marcador 0.	No figuran
S_{14}		No se encuentra condición. Puede, por tanto, insertarse una instrucción ficticia cualquiera, o no insertarse ninguna.		$0,S_{11},D$	Terminar el avance del marcador 0 del registro *M*, una unidad hacia la derecha; repetir el ciclo.	137
S_{15}	0,–,–	La rutina de multiplicación está terminada. Detención de la máquina, o espera de nuevas instrucciones.	188	$1,S_{15},I$	Desplazamiento hacia la izquierda, hasta el extremo izquierdo del registro *M*.	187

5. PODEMOS EFECTUAR LA MULTIPLICACION mediante una máquina de Turing integrando en ella otra máquina de Turing capaz de copiar cadenas de "unos". En este ejemplo se requieren 88 ciclos para hallar el producto de 2 por 3. Los saltos de numeración corresponden a ciclos de máquina que trasladan el control a estados definidos para la máquina copiadora (*véase la figu precedente*). En el último ciclo se presenta al producto de 2 por 3 en forma cadena de seis "unos" impresos en una sección de la cinta llamada registro que se halla situado inmediatamente detrás del 0 que separa los dos factore

procedió de forma mucho más directa y concreta que Church. Lo que se requería era un modelo a la vez preciso y sencillo del proceso de computación; las máquinas de Turing fueron diseñadas para satisfacer tal necesidad. Una vez especificadas cuáles habrían de ser sus propiedades, Turing estableció brillantemente la conexión entre la idea de función computable y los resultados de trabajos matemáticos realizados unos 50 años antes por el alemán Georg Cantor. Cantor había razonado que si bien no existe un número entero que sea el máximo, cualquier conjunto infinito de objetos que pudieran ser contados, esto es, biunívocamente asociados con los números enteros positivos, es un conjunto del mismo tamaño que el conjunto de los números enteros. Dado que cualquiera de las máquinas de Turing puede ser expresada por una cadena de caracteres de longitud finita, todas las posibles máquinas de Turing, y con ellas todas las posibles funciones computables, pueden ser inventariadas en orden numérico o lexicográfico; pueden por tanto ser puestas en correspondencia biunívoca con los números enteros [*véase la figura 2*]. Como es obvio, no existe una cota superior que limite el tamaño de las máquinas de Turing, ni por consiguiente tampoco hay límite para los números de posibles máquinas de Turing distintas. No obstante, el análisis de Cantor muestra que el conjunto de todas las posibles funciones computables es del mismo tamaño que el conjunto de todos los números enteros; ambos conjuntos son llamados contables o numerables.

Cantor había demostrado también que hay infinitos conjuntos que no son numerables; son conjuntos mayores que el conjunto de los números enteros, en el sentido de que no pueden ser biunívocamente asociados con éstos. Entre los ejemplos de tales conjuntos no numerables tenemos el conjunto de todas las funciones dependientes de números enteros, con valores también enteros. Un análisis cuidadoso demuestra que ha de haber más de tales funciones que números enteros. La consecuencia es que será imposible que existan suficientes programas como para computar toda posible función.

El problema de la detención

¿Cuáles son las funciones no computables? Desdichadamente, las demostraciones de Church y de Turing no proporcionan directamente ejemplos de funciones no computables. No obstante, durante los veinte últimos años, se ha sacado partido de las máquinas de Turing para construir varias de tales funciones. Uno de los primeros ejemplos de función no computable fue ideado por Tibor Rado, de la Universidad estatal de Ohio, en 1962. Fijémonos en todas las máquinas de Turing que tienen cierto número fijo N de estados. Supongamos que todas las máquinas se hacen comenzar a funcionar sobre una cinta en blanco, o más precisamente, sobre una cinta cuyos cuadros están todos ocupados por ceros. Imagi-

UN SIMBOLO

1, 2, 3, 9, π

DOS SIMBOLOS

11 99 $9!$ $(=1\times2\times3\times4\times5\times6\times7\times8\times9=362.880)$ 9^9 $(=387.420.489)$

TRES SIMBOLOS

UNO 9+9 $3\uparrow4$ $(=3\times(3\times(3\times3))=3^4=81)$ 9^{9^9} $(=9^{387.420.489} \doteq 10^{[illegible]})$

CUATRO SIMBOLOS

TRES $10^{[illegible]}$ $3\uparrow\uparrow4$ $(=3\uparrow(3\uparrow(3\uparrow3))=3^{3^{3^3}}=3^{3^{27}}=3^{7.625.597.484.987}$
$\doteq 10^{[illegible]})$

CINCO SIMBOLOS

CINCO $3\uparrow\uparrow\uparrow4$ $(=3\uparrow\uparrow(3\uparrow\uparrow(3\uparrow\uparrow3))=3\uparrow\uparrow(3\uparrow\uparrow3^{3^3})=3\uparrow\uparrow3^{3^{\cdot^{\cdot^{3}}}}\Big\}3^{3^3}$ NIVELES
$=3^{3^{\cdot^{\cdot^{3}}}}\Big\}3^{3^{\cdot^{\cdot^{3}}}}$ NIVELES $\Big\}3^{3^3}$ NIVELES)

⋮

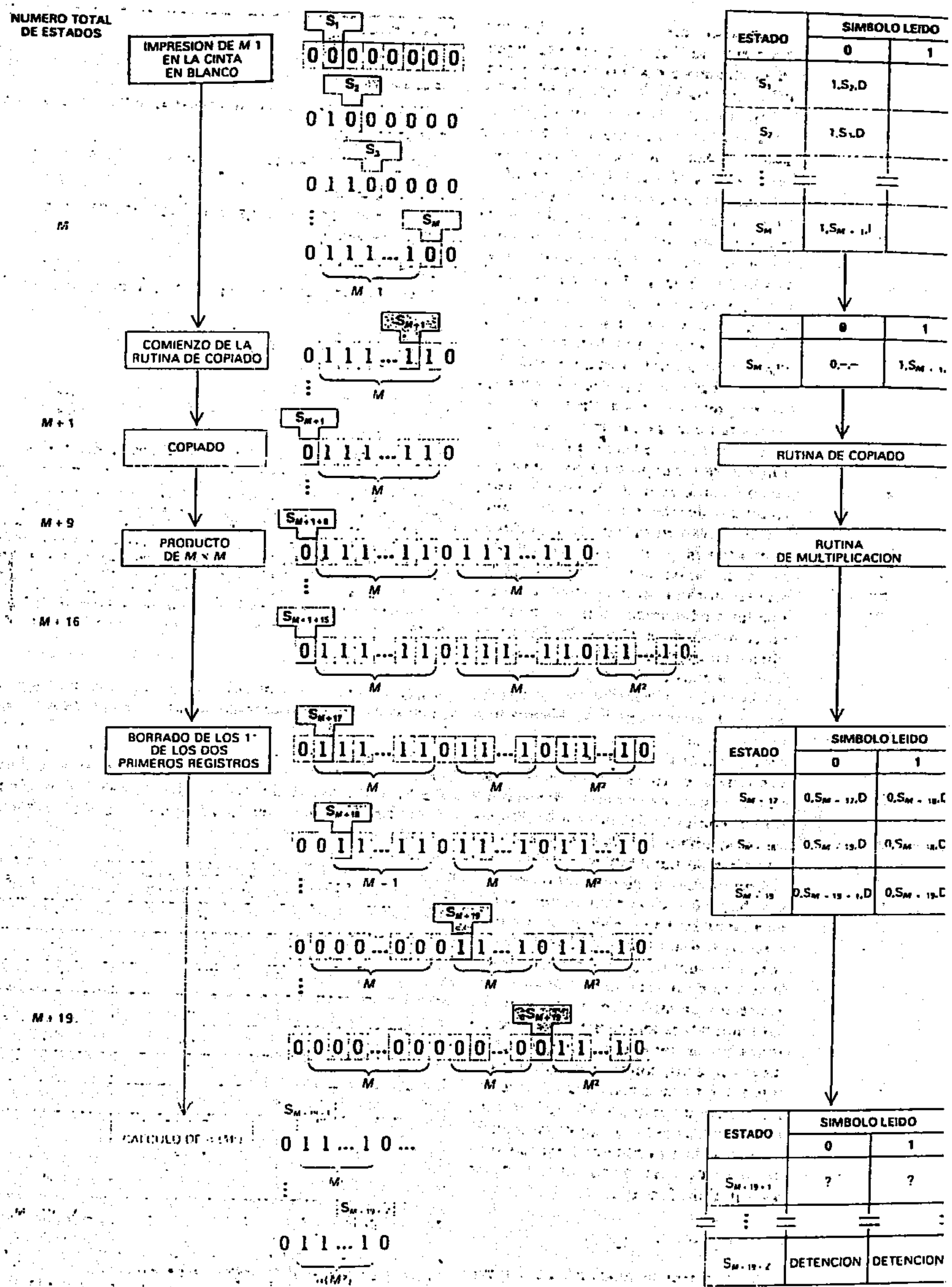
53 SIMBOLOS

1 ≠ MAS ≠ QUE ≠ EL ≠ MAXIMO ≠ NUMERO ≠ EXPRESABLE ≠ CON ≠ 53 ≠ SIMBOLOS

6. PARADOJA DE RICHARD, así llamada en honor del matemático francés Jules Richard. Se suscita al suponer que los enteros positivos pueden ser ordenados, o numerados en lista, de acuerdo con el número de símbolos necesarios para especificarlos. Es posible designar enteros aislados de considerable tamaño con ayuda de símbolos especiales, como la notación sagital ideada por Donald E. Knuth, de la Universidad de Stanford. Empero, de acuerdo con su propia descripción, el número supuestamente especificado arriba mediante 53 símbolos es mayor que cualquier otro que pueda ser especificado mediante 53 símbolos, lo que no deja de constituir una paradoja. Se presentan paradojas similares cuando se intenta hallar una máquina de Turing determinada en una lista de máquinas de Turing que se encuentran dispuestas según la longitud de la cadena de símbolos necesarios para codificar cada una de dichas máquinas.

NUMERO DE ESTADOS	NUMERO MAXIMO DE "UNOS" IMPRESOS	COTA INFERIOR PARA EL VALOR DE σ
3	$\sigma(3)$	6
4	$\sigma(4)$	12
5	$\sigma(5)$	17
6	$\sigma(6)$	35
7	$\sigma(7)$	22.961
8	$\sigma(8)$	$3^{92} \doteq 7.9 \times 10^{43}$
9	$\sigma(9)$	$3^{92}+1$
10	$\sigma(10)$	$a^{a^{\cdot^{\cdot^{a}}}}\Big\}a^{a^{\cdot^{\cdot^{a}}}}\Big\}\cdots\Big\}a^a$

7. EL PROBLEMA DEL "CASTOR AFANOSO" consiste en hallar el número máximo de "unos" que puede llegar a imprimir una máquina de Turing de N estados que comienza a funcionar con una cinta inicialmente ocupada sólo por "ceros" y que ha de terminar por detenerse. Tal número, que depende de N, constituye el valor de una función designada por $\sigma(N)$. En 1962, Tibor Rado, de la Universidad estatal de Ohio, demostró que tal función crecía con excesiva rapidez para poder ser computable. Vemos en la tabla cotas inferiores de la función, estimadas para valores pequeños de N. La cota inferior correspondiente a $\sigma(10)$ puede ser expresada mediante un número a cuyo valor es, aproximadamente, $\sqrt{8}$; $\sigma(10)$ es una pila de a-es exponencialmente acumuladas, donde el número de a-es de cada pila viene expresado por otro número de a-es. El proceso de definición de la altura de una pila mediante otra se reitera 10 veces.



8. LA DEMOSTRACION DE NO COMPUTABILIDAD de σ(*N*) comienza por calcular una estimación del número de estados necesarios para generar una cadena de M^2 signos 1 en una cinta inicialmente ocupada por "ceros". Las máquinas de Turing para copiar y multiplicar han sido combinadas aquí, formando una máquina de $M + 16$ estados. Otros tres estados más se encargan de borrar las dos cadenas de "unos" situadas a la izquierda, dejando una cadena de M^2 cifras 1. Se supone que para toda cadena de *N* cifras 1, hay una máquina de *Z* estados capaz de generar σ(*N*) "unos" (*texto en color*) nemos que de este conjunto hayamos podido excluir todas las máquinas de Turing que no llegan jamás a detenerse. De entre las restantes, seleccionemos la máquina o grupo de máquinas que hayan impreso sobre la cinta en blanco el máximo número de "unos" en sucesión antes de detenerse. Tal número de "unos", que está definido para cada valor de N, es el valor de la función de Rado, y es habitualmente designado por $\sigma(N)$. La demostración detallada de que $\sigma(N)$ no es computable consiste en suponer que sí puede ser computada y en deducir una contradicción. El razonamiento es bastante directo, pero los detalles técnicos son intrincados [*véanse las figuras 8 y 9*].

Pudiera pensarse que el razonamiento es deficiente, porque presume que las máquinas de Turing de N estados que no llegarían a detenerse pueden ser eliminadas de antemano. Tal objeción es verdaderamente seria. Fijémonos, así pues, en cómo podríamos intentar el cálculo de $\sigma(N)$ por la fuerza bruta. Damos primero la lista completa de todas las máquinas de Turing de N estados, ordenadas según cierto criterio. Simulemos cada una de ellas en una máquina universal de Turing; y seleccionemos la máquina o grupo de máquinas que más "unos" hayan impreso. Aunque este método de computación parezca eliminar la objeción, la dificultad planteada por las máquinas de Turing que no llegan a detenerse reaparece, esta vez en forma de no-tratable. Algunas de las máquinas de N estados que llegan a detenerse pueden quedar eliminadas mediante algoritmos sencillos, pero hay otras para las que no puede tomarse tal decisión. Si no podemos determinar si una máquina concreta se va o no a parar, no podremos eliminarla de la lista de máquinas de N estados y la simulación habrá de continuar. Y dado que muy bien puede suceder que la máquina no llegue a detenerse jamás, no existe garantía de que el cálculo de $\sigma(N)$ pueda llevarse a término.

Si bien la máquina de Turing produjo su impacto primero en la lógica matemática, ha desempeñado un papel dominante en informática desde hace unos 20 años. En 1965, Juris Hartmanis y Richard E. Stearns, a la sazón en los General Electric Research Laboratories, en Schenectady, Nueva York, demostraron que la máquina de Turing podía ayudar a establecer acotaciones muy estrictas de complejidad de computación. Investigadores subsiguientes comenzaron a clasificar los problemas según la forma de dependencia del tiempo de funcionamiento, o lo que viene a ser equivalente, del número de pasos de cómputo, respecto del tamaño del problema. Por ejemplo, supongamos que cierto número de puntos N estén interconectados mediante líneas, formando un grafo compuesto por vértices y lados. El problema consiste en colorear los vértices de modo que ningún par de vértices conectados por una línea sean de un mismo color. Supongamos, además, que el más rápido de los métodos conocidos para resolver el problema exija tiempos que varíen según cierta potencia de N, N^2, pongamos por ejemplo. Se dice entonces que el problema pertenece a la clase de problemas resolubles "en tiempo polinómico", clase denotada P. La clase P ha ido adquiriendo cada vez mayor importancia; hoy son muchos los especialistas en ciencias de cómputo que consideran no-tratables los problemas no pertenecientes a la clase P.

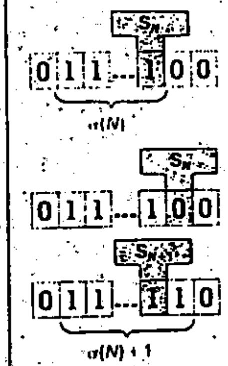
Teoría moderna de la complejidad

Observemos que, para que un problema sea asignado a la clase P, ningún caso particular del problema puede exigir para su resolución un tiempo mayor que el polinómico. Dicho con otras palabras, el método de solución es determinístico en el sentido de que garantiza la existencia de una solución, y ello en un tiempo inferior al de una cierta potencia fija del tamaño N del problema.

Podemos definir también máquinas de Turing no determinísticas. A tales máquinas se les permite resolver el problema por tanteo: se conjetura una posible solución y se verifica si la conjetura ha sido o no atinada. Por ejemplo, para determinar si un número es compuesto, la máquina no determinística tantea con un supuesto divisor, y si la división resulta exacta, da por comprobado que el número es compuesto. Una máquina determinística, por el contrario, habría de proceder a buscar sistemáticamente un divisor.

El tiempo requerido para resolver un problema con una máquina no determinística viene medido por la longitud del cálculo mínimo; por tanto, la máquina no determinística parecería tener enorme ventaja sobre la determinística. La experiencia ordinaria sugiere que es más fácil comprobar si algo es solución que empezar por encontrarla. Sin embargo, nadie ha podido demostrar que los problemas resolubles en tiempo polinómico con máquinas no determinísti-

"Observemos ante todo que σ es uniformemente creciente: es decir, si $X > Y$, $\sigma(X) > \sigma(Y)$. Si una máquina de Turing de N estados imprime un número dado de "unos", siempre podrá construirse una máquina de $N+1$ estados que añada un 1 a la anterior cadena de 1s, y después se detenga."



ESTADO	SIMBOLO LEIDO	
	0	1
S_N	$1, S_{N+1}, I$	$1, S_N, D$
S_{N+1}	DETENCION	DETENCION

Supongamos que exista una máquina de Turing, de Z estados, capaz de computar $\sigma(M^2)$.
Por la definición de σ y el cálculo mostrado en la ilustración de la página adyacente,

$\sigma(M + 19 + Z) \geq \sigma(M^2)$. (1).

Sin embargo, si M es suficientemente grande, el enunciado (1) contradice que σ sea uniformemente creciente.
Para demostrarlo, sea $M = Z + 20$, o sea, $Z = M - 20$
Entonces, $M + 19 + Z = M + 19 + M - 20 = 2M - 1$.
Como es evidente, $2M - 1 = M^2 - (M-1)^2$, y así $M + 19 + Z = M^2 - (M-1)^2$.
Dado que $(M-1)^2 > 1$, $M^2 - 1 > M^2 - (M-1)^2$.
Así pues, por ser σ uniformemente creciente,

$\sigma(M^2 - 1) > \sigma(M^2 - (M-1)^2) = \sigma(M + 19 + Z) \geq \sigma(M^2)$.

Sin embargo, como $\sigma(M^2 - 1) > \sigma(M^2)$, σ no puede ser uniformemente creciente, y por tanto, la suposición de que existe una máquina de Turing de Z estados que computa $\sigma(M^2)$ implica una contradicción.

9. **PASOS FINALES de la demostración de no computabilidad de $\sigma(N)$: extraen una contradicción de la hipótesis de que hay una máquina de Z estados tal que con una cadena de M^2 cifras 1 computa $\sigma(M^2)$.**

cas —que forman la clase de problemas NP— sean intrínsecamente de mayor dificultad que los problemas de la clase P. La cuestión de si la clase P es o no distinta de la clase NP, cuestión conocida por "problema P-NP", se ha convertido en una de las principales cuestiones no resueltas de las matemáticas de hoy.

En 1970, Stephen A. Cook, de la Universidad de Toronto, dio un importante paso en relación con el problema P-NP. Cook estaba investigando el problema de determinación de condiciones bajo las que una proposición lógica compuesta es verdadera. La proposición compuesta que se forma cuando dos proposiciones simples se conectan por la disyuntiva "o" es verdadera cuando una de las proposiciones, o ambas, son verdaderas. Es en general muy difícil determinar el campo de condiciones de veracidad que deben cumplir las proposiciones simples para satisfacer una proposición compuesta o, con otras palabras, los valores de veracidad o falsedad que deben asignarse a las proposiciones simples para que sea verdadera la proposición compuesta. Cook consiguió demostrar que este problema, llamado problema de satisfactibilidad, es tan difícil como cualquier otro problema de la clase NP. La condición necesaria y suficiente para que exista un algoritmo eficiente capaz de resolver el problema de satisfactibilidad es que exista un algoritmo tal para cada uno de los problemas de la clase NP. Todo problema que posea esta misma propiedad con relación a una clase dada de problemas se dice ser completo para tal clase.

Transcurrió un año hasta que la mayoría de los investigadores empezaran a captar la importancia del resultado de Cook. En 1971, Richard M. Karp, de la Universidad de California en Berkeley, comenzó a preguntar qué otros problemas naturales podrían desempeñar el mismo papel que desempeña el problema de satisfactibilidad con respecto a la clase NP. Karp descubrió que muchos problemas importantes de investigación operativa, incluido el de colorear un gráfico con tres colores, son también de la misma dificultad que cualquiera de los problemas que pueden ser asignados a la clase NP; con otras palabras, son NP-completos. Puede demostrarse directamente, trasladando un problema al dominio del otro, que el problema de coloreado del grafo y el problema de satisfactibilidad son equivalentes [*véase la figura 10*].

Se ha demostrado por métodos similares que varios centenares de problemas, que antes se tuvieron por distintos, son en realidad variantes unos de otros, cuyas diferencias son puramente notacionales. Todos estos problemas son equivalentes al de satisfactibilidad, y son todos, por tanto, NP-completos. Se han descubierto otras varias colecciones de tales problemas completos, tanto para la clase P como para las clases de problemas no tratables, en las cuales el número de pasos que para su resolución requiere una máquina de Turing crece exponencialmente con el tamaño del problema. Empero, parece todavía prematuro intentar el asalto directo del problema P-NP. Su dificultad puede, en parte, apreciarse a partir de desarrollos recientes en teoría de computación.

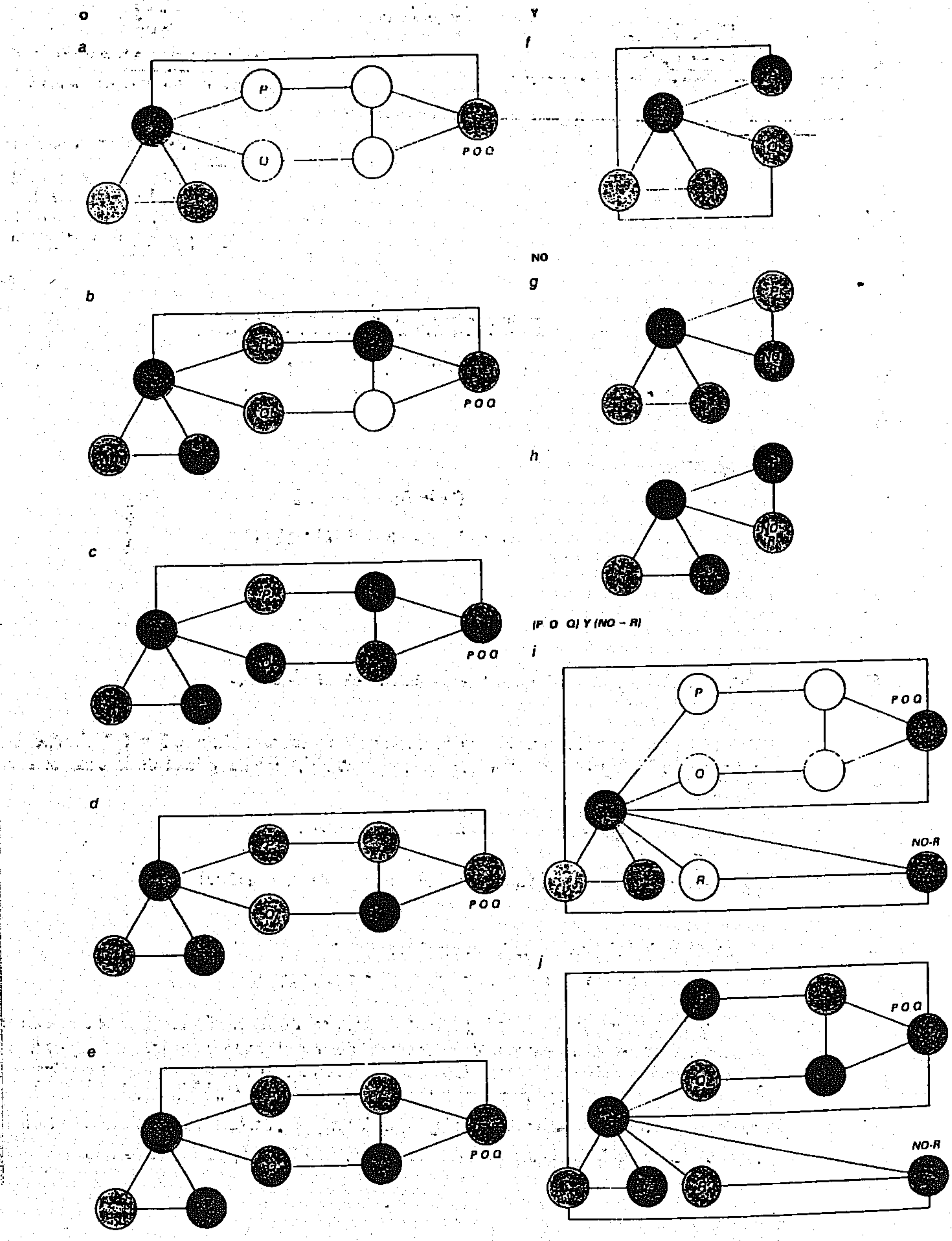
Computabilidad relativa

La idea de que una función pueda ser computada por la máquina de Turing es susceptible de generalización, haciendo que su computabilidad dependa de cadenas de símbolos que la máquina pueda encontrar en su cinta. Si una cadena detectada en la cinta pertenece a una colección A de cadenas previamente especificadas, la máquina de Turing puede ser instruida para que avance hasta un estado especial que proceda a computar la función de que se trate. Si la cadena no pertence al conjunto A, la máquina dictamina que tal función no es computable. En el primer caso, se dice que la función es computable relativamente al conjunto A. Si el conjunto A estuviera compuesto por todas las cadenas que codifican máquinas de Turing que llegan a detenerse cuando les proporcionamos como entrada una cinta en blanco, la función de Rado, $\sigma(N)$, sería computable respecto de A.

En 1974, Theodore P. Baker, [d]
Universidad de Cornell, John Gill
la Universidad de Stanford, y Ro
M. Solovay, de la de Berkeley, plan
ron si podría demostrarse la inequ
lencia de las clases P y NP para la c
putación relativa. En el transcurs
su trabajo hicieron un descubrimi
desconcertante: consiguieron exp
car dos conjuntos, A y B, para lo cu
las relaciones entre las clases P y
eran contradictorias. Dicho de
forma, para cómputos relativos
conjunto A, las clases P y NP
equivalentes, mientras que rela
mente al conjunto B, P y NP eran
ses no equivalentes. Se descubrió
más que para cualquier sistema fo
existían cálculos relativos para los
les tanto podía suponerse que P y
erán equivalentes como que no lo e
sin detrimento de la coherencia del
tema. Desde entonces, muchos o
investigadores han descubierto nu
rosos problemas que pueden ser re
vizados de modo tal que cada pos
resultado sea verdadero.

Esta paradójica situación es ob
mente incómoda. Todavía no ha p
do resolverse ninguno de los proble
relativizados de dos modos contradi
rios, hecho que suele tomarse c
prueba de que las soluciones a t
problemas se encuentran más allá
alcance actual de las matemáticas.
pero, debemos recordar que la n
formulación de estos problemas,
apariencia no tratables, ha sido po
litada por la sencilla solución que s
dio a un problema impenetrable p
una generación anterior. Tal vez, v
en retrospectiva, el próximo paso
portante llegue a parecernos tan se
llo como las máquinas creadas po
vivaz imaginación de A. M. Turing

10. SIGNIFICA LA EQUIVALENCIA DE DOS PROBLEMAS que la solución de uno cualquier
ellos proporciona inmediatamente la solución del otro. Aquí hemos transformado el problema de la d
minación de las condiciones en las que es verdadera una proposición lógica o fórmula booleana com
en otro problema, el de coloreado de los vértices de un grafo con tres colores, de modo tal que ningú
de vértices conectados por una línea sea del mismo color. La veracidad o falsedad de una fórmula bo
na cualquiera es función de la veracidad o falsedad de las proposiciones simples, o atómicas, q
componen, y de los modos en que los valores de veracidad están enlazados por las conectivas "o", '
"no". A cada posible fórmula booleana se le asocia un gráfico simple, compuesto por vértices y líne
cual puede ser coloreado con tres colores si y solamente si pueden asignarse a las proposiciones atóm
valores de verdad de modo tal que la fórmula compuesta sea verdadera. Fijémonos, por ejemplo,
fórmula compuesta P *o* Q, construida a partir de las proposiciones atómicas P y Q. La fórmula P *o*
verdadera solamente si es verdadera la proposición P, o lo es la Q, o lo son ambas. Estas condicion
reflejan en el coloreado del grafo del ángulo superior izquierdo (*a*) de la ilustración, en el cual el v
denota verdad y el azul, falso. El coloreado puede terminarse de modo tal que el vértice en pos
extrema derecha sea verde (verdadero) solamente si uno o ambos de los vértices rotulados están col
dos de verde (*b* - *e*). Análogamente, la fórmula compuesta P *y* Q es verdadera solamente si las proposicl
atómicas, P y Q, son ambas verdaderas, situación que se refleja en un grafo en el cual el vértice
representa a P y el vértice que representa a Q están ambos coloreados de verde (*f*). Finalmente, la fór
no-P es verdadera solamente si el vértice rotulado P está coloreado de azul (*g*); la fórmula P es verda
solamente si el vértice rotulado *no*-P está coloreado de azul (*h*). En el ángulo inferior derecho está el g
correspondiente a una fórmula booleana más compleja, (P *o* Q) *y* (*no*-R); el coloreado superior (*i*) re
senta las restricciones impuestas por las conectivas "o", "y" y "no". En la parte baja (*j*) vemos una d
cuatro formas de colorear el grafo, correspondiente a una forma de satisfacer la fórmula boole


o
a
P
O
P O Q
Y
f
b
P O Q
NO
g
h
c
P O Q
(P O Q) Y (NO – R)
i
P
P O Q
O
NO-R
R
d
P O Q
j
P O Q
e
P O Q
NO-R

Nº 246
0'20 €

DEPARTAMENTO DE INTELIGENCIA ARTIFICIAL

FACULTAD DE INFORMÁTICA
UNIVERSIDAD POLITÉCNICA DE MADRID

INFORMATICA TEORICA
Curso 2005 - 2006

Prácticas Tema 8
MAQUINAS DE TURING

Práctica 8.1: Construcción de Máquinas de Turing.

8.1.1.- Multiplicador unario. Dados dos números **x** e **y** sobre el alfabeto $\Sigma = \{1\}$, diseñar una máquina de Turing que los multiplique. Los contenidos inicial y final de la cinta son los siguientes:

$$\#y\#x\# \longrightarrow \#x.y\#y\#x\#$$

La cabeza se encuentra inicialmente sobre el primer blanco a la derecha del número **x**.

8.1.2.- Sea el alfabeto de la lengua castellana

$$\Sigma = \{a, b, c, \ldots, k, l, m, n, ñ, \ldots, r, s, \ldots, x, y, z\}$$

a) Construir una máquina de Turing que con la cabeza situada en el símbolo más a la izquierda de una palabra $w \in \Sigma^*$ escriba, a la izquierda de w y dejando un blanco de separación, la longitud de la palabra w, expresada como un número en sistema binario. La palabra w, al finalizar el proceso, deberá mantenerse en el lugar que ocupaba inicialmente.

b) Escribir el proceso que realiza la máquina de Turing construída para la palabra w = turing

Se considerará también la palabra vacía, que pertenece a Σ^* y cuya longitud es cero.

#carlos# → #110#carlos#

8.1.3.- a) Construir una Máquina de Turing tal que colocada en el dígito más a la izquierda de un número **w** expresado en unario ($w \in \{1\}^*$) y distinto de cero, lo copia a la derecha del número dado, sin dejar ninguna separación entre el original y la copia. Probar el funcionamiento con w=1111.

b) Construir una Máquina de Turing tal que colocada en el dígito más a la izquierda de un número natural **n** expresado en unario ($n \in \{1\}^*$) y distinto de cero, calcula $\mathbf{2^n}$, también expresado en unario, y lo coloca a la derecha del número dado **n**, dejando un blanco de separación. Probar el funcionamiento con n=3, i.e., n=111. (Indicación: 2^n en unario es igual a 2^{n-1} en unario seguido de 2^{n-1} en unario: $11111111 = 2^3 = 2^2\, 2^2 =$ 1111 1111).

(Examen Febrero 2000)

8.1.4.- Construir una máquina de Turing cuya cinta en la situación inicial contiene n números en unario, separados por asteriscos, y en la situación final deja la cinta con los mismos números en unario ordenados de menor a mayor y de izquierda a derecha y sobre el mismo espacio de la cinta en que se encontraban los números inicialmente.

8.1.5.- a) Construir una máquina de Turing que transforma un número entero expresado en sistema unario en el correspondiente número en sistema decimal. Dicha máquina, con la cabeza situada en el dígito más a la izquierda de una palabra sobre el alfabeto $\{a\}$, escribirá a la **izquierda** de este número y dejando un blanco de separación, el correspondiente número decimal (una palabra sobre el alfabeto $\{0,1,2,3,4,5,6,7,8,9\}$) . Notas: 1.- Para el número en unario se utiliza el símbolo **a** para evitar confusiones. 2.- El número unario debe mantenerse en su posición y tal como estaba al principio. (Indicación: por cada dígito del número unario que se va recorriendo se aumenta en una unidad el número decimal correspondiente).

(Examen Junio 2000)

b) Construir una máquina de Turing que realiza el mismo proceso que en el apartado a) pero colocando el número decimal a la **derecha** del número unario, dejando un blanco de separación.

c) Construir una máquina de Turing que transforme un número decimal en un número unario. Con la cabeza situada en el dígito más a la izquierda de un número decimal, destruye éste y escribe el correspondiente número en sistema unario. (Indicación: por cada dígito del número unario que se va añadiendo se resta una unidad al número decimal).

d) Diseñar una máquina que realiza la misma tarea que en el apartado c), pero dejando el número decimal tal y como estaba y en su mismo sitio y el número unario a la derecha y separado por un blanco.

8.1.6. Sea **n** un número unario tal que $\mathbf{n = 2^x}$, $\mathbf{x \geq 1}$. Construir una máquina de Turing que, con la cabeza situada inicialmente en el dígito más a la izquierda del número **n**, escriba a la derecha de **n**, y dejando una celda en blanco intermedia, el valor en unario de $\mathbf{log_2 n}$.

Indicación: Se aconseja utilizar la relación recursiva $\log_2 2n = \log_2 n + 1.\quad \log_2 2 = 1$

(Examen Febrero 2002)

8.1.7. Construir una Máquina de Turing que decida el predicado **mayor (x,y)**, donde **x** e **y** son números naturales mayores que cero, expresados en unario.

Es decir que, dados dos números **x** e **y**, escriba a continuación de ellos y dejando un blanco de separación un **1**, si **x > y**, y un **0** en caso contrario.

(Examen Junio 2005)

Práctica 8.2: Máquina de Turing Universal

8.2.1. Sea la máquina de Turing M definida por su función de transición:

	q'0	q0	q1	q2	q'1	q''1	q'2	q''2	q'''2
1	–	q1,1,I	q2,1,I	q0,1,I	–	–	–	–	–
#	q0,#,I	–	q'1,#,I	q'2,#,I	q''1,1,I	–	q''2,1,I	q'''2,1,I	–

a) ¿Qué función aritmética sobre el alfabeto {1} realiza esta MT?. Mostrar el funcionamiento de la MT cuando la situación inicial de la cinta y la posición de la cabeza de lectura-escritura son las siguientes:

...	#	1	1	1	1	1	#	...
							↑q'0	

b) Escribir las cintas inicial y final de la Máquina de Turing Universal cuando simula la máquina M con la computación correspondiente al apartado a), así como la posición de la cabeza de lectura-escritura de la MTU en ambas situaciones.

(Examen Junio 1997)

8.2.2. Las 3 líneas siguientes, leídas una a continuación de otra, forman la cinta inicial de la Máquina de Turing Universal (MTU) cuando simula una determinada Máquina de Turing M.

... # # * 1 1 1 0 0 0 0 0 0 ǂ 0 0 0 1 ǂ 0 0 0 1 0 0 1 0 0 ǂ 0 0 1 1 0 0 1 1 0 ǂ 0 0 1 0 0 1 0 0 0 ǂ ...

0 1 0 0 0 1 1 1 1 ǂ 0 1 1 0 1 0 0 0 1 ǂ 1 0 0 1 1 0 0 1 1 ǂ 1 0 0 0 0 0 0 1 0 ǂ 0 1 0 1 0 1 0 1 0 ǂ ...

0 1 1 1 0 1 1 1 1 ǂ 0 0 0 0 1 0 1 1 0 ǂ 1 0 1 1 1 0 1 1 0 ǂ 1 0 1 0 1 1 0 0 1 ǂ 1 1 0 1 1 1 0 0 0 ǂ # #

M tiene 7 estados que se han codificado de la siguiente forma:

q0	0 0 0	q4	1 0 0
q1	0 0 1	q5	1 0 1
q2	0 1 0	q6	1 1 0
q3	0 1 1		

SE PIDE:

1) ¿Cuál es el resultado que produce M sobre la entrada representada en la cinta de la MTU?

2) Cinta de la MTU cuando M se encuentra en la configuración 1 q4 # 1 1 # 1 1 y la MTU empieza a simular la transición correspondiente.

3) Cinta de la MTU al final de cada uno de los procesos localización, transcripción y simulación, cuando la MTU simula la transición del apartado 2).

(NOTA: Se sugiere que se utilice el propio enunciado -las 3 líneas de la cinta de la MTU- en la contestación del apartado 1))

(Examen Junio 2000)

Práctica 8.3: MT como reconocedor de lenguajes.

Construir sendas máquinas de Turing que reconozcan los siguientes lenguajes:

a) $\{ a^n b^{2n} \mid n > 0 \}$

b) $\{ ww \mid w \in \{a,b\}^* \}$

c) $L = \{x \in \{0,1\}^* \mid N_0(x) = 2.N_1(x) \}$ (Examen Febrero 2005)

Facultad de Informática
Universidad Politécnica de Madrid

Informática Teórica

Curso 2004-2005

Redes de Neuronas Artificiales (RNA)

Nº 249
1'47 €

jmbarreiro@fi.upm.es

Contenido

2

- ➔Introducción
- ◆ Fundamentos biológicos de las RNA
- ◆ La neurona artificial
- ◆ Componentes de una RNA
- ◆ Clases de neuronas
- ◆ Aprendizaje
- ◆ El Perceptron
- ◆ ADALINE

Redes de Neuronas Artificiales

- **Diferentes denominaciones**
  - Inteligencia Artificial Conexionista
  - "redes neuronales", "sistemas conexionistas", "sistemas neuromórficos", "sistemas de procesamiento paralelo", etc
- **Carácter multidisciplinario** → Intervienen varias ciencias.
  - neurociencias (medicina), ciencias de la computación, matemáticas, la cibernética, etc

Sistemas basados en la IA (Inteligencia Artificial)

- **Enfoque "Simbólico"**
  - Pretende simular cómo es el funcionamiento del cerebro, considerando a este como un procesador de información
    - » Heurístico o paradigma del poder
    - » Epistemológico o paradigma del conocimiento
- **Enfoque de "Emulación"**
  - Imita la estructura y el funcionamiento de los sistemas biológicos con comportamiento considerado inteligente
    - » RNA

Pilares en la evolución de las RNA

- **Tecnológicos**
  - Computadoras
- **Biológicos**
  - Sistema Nervioso
- **Computacionales**
  - Modelos Inteligentes

Pilares Tecnológicos

- **Babbage**
  - Computadora Analítica
- **Turing (1936)**
  - Colossus (descifrado de mensajes)
- **Von Newmann (1958)**
  - Computadoras Secuenciales Binarias

Pilares Biológicos

- **Golgi**
  - Identificación (Reazione Nera)
- **Cajal**
  - "Sinapsis" o unidades funcionales (Teoría de la Neurona) → Todas las funciones se realizan en la sinapsis.
  - Transmisión de la corriente nerviosa unidireccional (Teoría de la Polarización Dinámica)
  - Evolución de los axones estableciendo nuevas sinapsis (Teoría Neurotrópica) = nuevas conexiones.

Pilares Computacionales

- **McCulloch y Pitts (1943)**
  - Redes sencillas binarias
- **Hebb (1949)**
  - Regla de Aprendizaje de Hebb (variación en las "sinapsis")
- **Widrow y Hoff (1959)**
  - Regla Delta: ADALINE (Neurona Adaptativa Lineal) y MADALINE
- **Rosenblatt (1962)**
  - Perceptron: máquina adaptativa capaz de reconocer patrones

Declive (estancamiento)

- Minsky y Papert "Perceptrons" (1969)
- Las investigaciones se paralizan durante 15 años
  - No hay nuevos algoritmos de aprendizaje
  - No se soluciona el problema del Perceptron
  - La comunidad científica se vuelca en el enfoque simbólico

Resurgimiento

- Retropropagación del gradiente
  - Algoritmo de aprendizaje
- Hopfield y Grossberg (1982)
  - "Memoria Lineal Asociativa" y Teoría de la Resonancia Adaptativa (ART)
- Carpenter y Kohonen (1988)
  - "Feature Map"
- Daniel Hillis
  - "Máquina Conexionista"
- Rumelhart, McClelland & Group
  - Editan los famosos libros PDP (Parallel Distributed Processing)

Características de las RNA

- Capacidad para aprender automáticamente
- Capacidad de filtrar ruidos: Generalización → Le podemos dar información ruidosa, que sacará la respuesta correcta
- Sistemas tolerantes a fallos → Si falla una neurona puede seguir funcionando bien.
- Posibilidad de construir computadores paralelos
- Similitud con modelos neurofisiológicos del cerebro

Fundamentos biológicos de las RNA

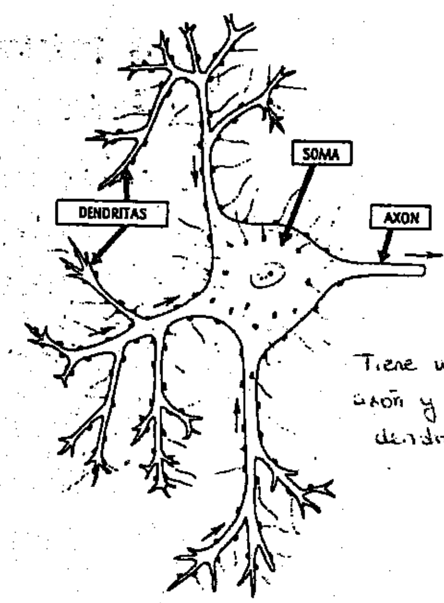
- ◆ **Órganos Receptores**
  - Recogen la información del exterior
- ◆ **Sistema Nervioso (SNC)**
  - Procesa la información
- ◆ **Órganos Efectores**
  - Ejecutan las respuestas

La Neurona

- **Axón o neurina termina en las Sinapsis o "botones terminales"**
- **Soma o pericarion "cuerpo celular"**
- **Dendritas "prolongaciones"**

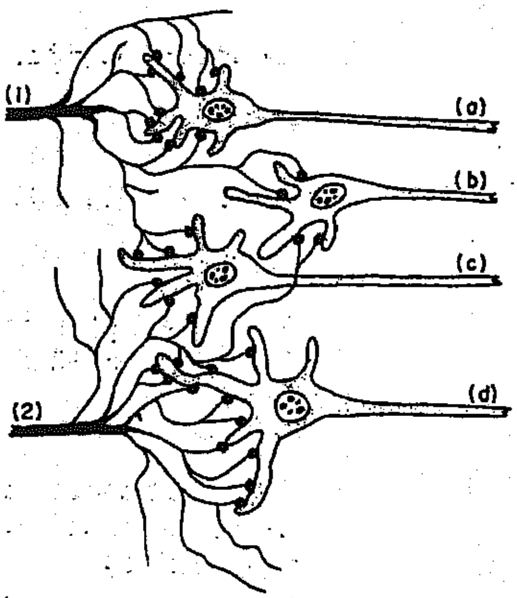
Son multipolares:

Poseen axón y dendritas



Tiene un solo axón y muchas dendritas.

La Neurona (2)



Prolongaciones que forman REDES (elaboran y almacenan información)

Su estructura es la RESULTANTE de las exigencias generales y especiales de su función

UNA Y SÓLO UNA rama (axón) se encarga de la conducción de los impulsos

La información entra por los dendritas y sale por el axón. El flujo de información siempre es en esta dirección.

La Neurona (3)

♦ Con la información que llega:

- La integra en un código de activación propio
- La transmiten codificada a través de su axón
- los impulsos pasan a las neuronas subsiguientes

♦ Desde un punto de vista funcional:

- Superficie receptora (región generadora)
- Trayecto de conducción (distribución)
- Zona de transmisión

Potenciales de neurona

Potencial de reposo:

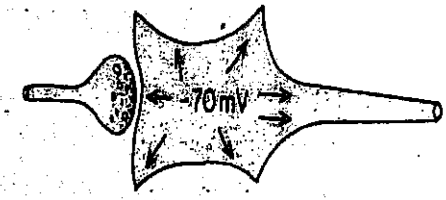
-70 mv

Se invierte hasta +35 mv (en sólo 0,0001 seg.)

Potencial de acción:

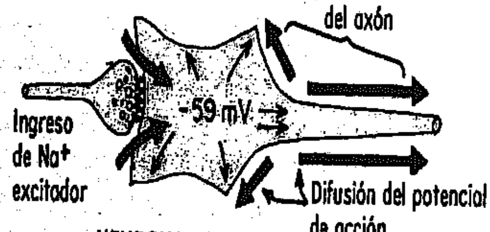
Despolarización sucesiva de la membrana del nervio (suma algebraica)

A



NEURONA EN REPOSO

B



NEURONA EXCITADA

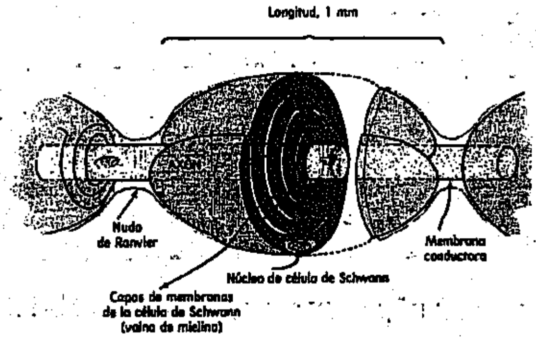
Neurotransmisión: Capacidad del SN

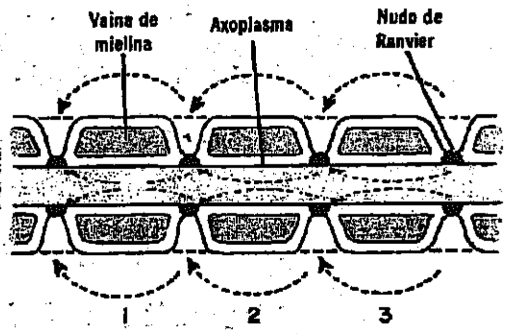
- **Flujo de corriente unidireccional**
- **Velocidad de conducción intracelular e intercelular**
- **Aislamiento de la membrana**
- **Mensajeros intercelulares diferentes y especializados**
  - **La rápida y precisa transmisión y desaparición de la señal**
  - **Los cambios de instalación lenta y de larga duración en la recepción postsináptica**

Neurotransmisión: Fibras mielínicas

Poseen axón central y vaina de mielina que lo rodea, con saltos o interrupciones (Nódulos de Ranvier)

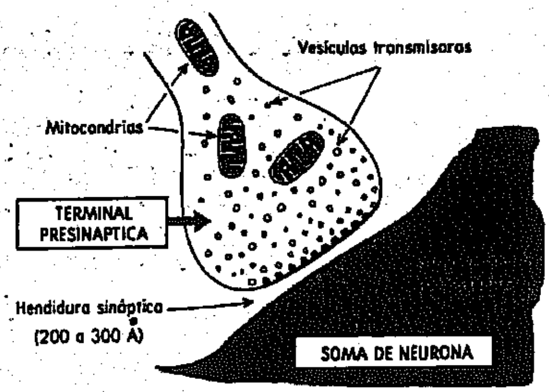
Conducción saltatoria, que ↑ su velocidad 5-7 veces.





Sinapsis: Anatomía

- Elemento presináptico
- Elemento postsináptico
- Espacio intersináptico



Potencial de acción:

- La membrana presináptica se *despolariza*
- *Vesículas sinápticas* se acercan a la membrana y *liberan su contenido*
- *Transmiten* la señal a la membrana postsináptica

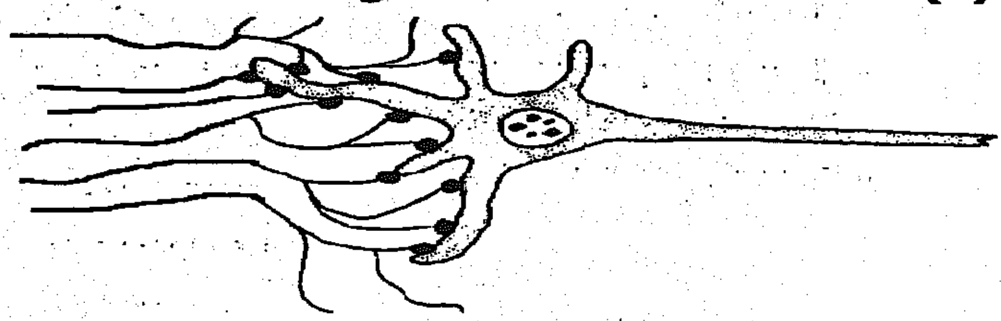
Sinapsis: Tipos

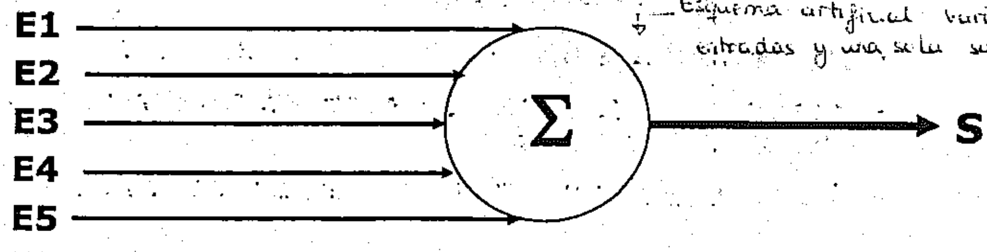
- **Química: Unidireccional (*neurotransmisores*)**
- **Eléctrica: Bidireccionales (*conductos directos*)**

PREDOMINIO (químicas):

- ***Cambio de codificación* (E-Q-E)** → Eléctrico – Químico – Eléctrico.
- **Su *superficie de contacto* (> 10.000 contactos)** ↳ contactos que puede establecer una neurona, con su vecina.
- **El *efecto válvula*** ⇒ Flujo en un único sentido.
- **La *plasticidad*** → capacidad para ir creando nuevas conexiones. Aprendizaje.
- **Permite la *intervención externa*** ~ Medicinas.

Neurona Biológica *versus* Artificial (I)





↳ Esquema artificial varias entradas y una sola salida.

Fundamentos biológicos 23

Biológica	Artificial
Conexiones aferentes	Entradas (inputs)
Potenciales sinápticos	Activación
Umbral de excitación	Umbral o sesgo
Suma de potenciales	Función de transferencia
Mediadores sinápticos	Pesos de conexión
Plasticidad	Aprendizaje

Superado este umbral transmite información.

Sustancias químicas..

Contenido 24

- ◆ Introducción
- ◆ Fundamentos biológicos de las RNA
- ➔ La neurona artificial
- ◆ Componentes de una RNA
- ◆ Clases de neuronas
- ◆ Aprendizaje
- ◆ El Perceptron
- ◆ ADALINE

Definiciones previas

Neurona: Unidad básica de la Red. Recibe múltiple información, la elabora y emite una única respuesta

Sinapsis: Conexión de una neurona con otras. Son unidireccionales: inhibidoras y excitadoras → Negatividad / Positividad.

Peso sináptico: Noción biológica de fuerza de unión de una sinapsis

Entrada (Input Total): Se denomina NET_i (t)

Activación: Grado o nivel de excitación de una neurona A(t)

Respuesta neuronal: Señal que emite una neurona

Aprendizaje: Proceso mediante el cual las neuronas ajustan sus respuestas para que el comportamiento de la red sea el esperado (PESOS SINÁPTICOS)

Ajusta los pesos para el aprendizaje.

Neurona Artificial

Máquina neuronal que elabora la información de ENTRADA para obtener una salida o RESPUESTA.

Se organizan en REDES.

Clasificación:

- **Redes alimentadas hacia delante**
- **Redes con retroalimentación**

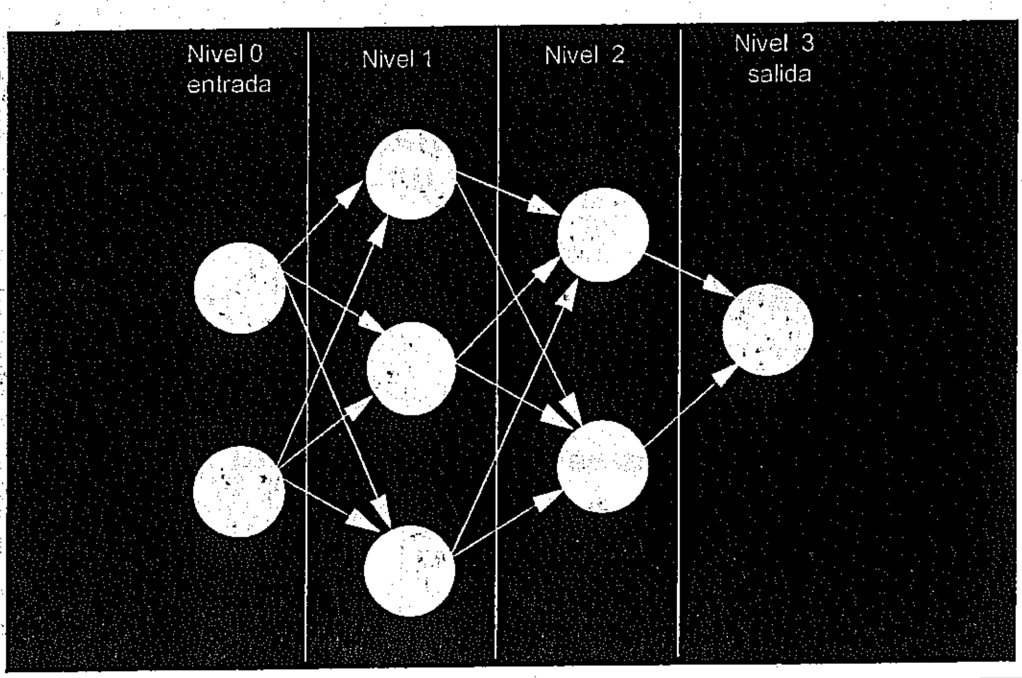
Redes alimentadas hacia delante

- La información va en un solo sentido: ATRÁS HACIA ADELANTE
- Organizadas en CAPAS:
  - Cada capa agrupa neuronas que reciben sinapsis de neuronas de la capa anterior y emiten salidas a la siguiente capa
  - Entre las neuronas de una misma capa no hay sinapsis
  - Existe al menos una capa de entrada y una de salida
  - Existen una o más capas intermedias
- Cada capa CALCULA EN PARALELO su estado de activación pasa su respuesta a la siguiente, que hará lo mismo
- Son REDES RÁPIDAS en sus cálculos

O Neuronas

↗ Conexiones

Se numeran de la capa de entrada a la capa de salida.

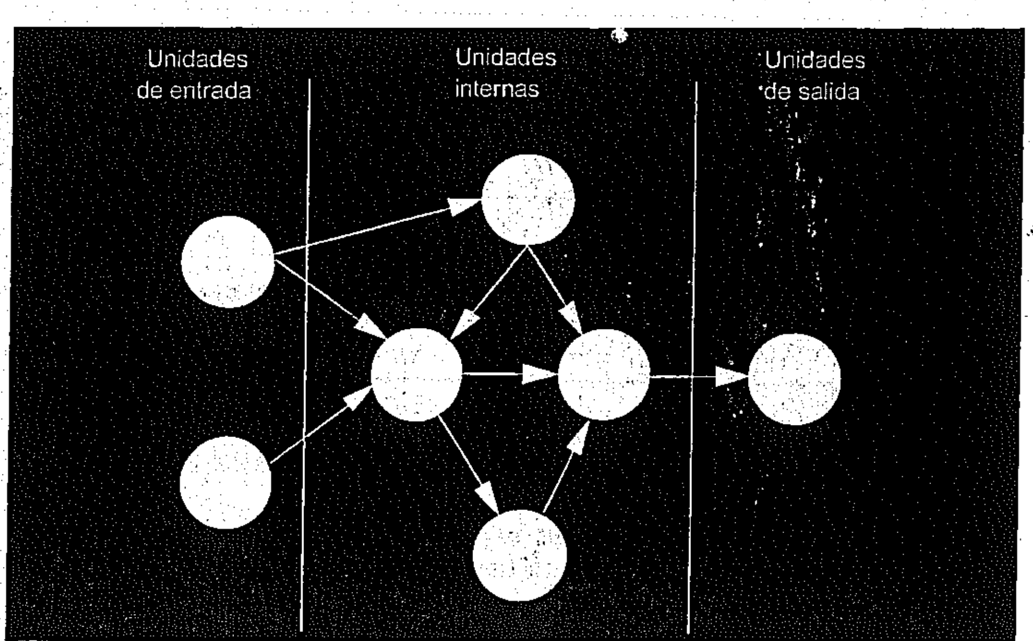


La Neurona Artificial 29

Redes con retroalimentación

- NO HAY CAPAS: la neurona está conectada a las demás
- CALCULAN Y RECALCULAN su estado varias veces
  - Una neurona recibe una señal, se activa y emite una respuesta que actua sobre otra(s) neurona(s), que podrán influir sobre la neurona inicial, cambiando su estado de activación
  - Estado Estable = no cambios en salida de ninguna neurona
  - Entradas serán también constantes (ESTADO GLOBAL ESTABLE)
- NO SE SABE CUÁNTO tardará en alcanzar un estado estable
- EMULAN MÁS FIELMENTE la estructura del cerebro humano donde la retroalimentación es fundamental

La Neurona Artificial 30



Sale la salida cuando es estable.

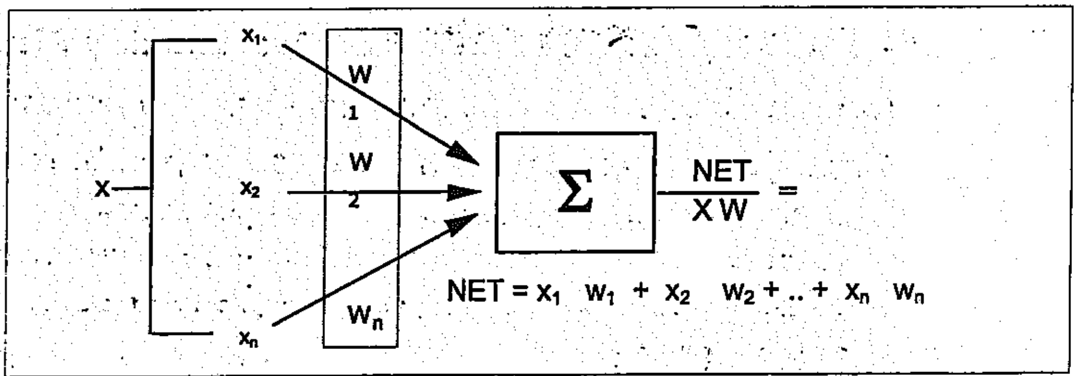
Neurona artificial, célula o autómata:

- **Elemento que posee un estado interno:**
  - S = el conjunto de estados posibles:

  Ejemplos:

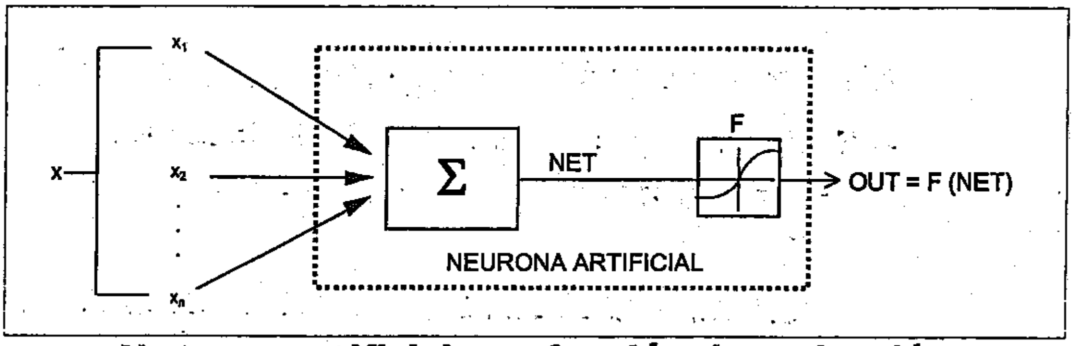
S = {0, 1}	*0 estado inactivo* *1 estado activo*
S' = {0, 1, 2 ..., n}	*imagen con n+1 niveles de gris*
S'' = [0 1]	*intervalo continuo de valores*

- **Recibe señales que le permiten cambiar de estado:**
  - Función de transición de estado o función de activación F



- **Vector X de Entradas: $x_1, x_2, \dots, x_n$**
  Señales de Entrada a la red
- **Vector W de Pesos: $w_1, w_2, \dots, w_n$**
  Fuerza de conexión sináptica

NET = SUMA (SEÑALES × PESOS asociados) $= \Sigma x_i w_i = X\ W$



Neurona artificial con función de activación

Funciones de activación

- Lineal: F (NET) = k NET, con k = cte.
- Umbral: F (NET) = 1 si NET $\geq \theta$, con θ = cte.
  F (NET) = 0 en otro caso
- Cualquier otra función (S, TH)

1er Modelo de Neurona: McCulloch y Pitts (1943)

- Modelo simplificado de las neuronas del cerebro
- Dispositivos binarios (sólo 2 estados posibles) → 0 : Inactivo; → 1 : Activo
  - m ENTRADAS ($m \geq 1$) $\{x_1, x_2, \ldots, x_m\}$ y una sola SALIDA Y
  - ASOCIADAS a m + 1 valores, de los cuales uno era su UMBRAL θ
  - m valores restantes: PESOS $\{w_1, w_2, \ldots, w_m\}$, asociados a las entradas x_i

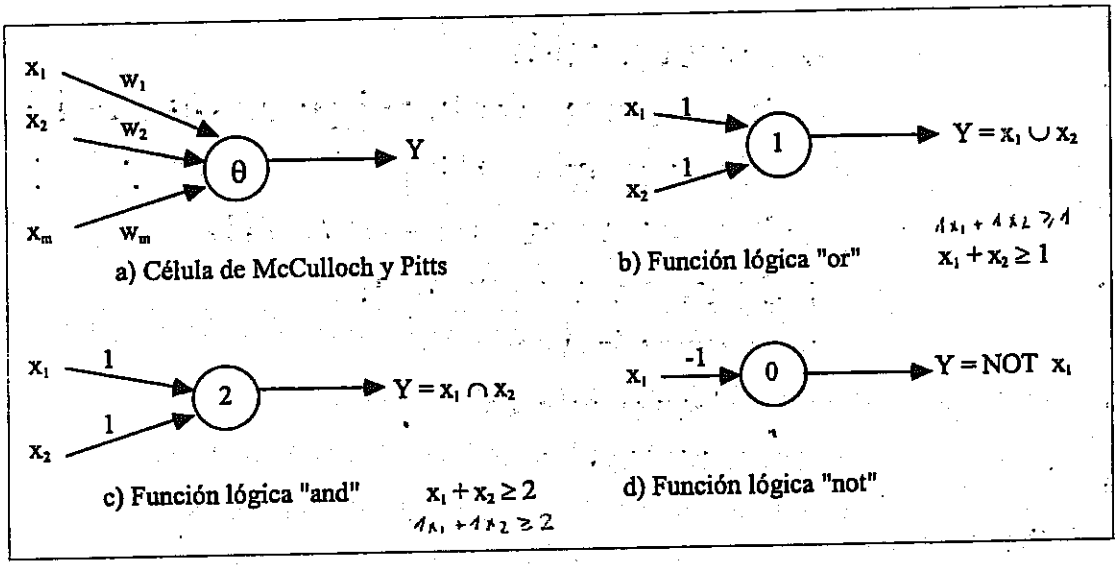
FUNCIONAMIENTO:

NET => Suma ponderada de los pesos de las entradas

- OUT (t+1) = 1 si y solo si $\Sigma_i\, w_i\, x_i(t) \geq \theta$ ó $\Sigma_i\, w_i\, x_i(t) - \theta \geq 0$
- OUT (t+1) = 0 en caso contrario

OUT => Resultado de aplicar la función de activación F a NET.

  - $w_i > 0$ = sinapsis excitadoras
  - $w_i < 0$ = sinapsis inhibidoras



Ejemplos de células de McCulloch y Pitts para las funciones lógicas AND, OR y NOT

Primer modelo de Red de Neuronas Artificial

- **Colección de neuronas de McCulloch y Pitts**
- **Las salidas están conectadas a entradas de otras neuronas**
  - **Una salida puede actuar sobe varias entradas**
  - **Una entrada viene a lo sumo de una salida**
- **La red contacta con el exterior a través de líneas de entrada y de salida**
  - **Las líneas de entrada forman parte de la entrada de una o de todas las neuronas de la red**
  - **Las líneas de salida proceden de una o todas las neuronas de la red**

Contenido 37

- Introducción
- Fundamentos biológicos de las RNA
- La neurona artificial
- →Componentes de una RNA
- Clases de neuronas
- Aprendizaje
- El Perceptron
- ADALINE

Componentes de una RNA 38

- Conjunto de unidades de proceso
- Estado de activación
- Función de salida para cada unidad
- Patrón de conexiones
- Regla de propagación
- Función de activación
- Regla de aprendizaje

Unidades de Proceso

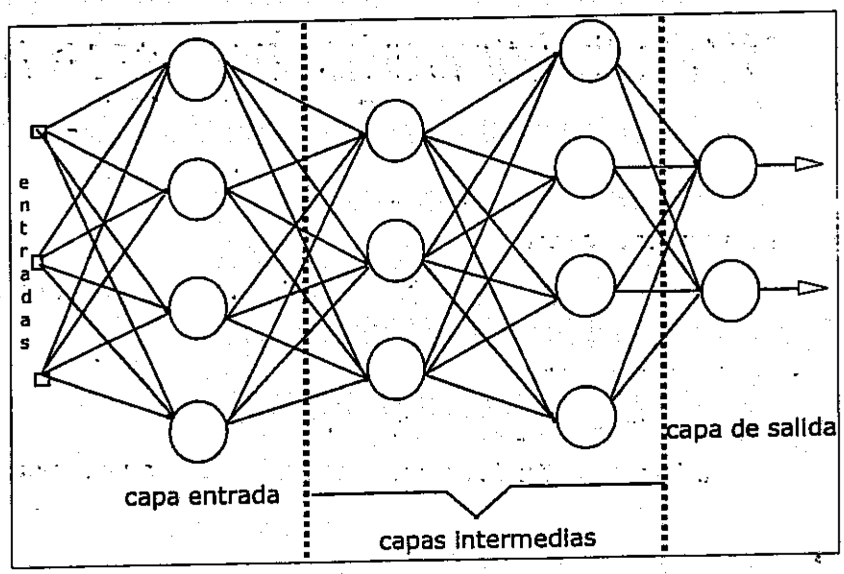
- Forman un Sistema de Procesamiento Distribuido
- Con n unidades, se designa la i-ésima como U_i

TIPOS:

Unidades de entrada	→	reciben señales desde el entorno
Unidades de salida	→	envían la señal fuera del sistema
Unidades ocultas (unidades intermedias)	→	sus entradas y salidas se encuentran dentro del sistema

Unidades de Proceso (2)

CONECTIVIDAD TOTAL: Cada neurona se comunica con todas las siguientes. →



Estado de Activación

Son los estados de cada unidad en el tiempo t.
Se especifica por un vector a(t)

$$\boxed{a(t) = (a_1(t), a_2(t), \ldots\ldots a_i(t), \ldots\ldots, a_n(t))}$$

$a_i(t)$ = activación de la neurona i.

Conjunto de estados de activación (S) que tiene cada unidad:

- **Finito:** $S = \{0,1\}$ (NEURONAS DIGITALES)
  - estado 0 = desactivación
  - estado 1 = activación
- **Infinito:** $S = [0,1]$ (NEURONAS ANALÓGICAS)
  - es continuo y coincide con un intervalo de activación

Función de Salida

Función asociada a cada unidad $f_i(a_i(t))$. Transforma el estado actual de activación $a_i(t)$ en señal de salida $O_i(t)$

$$\boxed{O_i = f_i(a_i(t))}$$

En notación vectorial:

$$O(t) = (f_1(a_1(t)), f_2(a_2(t)) \ldots\ldots f_n(a_n(t)))$$

Si la función f es la función identidad, f(x) = x

$$O(t) = (a_1(t), a_2(t) \ldots\ldots a_n(t))$$

La Función de Salida f_i no tiene que ser necesariamente distintas

Patrón de Conexiones

Arquitectura de la red (grafo orientado)

- Nodos constituyen las *unidades de procesamiento*
- Arcos orientados son los *canales de comunicación* entre unidades
- Cada arco se asocia con un valor: *peso de la conexión*

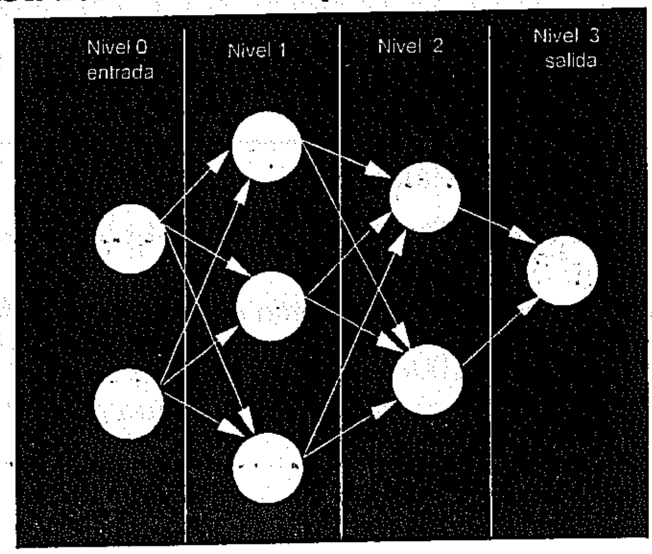
Se representa por una matriz W (n filas y m columnas)

- El elemento i,j (w_{ij}) es el peso de conexión entre la unidad u_j y la u_i
  - » Conexiones inhibitorias (pesos negativos) = W_i → $w_{ij} < 0$
  - » Conexiones excitatorias (pesos positivos) = W_e → $w_{ij} > 0$

Patrón de Conexiones: esquema jerárquico

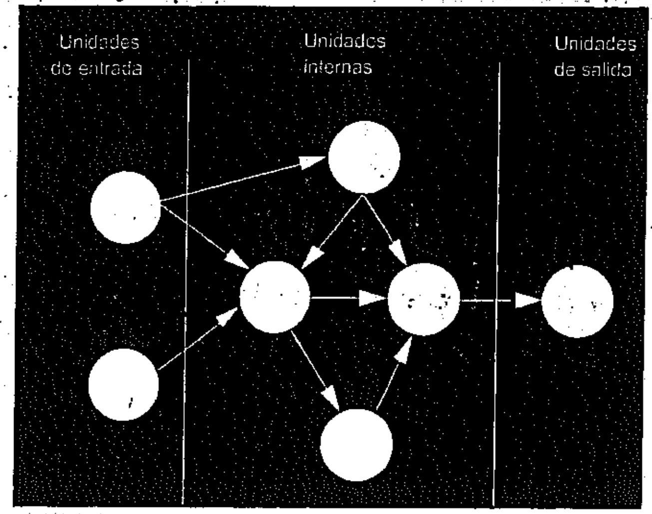
Unidades estructuradas en capas con un esquema *bottom-up*

↳ Hace referencia a cómo fluye la información: desde capa 0 a la salida



Patrón de Conexiones: esquema no jerárquico

No existe ninguna limitación en los esquemas de conexión entre células



Regla de Propagación

Determina los inputs totales Net_i para cada tipo de conexión entrante en una unidad.

Dado el vector de salida O(t) y las matrices de conexiones

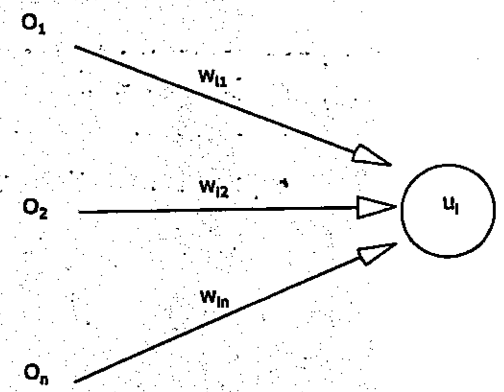
- Denominando $Net_{ij}(t)$ la entrada total de tipo i que llega a la unidad u_j en el instante t
- La regla de propagación más usada: se multiplican las señales de salida (O_j) que pasan por un canal por el peso a él asociado (W_{ij})

$$\boxed{Net_i = \sum_j W_{ij} O_j}$$

Regla de Propagación (2)

Si hay dos matrices de conexión, una para la conexión inhibitoria (W_I) y otra para la excitadora (W_e), habrá dos vectores de input $Net_i(t)$ y $Net_e(t)$:

- $Net_i(t) = W_I\, O(t)$
- $Net_e(t) = W_e\, O(t)$



Función de Activación

Dado el estado de activación $a_i(t)$ de la unidad u_i y los input que llegan a ella, $Net_{1i}(t)$, $Net_{2i}(t)$ ..., el estado de activación siguiente $a_i(t+1)$ se obtiene aplicando una función F_i llamada de activación o transición:

$$\boxed{a_{i\,(t+1)} = F_i\,(a_i(t), Net_{1i}(t), Net_{2i}(t) \ldots\ldots\ Net_{ni}(t))}$$

- **En notación vectorial:**

  $a_{(t+1)} = F\,(a(t), Net_1, Net_2, \ldots Net_n)$

  F = vector de las funciones de activación F_i, $F = (F_1, F_2, \ldots, F_n)$

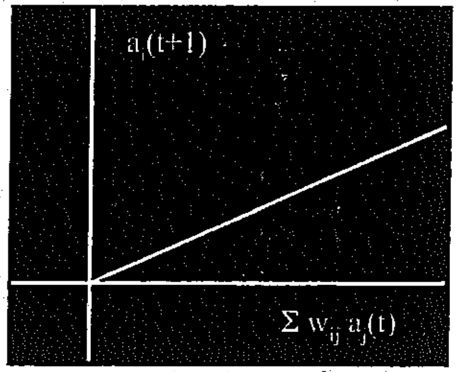
Función de Activación: Tipos

a) Lineal

- Conjunto de estados S puede contener cualquier número real
- Estado de activación coincide con el input total (Net) de la unidad:

$$\boxed{a_i(t+1) = \sum w_{ij}\, a_j(t)}$$

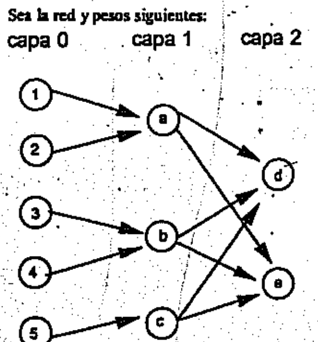
$$a(t+1) = W\, a(t)$$



Función de activación lineal: conjunto de unidades de input conectadas directamente al output
Más capas intermedias: no aumenta la potencia computacional (si esquema de conexión es *bottom-up*)

Ejemplo:

Sea la red y pesos siguientes:



$w_{a1} = 2$ $w_{da} = -1$
$w_{ea} = -2$
$w_{a2} = -1$ $w_{db} = 2$
$w_{b3} = 3$ $w_{eb} = 1$
$w_{b4} = -1$ $w_{dc} = -1$
$w_{c5} = 2$ $w_{ec} = 3$

Las matrices de las conexiones serán:

Fila: conexiones de → la neurona ⓐ.

$$w_1 = \begin{bmatrix} 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 3 & -1 & 0 \\ 0 & 0 & 0 & 0 & 2 \end{bmatrix};\quad w_2 = \begin{bmatrix} -1 & 2 & -1 \\ -2 & 1 & 3 \end{bmatrix}$$

→ conexiones de la neurona ⓔ.

vector de entrada → lo dan. Sea el vector de entrada [1 0 2 1 2] las activaciones de abc serán: → capa 1

$$\begin{bmatrix} 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 3 & -1 & 0 \\ 0 & 0 & 0 & 0 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 2 \\ 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 2 \\ 5 \\ 4 \end{bmatrix} \begin{matrix} \to a \\ \to b \\ \to c \end{matrix}$$

y las de d y e: → capa 2

$$\begin{bmatrix} -1 & 2 & -1 \\ -2 & 1 & 3 \end{bmatrix} \begin{bmatrix} 2 \\ 5 \\ 4 \end{bmatrix} = \begin{bmatrix} 4 \\ 13 \end{bmatrix} \begin{matrix} \to d \\ \to e \end{matrix}$$

que es equivalente a la utilización de una sola capa con matriz de conexiones:

$$w = w_2 \cdot w_1 = \begin{bmatrix} -1 & 2 & -1 \\ -2 & 1 & 3 \end{bmatrix} \begin{bmatrix} 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 3 & -1 & 0 \\ 0 & 0 & 0 & 0 & 2 \end{bmatrix} = \begin{bmatrix} -2 & 1 & 6 & -2 & -2 \\ -4 & 2 & 3 & -1 & 6 \end{bmatrix}$$

con lo que la activación de las células d y e será:

$$\begin{bmatrix} -2 & 1 & 6 & -2 & -2 \\ -4 & 2 & 3 & -1 & 6 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 2 \\ 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 4 \\ 13 \end{bmatrix}$$

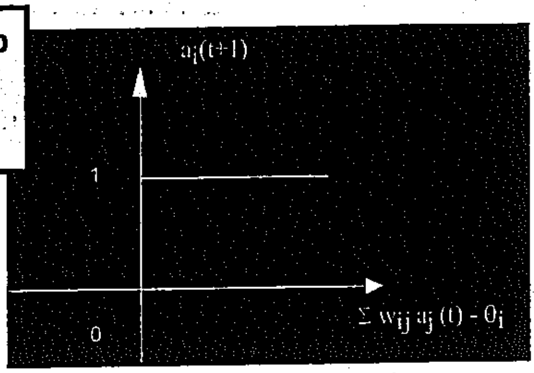
es decir la misma calculada anteriormente.

Función de Activación: Tipos

b) No lineal con función de activación binaria a umbral (1)

- El conjunto de los estados de activación S es binario y cada unidad u_i está caracterizada por un umbral θ_i (número real)

$$a_i(t+1) = \begin{cases} 1 & \text{si } [(\Sigma w_{ij} a_j(t)) - \theta_i] \geq 0 \\ 0 & \text{en caso contrario} \end{cases}$$



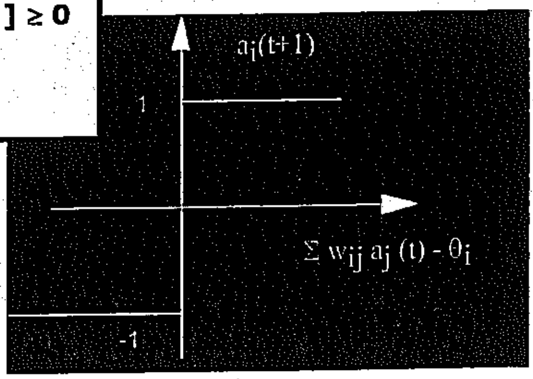
- Cualquiera que sea S = {0,1}

Función de Activación: Tipos

b) No lineal con función de activación binaria a umbral (2)

- Si S = {-1, 1} tendremos que:

$$a_i(t+1) = \begin{cases} 1 & \text{si } [(\Sigma w_{ij} a_j(t)) - \theta_i] \geq 0 \\ -1 & \text{en caso contrario} \end{cases}$$

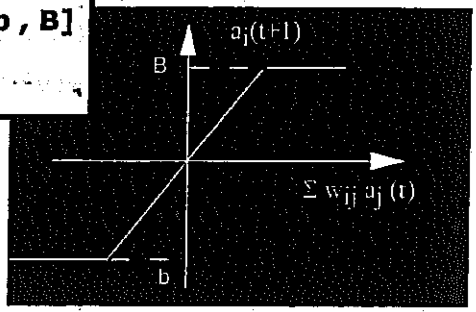


Función de Activación: Tipos

c) No lineal con función de activación a saturación

- El conjunto S es un intervalo del eje real, por ejemplo $S = [b,B]$
- Llamando $net_i = \Sigma w_{ij} a_j(t)$, la activación de la unidad es:

$$a_i(t+1) = \begin{cases} b & \text{si } net_i \le b \\ net_i & \text{si } net_i \text{ pertenece a } [b, B] \\ B & \text{si } net_i \ge B \end{cases}$$



Función de Activación: Tipos

d) No lineal con función de activación continua

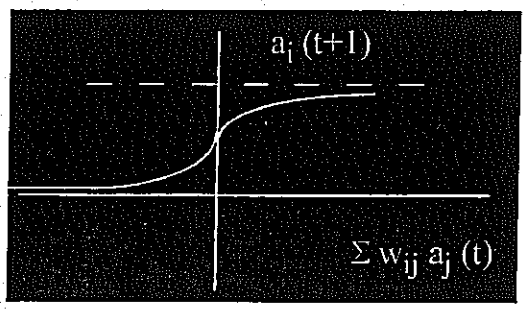
- El espacio de los estados de activación es un intervalo del eje real

Aplicar F al input total. → $\boxed{a_i(t+1) = F_i(\Sigma w_{ij} a_j(t))}$ siendo F_i una función continua

Se denomina "semilineal" si es no decreciente y diferenciable

Ejemplo: la función sigmoide

(Son las mas utilizadas)



Modificación de patrones de conectividad: Regla de aprendizaje

- **3 métodos:**
  - **Desarrollar nuevas conexiones** → Redes constructivas.
  - **Eliminación de conexiones existentes** → Se parte de una estructura de red y se eliminan las neuronas no necesarias.
  - **Modificación de intensidades en las conexiones existentes**
- **La RNA se caracteriza por su entrenabilidad. 3 tipos:**
  - **Supervisado: Utiliza un "profesor" externo al sistema** → Ajusta los pesos hasta que coincide con la respuesta correcta
  - **No supervisado: Se presentan a la red patrones de entrada** ← Ajusta los pesos hasta que son válidos para todas las entradas.
  - **Auto-supervisado: La red realiza auto-monitorización interna**

Regla de Hebb

Si una unidad u_i, recibe señal de otra u_j y si ambas están activas, entonces, el peso w_{ij} de sus conexiones aumenta

Regla de Hebb sin instructor (Aprendizaje ~~no~~ supervisado) no supervisado

$$\boxed{\Delta w_{ij} = \mu\, a_i(t)\, O_j}$$

Donde: μ = Tasa de Aprendizaje

$a_i(t)$ = activación actual

O_j = señal (entrada) que recibe de la unidad j

Componentes de una RNA

Regla de Widrow-Hoff (regla delta)

El aprendizaje es proporcional a la diferencia (DELTA) entre la activación actual $a_i(t)$ y la del patrón de supervisión $t_i(t)$

↳ con lo que comparamos (es la respuesta correcta).

(Aprendizaje supervisado).

$$\boxed{\Delta w_{ij} = \mu\,(t_i(t) - a_i(t))\,O_j(t)}$$

Esta es la generalización de la regla de aprendizaje del Perceptron para la cual se probó el teorema de convergencia

Contenido

- Introducción
- Fundamentos biológicos de las RNA
- La neurona artificial
- Componentes de una RNA
- →Clases de neuronas
- Aprendizaje
- El Perceptron
- ADALINE

Neuronas: Clases

- **Existe VARIEDAD DE FUNCIONES de activación y de transferencia que pueden implementarse en una red**
- **Criterio de clasificación (2 grandes grupos): FUNCIONES LINEALES y NO LINEALES**
- **La linealidad o no linealidad de las funciones definen el COMPORTAMIENTO de la neurona**
  - **Funciones lineales ⇒ Neuronas lineales**
  - **No lineales ⇒ Neuronas no lineales**

Neuronas Lineales

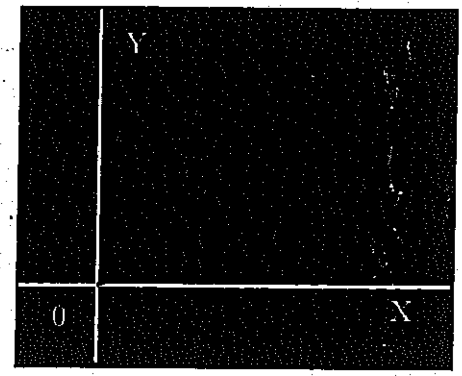
- **Su salida O_i es linealmente dependiente de sus entradas O_j**
- **Tienen funciones de activación y de transferencia lineales por lo que su composición da lugar a otra función lineal que regirá las respuestas**
- **La respuesta no está acotada y podrá tomar cualquier valor**
- **Primeras redes. Tienen 2 grandes problemas:**
  1. **Fluctuación en las respuestas**
  2. **Falta de adecuación simultánea a señales grandes y pequeñas**

Neuronas Lineales (2)

- Las salidas pueden tomar cualquier valor
- La función de activación: Lineal

$$\boxed{F(net) = K\ net}$$

Donde K es constante



Neuronas No Lineales

- En éstas, la función de activación, la de transferencia o ambas, son funciones no lineales y también su composición
- Producen respuestas acotadas
  - No problemas de fluctuación de la respuesta
  - Adecuación a señales grandes o pequeñas
- Con función sigmoide o tangente-hiperbólica tienen límites de saturación superior e inferior
  - Las respuestas sólo varían entre los dos niveles de saturación
  - Si alcanza uno de los límites pequeñas variaciones no alteran la respuesta
- Dos tipos: Digitales y analógicas

Neuronas Digitales

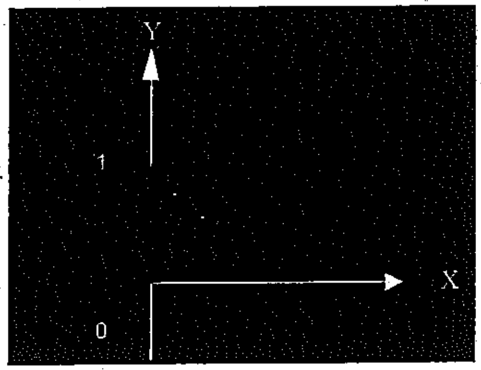
- **Las salidas son discretas:**

  Generalmente: (0 y 1) ó (-1 y +1)

- **La función de activación: Tipo Umbral**

$$F(net) = \begin{cases} 1 & \text{si } net \geq \theta \\ 0 & \text{si } net < \theta \end{cases}$$

Donde θ es el umbral



Neuronas Analógicas

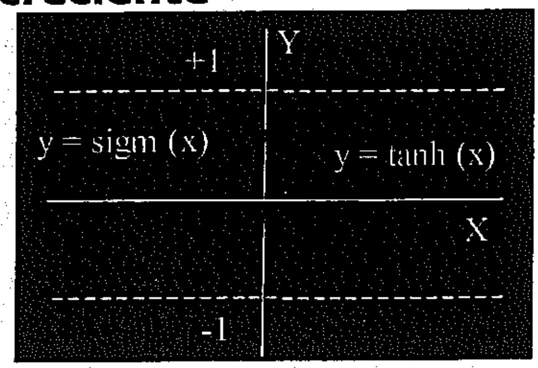
- **Las salidas son continuas:**

  Generalmente con valores en un intervalo: [0 , 1] ó [-1 , +1]

- **La función de activación: Continua diferenciable y no decreciente**

F(net) = sigm (net)

F(net) = tanh (net)



Contenido

- ◆ Introducción
- ◆ Fundamentos biológicos de las RNA
- ◆ La neurona artificial
- ◆ Componentes de una RNA
- ◆ Clases de neuronas
- →Aprendizaje
- ◆ El Perceptron
- ◆ ADALINE

Aprendizaje

Aprendizaje: Definición en RNA

- ◆ Proceso en el que la red va modificando sus respuestas ante sus entradas para irse adaptando hacia un funcionamiento considerado correcto
  - » Ajuste de Pesos de conexión
  - » Algoritmo de aprendizaje

Aprendizaje

Tipos de Aprendizaje

- **Aprendizaje supervisado**
  - Autoasociación
  - Asociación de patrones
  - Clasificación
- **Aprendizaje no supervisado**
  - Reconocimiento de regularidades

✓ **Ejecución: Generalización**

Aprendizaje

Aprendizaje supervisado

- En éste el entrenamiento consiste en presentar de forma repetida a la red patrones de entrada: juego de ensayo (JE)
  - JE son parejas de patrón de estímulos-respuesta correcta
  - En el JE debe de estar representada toda la información
  - La respuesta a cada patrón se compara con respuesta correcta
  - En virtud de ello se reajustan los pesos
  - Cuando se termina con el último patrón del JE, se vuelve a empezar con el primero
- Los pesos se estabilizan en torno a valores óptimos
- Algoritmo converge: red entrenada (finaliza fase aprendizaje)

Aprendizaje supervisado (2)

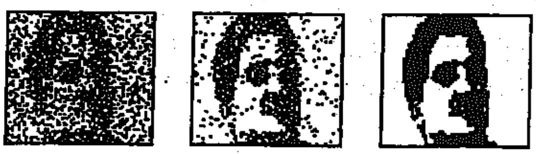
Autoasociación

- El objeto del aprendizaje es la memorización del patrón
- En el entrenamiento al presentar parte del patrón se reconstruye lo que falta

Aprendizaje supervisado (3)

Asociación de patrones

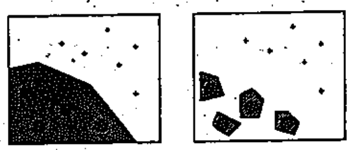
- El objeto del aprendizaje es la memorización de la copia del patrón
- Para patrón de *input* presentado, se enseña el *output* correspondiente
- Al final al presentar un patrón devolverá el *output* que ha aprendido



Aprendizaje supervisado (4)

Clasificación

- El objeto de aprendizaje es la clasificación del input en categorías prefijadas



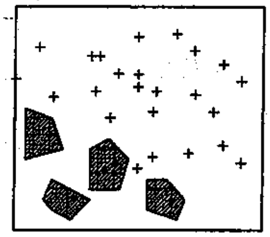
Aprendizaje no supervisado

- No se le especifica a la red cuál debe ser la respuesta correcta
  - No hay una comparación entre la respuesta de la red y la deseada
  - No existe influencia externa a la red, no se le informa si un resultado es correcto o no
- Hay que suministrar grandes cantidades de datos para que la red realice asociaciones
- Necesita muchos más patrones en el entrenamiento para que la red ajuste correctamente sus pesos

Aprendizaje no supervisado (2)

Reconocimiento de regularidades

- No hay categorías prefijadas y no se supervisa con un enseñante
- Aprenden a responder a patrones en la entrada
- Es la base para detectores de cualidades o características



Ejecución: Generalización

- El esquema de fase de entrenamiento y fase de ejecución es común en las redes con aprendizaje supervisado
- En la fase de ejecución la red responderá a estímulos diferentes a los de entrenamiento
- La red deberá generalizar y ante entradas similares a las del JE dará salidas correctas
- El proceso con una red entrenada, es el mismo que en el entrenamiento, sólo que ahora no se realizará ningún ajuste en los pesos
  - La red RECONOCERÁ O EVALUARÁ y dará una respuesta
  - Ahora no conoce respuesta correcta y NO PUEDE COMPARARLA

Aprendizaje: Métodos de ajuste de RNA

- **Métodos iterativos**
  - Deterministas
  - Aleatorios
- **No iterativos**
  - Lineales
  - Heurísticos
- **Constructivos**

Métodos iterativos

- **Error en la red para cada patrón de entrada**

  $$Ep = \frac{1}{2} \sum (Dip - Sip)^2$$

  Dip: Salida deseada de la i-ésima unidad de respuesta

  Sip: Salida real de la i-ésima unidad de respuesta
- **Error total de la red**

  $$E = \sum Ep$$

  - El objetivo de métodos de ajuste es hacer mínimo el error de la red
  - Se considera dicho error como una función de los pesos
  - Se aplica cualquier método conocido de optimización de funciones

Algoritmo general

1. **Asignar valores iniciales aleatorios a los pesos sinápticos**
2. **Repetir los siguientes pasos hasta que el error total sea inferior a cierta cantidad prefijada**
   - 2.1. Para todos los patrones de entrada hacer lo siguiente
     - 2.1.1. Hallar las salidas de la red
     - 2.1.2. Comparar las salidas reales con las deseadas
     - 2.1.3. Calcular incrementos de pesos: ΔW_i
   - 2.2. Actualizar los pesos
3. **Fin**

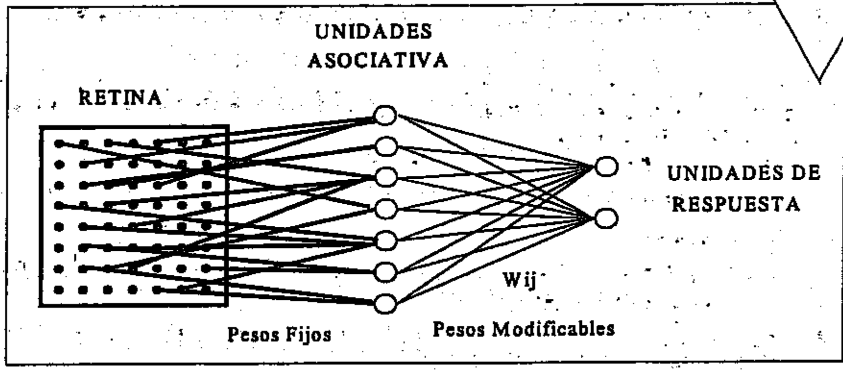
La actualización de los pesos se hace así: $W_i = W_i + \Delta W_i$

Contenido

Perceptron

- ✓ Una retina de unidades sensitivas
- ✓ Una capa de unidades asociativas
- ✓ Una capa de unidades de respuesta

Solo son ajustables las sinapsis entre las unidades asociativas y las unidades de respuesta



Algoritmo de aprendizaje del Perceptron

- ◆ Para construir y utilizar un Perceptron hay que responder a una serie de preguntas:
  - a) ¿Cuál es el número de unidades de la retina?
  - b) ¿Cuál es el número de unidades de respuesta?
  - c) ¿Cuál es el número de unidades asociativas?
  - d) ¿Qué valor tienen las conexiones entre la retina y las unidades asociativas?
  - e) ¿Qué valor tienen las conexiones entre las unidades asociativas y las unidades de respuesta?

Algoritmo de aprendizaje del perceptron

1. **Fijar de alguna manera los pesos de la capa asociativa**
2. **Inicializar aleatoriamente los pesos de la capa de salida**
3. **Introducir patrón de entrada, copiando datos en células x_i de la retina**
4. **Calcular la activación a_i de las unidades asociativas usando la ecuación de la neurona:**

$$a_i = U\left(\sum_{j=0}^{N} w_{ij} x_j\right) \qquad U(n) = \begin{cases} +1 & \text{si } n \geq 0 \\ -1 & \text{si } n < 0 \end{cases}$$

5. **Calcular la activación y_i de las unidades de salida con** $y_i = U\left(\sum_{j=0}^{N} w_{ij} a_j\right)$
6. **Para cada uno de los pesos w_{ij} que conecta una neurona asociativa j con una neurona de salida i, y para el peso del sesgo de esta neurona i, si la activación de la neurona i es:**
   - **6.1. y_i = correcta, no hacer nada**
   - **6.2. y_i = -1 y debía ser +1, entonces cambiar valor del peso con:**
     $w_{ij} = w_{ij} + \alpha\, a_j$
   - **6.3. y_i = +1 y debía ser -1, entonces cambiar valor del peso con:**
     $w_{ij} = w_{ij} - \alpha\, a_j$
7. **Repetir pasos 3 a 6 con todos los patrones hasta que el Perceptrón calcule correctamente la salida**

donde α es una constante entre 0 y 1 y a_j es la activación de la unidad de asociación ó +1 si se trata del sesgo

El Perceptron y la Separabilidad Lineal

◆ Cálculo del Perceptron:

Ejemplo: Perceptron sin capa de unidades de asociación, y con todas las neuronas de salida conectadas con todas las de entrada

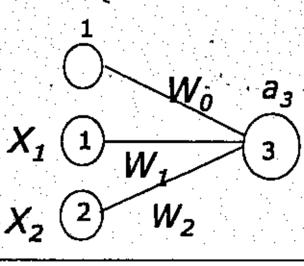
Patrones $Y = \{Y_1\ Y_2\ Y_3\}$

$$a_3 = U(x_0 w_0 + x_1 w_1 + x_2 w_2) \qquad net = X\,W$$

$$U(net) = \begin{cases} +1 & si\ \ net \geq 0 \quad Patrón\ Y_1 \\ -1 & si\ \ net < 0 \quad Patrón\ Y_2 \end{cases}$$

[redacted]

Patrones $Y_0 = \{Y_4\ Y_5\ Y_6\}$

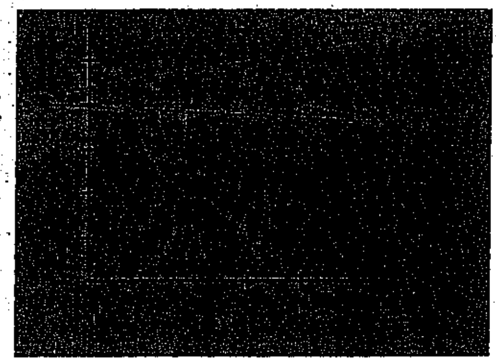


[redacted]

El Perceptron y la Separabilidad Lineal(2)

- **Esta ecuación nos dice que:**
  - a_3 será +1 si el valor de x_2 es mayor que el valor de la recta en x_1
  - a_3 será -1 si ocurre lo contrario
  - La salida a_3 indica si el patrón representado por los datos x_1 y x_2 está por encima o por debajo de esa recta

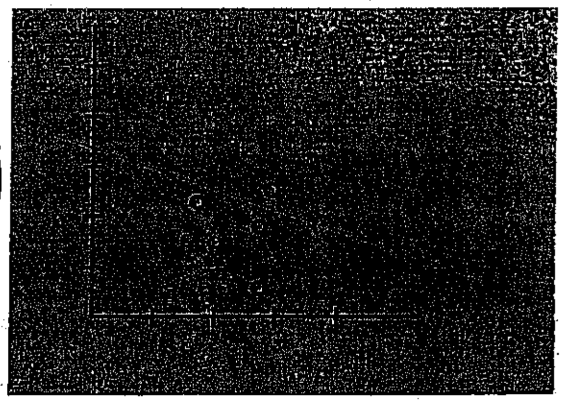
El algoritmo lo que hace es mover la recta (plano o hiperplano) hasta que divide a los patrones en dos



Limitaciones del perceptron

El *problema de la generalización*: La red funciona muy bien para los patrones de *Entrenamiento* (con los que se ha entrenado a la red y se han calculado los pesos)

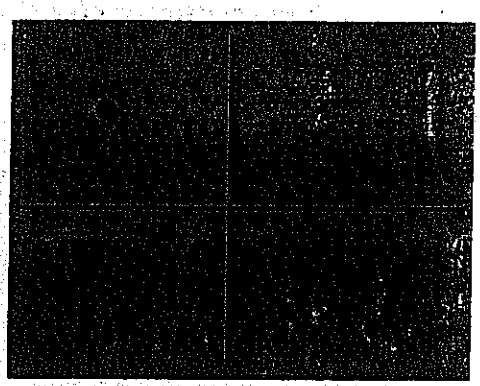
Los problemas *linealmente separables* los resuelve bien con una recta, plano o un hiperplano



Limitaciones del perceptron (2)

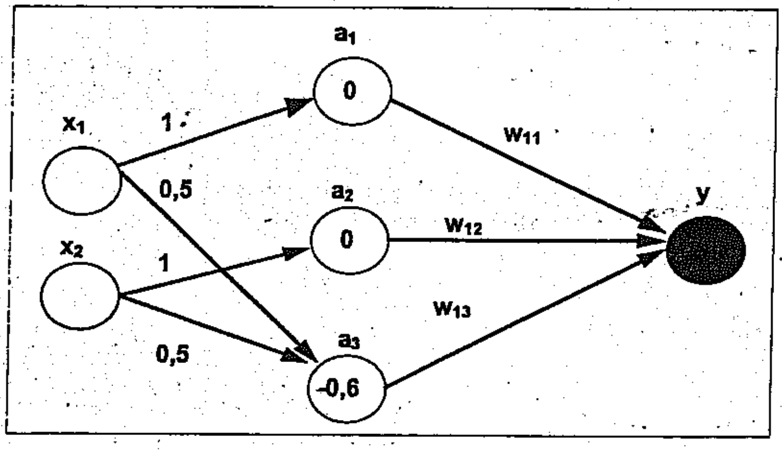
El problema del XOR lógico: **No es un problema linealmente separable**

Nº variables	Funciones	Lineales	No lineales
2 variables binarias	16 funciones boolenas	14 son linealmente separables	2 no el XOR y su negación
6 variables	1,8*10^{19} funciones	5.028.134 son linealmente separables ([illegible])	90%
n variables		Disminuyen	Aumentan



Limitaciones del perceptron (3)

Solución al problema del XOR lógico: **con un Perceptrón con una capa asociativa sería posible resolverlo**

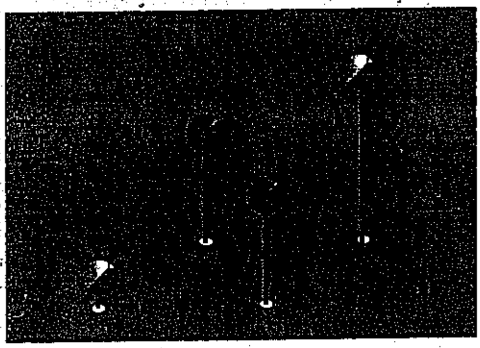


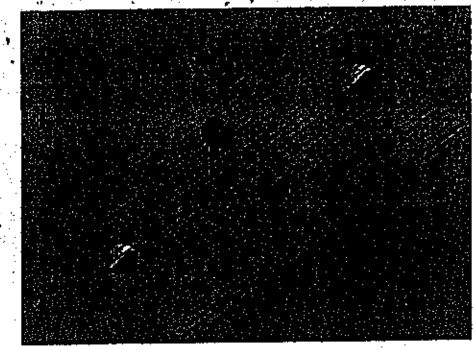
x_1	x_2	y
-1	-1	-1
-1	+1	+1
+1	-1	+1
+1	+1	-1

Limitaciones del perceptron (4)

***Solución al problema del XOR lógico:* ¿Cómo lo ha hecho?**

- **Problema de 2 dimensiones lo ha convertido en uno tridimensional: neurona de salida Y tiene 3 entradas a_1, a_2 y a_3, en vez de sólo las 2 entradas x_1 y x_2**
- **Las 3 entradas tienen un valor distinto para los 4 valores del XOR**

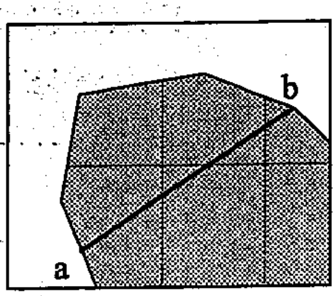




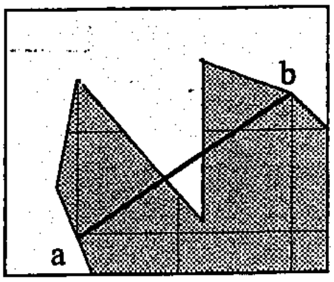
Perceptron Multicapa

Superficies convexas y no convexas

La región es convexa porque para ir de a a b mediante una línea sólo pasamos por puntos de la misma región



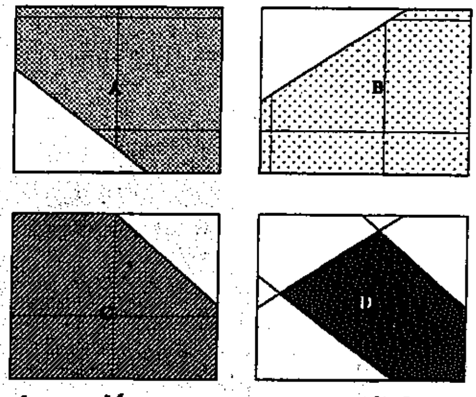
La región no es convexa porque para ir de a a b mediante una línea salimos de la región



Perceptron Multicapa

Superficies convexas y no convexas

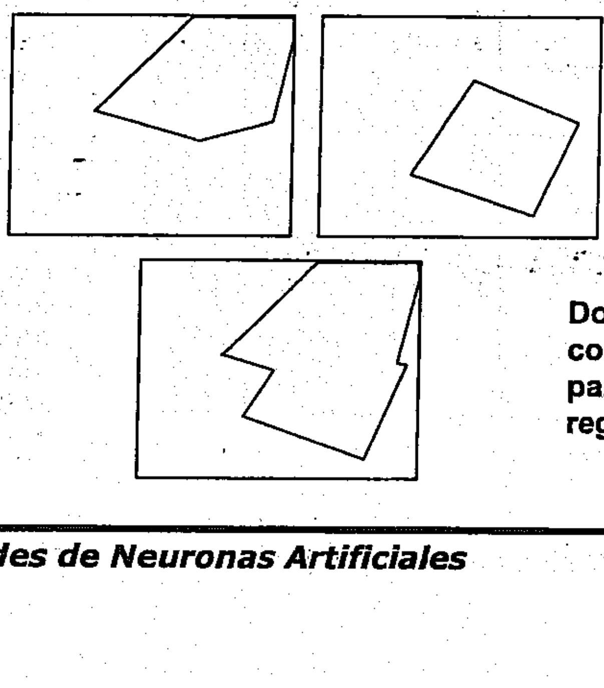
- **Las regiones convexas se forman interseccionando medios planos**
- **Los medios planos que forman la región convexa de un Perceptrón de tres capas están determinados por las neuronas de la capa de asociación**
- **Cada una de estas neuronas se activará sólo si la entrada cae en el lado correcto del hiperplano delimitado por sus pesos**



La región convexa oscura (D) se forma uniendo las regiones A, B y C que dividen cada una el plano en dos

Perceptron Multicapa

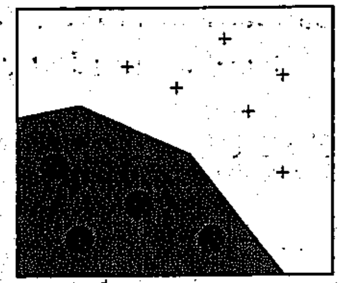
Superficies convexas y no convexas

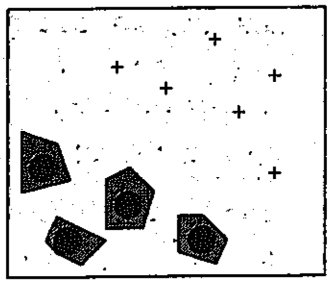


Dos regiones convexas se unen para formar una región no convexa

Perceptron Multicapa

Superficies convexas y no convexas

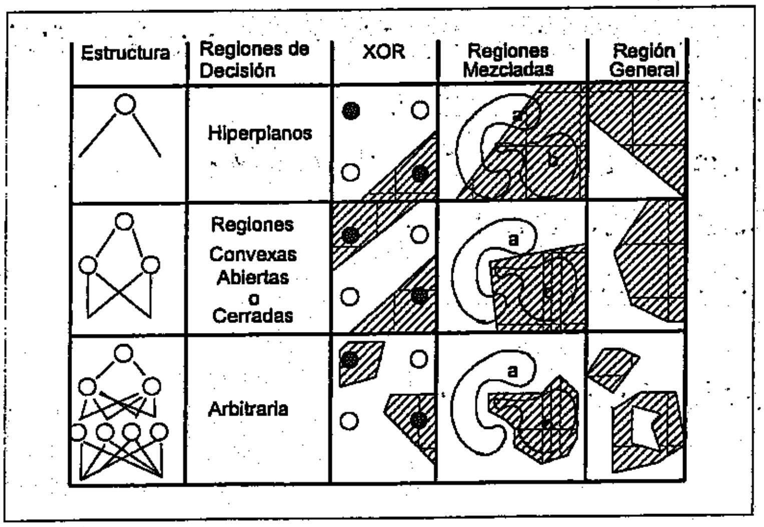




Separación de círculos y cruces:

La solución de la izquierda es más general que la de la derecha, donde se han generado regiones particulares para cada círculo

Perceptron Multicapa



A medida que se dota a un Perceptrón de más y más capas gana más y más capacidad de resolución de problemas.

No todo depende de las capas sino también del número de neuronas por capa.

Perceptron: Conclusiones

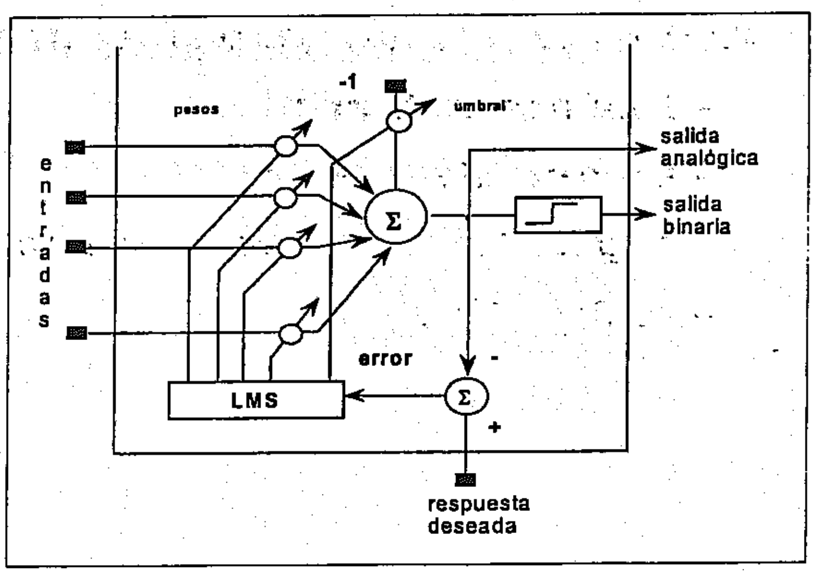
- Red constituida por 3 capas
- Dispone de un algoritmo de aprendizaje iterativo
- Los problemas que resuelve son siempre de clasificación
- Los problemas de clasificación pueden ser linealmente separables o no
- Un Perceptron con sólo 2 capas resuelve cualquier problema lineal y el algoritmo de aprendizaje permite encontrar los pesos
- Un Perceptron con 3 capas o más podrá resolver los problemas no lineales que aparecen más habitualmente en la vida real
- El algoritmo de aprendizaje del Perceptron NO es capaz de encontrar el juego de pesos necesario

ADALINE: (ADAptative LInear NEuron)

- Desarrollado por Widrow y Hoff (U. Stanford)
- Adición de estos elementos: MADALINE
- Algoritmo: LMS (Least Mean Squares) o mínimos cuadrados para ajuste de un ADALINE
- Tienen 2 salidas:

— Binaria:
Salida q = Neurona umbral $q = \begin{cases} +1 & si\ y \geq 0 \\ -1 & si\ y < 0 \end{cases}$
— Analógica:
Salida y = NET (input total) $y = x_0 w_0 + x_1 w_1 + x_2 w_2 = \sum w_{ij} x_i$

ADALINE: Esquema



ADALINE: Neurona Adaptativa Lineal

En las MADALINE la función de transferencia es lineal:

$$Net_i = S_i = \Sigma\, w_{ij}\, x_j$$

[redacted]

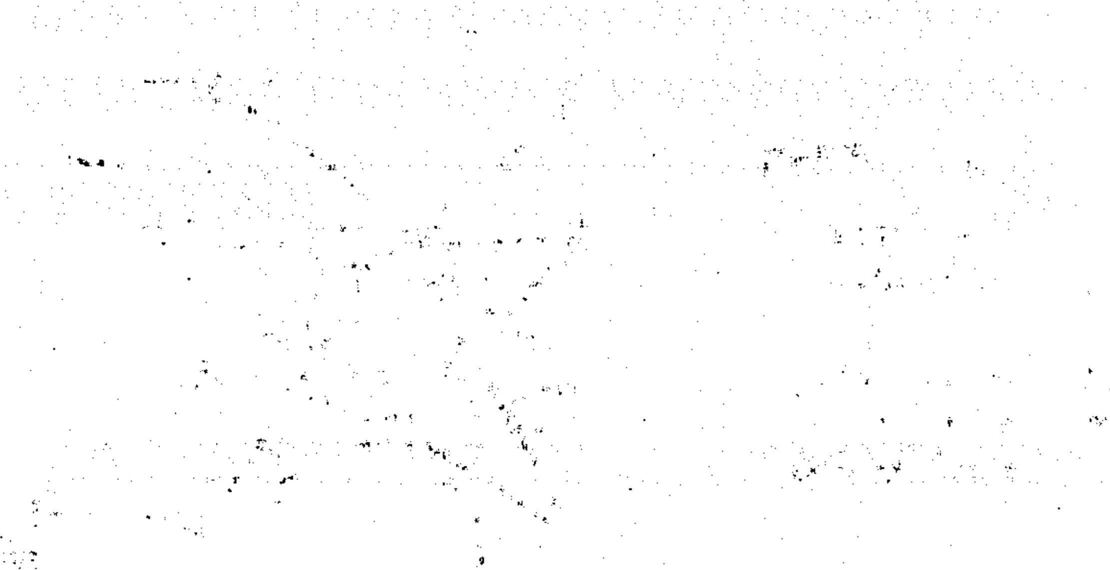
$$E = \tfrac{1}{2}\,(t_i - s_i)^2 = \tfrac{1}{2}\,(t_i - \Sigma\, w_{ij}\, x_j)^2$$

$$\boxed{\Delta\, w_{ij} = \mu\,(t_i - s_i)\, x_j}$$ **REGLA DELTA**

ADALINE: Aplicaciones de Adaline/Madaline

Widrow y col. (1963-64) estudiaron su aplicación a:

- **Previsión del tiempo**
- **Reconocimiento de voz**
- **Diagnóstico de electrocardiogramas**
- **Clasificación de imágenes**



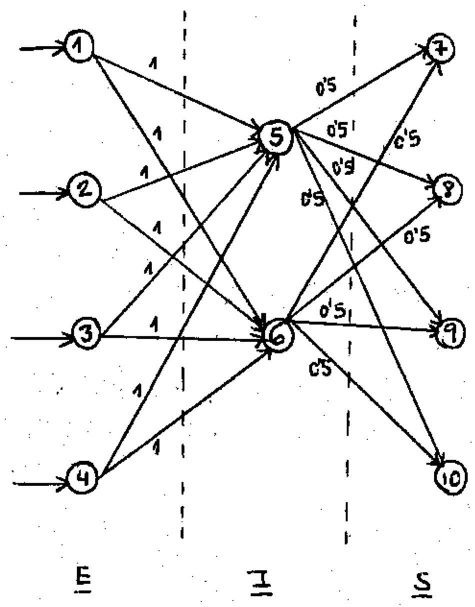
TEMA 9: REDES DE NEURONAS ARTIFICIALES.

1. Introducción.
2. Fundamentos biológicos de las RNA.
3. La neurona artificial.
4. Componentes de una RNA.
5. Clases de neuronas.
6. Aprendizaje.
7. El perceptron.
8. ADALINE.

TRANSPARENCIAS.

EJ:

a)



Vector de entrada:

$$\begin{array}{ccccc} & x_1 & x_2 & x_3 & x_4 \\ \overline{X} = & [\,0'7 & 0'3 & 0'4 & 0'6\,] \end{array}$$

F(0'88) = 0'7
F(2) = 0'88

DATOS

$$Net_5 = [W_{51}\ W_{52}\ W_{53}\ W_{54}] \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = [1\ 1\ 1\ 1] \cdot \begin{bmatrix} 0'7 \\ 0'3 \\ 0'4 \\ 0'6 \end{bmatrix} = 2$$

$$Net_6 = [W_{61}\ W_{62}\ W_{63}\ W_{64}] \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = [1\ 1\ 1\ 1] \cdot \begin{bmatrix} 0'7 \\ 0'3 \\ 0'4 \\ 0'6 \end{bmatrix} = 2$$

$O_5 = F(Net_5) = F(2) = 0'88$

$O_6 = F(Net_6) = F(2) = 0'88$

$$Net_7 = [\,W_{75} \quad W_{76}\,] \cdot \begin{bmatrix} O_5 \\ O_6 \end{bmatrix} = [\,0'5 \quad 0'5\,] \cdot \begin{bmatrix} 0'88 \\ 0'88 \end{bmatrix} = 0'88$$

La salida de las neuronas con las que conecta!! NO el vector de entrada inicial.

$$Net_8 = [\,W_{85} \quad W_{86}\,] \cdot \begin{bmatrix} O_5 \\ O_6 \end{bmatrix} = [\,0'5 \quad 0'5\,] \cdot \begin{bmatrix} 0'88 \\ 0'88 \end{bmatrix} = 0'88$$

$$Net_9 = [\,W_{95} \quad W_{96}\,] \cdot \begin{bmatrix} O_5 \\ O_6 \end{bmatrix} = [\,0'5 \quad 0'5\,] \cdot \begin{bmatrix} 0'88 \\ 0'88 \end{bmatrix} = 0'88$$

$$Net_{10} = [\,W_{10\,5} \quad W_{10\,6}\,] \begin{bmatrix} O_5 \\ O_6 \end{bmatrix} = [\,0'5 \quad 0'5\,] \cdot \begin{bmatrix} 0'88 \\ 0'88 \end{bmatrix} = 0'88$$

$O_7 = F(Net_7) = F(0'88) = 0'7$ $\qquad$ $O_9 = F(Net_9) = F(0'88) = 0'7$

$O_8 = F(Net_8) = F(0'88) = 0'7$ $\qquad$ $O_{10} = F(Net_{10}) = F(0'88) = 0'7$

b) Si la función de activación fuese lineal:

$Net_5 = 2 \qquad O_5 = Net_5 = 2$

$Net_6 = 2 \qquad O_6 = Net_6 = 2$

Si es lineal, el estado de activación coincide con el input total (net) de la unidad.

$Net_7 = [\,0'5 \quad 0'5\,] \begin{bmatrix} 2 \\ 2 \end{bmatrix} = 2$ $\qquad$ $O_7 = Net_7 = 2$

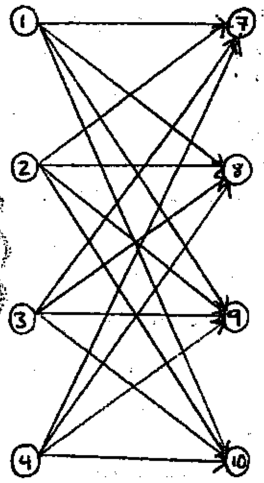
$Net_8 =$ " " $= 2$ $\qquad$ $O_8 = Net_8 = 2$

$Net_9 =$ " " $= 2$ $\qquad$ $O_9 = Net_9 = 2$

$Net_{10} =$ " " $= 2$ $\qquad$ $O_{10} = Net_{10} = 2$

c)

Quito las neuronas intermedias.



$$W_1 = \begin{bmatrix} W_{51} & W_{52} & W_{53} & W_{54} \\ W_{61} & W_{62} & W_{63} & W_{64} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

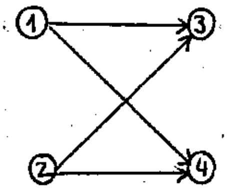
$$W_2 = \begin{bmatrix} W_{75} & W_{76} \\ W_{85} & W_{86} \\ W_{95} & W_{96} \\ W_{10\,5} & W_{10\,6} \end{bmatrix} = \begin{bmatrix} 0'5 & 0'5 \\ 0'5 & 0'5 \\ 0'5 & 0'5 \\ 0'5 & 0'5 \end{bmatrix} \qquad \begin{array}{l} O_5 = Net_5 = 2 \\ O_6 = Net_6 = 2 \end{array}$$

$$W_1 \cdot W_2 = W$$

$$\underset{W_2}{\begin{bmatrix} 0'5 & 0'5 \\ 0'5 & 0'5 \\ 0'5 & 0'5 \\ 0'5 & 0'5 \end{bmatrix}} \cdot \underset{W_1}{\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}} = \underset{W}{\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}}$$

$$\begin{array}{l} O_7 = \\ O_8 = \\ O_9 = \\ O_{10} = \end{array} \begin{bmatrix} Net_7 \\ Net_8 \\ Net_9 \\ Net_{10} \end{bmatrix} = \begin{bmatrix} W_{71} & W_{72} & W_{72} & W_{74} \\ W_{81} & W_{82} & W_{83} & W_{84} \\ W_{91} & W_{92} & W_{93} & W_{94} \\ W_{10\,1} & W_{10\,2} & W_{10\,3} & W_{10\,4} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0'7 \\ 0'3 \\ 0'4 \\ 0'6 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \\ 2 \\ 2 \end{bmatrix}$$

EJ: REGLA DE HEBB.



incremento de los pesos

$$\boxed{\Delta w_{ij} = \mu\, a_i\, O_j}$$

Este valor lo tienen que dar.

DATOS.- Pesos: $w_{31} = 2$ $\quad \mu = 0'1$

$w_{32} = -1$

$w_{41} = -1$

$w_{42} = 1$

Función de activación umbral: $a_i = \begin{cases} 0 & \text{si } \sum w_{ij} O_j < 1 \\ 1 & \text{si } \underbrace{\sum w_{ij} O_j}_{Net} > 1 \end{cases}$

Para aplicar la Regla de Hebb vamos a ver lo que pasa en las neuronas 3 y 4. Para ello considero las posibles salidas de 1 y 2:

$\left.\begin{matrix} O_1 = 1 \\ O_2 = 0 \end{matrix}\right\}$ $\quad \left.\begin{matrix} O_1 = 0 \\ O_2 = 1 \end{matrix}\right\}$ $\quad \left.\begin{matrix} O_1 = 1 \\ O_2 = 1 \end{matrix}\right\}$

+ Lo primero es calcular el NET:

Primer patrón → $\left.\begin{matrix} O_1 = 1 \\ O_2 = 0 \end{matrix}\right\}$ $\quad Net_3 = [2 \ \ -1] \begin{bmatrix} 1 \\ 0 \end{bmatrix} = 2$ $\qquad a_3 = U(Net_3) = U(2) = 1$ (porque es > 1)

$Net_4 = [-1 \ \ 0] \begin{bmatrix} 1 \\ 0 \end{bmatrix} = -1$ $\qquad a_4 = U(Net_4) = U(-1) = 0$

vector entrada.

+ A continuación hay que calcular los incrementos de peso y pesos nuevos.

$$\begin{aligned} &\quad\ \ \mu \quad a_3 \quad O_1 \\ \Delta w_{31} &= 0'1 \cdot 1 \cdot 1 = 0'1 & w_{31} &= 2 + 0'1 = 2'1 \\ \Delta w_{32} &= 0'1 \cdot 1 \cdot 0 = 0 & w_{32} &= -1 + 0 = -1 \\ \Delta w_{41} &= 0'1 \cdot 0 \cdot 1 = 0 & w_{41} &= -1 + 0 = -1 \\ \Delta w_{42} &= 0'1 \cdot 0 \cdot 0 = 0 & w_{42} &= 1 + 0 = 1 \end{aligned}$$

Pesos nuevos

Se ponen los nuevos pesos — patrón de entrada

Siguiente patrón $\rightarrow$ $\left.\begin{array}{l} O_1 = 0 \\ O_2 = 1 \end{array}\right\}$ * Calculo el NET: $Net_3 = \begin{bmatrix} 2'1 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = -1$

$$Net_4 = \begin{bmatrix} -1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = 1$$

$$a_3 = U(-1) = 0 \qquad a_4 = U(1) = 1$$

$$\left.\begin{array}{ll} \Delta w_{31} = 0'1 \cdot 0 \cdot 0 = 0 & w_{31} = 2'1 + 0 = 2'1 \\ \Delta w_{32} = 0'1 \cdot 0 \cdot 1 = 0 & w_{32} = -1 + 0 = -1 \\ \Delta w_{41} = 0'1 \cdot 1 \cdot 0 = 0 & w_{41} = -1 + 0 = -1 \\ \Delta w_{42} = 0'1 \cdot 1 \cdot 1 = 0'1 & w_{42} = 1 + 0'1 = 1'1 \end{array}\right\} \text{Pesos nuevos.}$$

Siguiente patrón $\rightarrow$ $\left.\begin{array}{l} O_1 = 1 \\ O_2 = 1 \end{array}\right\}$

$$Net_3 = \begin{bmatrix} 2'1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 1'1 \qquad a_3 = U(1'1) = 1$$

$$Net_4 = \begin{bmatrix} -1 & 1'1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 0'1 \qquad a_4 = U(0'1) = 0$$

$$\begin{array}{ll} \Delta w_{31} = 0'1 \cdot 1 \cdot 1 = 0'1 & w_{31} = 2'1 + 0'1 = \boxed{2'2} \\ \Delta w_{32} = 0'1 \cdot 1 \cdot 1 = 0'1 & w_{32} = -1 + 0'1 = \boxed{-0'9} \\ \Delta w_{41} = 0'1 \cdot 0 \cdot 1 = 0 & w_{41} = -1 + 0 = \boxed{-1} \\ \Delta w_{42} = 0'1 \cdot 0 \cdot 1 = 0 & w_{42} = 1'1 + 0 = \boxed{1'1} \end{array}$$

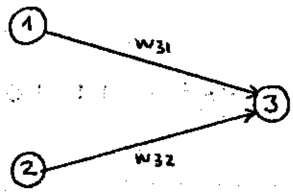
NUEVOS PESOS DE CONEXIÓN DESPUÉS DE PASARLE LOS 3 PATRONES

Si las neuronas 3 y 4 recibieran siempre estos tres patrones, consideraríamos que la red ha aprendido cuando no se modifican los pesos para ninguno de dichos patrones. Tendríamos que continuar el ejercicio pasándoselos hasta que los pesos no cambien, es decir, hasta que los pesos son válidos para todas las entradas. Esto es aprendizaje no supervisado (no sabe la respuesta correcta, sólo sabe que debe ajustar sus pesos).

EJ: **REGLA DELTA.**

$$\boxed{\Delta w_{ij} = \mu\,(t_i - a_i)\,O_j}$$

Aprendizaje supervisado.



DATOS.- $\mu = 0'5$

$w_{31} = 1$

$w_{32} = 1$

Función de activación → FA lineal : $a_3 = \sum w_{3i}\, O_j = Net_3$

FS umbral : $O_i = \begin{cases} -1 & \text{si } a_3 < 0 \\ 1 & \text{si } a_3 \geq 0 \end{cases}$

de entrada ↓

Estímulos	Act. Deseada t_i	Salida O_i
[-1 -1]	-0'2	-1
[1 -1]	-0'6	-1
[-1 1]	0'6	1
[1 1]	0'2	1

Aplicando la función debe sacar lo que deseamos.

$\Delta w_{ij} = \mu\,(t_i - a_i)\,O_j \qquad \mu = 0'5$

- 1ª ITERACIÓN

* P [-1 -1]

$a_3 = w_{31}O_1 + w_{32}O_2 = 1(-1) + 1(-1) = -2 \qquad$ Act. deseada $= -0'2$

$\Delta w_{31} = 0'5\,(-0'2 - (-2))\,(-1) = -0'9 \qquad w_{31} = 1 + (-0'9) = 0'1$

$\Delta w_{32} = 0'5\,(-0'2 - (-2))\,(-1) = -0'9 \qquad w_{32} = 1 + (-0'9) = 0'1$

* P [1 -1]

$a_3 = w_{31}O_1 + w_{32}O_2 = 0'1(+1) + 0'1(-1) = 0 \qquad$ A. deseada $= -0'6$

$\Delta w_{31} = 0'5\,(-0'6 - 0)\,(1) = -0'3 \qquad w_{31} = 0'1 + (-0'3) = \boxed{-0'2}$

$\Delta w_{32} = 0'5\,(-0'6 - 0)\,(-1) = 0'3 \qquad w_{32} = 0'1 + (0'3) = \boxed{0'4}$

Pesos para que aprendan los 4 patrones

* P[-1 1]

$a_3 = -0'2(-1) + 0'4(1) = 0'6$ AD = 0'6   No modificar pesos.

Si la salida es correcta no hay que hacer nada.

* P[1 1]

$a_3 = -0'2 \cdot 1 + 0'4 \cdot 1 = 0'2$ AD = 0'2   No modificar pesos.

· 2ª ITERACIÓN.

* P[-1 -1]

$a_3 = -0'2(-1) + 0'4(-1) = -0'2$ AD = -0'2   No modificar pesos.

* P[1 -1]

$a_3 = -0'2(1) + 0'4(-1) = -0'6$ AD = -0'6   No modificar pesos.

* P[-1 1]

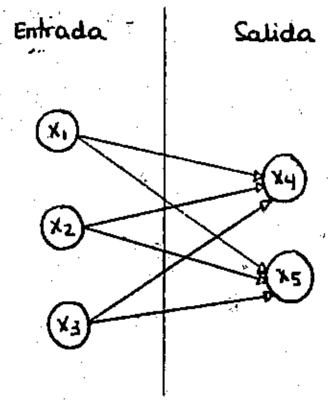
$a_3 = 0'6$ AD = 0'6   No modificar pesos.

* P[1 1]

$a_3 = 0'2$ AD = 0'2   No modificar pesos.

EJ: PERCEPTRÓN

Sea el perceptrón de la figura. Calcular utilizando el algoritmo de aprendizaje los pesos después de la 1ª iteración.



	x_0	x_1	x_2	x_3	y_4	y_5
P_1	1	1	1	1	1	1
P_2	1	1	1	-1	1	-1
P_3	1	1	-1	1	-1	1
P_4	1	-1	1	1	-1	-1

$$Y_i = \underset{\substack{\uparrow \\ \text{función umbral}}}{U}(Net_i) = \begin{cases} 1 & \text{si } Net_i \geq 0 \\ -1 & \text{si } Net_i < 0 \end{cases}$$

Calcular pesos conexión:

Pesos w_{ij} a cero

$\alpha = 1$

$$w_{t+1} = \begin{cases} w_t + \alpha X & \text{si } Y_i = -1 \text{ (Debía ser +1)} \\ w_t - \alpha X & \text{si } Y_i = +1 \text{ (" " -1)} \\ w_t & \text{si correcto} \end{cases}$$

- $P_1 (x_0 \; x_1 \; x_2 \; x_3) = (1 \; 1 \; 1 \; 1)$

Salida $(Y_4 \; Y_5) = (1 \; 1)$

$w_{4j} = (w_{40} \; w_{41} \; w_{42} \; w_{43}) = (0 \; 0 \; 0 \; 0)$

$w_{5j} = (w_{50} \; w_{51} \; w_{52} \; w_{53}) = (0 \; 0 \; 0 \; 0)$

$Net_4 = w_{40} \cdot x_0 + w_{41} \cdot x_1 + w_{42} \cdot x_2 + w_{43} \cdot x_3 = 0$

$Net_5 = w_{50} \cdot x_0 + w_{51} \cdot x_1 + w_{52} \cdot x_2 + w_{53} \cdot x_3 = 0$

$Y_4 = U(Net_4) = U(0) = 1$ Bien calificada

$Y_5 = U(Net_5) = U(0) = 1$ B.C.

* P2 $(x_0\ x_1\ x_2\ x_3) = (1\ 1\ 1\ -1)$

Salida $(Y_4\ Y_5) = (1\ -1)$

$w_{4j} = (0\ 0\ 0\ 0)$ $\quad Net_4 = 0 \quad Y_4 = U(Net_4) = 1$ $\quad$ BC

$w_{5j} = (0\ 0\ 0\ 0)$ $\quad Net_5 = 0 \quad Y_5 = U(Net_5) = 1$ $\quad$ MC (Debía ser -1)

$w_{4j} = (0\ 0\ 0\ 0)$

$w_{5j} = (0\ 0\ 0\ 0) - 1(1\ 1\ 1\ -1) = (-1\ -1\ -1\ 1)$ Hemos cambiado los pesos de W_{5j}

* P3 $(1\ 1\ -1\ 1)$

Salida $(-1\ 1)$

$Net_4 = 0 \quad Y_4 = U(0) = 1$ $\quad$ MC (Debía ser -1)

$w_{4j} = (0\ 0\ 0\ 0) - 1(1\ 1\ -1\ 1) = (-1\ -1\ 1\ -1)$

$Net_5 = (-1)(1) + (-1)(1) + (-1)(-1) + (1)(1) = -1 - 1 + 1 + 1 = 0 \quad Y_5 = U(0) = 1$ $\quad$ BC

* P4 $(1\ -1\ 1\ 1)$

Salida $(-1\ -1)$

$Net_4 = (-1)(1) + (-1)(-1) + (1)(1) + (-1)(1) = -1 + 1 + 1 - 1 = 0 \quad Y_4 = U(0) = 1$ $\quad$ MC (-1)

$w_{4j} = (-1\ -1\ 1\ -1) - 1(1\ -1\ 1\ 1) = \underbrace{(-2\ 0\ 0\ -2)}$

Esto es lo que se pide

$Net_5 = (-1)(1) + (-1)(-1) + (-1)(1) + (1)(1) = -1 + 1 - 1 + 1 = 0 \quad Y_5 = U(0) = 1$ $\quad$ MC (-1)

$w_{5j} = (-1\ -1\ -1\ 1) - 1(1\ -1\ 1\ 1) = \underbrace{(-2\ 0\ -2\ 0)}$

Esto es lo que se pide

EJ: JUNIO 04

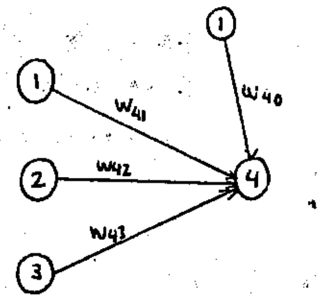
Construir un perceptron que clasifique correctamente los siguientes patrones:

$Y_1 = \{ y_1 = (-1\ -1\ -1),\ y_2 = (1\ 1\ 1) \}$

$Y_2 = \{ y_3 = (-1\ 1\ -1),\ y_4 = (1\ -1\ -1) \}$

$w_{ij} = 0$

$\alpha = 1$



Sesgo → X_0; cosa nuestra → X_4

	X_0	X_1	X_2	X_3	X_4	
P1	1	-1	-1	-1	1	Y_1
P2	1	1	1	1	1	Y_1
P3	1	-1	1	-1	-1	Y_2
P4	1	1	-1	-1	-1	Y_2

Patrones: X_0–X_3

Lo del enunciado: X_1–X_3

$w = (0\,0\,0\,0)$

$$Y_4 = U(Net_4) = \begin{cases} +1 & Net_4 \geq 0 \\ -1 & Net_4 < 0 \end{cases}$$

$$Net_4 = w_{40}x_0 + w_{41}x_1 + w_{42}x_2 + w_{43}x_3$$

$$w_{(t+1)} = \begin{cases} w_t + \alpha X & (MC) \quad Y_4 = -1 \in Y_1 \text{ (debía ser 1)} \\ w_t - \alpha X & (MC) \quad Y_4 = 1 \in Y_2 \text{ (debía ser -1)} \\ w_t & (BC) \end{cases}$$

• 1^{a} ITERACIÓN

P1 $(1\ -1\ -1\ -1)$ Salida = 1

$$Net_4 = [0\ 0\ 0\ 0]\begin{bmatrix} 1 \\ -1 \\ -1 \\ -1 \end{bmatrix} = 0 \qquad Y_4 = U(0) = 1 \quad BC$$

P2 $(1\ 1\ 1\ 1)$ Salida = 1

$Net_4 = 0 \qquad Y_4 = U(0) = 1 \quad BC$

P_3 (1 -1 1 -1) Salida = -1

$Net_4 = 0$. $Y_4 = U(0) = 1$ MC

$w_{4j} = (0\ 0\ 0\ 0) - 1(1\ -1\ 1\ -1) = (-1\ 1\ -1\ 1)$

P_4 (1 1 -1 -1) Salida = -1

$Net_4 = (-1)(1) + (1)(1) + (-1)(-1) + (1)(-1) = -1 + 1 + 1 - 1 = 0$

$Y_4 = U(0) = 1$ MC

$w_{4j} = (-1\ 1\ -1\ 1) - 1(1\ 1\ -1\ -1) = (-2\ 0\ 0\ 2)$

• 2ª ITERACIÓN.

P1 (1 -1 -1 -1) Salida = 1

$Net_4 = (-2)(1) + 0 + 0 + (2)(-1) = -4$ $Y_4 = U(-4) = -1$ MC

$w = (-2\ 0\ 0\ 2) + 1(1\ -1\ -1\ -1) = (-1\ -1\ -1\ 1)$

P2 (1 1 1 1) Salida = 1

$Net_4 = (-1)(1) + (-1)(1) + (-1)(1) + (1)(1) = -1 -1 -1 + 1 = -2$

$Y_4 = U(-4) = -1$ MC

$w = (-1\ -1\ -1\ 1) + 1(1\ 1\ 1\ 1) = (0\ 0\ 0\ 2)$

P3 (1 -1 1 -1) Salida = -1

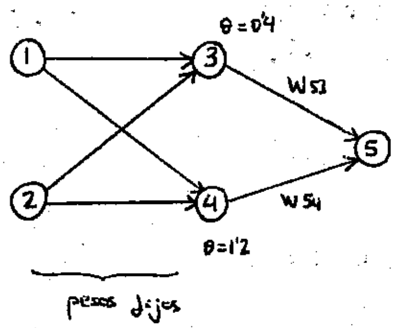
$Net_4 = 0 + 0 + 0 + (2)(-1) = -2$ $Y_4 = U(-2) = -1$ BC

P4 (1 1 -1 -1) Salida = -1

$Net_4 = 0 + 0 + 0 + (2)(-1) = -2$ $Y_4 = U(-2) = -1$ BC

PERCEPTRÓN CON CAPA ASOCIATIVA.

Calcular los pesos w_{53}, w_{54} para resolver el problema del XOR.



x_0	x_1	x_2	Y_5
1	0	0	0
1	1	0	1
1	0	1	1
1	1	1	0

$W_{(0)} = (0\ 0\ 0)$

X_0	a_3	a_4	Y_5
1	0	0	0
1	1	0	1
1	1	0	1
1	1	1	0

$$\begin{pmatrix} \text{act.} \\ \text{neu } 3 \end{pmatrix} a_3 = U(Net_3) \begin{cases} 1 & \text{si } Net_3 \geq 0 \rightarrow Net_3 - \theta \geq 0 \\ 0 & \text{si } Net_3 < 0 \rightarrow Net_3 - \theta < 0 \end{cases}$$

$\alpha = 0'5$

$w_{ij} = 0$

$$a_4 = U(Net_4) \begin{cases} 1 & \text{si } Net_4 \geq 0 \rightarrow Net_4 - \theta \geq 0 \\ 0 & \text{si } Net_4 < 0 \rightarrow Net_4 - \theta < 0 \end{cases}$$

$Net_3 = w_{30} x_0 + w_{31} x_1 + w_{32} x_2 \qquad w_{30} = -0'4$

$Net_4 = w_{40} x_0 + w_{41} x_1 + w_{42} x_2 \qquad w_{40} = -1'2$

1 [0 0]

$a_3 = U(Net_3) = U(-0'4 x_0 + 1 x_1 + 1 x_2) = U(-0'4) = 0$

$a_4 = U(Net_4) = U(-1'2) = 0$

$Net_3 = -0'4 + 0 + 0 = -0'4$

$Net_4 = -1'2 + 0 + 0 = -1'2$

1 [1 0] $\quad Net_3 = -0'4 + 1 + 0 = 0'6 \qquad a_3 = U(0'6) = 1$

$Net_4 = -1'2 + 1 + 0 = -0'2 \qquad a_4 = U(-0'2) = 0$

1 [0 1] $\quad Net_3 = -0'4 + 0 + 1 = 0'6 \qquad a_3 = U(0'6) = 1$

$Net_4 = -1'2 + 0 + 1 = -0'2 \qquad a_4 = U(-0'2) = 0$

1 [1 1] $\quad Net_3 = -0'4 + 1 + 1 = 1'6 \qquad a_3 = U(1'6) = 1$

$Net_4 = -1'2 + 1 + 1 = 0'8 \qquad a_4 = U(0'8) = 1$

· 1^{a} ITERACIÓN.

P1 (1 0 0)

$$Net_5 = [w_{50}\ w_{53}\ w_{54}] \begin{bmatrix} x_0 \\ a_3 \\ a_4 \end{bmatrix} = [0\ 0\ 0] \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = 0 \qquad Y_5 = U(Net_5) = U(0) = 1 \quad MC$$

$$U(Net_5) = \begin{cases} 1 & si\ Net_5 \geq 0 \\ 0 & si\ Net_5 < 0 \end{cases}$$

$$w_1 = (0\ 0\ 0) - 0'5(1\ 0\ 0) = (-0'5\ \ 0\ \ 0)$$

P2 (1 1 0)

$$Net_5 = [-0'5\ \ 0\ \ 0] \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} = -0'5 \qquad Y_5 = U(-0'5) = 0 \quad MC$$

$$w_2 = (-0'5\ \ 0\ \ 0) + 0'5(1\ 1\ 0) = (0\ \ 0'5\ \ 0)$$

P3 (1 1 0)

$$Net_5 = [0\ \ 0'5\ \ 0] \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} = 0'5 \qquad Y_5 = U(0'5) = 1 \quad BC \qquad w_3 = w_2$$

P4 (1 1 1)

$$Net_5 = [0\ \ 0'5\ \ 0] \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = 0'5 \qquad Y_5 = U(0'5) = 1 \quad MC$$

$$w_4 = [0\ \ 0'5\ \ 0] - 0'5[1\ 1\ 1] = [-0'5\ \ 0\ \ -0'5]$$

· 2^{a} ITERACIÓN.

P1 (1 0 0)

$$Net_5 = (-0'5\ \ 0\ \ 0'5) \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = -0'5 \qquad Y_5 = U(-0'5) = 0 \quad BC$$

$$w_5 = w_4$$

P2 (1 1 0)

$$Net_5 = (-0'5 \quad 0 \quad -0'5)\begin{pmatrix}1\\0\\0\end{pmatrix} = -0'5 \qquad Y_5 = U(-0'5) = 0 \ \ MC$$

$$\omega_6 = (-0'5 \quad 0 \quad -0'5) + 0'5(1 \ 1 \ 0) = (0 \quad 0'5 \quad -0'5)$$

P3 (1 1 0)

$$Net_5 = (0 \quad 0'5 - 0'5)\begin{pmatrix}1\\1\\0\end{pmatrix} = 0'5 \qquad Y_5 = U(0'5) = 1 \ \ BC$$

$$\omega_7 = \omega_6$$

P4 (1 1 1)

$$Net_5 = (0 \quad 0'5 \quad -0'5)\begin{pmatrix}1\\1\\1\end{pmatrix} = 0 \qquad Y_5 = U(0) = 1 \ \ MC$$

$$\omega_8 = (0 \quad 0'5 \quad -0'5) - 0'5(1 \ 1 \ 1) = (-0'5 \quad 0 \quad -1)$$

• 3ª iteración.

P1 (1 0 0)

$$Net_5 = (-0'5 \quad 0 \quad -1)\begin{pmatrix}1\\0\\0\end{pmatrix} = -0'5 \qquad Y_5 = U(-0'5) = 0 \ \ BC \qquad \omega_9 = \omega_8$$

P2 (1 1 0)

$$Net_5 = (-0'5 \quad 0 \quad -1)\begin{pmatrix}1\\1\\0\end{pmatrix} = -0'5 \qquad Y_5 = U(-0'5) = 0 \ \ MC$$

$$\omega_{10} = (-0'5 \quad 0 \quad -1) + 0'5(1 \ 1 \ 0) = (0 \quad 0'5 \quad -1)$$

P3 (1 1 0)

$$Net_5 = (0 \quad 0'5 \quad -1)\begin{pmatrix}1\\1\\0\end{pmatrix} = 0'5 \qquad Y_5 = U(0'5) = 1 \ \ BC$$

$$\omega_{11} = \omega_{10}$$

P4 (1 1 1)

$$Net_5 = (0\ \ 0'5\ \ -1)\begin{pmatrix}1\\1\\1\end{pmatrix} = 0'5 \qquad Y_5 = U(0'5) = 0 \quad BC$$

• 4ª ITERACIÓN

P1 (1 0 0)

$$Net_5 = (0\ \ 0'5\ \ -1)\begin{pmatrix}1\\0\\0\end{pmatrix} = 0 \qquad Y_5 = U(0) = 1 \quad MC$$

$$w_{13} = (0\ \ 0'5\ \ -1) - 0'5(1\ 0\ 0) = (-0'5\ \ 0'5\ \ -1)$$

P2 (110)

$$Net_5 = (-0'5\ \ 0'5\ \ -1)\begin{pmatrix}1\\1\\0\end{pmatrix} = 0 \qquad Y_5 = U(0) = 1 \quad BC$$

$w_{14} = w_{13}$

P3 (110)

$Net_5 = \ldots$ BC $w_{15} = w_{14}$

P4 (1 1 1)

$$Net_5 = (-0'5\ \ 0'5\ \ -1)\begin{pmatrix}1\\1\\1\end{pmatrix} = -1 \qquad Y_5 = U(-1) = 0 \quad BC$$

$$w = (-0'5\ \ 0'5\ \ -1)$$
$$\quad w_{50}\ \ w_{53}\ \ w_{54}$$

Facultad de Informática de Madrid

INFORMÁTICA TEÓRICA – GRUPO 21-M

EXAMEN 1er PARCIAL- Febrero 2005

Apellidos: ____________________ Nombre: ____________________

Ejercicio 1:

Dado el Lenguaje: $L = \{ a^n b^m \,/\, m = n \text{ ó } m = 2n \text{ ó } n \le m \le 2n \quad n, m \ge 0 \}$

a) Obtener una Gramática Independiente de Contexto (G) que genere dicho lenguaje (solo son necesarios tres símbolos no-terminales incluido el axioma).

b) Depurar dicha gramática G.

c) Obtener una gramática equivalente en Forma Normal de Greibach (FNG).

d) Obtener una gramática equivalente en Forma Normal de Chomsky (FNCH).

40 minutos – 15 puntos

Facultad de Informática de Madrid

INFORMÁTICA TEÓRICA – GRUPO 21-M

EXAMEN 1er PARCIAL- Febrero 2005

Apellidos: Nombre:

Ejercicio 2:

a) Demostrar si se verifica o no la siguiente implicación:

Sean p y q dos estados de un autómata finito determinista,

$$\forall\, x \in \Sigma^{+},\ f(p, x)\ E\ f(q, x) \Rightarrow pEq$$

b) Construir un autómata finito A:

b.1) Que acepte los números naturales (codificados en sistema decimal) múltiplos de 4.

Es decir, $L(A)= \{ n \mid n \bmod 4 = 0 \}$.

(Se aceptan como válidos los números que comiencen por 0: por ejemplo, 04, 08, 012, etc. y se valorarán positivamente los diseños mas simples).

b.2) Minimizar dicho autómata.

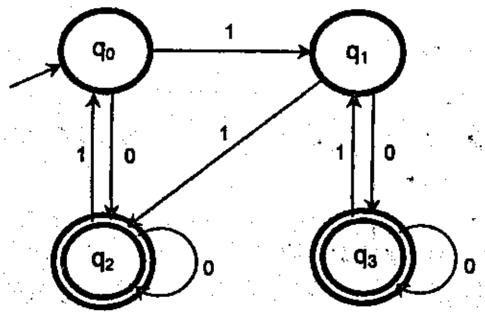
50 minutos – 20 puntos

Facultad de Informática de Madrid
INFORMÁTICA TEÓRICA – GRUPO 21-M
EXAMEN 1^er^ PARCIAL- Febrero 2005

Apellidos: Nombre:

Ejercicio 3:

Dado el siguiente Autómata Finito (AFD) descrito mediante el siguiente diagrama de estados:



Obtener, utilizando los algoritmos explicados en clase:

a) Una Gramática Lineal Izquierda (GLI) que genere el mismo lenguaje L que acepta el AF, justificando las producciones obtenidas (explicando, paso a paso, la aplicación del algoritmo).

b) Una Gramática Lineal Derecha (GLD) que genere el mismo lenguaje L que acepta el AF, justificando, también en este caso, las producciones obtenidas.

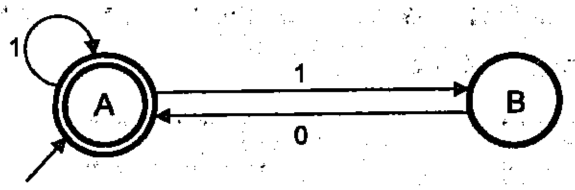
40 minutos – 15 puntos

Facultad de Informática de Madrid
INFORMÁTICA TEÓRICA – GRUPO 21-M
EXAMEN 2º PARCIAL - Junio 2005

Apellidos: Nombre:

Ejercicio 1:

Sea A un Autómata Finito que acepta un Lenguaje L:



Se pide:

a) Obtener una Expresión Regular que describa el Lenguaje L (utilizar ecuaciones características) y una gramática G que genere el Lenguaje L (utilizar derivadas).

b) Construir un autómata A^{-1} que acepte L^{-1}, esto es, el lenguaje inverso de L.

c) Lo mismo del apartado a) pero ahora para el Lenguaje L^{-1} y el autómata A^{-1} del apartado b).

d) Construir, utilizando combinación paralela de autómatas, un autómata que acepte el Lenguaje $L \cap L^{-1}$.

50 minutos – 20 puntos

Facultad de Informática de Madrid
INFORMÁTICA TEÓRICA – GRUPO 21-M
EXAMEN 2º PARCIAL - Junio 2005

Apellidos: ____________________ Nombre: ____________________

Ejercicio 2:

Construir una MÁQUINA DE TURING (MT), que recibiendo en su cinta un número entero positivo par w, codificado en unario, se para, conteniendo en la cinta w/2, codificado también en unario:

Ej.: Para w = 111111 = 6 → w/2 = 111 = 3

C. I. # 1 1 1 1 1 1 # ($\uparrow q_0$) C. F. # 1 1 1 #

Comprobar y describir el funcionamiento de la máquina para w = 11 y w = 111111.

Sugerencia: Para el Diseño de la MT es útil pensar que, en unario, la mitad del número ww es w. Esto es, si ww = 111111, entonces, ww/2 = w = 111.

NOTA: Se valorarán positivamente los diseños más simples (5 estados son suficientes). No se considerarán, como válidos, los diseños con más de 12 estados.

45 minutos – 15 puntos

Facultad de Informática de Madrid
INFORMÁTICA TEÓRICA – GRUPO 21-M
EXAMEN 2º PARCIAL - Junio 2005

Apellidos: Nombre:

Ejercicio 3:

Dado el lenguaje: $L = \{ x \in (a, b)^* \ / \ N_a(x) = N_b(x) \}$

a) Construir un Autómata a Pila (AP), con un único estado, que por vaciado de pila acepte el lenguaje L.

b) Construir a partir de dicho AP una Gramática Independiente de Contexto (G) que genere el mismo lenguaje L.

35 minutos – 15 puntos

Facultad de Informática de Madrid
INFORMÁTICA TEÓRICA – GRUPO 21-M
EXAMEN - Septiembre 2005

Apellidos: Nombre:

Ejercicio 1:

Dada la expresión regular:

$$L = \lambda + aa^*b\,(aa^*b)^* + aa^*$$

Se pide:

a) Gramática G que genere Lenguaje L.

b) Construir un autómata finito determinista A que reconozca L.

c) Construir un autómata finito A^{-1} que acepte L^{-1}, esto es, el lenguaje inverso de L.

d) Construir un autómata finito $\overline{A}$ que reconozca $\overline{L}$.

Razonar todas las respuestas y mostrar el algoritmo utilizado.

40 minutos – 10 puntos

Facultad de Informática de Madrid

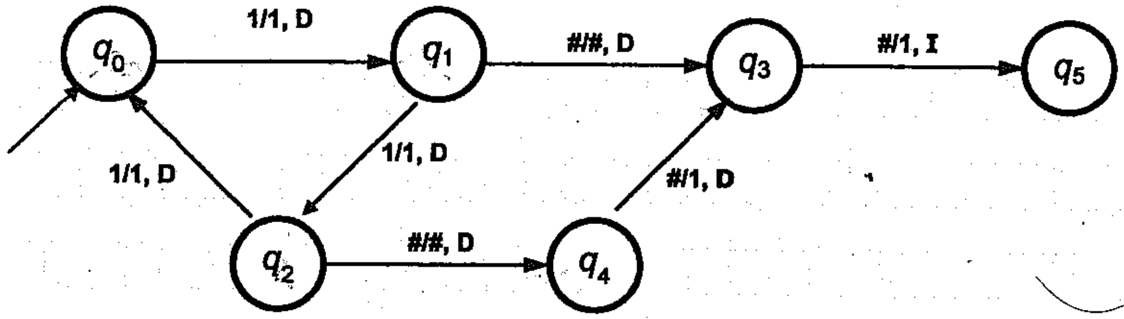
INFORMÁTICA TEÓRICA – GRUPO 21-M

EXAMEN - Septiembre 2005

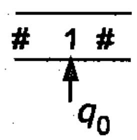
Apellidos: Nombre:

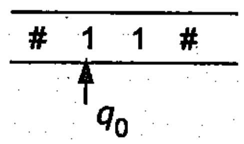
Ejercicio 2:

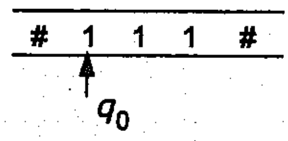
Sea la Máquina de Turing M definida según el siguiente grafo:



a) ¿Qué función aritmética sobre el alfabeto {1} calcula M? Mostrar las diferentes configuraciones a las que accede M cuando la situación inicial de la cinta y la posición de la cabeza de lectura-escritura son las siguientes:

a.1) 

a.2) 

a.3) 

b) Escribir (y describir brevemente) el contenido inicial de la cinta de una Máquina de Turing Universal cuando simula a la máquina M y ésta recibe como entrada la palabra unaria del apartado a.2). Utilizar la siguiente codificación binaria:

$q_0 \equiv 000$; $q_1 \equiv 001$; $q_2 \equiv 010$; $q_3 \equiv 011$, $q_4 \equiv 100$ $q_5 \equiv 101$

Desplazamiento a la izquierda I ≡ 0; Desplazamiento a la derecha D ≡ 1

c) Escribir (y describir brevemente) el contenido de la cinta de esa Máquina de Turing Universal tras simular el primer movimiento que realiza la máquina M con la entrada del apartado a.2).

d) Escribir (y describir brevemente) el contenido de la cinta de la Máquina de Turing Universal cuando termine de simular a la máquina M con la entrada del apartado a.2).

NOTA: Los apartados b), c) y d) se responderán en la hoja que se adjunta.

45 minutos – 10 puntos

Facultad de Informática de Madrid
INFORMÁTICA TEÓRICA – GRUPO 21-M
EXAMEN - Septiembre 2005

Apellidos: Nombre:

Continuación ejercicio 2.

Apartado b)

Apartado c) (sólo es necesario escribir lo que haya cambiado en la cinta respecto al apartado b) anterior)

Apartado d)

NOTA: **Describa lo escrito en las cintas de los apartados b), c) y d) en la carilla de atrás de esta hoja.**

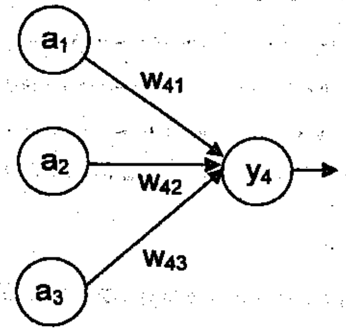
Facultad de Informática de Madrid
INFORMÁTICA TEÓRICA – GRUPO 21-M
EXAMEN - Septiembre 2005

Apellidos: Nombre:

Ejercicio 3:

Sea el Perceptron de la figura con 3 neuronas en la capa de entrada y 1 en la de salida.

Aplicar el algoritmo de aprendizaje para encontrar los valores de los pesos w_{ij} que conectan a las neuronas de la capa de entrada con la neurona de salida, de modo que sea capaz de resolver el problema mostrado en el siguiente cuadro:



a_1	a_2	a_3	y_4
-1	-1	-1	-1
-1	+1	-1	+1
+1	-1	-1	+1
+1	+1	+1	-1

Aplicar el algoritmo inicializando los pesos w_{ij} a cero y con $\alpha = 1$.

35 minutos – 10 puntos