Logica e Calcolo Proposizionale

Fondamenti di Informatica I
Corso di laurea in Ingegneria Informatica e Automatica
Sapienza Università di Roma

Domenico Lembo, Paolo Liberatore,
Alberto Marchetti Spaccamela, Marco Schaerf

Introduzione

Abbiamo già visto le rappresentazioni di numeri, caratteri e altro: ogni dato (di qualsiasi genere: numero, carattere, stringa, …) si può rappresentare usando sequenze opportune di zeri e uni

Ora: come memorizzare ed elaborare queste sequenze con circuiti elettronici

Esempio di elaborazione: somma

```
1111 ← riporti
 1011 +
 1101 =
--------
11000
```

quattro somme di singole cifre

esempio: terza colonna 1+0+1 → 0 e riporto 1

Somma singola cifra

riporto	prima cifra	seconda cifra		risultato	nuovo riporto
0	0	0	⇒	0	0
0	0	1	⇒	1	0
0	1	0	⇒	1	0
0	1	1	⇒	0	1
1	0	0	⇒	1	0
1	0	1	⇒	0	1
1	1	0	⇒	0	1
1	1	1	⇒	1	1

es, penultima riga:
riporto=1 prima cifra=1 seconda cifra=0
⇒
risultato=0 nuovo riporto=1

Tabella, a parole

il **riporto** vale uno se:

- le due cifre valgono entrambe uno, oppure
- il vecchio riporto vale uno e una delle due cifre vale uno

più formale: nuovo riporto = 1 se

- prima cifra = 1 AND seconda cifra = 1, oppure
- riporto = 1 AND poi prima cifra = 1 OR seconda cifra = 1

Operatori di combinazione

nuovo riporto = 1 se

- (prima cifra = 1) AND (seconda cifra = 1)
  OR
- (riporto = 1) AND (prima cifra = 1 OR seconda cifra = 1)

condizioni: prima cifra=1, seconda cifra=1, riporto=1

combinate con AND e OR

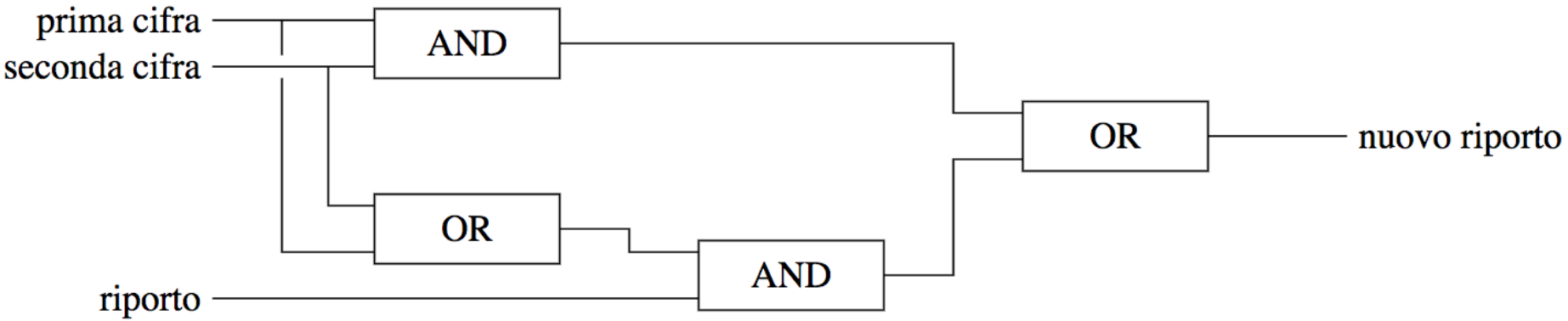
Circuito elettronico per il nuovo riporto

nuovo riporto = 1 se

- (prima cifra = 1) AND (seconda cifra = 1)
  OR
- (riporto = 1) AND (prima cifra = 1 OR seconda cifra = 1)

valori 0 e 1 rappresentati con grandezze elettriche

esistono componenti elettronici che combinano i valori in modo AND
altri che combinano in modo OR



in alto: prima cifra=1 AND seconda cifra=1 ecc.

Cifra del risultato

riporto	prima cifra	seconda cifra		risultato	nuovo riporto
0	0	0	⇒	0	0
0	0	1	⇒	1	0
0	1	0	⇒	1	0
0	1	1	⇒	0	1
1	0	0	⇒	1	0
1	0	1	⇒	0	1
1	1	0	⇒	0	1
1	1	1	⇒	1	1

altro modo di esprimerlo:

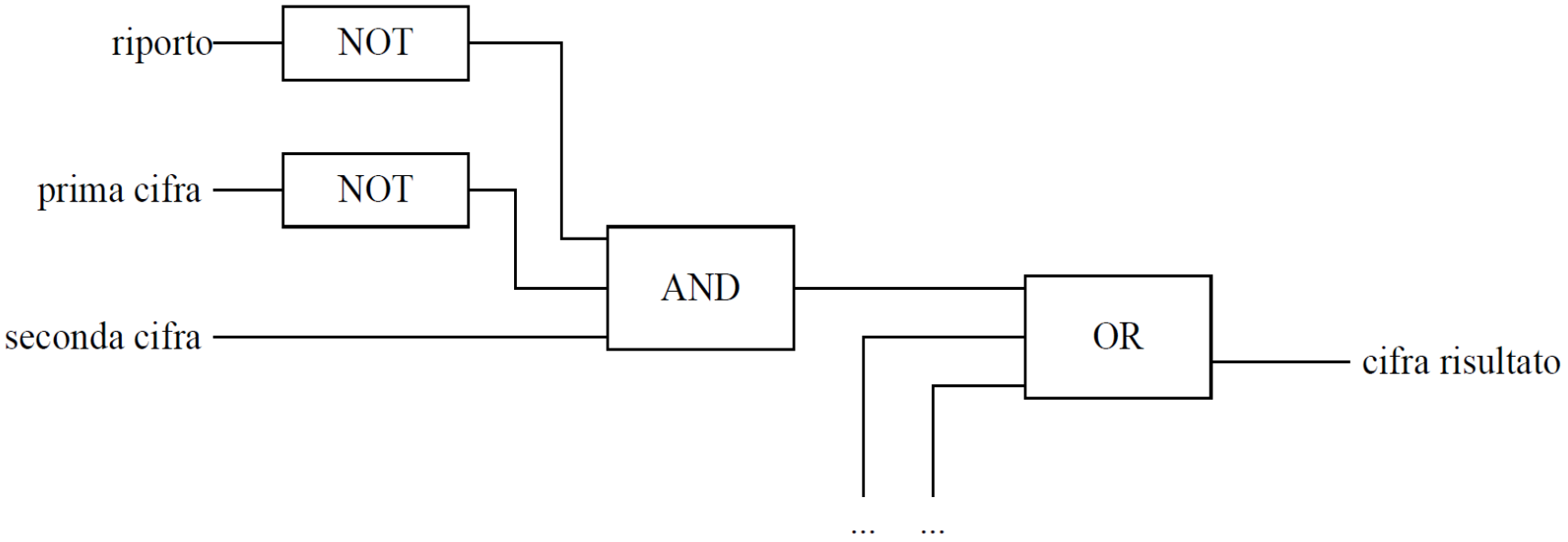
risultato=1 se

(riporto=0 AND prima cifra=0 AND seconda cifra=1) OR
(riporto=0 AND prima cifra=1 AND seconda cifra=0) OR
(riporto=1 AND prima cifra=0 AND seconda cifra=0) OR
(riporto=1 AND prima cifra=1 AND seconda cifra=1)

riporto=0 → NOT riporto=1

esiste un componente elettronico che realizza NOT

Cifra del risultato: circuito



poi ci sarebbero gli altri tre pezzi del circuito (omessi)

Elaborazioni, in generale

Dati:

sequenze di bit

Elaborazione:

da dati ottenere altri dati

quindi: da sequenze di bit ottenerne altre

come: usando NOT, AND, OR

analisi dei blocchi

Blocchi che utilizzeremo

- NOT
- AND
- OR

Tutti hanno

- ingressi 0 o 1
- uscite 0 o 1

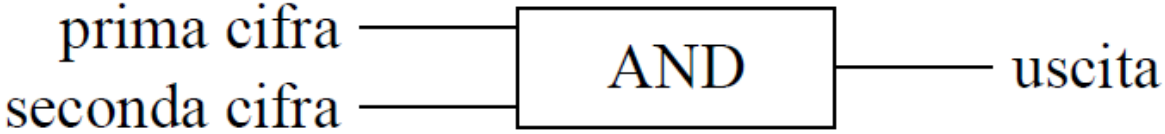
Blocco NOT

uscita 1 se riporto NOT uguale a 1

riporto	uscita
0	1
1	0

detto anche: negazione, not, !, ¬

Blocco AND

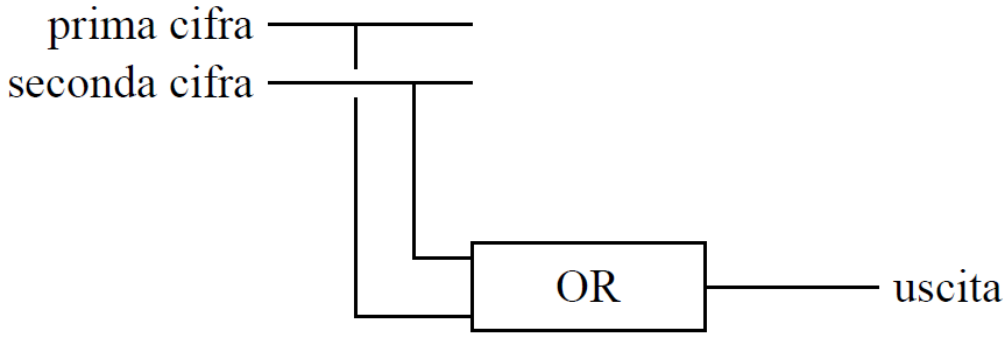


uscita 1 se prima cifra uguale a 1 AND seconda cifra uguale a 1

prima cifra	seconda cifra	uscita
0	0	0
0	1	0
1	0	0
1	1	1

detto anche: congiunzione, and, &&, ∧

Blocco OR



uscita 1 se prima cifra uguale a 1 OR (oppure) seconda cifra uguale a 1

prima cifra	seconda cifra	uscita
0	0	0
0	1	1
1	0	1
1	1	1

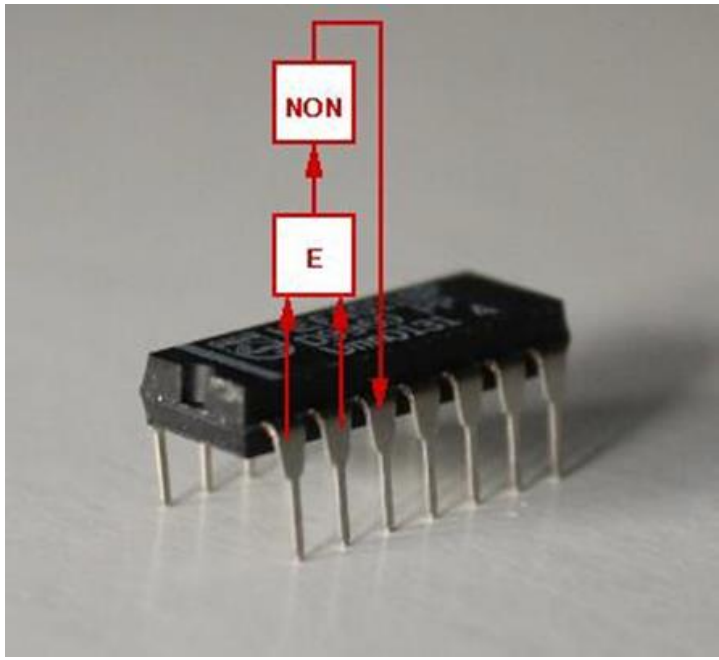
detto anche: disgiunzione, or, ||, V

Circuiti integrati

esistono componenti elettronici che realizzano AND, OR, NOT.

Esempio:

il circuito integrato 4011 contiene quattro unità AND+NOT (in figura).



Altri contengono quattro unità OR+NOT, sei unità NOT, ecc.

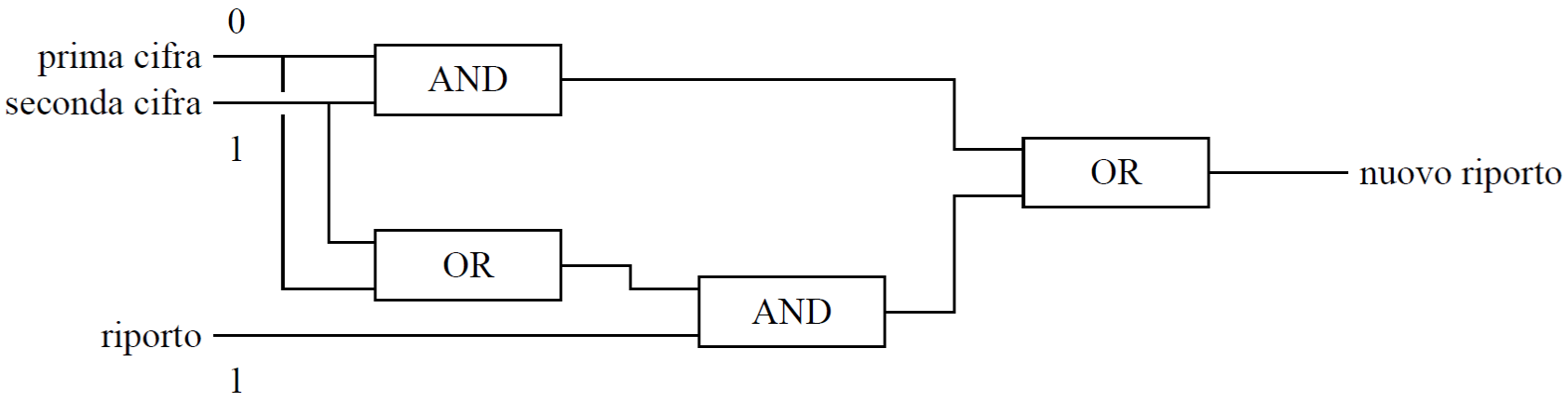
un microprocessore è un componente che contiene milioni di unità del genere

Elaborazione

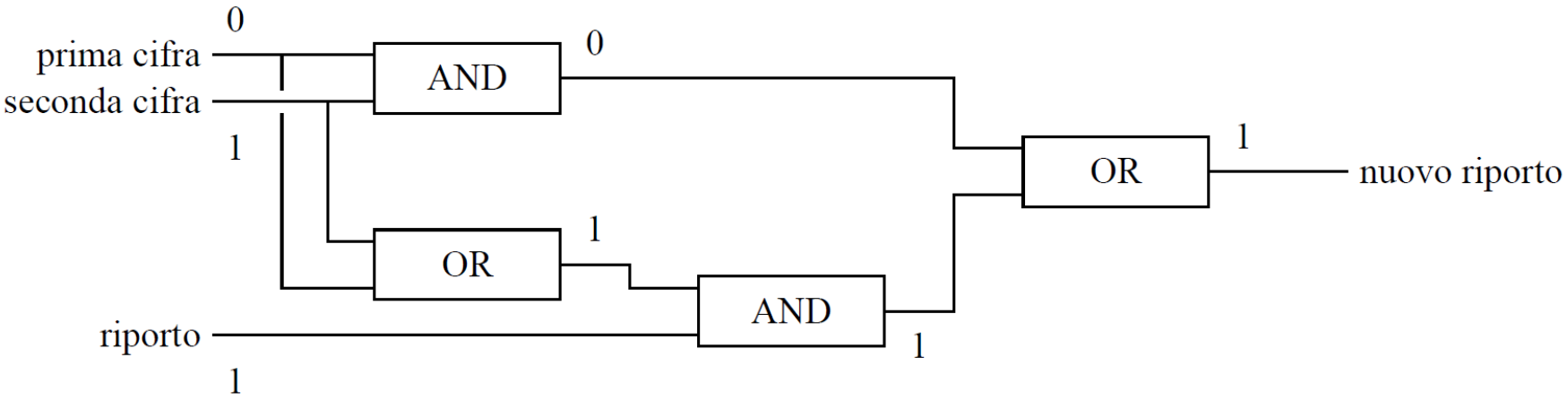
dati i valori degli ingressi...

... con le tabelle si possono calcolare i valori delle uscite anche per circuiti complicati

esempio: calcola nuovo riporto, con prima cifra zero, seconda cifra uno, riporto entrante uno



Uscita complessiva



In generale, dati gli ingressi, si può calcolare l'uscita:

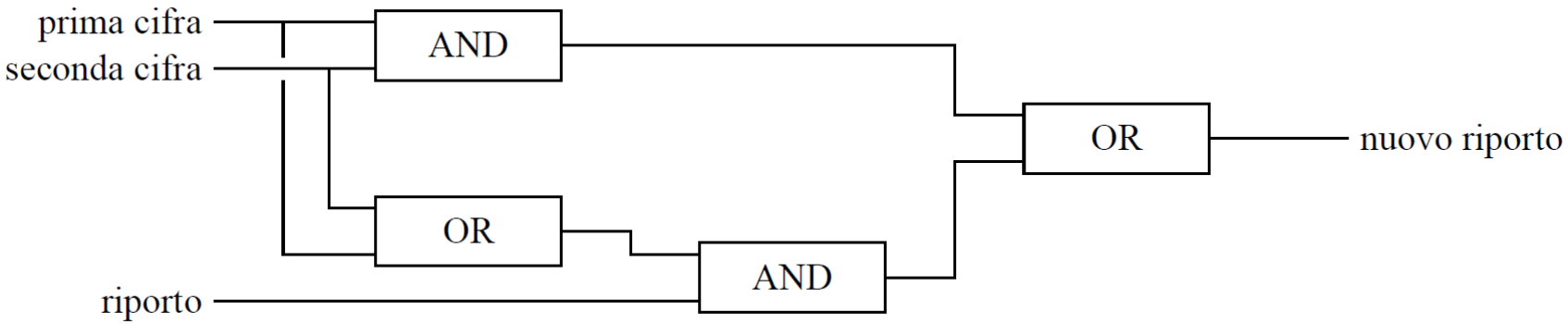
NOT: uscita 1 se l'ingresso NOT vale 1
AND: uscita 1 se primo ingresso vale 1 AND secondo ingresso vale 1
OR: uscita 1 se primo ingresso vale 1 OR secondo ingresso vale 1

Valutazione e sintesi

Ci sono 2 operazioni fondamentali dei circuiti logici:

- Valutazione
  - dato un circuito e i valori degli ingressi, calcolare l'uscita
  - si scrivono i valori 0/1 degli ingressi, si usano le tabelline di NOT, AND, OR per calcolare le uscite
- Sintesi
  - data la specifica di una elaborazione da fare (es. calcolare il riporto), disegnare il circuito usando NOT, AND e OR
  - analisi "a occhio" della tabellina somme/riporti; regola già (quasi) espressa con AND, NOT, OR
  - in generale, serve un metodo per passare dalla tabella al circuito

Dal circuito alla formula



formula complessiva:

```
(primacifra AND secondacifra) OR
((primacifra OR secondacifra) AND riporto)
```

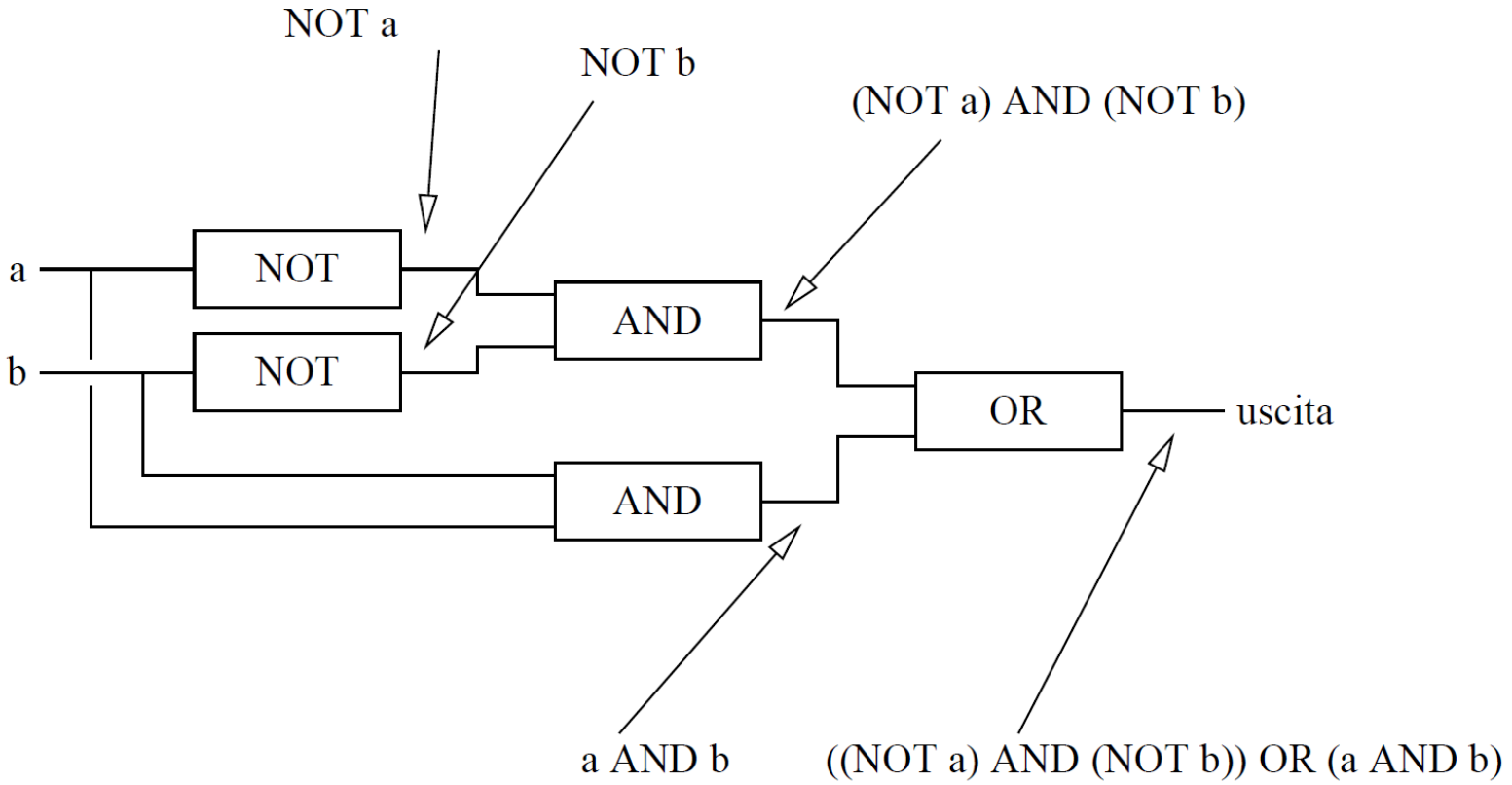
Formule e circuiti

dato un circuito, si può scrivere la formula

data una formula, si può disegnare il circuito

(visto con il riporto: disegnato il circuito a partire dalla definizione con AND e OR)

Dal circuito alla formula



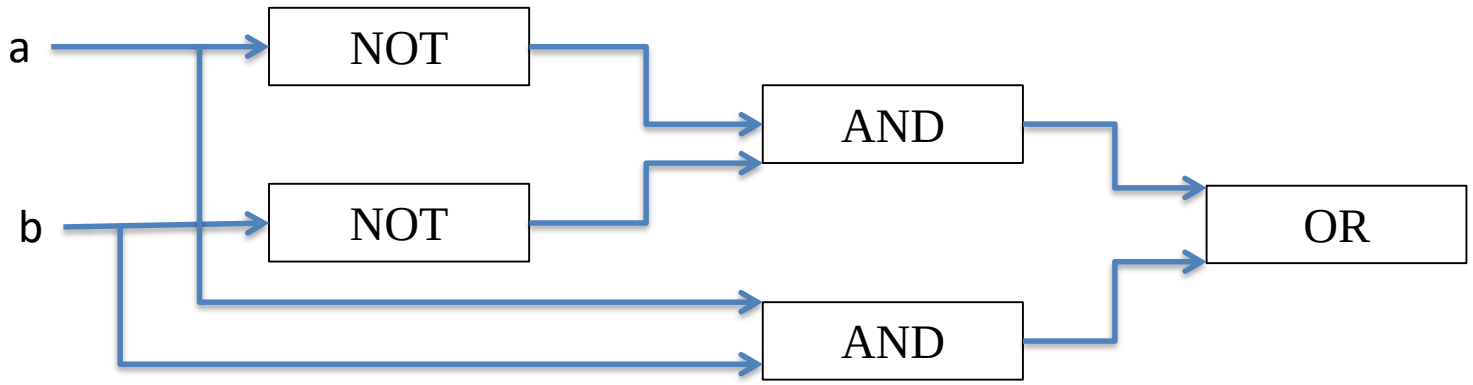
Sintesi: caso semplice

a	b		uscita
0	0	→	1
0	1	→	0
1	0	→	0
1	1	→	1

Analisi:
uscita vale uno se i due ingressi sono uguali

in questo caso, si poteva anche scrivere la condizione:
uguali = entrambi zero OR entrambi uno
con circuiti più complessi non è possibile

Sintesi del Circuito



Caso più complesso

a	b	c		uscita
0	0	0	→	0
0	0	1	→	0
0	1	0	→	1
0	1	1	→	1
1	0	0	→	1
1	0	1	→	1
1	1	0	→	0
1	1	1	→	0

a occhio: impossibile

metodo: in quali casi l'uscita vale 1?

Caso più complesso

a	b	c		uscita
0	0	0	→	0
0	0	1	→	0
0	1	0	→	1
0	1	1	→	1
1	0	0	→	1
1	0	1	→	1
1	1	0	→	0
1	1	1	→	0

Per ogni riga con uscita 1:

a==0 → (NOT a) a==1 → a b==0 → (NOT b) ecc.

queste sono poi composte in AND. Ad es., (NOT a) AND b AND (NOT c)

Tutte le formulecosì ottenute si compongono in OR

```
((NOT a) AND b AND (NOT c))   OR   (linea 010)
((NOT a) AND b AND c)         OR   (linea 011)
(a AND (NOT b) AND (NOT c))   OR   (linea 100)
(a AND (NOT b) AND c)              (linea 101)
```

Semplificazione

```
(a AND (NOT b) AND (NOT c)) OR (a AND (NOT b) AND c)
```

la formula precedente sta a indicare due possibilità:

1. `a AND (NOT b) AND (NOT c)`
2. `a AND (NOT b) AND c`

ossia `a AND (NOT b)`, e poi o `c` oppure `NOT c`

```
(a AND (NOT b) AND (NOT c)) OR (a AND (NOT b) AND c) =
(a AND (NOT b)) AND (c OR (NOT c)) =
a AND (NOT b)
```

lo stesso per l'altra sottoformula

Formula semplificata

```
((NOT a) AND b AND (NOT c))   OR
((NOT a) AND b AND c)         OR
(a AND (NOT b) AND (NOT c))   OR
(a AND (NOT b) AND c)         =
((NOT a) AND b) OR (a AND (NOT b))
```

circuito più piccolo!!

problema: semplificazione «a occhio»

con *100000* variabili invece di tre?

servono:

1) regole per semplificare
2) un metodo automatico

Logica proposizionale

valori: `falso`, `vero` (oppure: `0` e `1`)

variabili: `a, b, c, …`

connettivi: `¬ ∧ ∨` (cioè NOT, AND, OR)

formula (es):

`(a ∨ ¬b) ∧ c ∧ ¬a`

Regole logica proposizionale

- $a \wedge a = a$; $a \vee a = a$ (idempotenza dell'AND e dell'OR)
- $a \wedge b = b \wedge a$; $a \vee b = b \vee a$ (commutativa dell'AND e dell'OR)
- $a \wedge (b \wedge c) = (a \wedge b) \wedge c$; $a \vee (b \vee c) = (a \vee b) \vee c$ (associative)
- $a \wedge 0 = 0$
- $a \wedge 1 = a$
- $a \vee 0 = a$
- $a \vee 1 = 1$
- $a \wedge \neg a = 0$
- $a \vee \neg a = 1$
- $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$ (prop. Distributiva dell'AND rispetto all'OR)
- $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$ (prop. Distributiva dell'OR rispetto all'AND)
- $\neg(a \wedge b) = \neg a \vee \neg b$ (primo teorema di De Morgan)
- $\neg(a \vee b) = \neg a \wedge \neg b$ (secondo teorema di De Morgan)
- …

Interpretazioni e modelli

esempio di formula:

$(a \vee \neg b) \wedge c \wedge \neg a$

valore: 0 o 1 a seconda dei valori delle variabili

valore delle variabili = interpretazione

Una interpretazione assegna il valore 0 o 1 ad ogni variabile

esempio:

{a=1, b=0, c=0} è una interpretazione

{a=0, b=0, c=1} è un'altra interpretazione

Valutazione di una formula

$(a \vee \neg b) \wedge c \wedge \neg a$

data l'interpretazione: $\{a=1, b=0, c=0\}$

la formula vale:

$(a \vee \neg b) \wedge c \wedge \neg a =$
$(1 \vee \neg 0) \wedge 0 \wedge \neg 1 =$
$(1 \vee 1) \wedge 0 \wedge 0 =$
$1 \wedge 0 \wedge 0 =$
0

con questa interpretazione la formula vale 0

Valutazione di una formula

$(a \vee \neg b) \wedge c \wedge \neg a$

altra interpretazione: $\{a=0, b=0, c=1\}$

$(a \vee \neg b) \wedge c \wedge \neg a =$
$(0 \vee \neg 0) \wedge 1 \wedge \neg 0 =$
$(0 \vee 1) \wedge 1 \wedge 1 =$
$1 \wedge 1 \wedge 1 =$
1

questa volta la formula vale 1

Modello di una formula

stessa formula

interpretazione {a=1, b=0, c=0} → formula vale 0

interpretazione {a=0, b=0, c=1} → formula vale 1

il valore della formula dipende dall'interpretazione

una interpretazione che rende vera la formula si chiama **modello** della formula

Uso dei modelli

I modelli servono a definire:

1. data una interpretazione e una formula, valutare la formula (= trovare l'uscita di un circuito dati i suoi ingressi)
2. dato un insieme di interpretazioni, trovare la formula che vale 1 solo per quelli (= sintesi di un circuito, a partire dalle righe 1 di una tabella)
3. data una formula, decidere se ha un modello
4. trovare le conseguenze logiche di una formula

Ultimi due problemi: ancora non visti

ora: uno per volta i punti 1-4

Usi della logica

1. **valutare una formula data una interpretazione**
2. sintetizzare una formula
3. trovare (se esiste) un modello di una formula
4. effettuare deduzioni

iniziamo dal primo punto

Valutazione di una formula

interpretazione {a=1, …} → si mette 1 al posto di a nella formula

Usato per:

- Valore di una condizione in Python
- Composizione di condizioni in Python
- Verifica errori
- Test parità
- …..

Usi della logica

1. valutare una formula data una interpretazione
2. **sintetizzare una formula**
3. trovare (se esiste) un modello di una formula
4. effettuare deduzioni

Sintesi

formula → circuito è facile

tabella → formula già visto:

A	B	….	Uscita
0	0	….	….
1	0	….	….
1	1	….	….

Si prendono le righe con uscita 1, ognuna è un modello (valori delle variabili per cui la formula vale 1)

Si compone l'OR (V) dei modelli

Semplificazione

sulla tabella:

se due righe con uscita 1 differiscono solo per una variabile, si possono prendere solo le altre variabili e formare così un unico Λ invece di due Λ in $\vee$ (cf. slide 25)

vale anche su gruppi:

se in un gruppo di righe le variabili A hanno tutte lo stesso valore e le variabili B tutti i valori possibili, si può prendere solo la AND dei valori A

Questa semplificazione corrisponde alla applicazione ripetuta della regola:

$(F \Lambda c) \vee (F \Lambda \neg c) = F$

metodi automatici per trovare righe simili

(ma: semplificazione ottima è un problema intrattabile)

Semplificazione: esempio

a b c	uscita
0 0 0 →	0
0 0 1 →	0
0 1 0 →	1
0 1 1 →	0
1 0 0 →	1
1 0 1 →	1
1 1 0 →	1
1 1 1 →	1

disegnare il circuito che realizza questa funzione

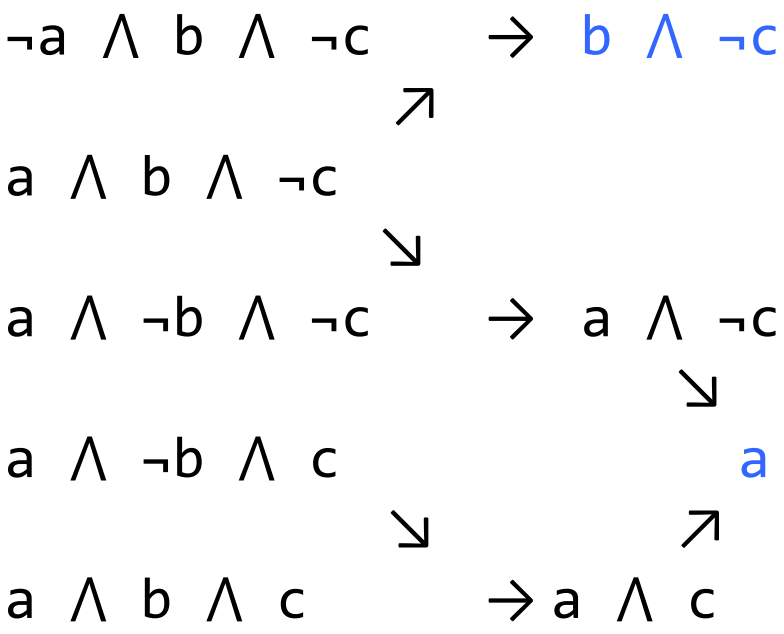
prima riga:

a=0 b=1 c=0 → ¬a ∧ b ∧ ¬c

principio: variabile=0 diventa not variabile
variabile=1 diventa variabile

010 → ¬a ∧ b ∧ ¬c
100 → a ∧ ¬b ∧ ¬c
101 → a ∧ ¬b ∧ c
110 → a ∧ b ∧ ¬c
111 → a ∧ b ∧ c

Semplificazioni



risultato: (b ∧ ¬c) ∨ a

nota: a ∧ ¬c e a ∧ c si semplificano in a

domanda: ma a ∧ b ∧ ¬c non è stato "preso due volte"?

Modelli presi due volte

dato che F = F ∨ F:

(¬a∧b∧¬c) ∨ (a∧b∧¬c) ∨ (a∧¬b∧¬c) ∨ …

(¬a∧b∧¬c) ∨ (a∧b∧¬c) ∨ (a∧b∧¬c) ∨ (a∧¬b∧¬c) ∨ …

|----------| |----------|

primi due si semplificano in: b ∧ ¬c

secondi due si semplificano in: a ∧ ¬c

Usi della logica

1. valutare una formula data una interpretazione
2. sintetizzare una formula
3. **trovare (se esiste) un modello di una formula**
4. effettuare deduzioni

Soddisfacibilità

formula F

verificare se ha un modello

oppure: trovare un modello (se esiste)

modello = valori di ingresso che producono 1 in uscita

Esempio: $a \wedge b \wedge \neg a$

vale 1 solo se sia a che ¬a valgono 1

impossibile

formula insoddisfacibile

Soddisfacibilità: esempi

$\neg b \wedge (a \vee b)$

con a=1 e b=0 la formula è vera → soddisfacibile

non è sempre così facile!

$(a \vee \neg b) \wedge \neg a \wedge (b \vee a)$

guardando tutti i possibili valori di a e b:

$0\ 0 \rightarrow (0 \vee \neg 0) \wedge \neg 0 \wedge (0 \vee 0) = 0$

$0\ 1 \rightarrow (0 \vee \neg 1) \wedge \neg 0 \wedge (1 \vee 0) = 0$

$1\ 0 \rightarrow (1 \vee \neg 0) \wedge \neg 1 \wedge (0 \vee 1) = 0$

$1\ 1 \rightarrow (1 \vee \neg 1) \wedge \neg 1 \wedge (1 \vee 1) = 0$

formula sempre falsa → insoddisfacibile

Numero di interpretazioni

n variabili = 2^n interpretazioni (tante quanti sono i numeri rappresentabili con n bit)

n	2^n
1	2
2	4
3	8
4	16
5	32
…	
10	1024
11	2048
…	

Crescita esponenziale

Metodi automatici

$(a \vee \neg b) \wedge \neg a \wedge (b \vee a)$

applicando la regola $(F \vee c) \wedge (F \vee \neg c) = F$:

$(a \vee \neg b) \wedge \neg a \wedge (b \vee a) =$
$(a \vee \neg b) \wedge \neg a \wedge (b \vee a) \wedge a = 0$

infatti: $a \wedge \neg a = 0$

formula insoddisfacibile

Risoluzione

Generalizzazione del metodo precedente:

la formula deve essere del tipo:

$(a \vee c \vee \neg b) \wedge (b \vee d) \wedge \ldots$

ossia un $\wedge$ di sottoformule che sono $\vee$ di variabili e variabili negate.

La forma precedente si chiama *Forma Normale Congiuntiva (CNF).*
Teorema*:* Ogni formula può sempre essere trasformata in una formula equivalente espressa in questa forma.

si applica la ***regola di risoluzione***:

$(F \vee \neg b) \wedge (b \vee D) \Rightarrow$ aggiungi $F \vee D$

se formula insoddisfacibile: alla fine si ottiene 0

metodo automatico per vedere se una formula è soddisfacibile: metodo di *risoluzione*

Esempio

$(\neg a \vee c \vee d) \wedge (a \vee d \vee e) \wedge (\neg a \vee \neg c) \wedge \neg d \wedge \neg e$

Per comodità di esposizione numeriamo le formule in AND (dette *clausole*)

$(\neg a \vee c \vee d)$	(1)
$(a \vee d \vee e)$	(2)
$(\neg a \vee \neg c)$	(3)
$\neg d$	(4)
$\neg e$	(5)

Applichiamo la regola di risoluzione in tutti i possibili modi ed aggiungiamo

$(c \vee d \vee e)$	(6)	da (1) e (2)
$(d \vee e \vee \neg c)$	(7)	da (2) e (3)
$(\neg a \vee c)$	(8)	da (1) e (4)
$(a \vee e)$	(9)	da (2) e (4)
$(a \vee d)$	(10)	da (2) e (5)
$(\neg a \vee d)$	(11)	da (1) e (3)

Applichiamo la risoluzione considerando il nuovo insieme di clausole e otteniamo ad es.,

d	(12)	da (10) e (11)

(12) e (4) generano, per risoluzione, una clausola vuota (che vale $\emptyset$). La formula è insoddisfacibile.

Soddisfacibilità: altri metodi

oltre a risoluzione:

- local search
- DPLL
- tableau
- …

trovano un modello (se esiste) di una formula

Soddisfacibilità: uso pratico

problema → formula

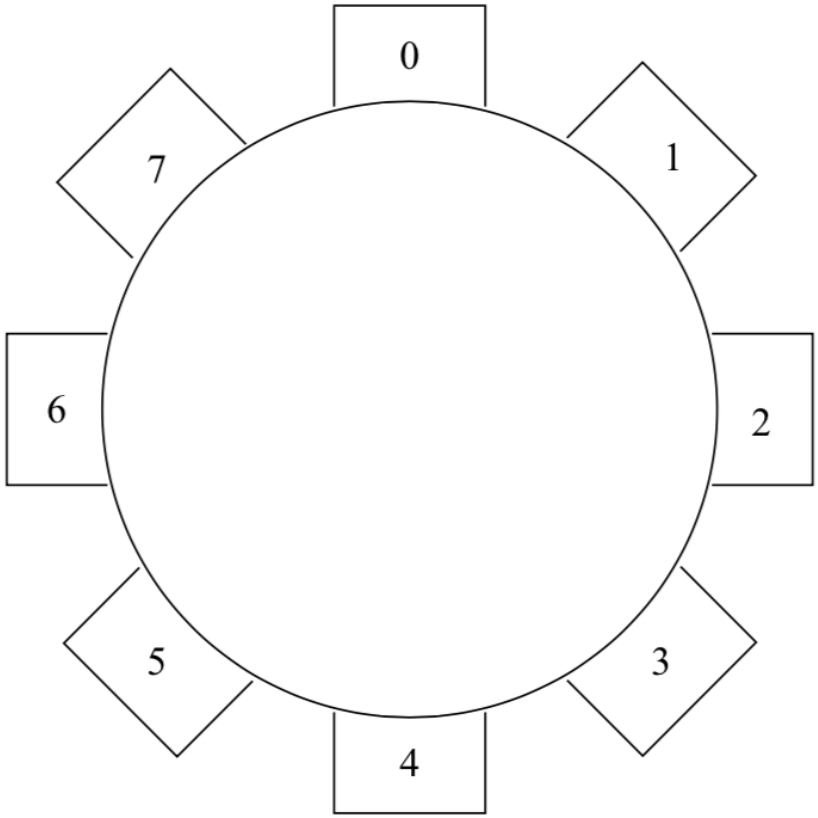
formula → modello

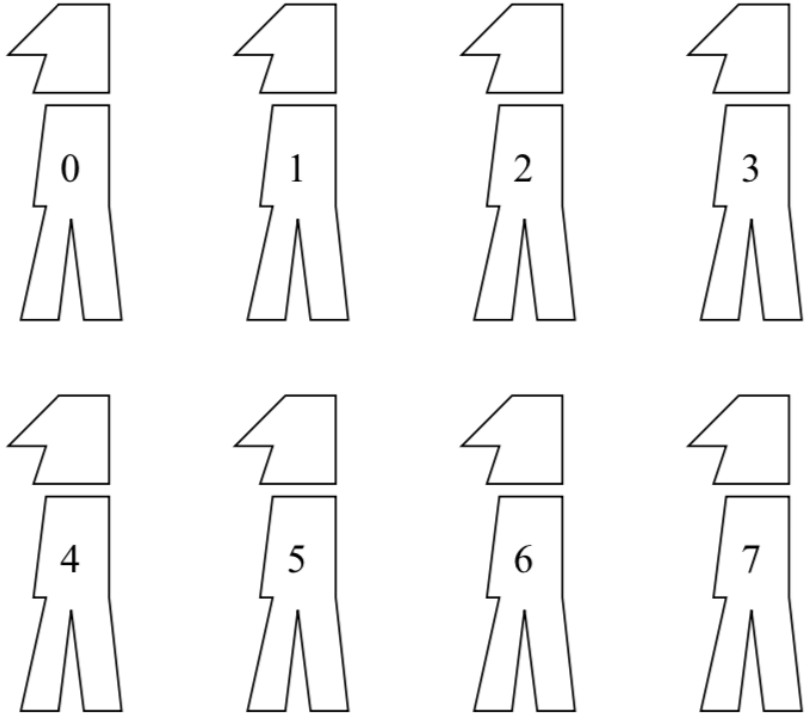
il modello è una soluzione del problema

secondo passo: si fa con un programma per la soddisfacibilità

Esempio: i cavalieri della tavola rotonda

Re Artù deve far sedere i suoi 150 cavalieri alla tavola rotonda, ma deve essere sicuro che nessuno sia seduto vicino a un rivale. I posti sono 150 e le rivalità sono note.





Codifica della soluzione

soluzione = posizione per ogni cavaliere

in binario:

- un numero da 0 a 149 per ogni cavaliere, oppure
- un bit per ogni cavaliere e posto

usiamo il secondo sistema

formula che esprime il problema

- ogni cavaliere ha un posto
- mai due cavalieri nello stesso posto
- mai rivali vicini

vediamo nel dettaglio come si fa

Codifica binaria della soluzione

una variabile per ogni cavaliere x e posto a

`x_siede_a` = cavaliere x siede al posto a

Esempi:

`2_siede_4 = 1` significa che il cavaliere 2 sta al posto 4

`9_siede_121 = 0` significa che il cavaliere 9 *non* sta al posto 121

(= sta a un altro posto)

Formula che esprime la soluzione

Ogni cavaliere ha un posto:

$$x\_siede\_0 \vee x\_siede\_1 \vee x\_siede\_2 \vee \ldots \vee x\_siede\_149$$

no due cavalieri nello stesso posto: per ogni posto a

$$\neg(x\_siede\_a \wedge y\_siede\_a)$$

no rivali vicini: se x e y sono rivali

$\neg(x\_siede\_a \wedge y\_siede\_b)$ dove b=(a+1)%150:

Non inviterò mai a cena 150 cavalieri

- era solo un esempio!
- forma generale:
  - assegnare delle risorse (es. posti) rispettando dei vincoli (es. no rivali vicini)
  - assegnare dei camion per trasportare delle merci
  - assegnare le aule in modo che il canale L-Z non sia mai in aula 1
  - …

Usi della logica

- valutare una formula data una interpretazione
- sintetizzare una formula
- trovare (se esiste) un modello di una formula
- effettuare deduzioni

Implicazione

altra applicazione della logica proposizionale: effettuare ragionamenti

Esempio:

- Gino o sta a casa o sta al bar
- Gino non sta a casa
- conseguenza: Gino sta al bar

si esprime in logica proposizionale:

In logica

- Variabili:
  - `casa=1` denota Gino sta a casa
  - `bar=1` denota Gino sta al bar
- Formule:
  - Gino o sta a casa o sta al bar: `casa ∨ bar`
  - Gino non sta a casa: `¬casa`
  - Gino sta al bar: `bar`
- Ragionamento:
  - *se* `casa ∨ bar` *e* `¬casa` *allora* `bar`

In generale

se a∨b *e* ¬a *allora* b

a	b	a∨b	¬a
0	0	0	1
0	1	1	1
1	0	1	0
1	1	1	0

riga gialla: unica in cui vale sia a∨b=1 che ¬a=1

in quella riga b=1

quindi: implicazione valida

Implicazione

- Forma generale dell'implicazione:

  se formula, formula, … formula *allora* formula

- per ogni valore delle variabili:
  - se le formule prima di "*allora*" valgono tutte 1
  - Ne consegue che la formula dopo vale 1

Semantica:

$A, B, C \models Z$

se A=1, B=1 e C=1 allora Z=1

Formalmente: ogni interpretazione per cui A=1, B=1 e C=1 è tale per cui Z=1

Implicazione

Esempi:

$a \vee b, \neg a \models b$

$a \models a \vee b$

$a \wedge b \models b$

$a \vee b, a \vee \neg b \models a$

$a \vee b, a \vee \neg b, \neg a \vee b \models a \wedge b$

$a \vee b, \neg b \vee c \models a \vee c$

Controesempi:

implicazione non vera: $a \vee b \not\models a$

implicazione non vera: $a \vee b, b \vee c \not\models a \vee c$

implicazione non vera: $a \not\models a \wedge b$

Esempio di dimostrazione

$a \vee b, \neg b \vee c \models a \vee c$ è vero?

a	b	c	$a \vee b$	$\neg b \vee c$	$a \vee c$
0	0	0	0	1	0
0	0	1	0	1	1
0	1	0	1	0	0
0	1	1	1	1	1
1	0	0	1	1	1
1	0	1	1	1	1
1	1	0	1	0	1
1	1	1	1	1	1

linee gialle: quelle dove $a \vee b=1$ e $\neg b \vee c=1$

in tutte vale anche $a \vee c=1$

l'implicazione è quindi valida

Esempio di confutazione

$a \vee b, b \vee c \models a \vee c$ è vero?

a	b	c	$a \vee b$	$b \vee c$	$a \vee c$
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	1	1	0
0	1	1	1	1	1
1	0	0	1	0	1
1	0	1	1	1	1
1	1	0	1	1	1
1	1	1	1	1	1

Linea rossa: $a \vee b = 1$ e $b \vee c = 1$ ma $a \vee c = 0$

l'implicazione quindi NON è valida

altro modo di dire "se"

Gino o è a casa o è al bar

=

se Gino non è a casa allora è al bar

`casa ∨ bar` uguale a "se non casa allora bar"

viceversa: "se non studio verrò bocciato":

- se `¬studio` allora `bocciato`
- `studio ∨ bocciato`

(= o studio o verrò bocciato)

Implicazione materiale

- è una formula che si esprime con il simbolo → e consente di esprimere in modo esplicito formule nella forma *se* _ *allora* _
- è una **abbreviazione**: _→_ indica ¬_∨_
- a → b indica quindi ¬a ∨ b
- quindi:
  - `casa ∨ bar` si può anche scrivere come `¬casa → bar`
  - `studio ∨ bocciato` si può anche scrivere come `¬studio → bocciato`

Implicazione materiale vs conseguenza logica

- $aVb \land \neg bVc \models aVc$ sempre vero (aVc è una conseguenza logica di $aVb \land \neg bV$)
- $a \rightarrow b$ è una formula che risulta vera o falsa a seconda del valore assegnato ad a ed a b. In particolare, se ad a assegniamo 0 la formula è vera qualunque sia il valore di b. Invece è falsa se ad a assegniamo 1 ed a b assegniamo 0.
- a seconda del contesto in cui è specificata la formula, l'implicazione può essere "sensata" oppure no. Ad esempio se a rappresenta il concetto "studio" e b indica "bocciato", $\neg a \rightarrow b$ è una implicazione materiale che rappresenta una situazione reale, ma se a denota "vittoria" e b denota "sconfitta", l'implicazione potrebbe non catturare correttamente la realtà, perché in molti sport esiste anche il pareggio.
- la semantica della logica non include informazioni sul contesto

Conseguenze logiche

¬casa ⊭ bar

lo si vede dalla tabella di verità:

casa	bar	¬casa	bar
0	0	1	0
0	1	1	1
1	0	0	0
1	1	0	1

righe gialle: dove ¬casa è vero

in una bar è falso

¬casa ⊭ bar

bar non è *conseguenza logica* di ¬casa

Indipendenza dal contesto

perché ¬casa ⊨ bar è falso?

- formula1 ⊨ formula2 indica un'implicazione vera *a prescindere dal significato delle variabili*
- come aggiungere informazioni relative al significato delle variabili?

Informazioni relative al contesto

¬casa ⊭ bar

ma:

¬casa ∧ (casa∨bar) ⊨ bar

oppure:

¬casa ∧ (¬casa→bar) ⊨ bar

le implicazioni che valgono solo nello specifico contesto vanno inserite come formule A → B

Informazioni generali e specifiche

la logica proposizionale distingue fra:

- ragionamento logico

  indipendente dal significato delle variabili: ⊨
- informazioni specifiche

  relative al contesto, quindi legate al significato delle variabili: ¬_∨_, →, ecc.

ma *non* distingue fra:

- informazioni sempre vere nel contesto considerato

  es: è sempre vero che Gino sta o a casa o al bar
- informazioni vere solo nello specifico caso

  es: oggi Gino non è a casa

in entrambi i casi sono formule: casa∨bar (oppure ¬casa→bar) e ¬casa

Implicazione e soddisfacibilità

$A \models B$ vero se ogni interpretazione per cui $A=1$ è tale per cui $B=1$

riformulato:

non esiste una interpretazione per cui $A=1$ ma $B=0$

equivale alla *non soddisfacibilità* di $A \wedge \neg B$

$a \vee b, \neg b \vee c \models a \vee c$ se e solo se $a \vee b \wedge \neg b \vee c \wedge \neg(a \vee c)$ non soddisfacibile

Vale anche il contrario

Equivalenza

due formule sono equivalenti se hanno gli stessi modelli (si può vedere sulle tabelle di verità)

equivalenze di De Morgan

$\neg(a \vee b)$ equivalente a $\neg a \wedge \neg b$

$\neg(a \wedge b)$ equivalente a $\neg a \vee \neg b$

Equivalenze di De Morgan: dimostrazione

per ¬(aVb) e ¬a ⋀ ¬b:

a	b	¬(aVb)	¬a⋀¬b
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	0

ultime due colonne uguali: formule equivalenti

stesso sistema per l'altra equivalenza

Equivalenza e implicazione

se A e B sono equivalenti:

$A \models B$

$B \models A$

e viceversa

Logica e computer

visto finora:

uso di blocchetti NOT, AND, OR per realizzare la somma

in generale: circuiti ¬, ∧, ∨ realizzano qualsiasi funzione (sintesi) in un computer,

i dati sono:

- elaborati
- memorizzati

Memorizzazione dati

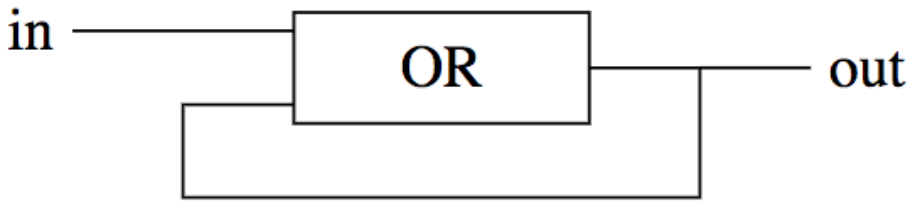
elaborazione = funzioni realizzate con ¬, Λ, V

Memorizzazione:

circuito che memorizza un bit

memorizzare 64 bit = 64 di questi circuiti

Retroazione



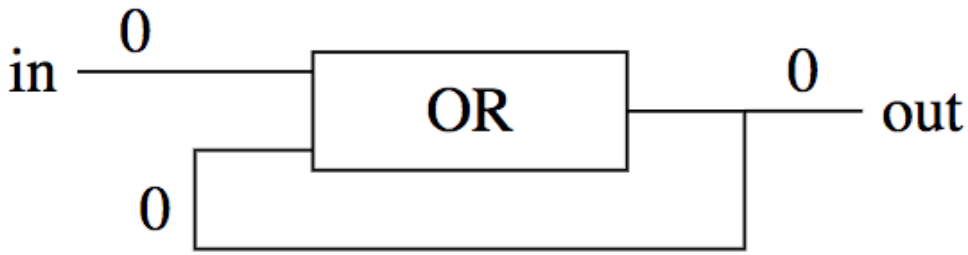
l'uscita ritorna indietro

come funziona?

i blocchi non sono istantanei

una variazione dell'ingresso ci mette un certo tempo (per quanto piccolo) a riflettersi sull'uscita

Stato iniziale



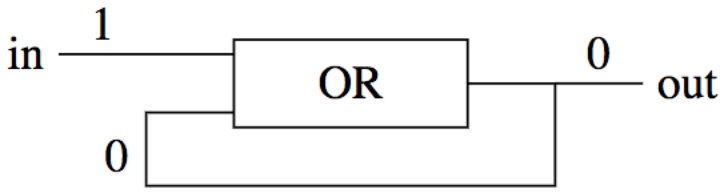
assumendo in=0 e out=0:

0 OR 0 = 0

configurazione stabile: l'uscita si mantiene a 0

se si cambia il valore di **in** ?

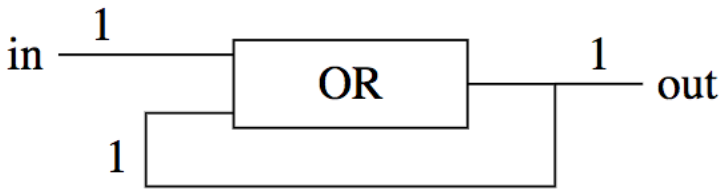
cambia in: diventa 1



1 OR 0 = 1

Cambia il valore dell’uscita

Propagazione del valore

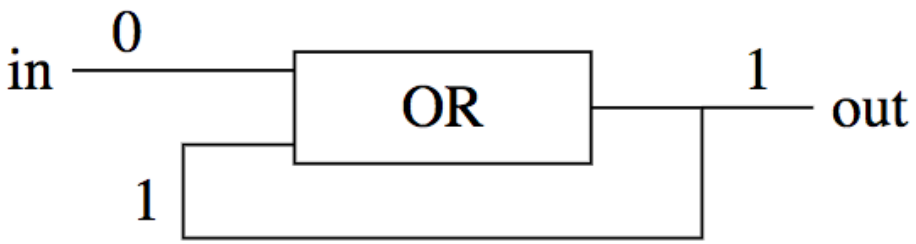


out passa a 1

configurazione stabile: 1 OR 1 = 1 out

mantiene il valore

in torna a zero



0 OR 1 = out

Rimane a 1

Circuito con retroazione: analisi

- inizio: in=0 out=0
- cambia in=1 → out=1
- cambia in=0 → out=1

una volta a 1, l'uscita *mantiene il valore*

circuito che memorizza l'arrivo di un 1 in ingresso

ricorda se è arrivato un uno

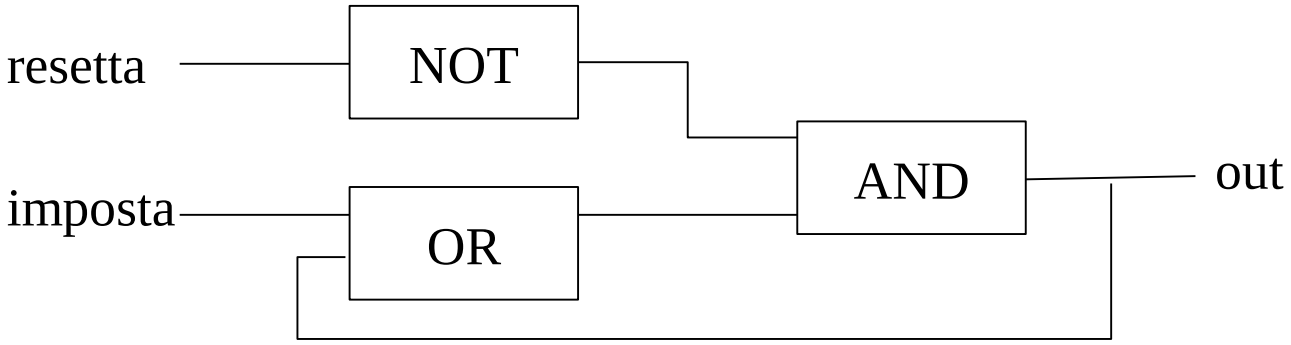
Memoria resettabile

una volta che out=1, il valore dell'uscita non cambia

vera memoria = ci si può mettere 0 oppure 1

serve un modo per riportare out a zero

Blocco di azzeramento

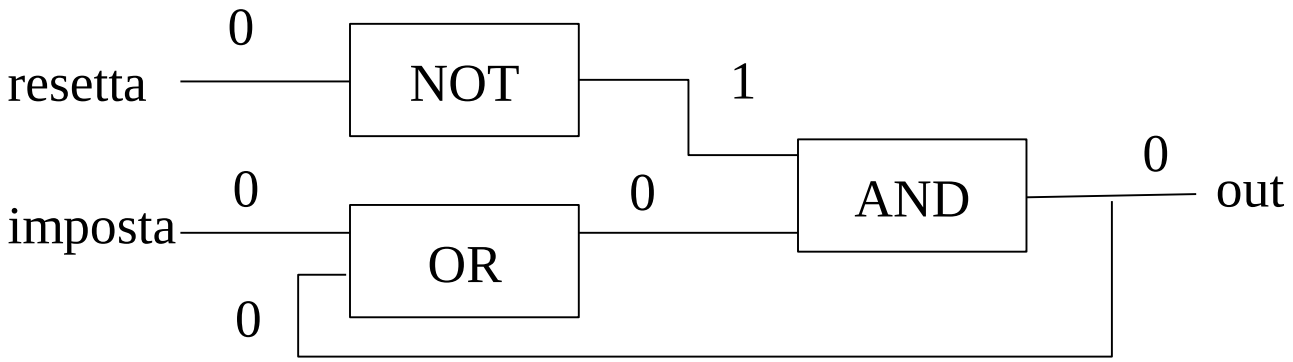


reset=1 →

primo ingresso di AND va a zero →

out va a zero

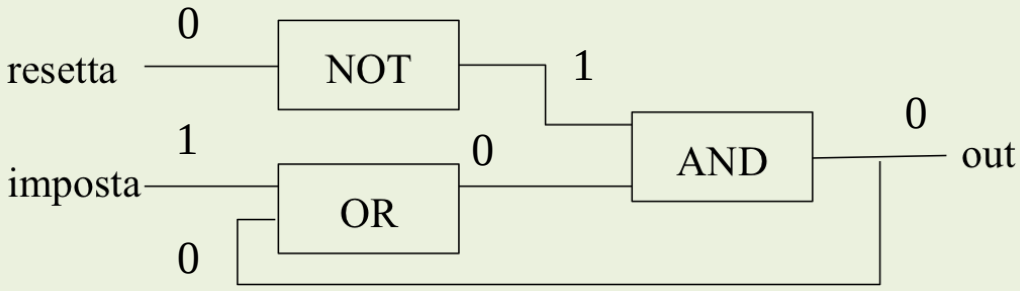
Situazione iniziale



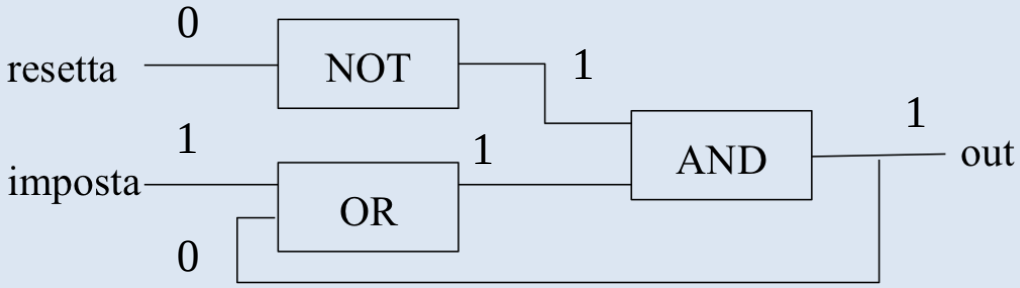
ingressi: imposta=0 resetta=0

si assume uscita = 0

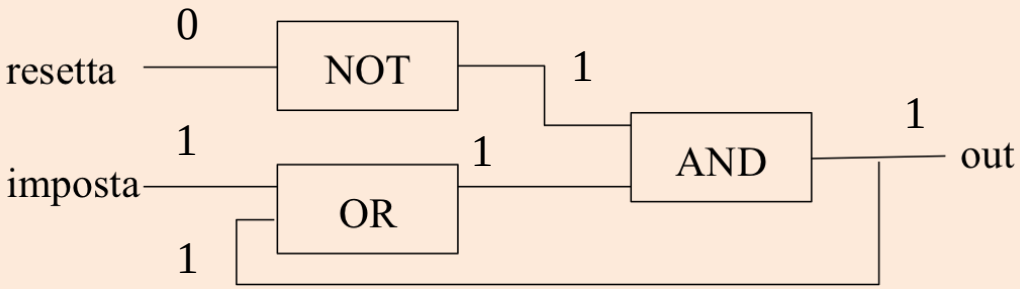
Imposta va a 1



tempo t_0 : ingressi: imposta=1 e resetta=0



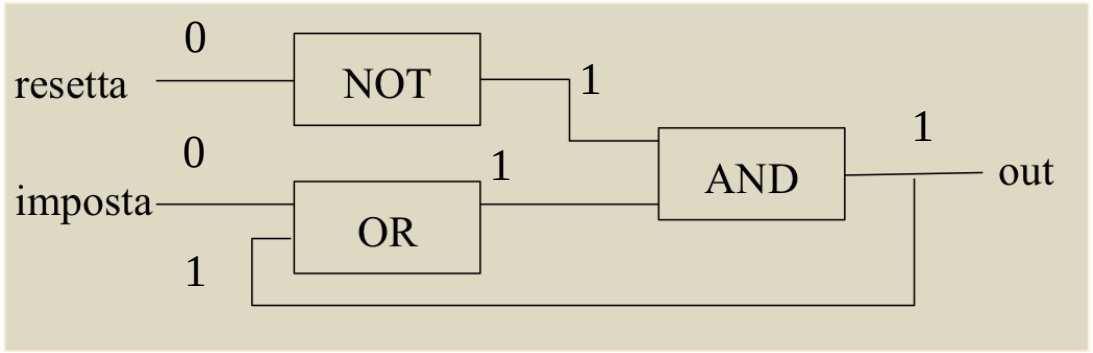
tempo t_1 : 1 OR qualsiasi = 1 1 AND 1 = 1 l'uscita va a 1



tempo t_2 : propagazione della retroazione

Memorizzazione

Imposta torna a 0



ingressi: imposta=0 resetta=0

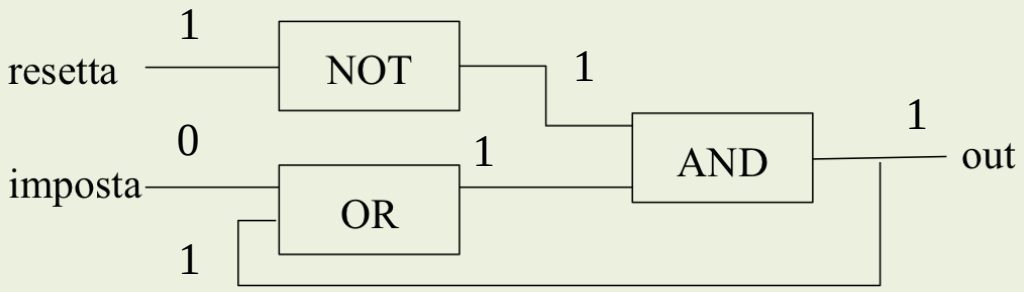
0 OR 1 = 1

1 AND 1 = 1

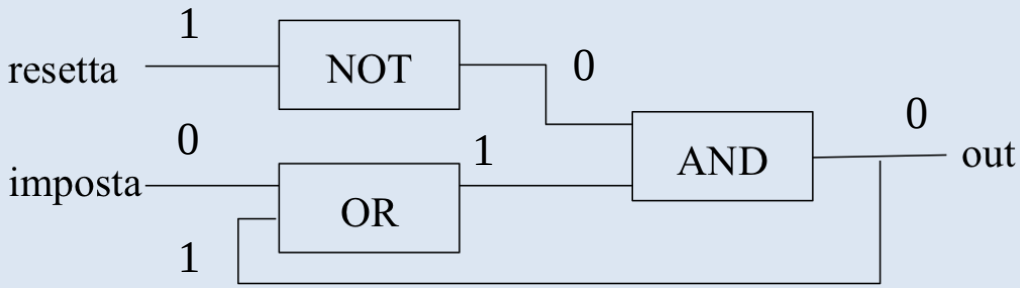
l'uscita si mantiene a 1

Nota: in questo caso gli ingressi ed uscite nel circuito sono uguali nei tempi t_0, t_1 e t_2

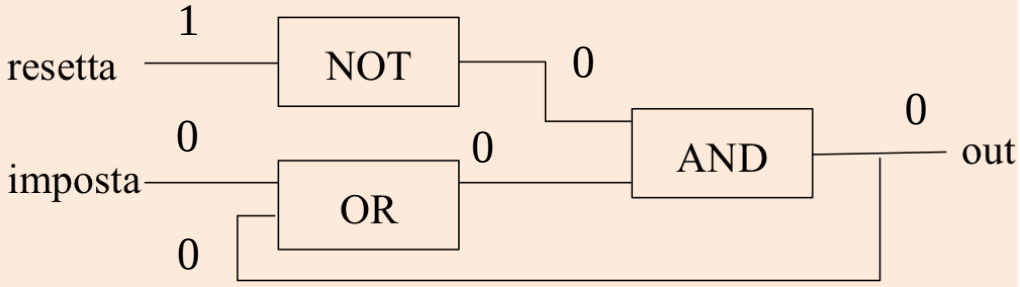
Resetta diventa 1



tempo t_0 : ingressi: imposta = 0 e resetta = 1



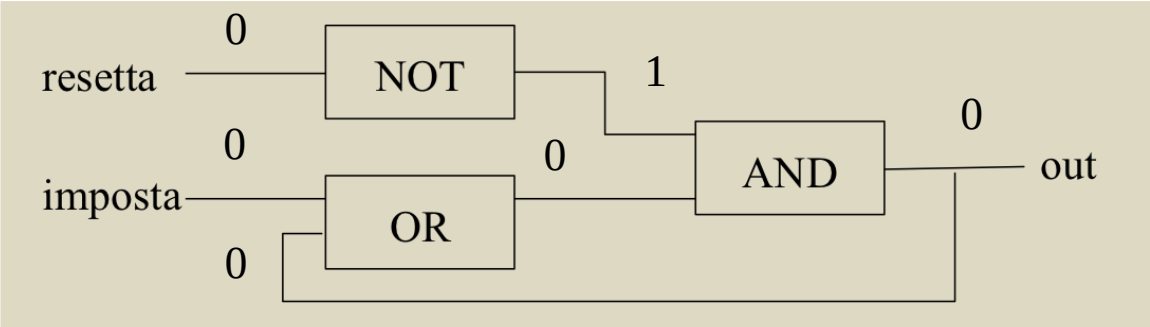
tempo t_1 : NOT 1 = 0 0 AND 1 = 1 l'uscita va a 0



tempo t_2 : propagazione della retroazione

Memorizzazione

Resetta torna a 0



Imposta = 0 resetta = 0

L'uscita si mantiene a 0

Nota: in questo caso gli ingressi ed uscite nel circuito sono uguali nei tempi t_1 *e* t_2

Riassunto

situazione iniziale: si assume uscita 0

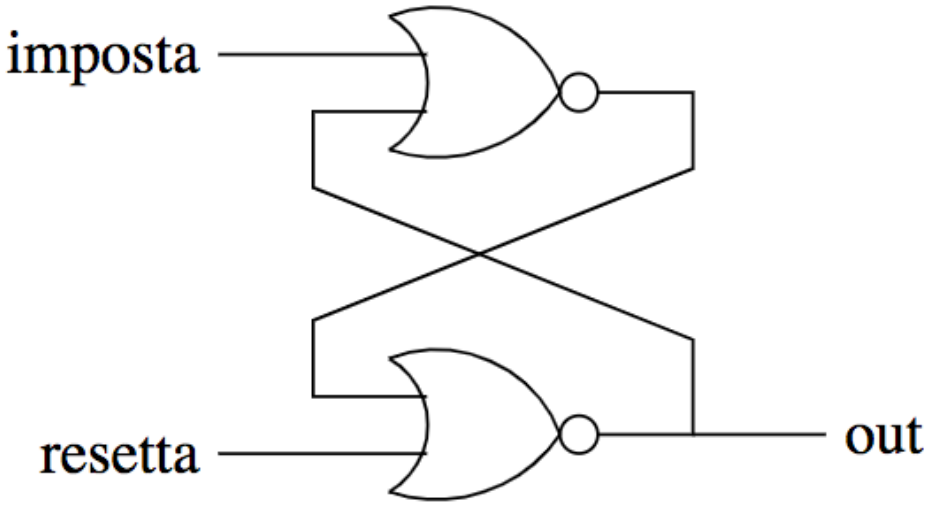
1. imposta=0 resetta=0: l'uscita si mantiene a 0
2. imposta=1 resetta=0: l'uscita va a 1
3. imposta=0 resetta=0: l'uscita si mantiene a 1
4. imposta=0 resetta=1: l'uscita va a 0
5. imposta=0 resetta=0: l'uscita si mantiene a 0

Quindi:

- imposta=1 → out=1
- resetta=1 → out=0
- con ingressi zero l'uscita *mantiene il valore*

Flip-flop

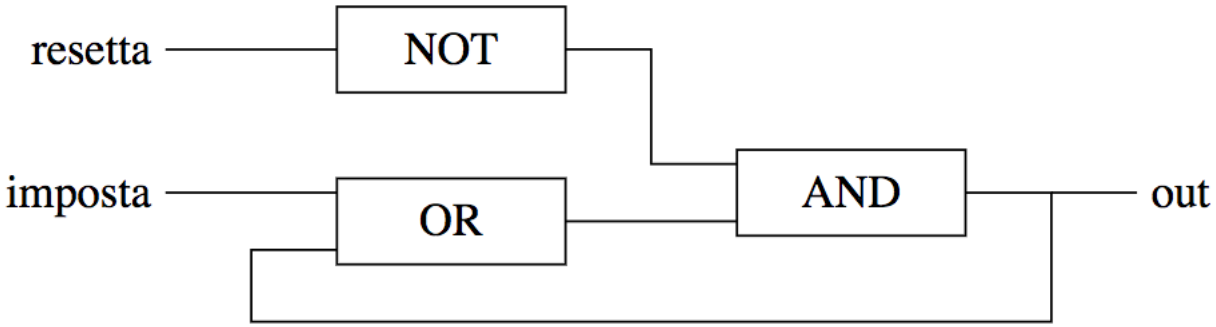
- circuito che memorizza un bit.
- di solito si disegna come:



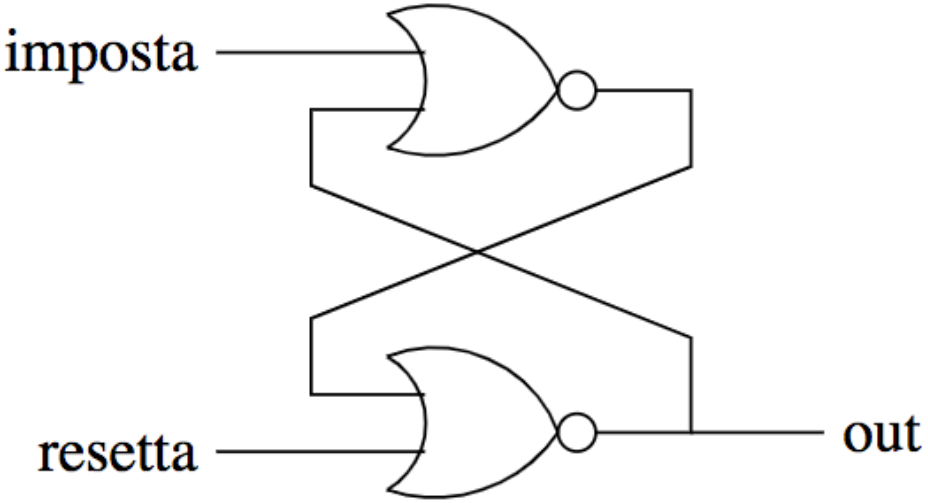
Blocchi NOR

a NOR b = NOT (a OR b)

Flip-flop



out = NOT resetta AND (imposta OR out)



applicando De Morgan

out = NOT (resetta OR (NOT (imposta OR out))) =

NOT resetta AND (imposta OR out)

Riassunto

nei computer tutti i dati sono espressi in forma di sequenze di bit

bit=0 oppure 1

con elementi NOT, AND, OR si possono realizzare:

- circuiti che memorizzano bit (flip-flop)
- circuiti che elaborano (es. somma)