

Automi e Linguaggi Formali

a.a. 2017/2018

LT in Informatica
24 Maggio 2018

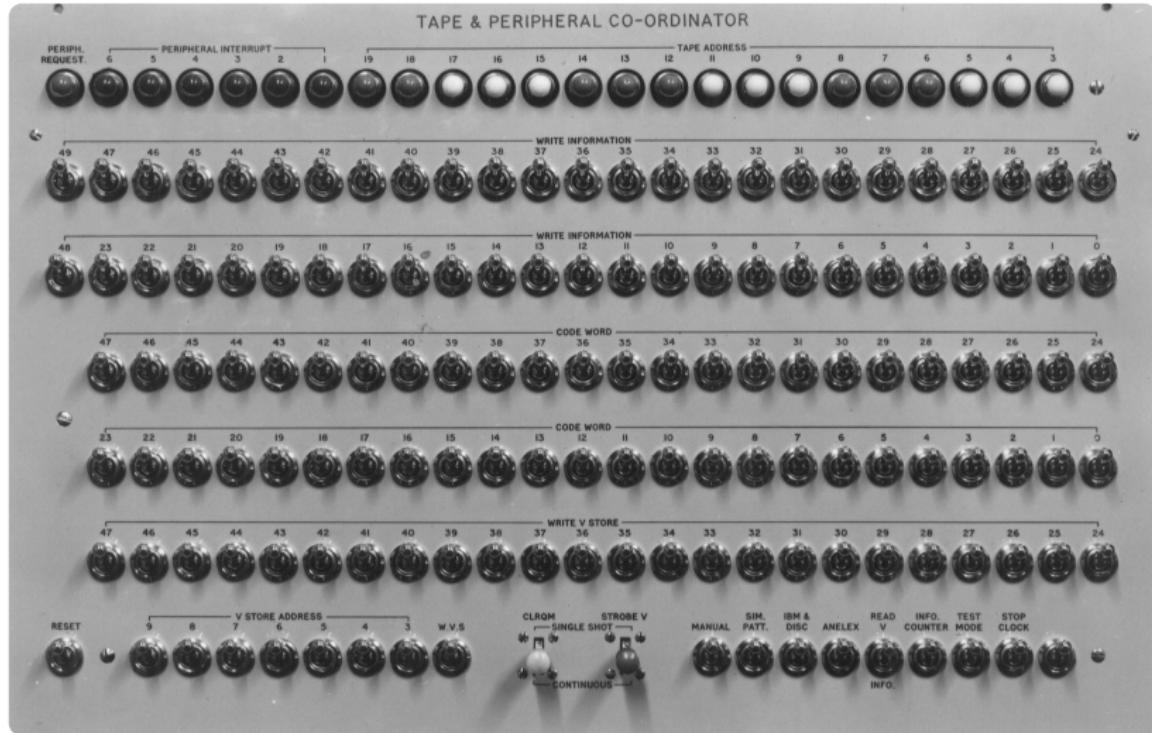


UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Un gioco che non potete vincere



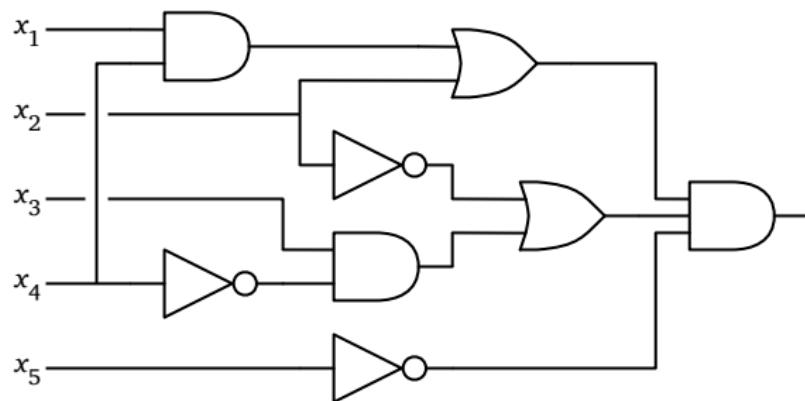
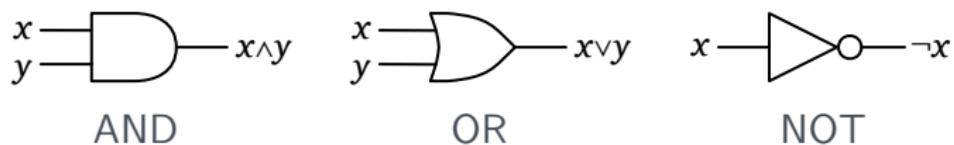
Un gioco che non potete vincere



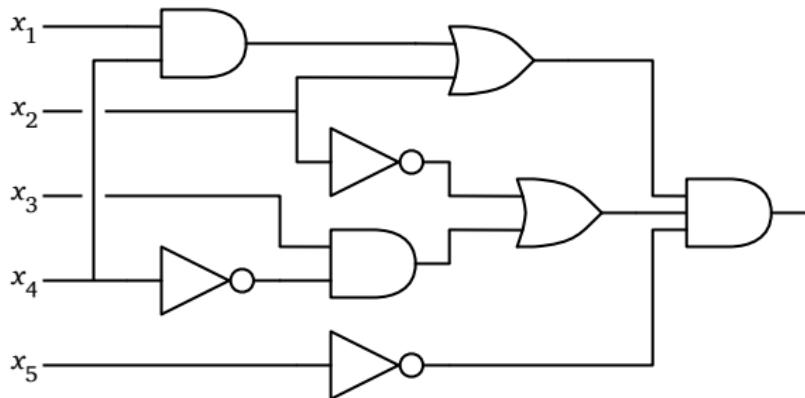
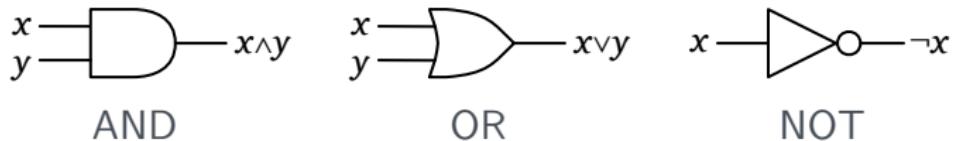
Un gioco che non potete vincere



Un gioco più facile?



Un gioco più facile?



CircuitSAT: dato un circuito Booleano, esistono dei **valori di input** che permettono di ottenere **output = 1**?

Algoritmi efficienti e non efficienti

- Gli algoritmi che si studiano sono tipicamente **algoritmi con complessità polinomiale**: il loro tempo di esecuzione è $O(n^k)$ per qualche costante k .
- Avere complessità polinomiale è un requisito minimo per considerare un algoritmo **efficiente**
- Un algoritmo con complessità più che polinomiale è un algoritmo **non efficiente** perché non è scalabile.

Problemi trattabili e problemi intrattabili

- I problemi per i quali esiste una soluzione polinomiale vengono considerati **trattabili**
- quelli che richiedono un algoritmo più che polinomiale sono detti **intrattabili** o anche **difficili**.
- Sappiamo che ci sono problemi che non possono essere risolti da **nessun algoritmo**:
 - “Halting Problem” di Turing
- Ci sono problemi che richiedono un tempo **esponenziale**:
 - il gioco della Torre di Hanoi

Problemi trattabili e problemi intrattabili

- I problemi per i quali esiste una soluzione polinomiale vengono considerati **trattabili**
- quelli che richiedono un algoritmo più che polinomiale sono detti **intrattabili** o anche **difficili**.
- Sappiamo che ci sono problemi che non possono essere risolti da **nessun algoritmo**:
 - “Halting Problem” di Turing
- Ci sono problemi che richiedono un tempo **esponenziale**:
 - il gioco della Torre di Hanoi

Stabilire con precisione qual'è il confine tra problemi trattabili ed intrattabili è piuttosto difficile



Problemi di decisione

Per semplificare lo studio della complessità dei problemi, limitiamo la nostra attenzione alla seguente classe di problemi:

Problemi di decisione

Problemi che hanno come output un singolo valore booleano: Si/No

Problemi di decisione

Per semplificare lo studio della complessità dei problemi, limitiamo la nostra attenzione alla seguente classe di problemi:

Problemi di decisione

Problemi che hanno come output un singolo valore booleano: Si/No

Ogni problema di decisione **corrisponde ad un linguaggio**:

Tutte le parole che **rappresentano** istanze con **risposta Si**

Problemi P, NP, e coNP

- **P** è la classe dei linguaggi tali che l'**appartenenza** di una stringa $x \in \Sigma^*$ al linguaggio può essere stabilita da una macchina di Turing che impiega **tempo** $O(|x|^k)$.

Problemi P, NP, e coNP

- **P** è la classe dei linguaggi tali che l'**appartenenza** di una stringa $x \in \Sigma^*$ al linguaggio può essere stabilita da una macchina di Turing che impiega **tempo** $O(|x|^k)$.
- **NP** è la classe dei linguaggi caratterizzati dalla seguente proprietà:
 - se una stringa $x \in \Sigma^*$ **appartiene** al linguaggio, allora esiste un **certificato** di questo fatto che può essere **verificato** in tempo polinomiale.

Problemi P, NP, e coNP

- **P** è la classe dei linguaggi tali che l'**appartenenza** di una stringa $x \in \Sigma^*$ al linguaggio può essere stabilita da una macchina di Turing che impiega **tempo** $O(|x|^k)$.
- **NP** è la classe dei linguaggi caratterizzati dalla seguente proprietà:
 - se una stringa $x \in \Sigma^*$ **appartiene** al linguaggio, allora esiste un **certificato** di questo fatto che può essere **verificato** in tempo polinomiale.
 - **Equivalente:** l'**appartenenza** di una stringa $x \in \Sigma^*$ al linguaggio può essere stabilita da una macchina di Turing **nondeterministica** che impiega **tempo** $O(|x|^k)$.

Esempi di problemi in P

- Alcuni problemi di decisione su **linguaggi regolari e context-free**:
 - **Appartenenza** di una parola ad un linguaggio regolare o context-free
 - Controllare se un linguaggio (regolare o CF) è **vuoto**
- I problemi di base del corso di Algoritmi e Strutture Dati:
 - ordinamento di un vettore, ricerca in un vettore o in un albero,
 - ...

Esempi di problemi in P

- Alcuni problemi di decisione su **linguaggi regolari e context-free**:
 - **Appartenenza** di una parola ad un linguaggio regolare o context-free
 - Controllare se un linguaggio (regolare o CF) è **vuoto**
- I problemi di base del corso di Algoritmi e Strutture Dati:
 - ordinamento di un vettore, ricerca in un vettore o in un albero,
 - ...

Per dimostrare che un problema è in P basta fornire un algoritmo **polinomiale** per risolverlo.

CircuitSAT è un problema NP

- 1 Codificare un circuito come una **stringa** in un alfabeto Σ
⇒ tra le infinite codifiche possibili, **vanno bene tutte** quelle polinomiali nella dimensione del circuito

CircuitSAT è un problema NP

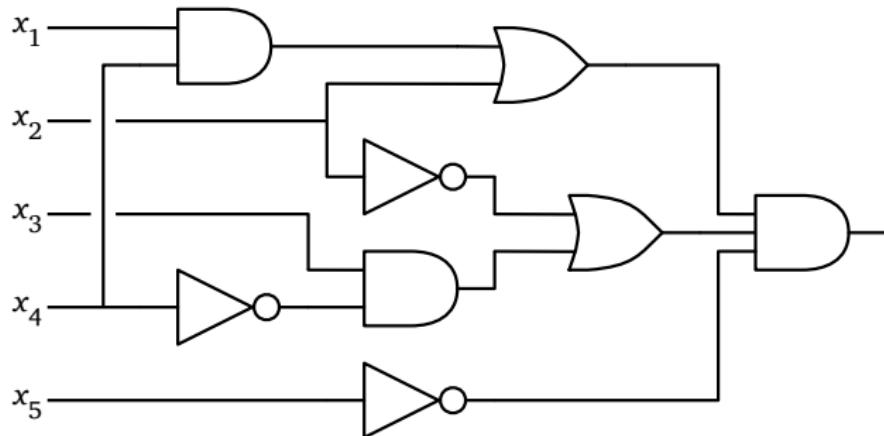
- 1 Codificare un circuito come una **stringa** in un alfabeto Σ
⇒ tra le infinite codifiche possibili, **vanno bene tutte** quelle **polinomiali** nella dimensione del circuito
- 2 Trovare un **certificato** di appartenenza:
⇒ i **valori degli input** x_1, x_2, \dots

CircuitSAT è un problema NP

- 1 Codificare un circuito come una **stringa** in un alfabeto Σ
⇒ tra le infinite codifiche possibili, **vanno bene tutte** quelle **polinomiali** nella dimensione del circuito
- 2 Trovare un **certificato** di appartenenza:
⇒ i **valori degli input** x_1, x_2, \dots
- 3 Trovare un **algoritmo polinomiale** per verificare il certificato.

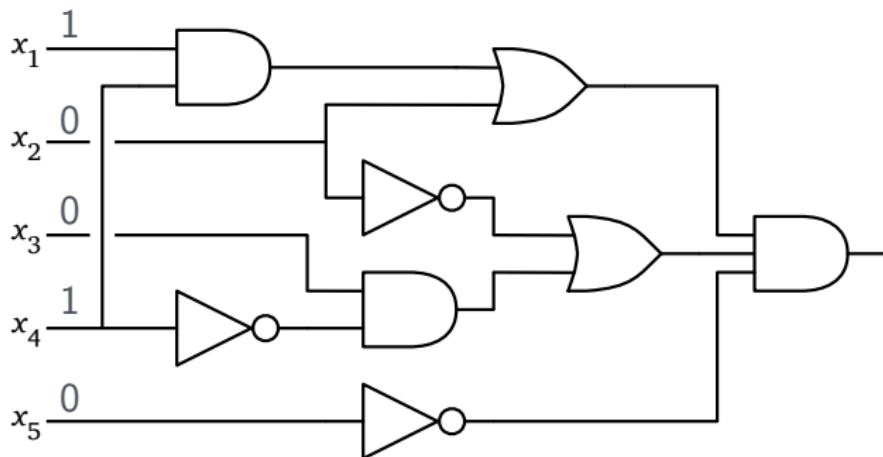
Verifica del certificato

Certificato: $x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 1, x_5 = 0$



Verifica del certificato

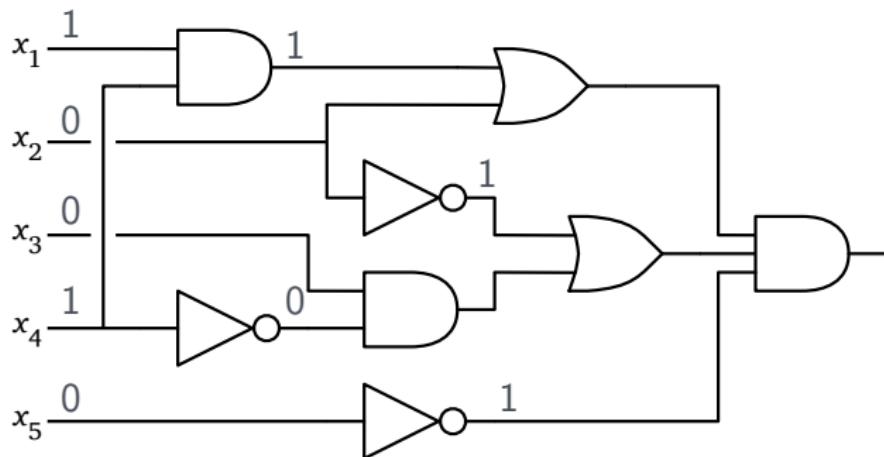
Certificato: $x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 1, x_5 = 0$



- 1 Assegna gli input come stabilito nel certificato

Verifica del certificato

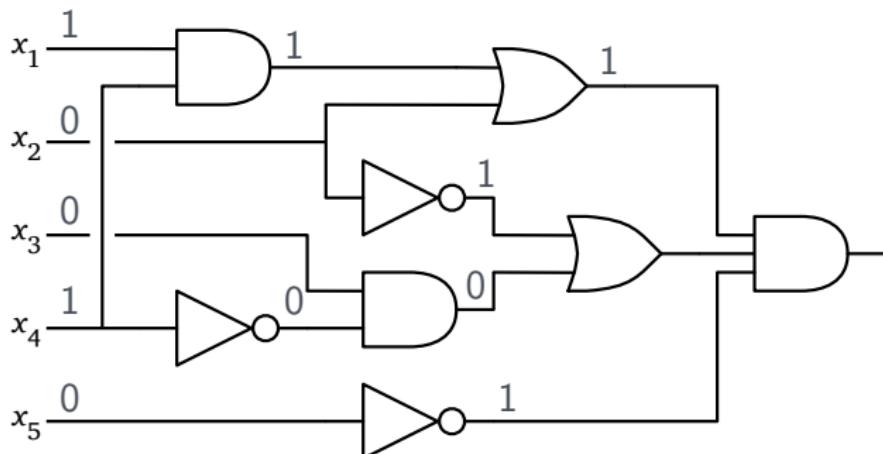
Certificato: $x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 1, x_5 = 0$



- 2 Calcola gli output delle porte logiche di primo livello

Verifica del certificato

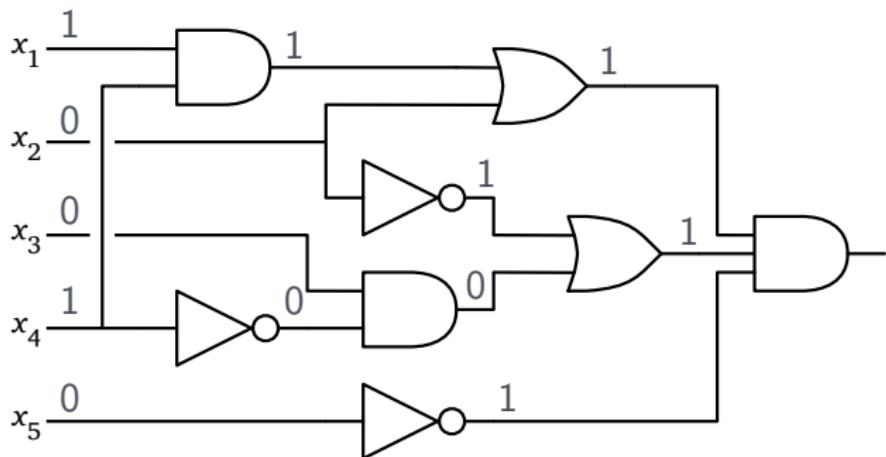
Certificato: $x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 1, x_5 = 0$



- 3 Calcola gli output delle porte logiche di secondo livello

Verifica del certificato

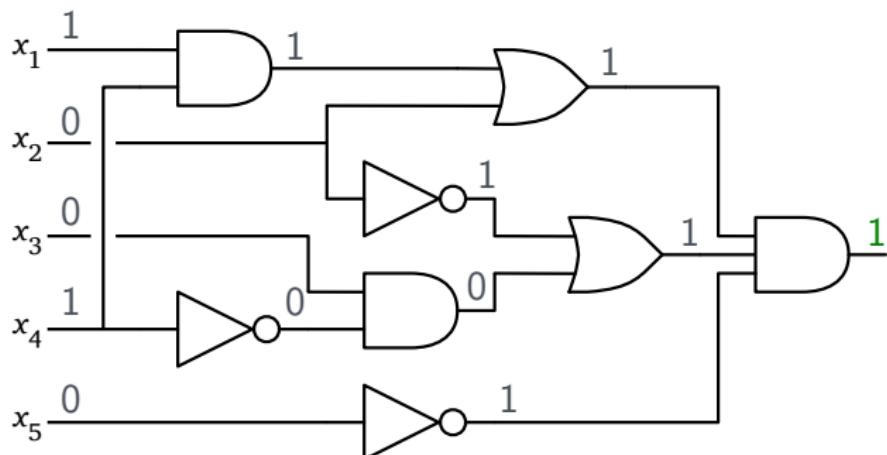
Certificato: $x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 1, x_5 = 0$



- 4 Calcola gli output delle porte logiche di terzo livello

Verifica del certificato

Certificato: $x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 1, x_5 = 0$



- 5 Calcola l'output dell'intero circuito: se = 1, **S1**, altrimenti **NO**

Problemi NP-hard

- Un problema è **NP-hard** se l'esistenza di un algoritmo polinomiale per risolverlo implica l'esistenza di un algoritmo polinomiale **per ogni problema in NP**.
- Se siamo in grado di risolvere un problema **NP-hard** in modo efficiente, allora possiamo risolvere in modo efficiente **ogni problema** di cui possiamo verificare facilmente una soluzione, usando la soluzione del problema **NP-hard** come sottoprocedura.
- Un problema è **NP-completo** se è sia **NP-hard** che appartenente alla classe **NP** (o “**NP-easy**”).

Dimostrare che un problema è NP-completo

Ogni dimostrazione di **NP-completezza** di compone di due parti:

- 1 dimostrare che il problema appartiene alla classe **NP**;
- 2 dimostrare che il problema è **NP-hard**.

Dimostrare che un problema è NP-completo

Ogni dimostrazione di **NP-completezza** di compone di due parti:

- 1 dimostrare che il problema appartiene alla classe **NP**;
 - 2 dimostrare che il problema è **NP-hard**.
- Dimostrare che un problema è in **NP** vuol dire dimostrare che esiste un algoritmo polinomiale per verificare un certificato per il Si.

Dimostrare che un problema è NP-completo

Ogni dimostrazione di **NP-completezza** di compone di due parti:

- 1 dimostrare che il problema appartiene alla classe **NP**;
 - 2 dimostrare che il problema è **NP-hard**.
-
- Dimostrare che un problema è in **NP** vuol dire dimostrare che esiste un algoritmo polinomiale per verificare un certificato per il Si.
 - Le tecniche che si usano per dimostrare che un problema è **NP-hard** sono fondamentalmente diverse.

Riduzioni tra problemi diversi

- Per dimostrare che un certo problema è **NP-hard** si procede tipicamente con una **dimostrazione per riduzione**
- **Ridurre** un problema B ad un altro problema A significa descrivere un **algoritmo polinomiale** che **risolve il problema B** sotto l'assunzione che **esista un algoritmo** per risolvere il **problema A** .

Riduzioni tra problemi diversi

- Per dimostrare che un certo problema è **NP-hard** si procede tipicamente con una **dimostrazione per riduzione**
- **Ridurre** un problema B ad un altro problema A significa descrivere un **algoritmo polinomiale** che **risolve il problema B** sotto l'assunzione che **esista un algoritmo** per risolvere il problema A .

Per dimostrare che X è **NP-hard** dobbiamo ridurre un problema **NP-hard** a X .

Riduzioni tra problemi diversi

- Per dimostrare che un certo problema è **NP-hard** si procede tipicamente con una **dimostrazione per riduzione**
- **Ridurre** un problema B ad un altro problema A significa descrivere un **algoritmo polinomiale** che **risolve il problema B** sotto l'assunzione che **esista un algoritmo** per risolvere il problema A .

Per dimostrare che X è **NP-hard** dobbiamo ridurre un problema **NP-hard** a X .

- Abbiamo bisogno di un problema **NP-hard** da cui partire: **CircuitSAT**

Teorema di Cook-Levin

Theorem (Cook e Levin, 1973)

*L'esistenza di una **macchina di Turing polinomiale** per risolvere **CircuitSAT** implica che $P = NP$.*

Teorema di Cook-Levin



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Theorem (Cook e Levin, 1973)

L'esistenza di una **macchina di Turing polinomiale** per risolvere **CircuitSAT** implica che $P = NP$.

P contro NP è uno dei problemi del millennio del Clay Institute:

 CMI ABOUT PROGRAMS MILLENNIUM PROBLEMS PEOPLE PUBLICATIONS EVENTS EUCLID

Ps vs NP Problem



Suppose that you are organizing housing accommodations for a group of four hundred university students. Space is limited and only one hundred of the students will receive places in the dormitory. To complicate matters, the Dean has provided you with a list of pairs of incompatible students, and requested that no pair from this list appear in your final choices. This is an example of what computer scientists call an NP-problem, since problems proposed by a computer scientist (in this case, the Dean's office), however the task of generating them is relatively impractical. Indeed, the total number of ways applicants is greater than the number of atoms in the observable universe, so it would be impractical to attempt to build a supercomputer capable of solving the problem for 100 students. However, this is

Rules:

[Related Documents](#)

- ## Official Problem Description

**1.000.000 US\$ di taglia
per chi lo risolve!**

Problema della soddisfacibilità Booleana (SAT)

- **Input:** una formula Booleana come

$$(a \vee b \vee c \vee \overline{d}) \leftrightarrow ((b \wedge \overline{c}) \vee (\overline{\overline{a} \rightarrow d}) \vee (c \neq a \wedge b)),$$

- **Output:** è possibile assegnare dei valori booleani (Vero/Falso) alle variabili a, b, c, \dots , in modo che il valore di verità della formula sia Vero?



SAT è NP-completo

- SAT è in NP:

SAT è NP-completo

■ SAT è in NP:

il **certificato** è l'assegnamento di verità alle variabili a, b, c, \dots

SAT è NP-completo

- **SAT** è in NP:
il **certificato** è l'assegnamento di verità alle variabili a, b, c, \dots
- **SAT** è NP-hard:

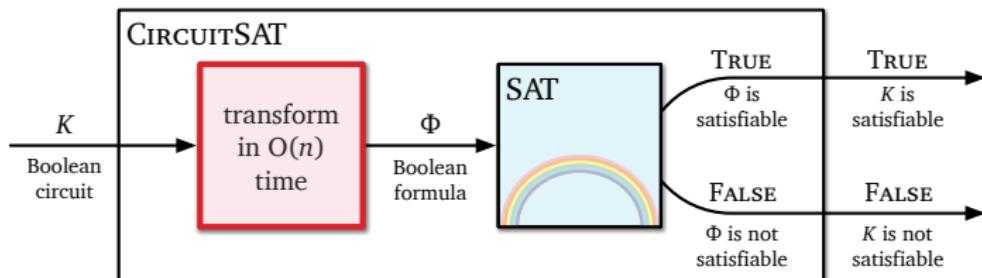
SAT è NP-completo

- **SAT è in NP:**

il **certificato** è l'assegnamento di verità alle variabili a, b, c, \dots

- **SAT è NP-hard:**

dimostrazione per **riduzione** di **CircuitSAT** a SAT

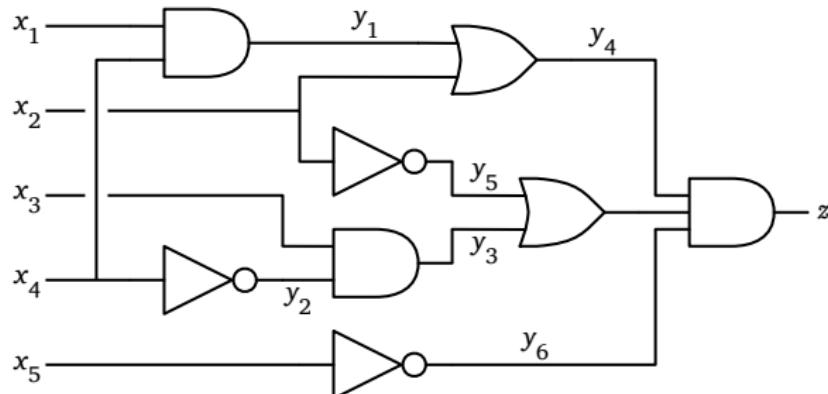


Riduzione di CircuitSAT a SAT



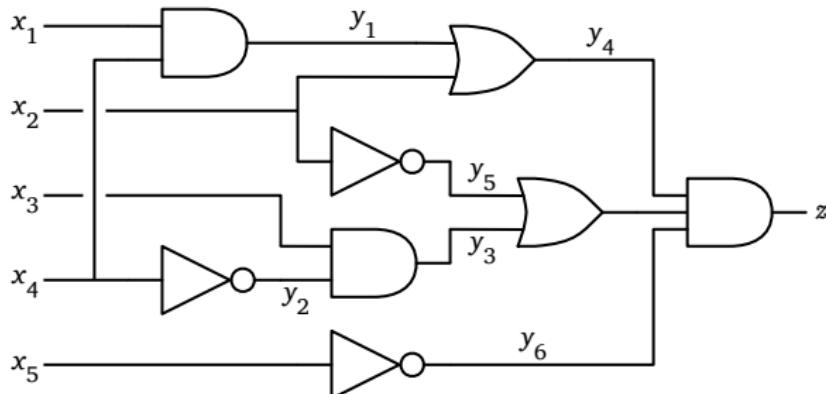
UNIVERSITÀ
DEGLI STUDI
DI PADOVA

- 1 Dare un nome agli output delle porte logiche:



Riduzione di CircuitSAT a SAT

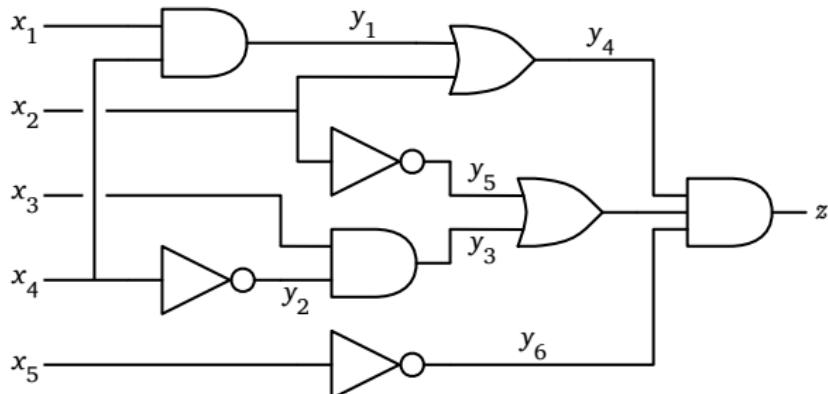
- 1 Dare un nome agli **output** delle porte logiche:



- 2 Scrivere le **espressioni booleane** per ogni porta logica e metterle in **and** logico:

Riduzione di CircuitSAT a SAT

- 1 Dare un nome agli **output** delle porte logiche:



- 2 Scrivere le **espressioni booleane** per ogni porta logica e metterle in **and** logico:

$$(y_1 = x_1 \wedge x_4) \wedge (y_2 = \overline{x_4}) \wedge (y_3 = x_3 \wedge y_2) \wedge (y_4 = y_1 \vee x_2) \wedge \\ (y_5 = \overline{x_2}) \wedge (y_6 = \overline{x_5}) \wedge (y_7 = y_3 \vee y_5) \wedge (z = y_4 \wedge y_7 \wedge y_6) \wedge z$$

Correttezza della riduzione

Ora dobbiamo mostrare che il circuito originale K è soddisfacibile se e solo se la formula risultante Φ è soddisfacibile.

Correttezza della riduzione

Ora dobbiamo mostrare che il circuito originale K è soddisfacibile
se e solo se la formula risultante Φ è soddisfacibile.

Dimostriamo questa affermazione **in due passaggi**:

Correttezza della riduzione

Ora dobbiamo mostrare che il circuito originale K è soddisfacibile **se e solo se** la formula risultante Φ è soddisfacibile.

Dimostriamo questa affermazione **in due passaggi**:

- ⇒ Dato un insieme di input che rende vero il circuito K , possiamo ottenere i valori di verità per le variabili nella formula Φ calcolando l'output di ogni porta logica di K .

Correttezza della riduzione

Ora dobbiamo mostrare che il circuito originale K è soddisfacibile **se e solo se** la formula risultante Φ è soddisfacibile.

Dimostriamo questa affermazione **in due passaggi**:

- ⇒ Dato un insieme di input che rende vero il circuito K , possiamo ottenere i valori di verità per le variabili nella formula Φ calcolando l'output di ogni porta logica di K .
- ⇐ Dati i valori di verità delle variabili nella formula Φ , possiamo ottenere gli input del circuito semplicemente ignorando le variabili delle porte logiche interne y_i e la variabile di uscita z .

Correttezza della riduzione

Ora dobbiamo mostrare che il circuito originale K è soddisfacibile **se e solo se** la formula risultante Φ è soddisfacibile.

Dimostriamo questa affermazione **in due passaggi**:

- ⇒ Dato un insieme di input che rende vero il circuito K , possiamo ottenere i valori di verità per le variabili nella formula Φ calcolando l'output di ogni porta logica di K .
- ⇐ Dati i valori di verità delle variabili nella formula Φ , possiamo ottenere gli input del circuito semplicemente ignorando le variabili delle porte logiche interne y_i e la variabile di uscita z .

L'intera trasformazione da un circuito all'altro può essere eseguita **in tempo lineare**. Inoltre, la dimensione della formula risultante cresce di **un fattore costante** rispetto a qualsiasi ragionevole rappresentazione del circuito.