Automi e Linguaggi Formali

a.a. 2017/2018

LT in Informatica
28 Maggio 2018

Università degli Studi di Padova

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

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

Tesi di Church computazionale

Tutti i formalismi di calcolo ragionevoli sono computazionalmente equivalenti a meno di fattori polinomiali.

Tesi di Church computazionale

Tutti i formalismi di calcolo ragionevoli sono computazionalmente equivalenti a meno di fattori polinomiali.

Esempi:

- Macchine di Turing Deterministiche
- Linguaggi di programmazione concreti: Java, C++, Python, . . .

Tesi di Church computazionale

Tutti i formalismi di calcolo ragionevoli sono computazionalmente equivalenti a meno di fattori polinomiali.

Esempi:

- Macchine di Turing Deterministiche
- Linguaggi di programmazione concreti: Java, C++, Python, . . .

Eccezioni:

- Computer quantistici
- DNA Computing, Bio Computing

Problemi/Linguaggi P e NP

- **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 deterministica che impiega tempo $O(|x|^k)$.

Problemi/Linguaggi P e NP

- **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 deterministica 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/Linguaggi P e NP

- **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 deterministica 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 $O(|x|^k)$.

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").

Perché studiare la NP-completezza?

- Progettare ed implementare algoritmi richiede la conoscenza dei principi di base della teoria della complessità
- Stabilire che un problema è NP-completo costituisce una prova piuttosto forte della sua intrattabilità
- Conviene cercare di risolverlo con un approccio diverso ...
  - identificare un caso particolare trattabile
  - cercare una soluzione approssimata
- ... invece di cercare un algoritmo efficiente per il caso generale che probabilmente non esiste nemmeno

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.

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 Y ad un altro problema X significa descrivere un algoritmo polinomiale che risolve il problema Y sotto l'assunzione che esista un algoritmo per risolvere il problema 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 Y ad un altro problema X significa descrivere un algoritmo polinomiale che risolve il problema Y sotto l'assunzione che esista un algoritmo per risolvere il problema X.

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 Y ad un altro problema X significa descrivere un algoritmo polinomiale che risolve il problema Y sotto l'assunzione che esista un algoritmo per risolvere il problema X.

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

Theorem (Cook e Levin, 1973)

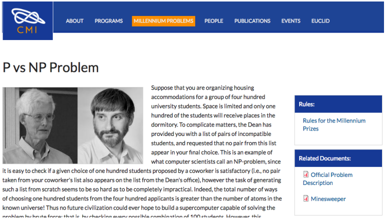
*L'esistenza di una macchina di Turing polinomiale per risolvere **CircuitSAT** implica che $P = NP$.*

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:



1.000.000 US$ di taglia per chi lo risolve!

Schema di riduzione polinomiale

Per dimostrare che un problema X è NP-hard:

1. Scegli un problema Y che sai essere NP-hard.
2. Descrivi un algoritmo per risolvere Y usando un algoritmo per X come subroutine:
   - data un'istanza di Y, trasformala in un'istanza di X,
   - quindi chiama l'algoritmo magico black-box per X.
3. Dimostra che l'algoritmo è corretto:
   - Dimostra che il tuo algoritmo trasforma istanze "buone" di Y in istanze "buone" di X.
   - Dimostra che il tuo algoritmo trasforma istanze "cattive" di Y in istanze "cattive" di X.
     Equivalente: se la tua trasformazione produce un'istanza "buona" di X, allora era partita da un'istanza "buona" di Y.
4. Mostra che la trasformazione funziona in tempo polinomiale.

Problema della soddisfaciblità 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: Si, se è possibile assegnare dei valori booleani (Vero/Falso) alle variabili $a, b, c, \ldots,$ in modo che il valore di verità della formula sia Vero; No altrimenti.

SAT è NP-completo

- **SAT** è in NP:

SAT è NP-completo

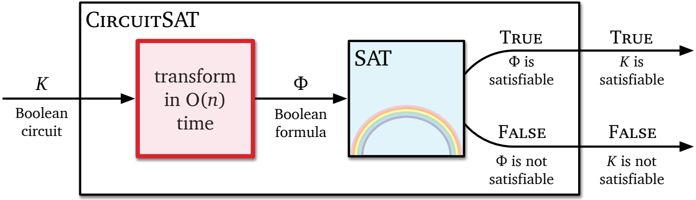
- **SAT** è in NP:
  il certificato è l'assegnamento di verità alle variabili $a, b, c, \ldots$

SAT è NP-completo

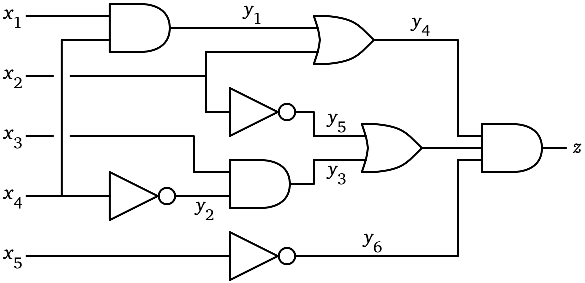
- **SAT** è in NP:
  il certificato è l'assegnamento di verità alle variabili $a, b, c, \ldots$
- **SAT** è NP-hard:

SAT è NP-completo

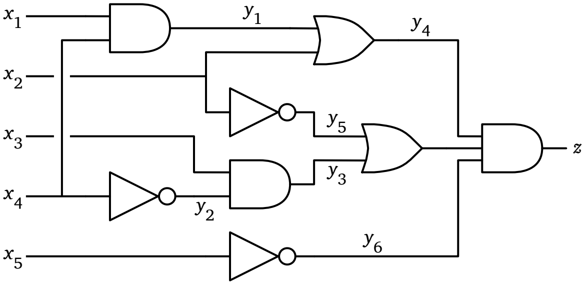
- **SAT** è in NP:
  il certificato è l'assegnamento di verità alle variabili $a, b, c, \ldots$
- **SAT** è NP-hard:
  dimostrazione per riduzione di **CircuitSAT** a **SAT**



1. Dare un nome agli output delle porte logiche:

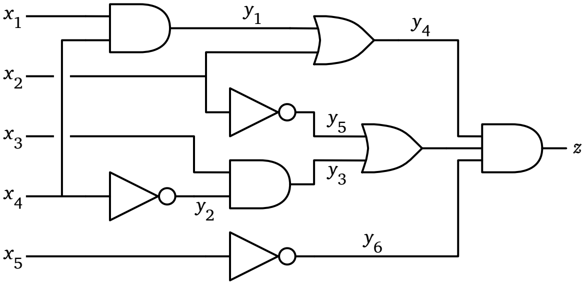


1. Dare un nome agli output delle porte logiche:



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

1. Dare un nome agli output delle porte logiche:



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

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

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**:

$\Rightarrow$ 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:

- $\Rightarrow$ 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.
- $\Leftarrow$ 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:

- $\Rightarrow$ 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.
- $\Leftarrow$ 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.

Una versione ristretta di soddisfacibilità

- Intendiamo dimostrare l'NP-completezza di un'ampia gamma di problemi
- Dovremmo procedere per riduzione polinomiale da **SAT** o **CircuitSAT** al problema in esame
- Esiste però un importante problema "intermedio", detto **3SAT**, molto più facile da ridurre ai problemi tipici rispetto a **SAT**:
  - anche **3SAT** è un problema di soddisfacibilità di espressioni booleane
  - **3SAT** però richiede che le espressioni siano di una forma ben precisa, formate cioè da congiunzione logica di clausole ognuna delle quali è disgiunzione logica di tre variabili (anche negate)

Forme normali di espressioni booleane

- Un letterale è una variabile o una variabile negata, ad es. x, $\overline{y}$
- Una clausola è una disgiunzione logica (OR) di uno o più letterali, ad es. x, $x \vee \overline{y}$
- Una espressione booleana si dice in forma normale congiuntiva (CNF), se è la congiunzione logica (AND) di una o più clausole:
  - $(x \vee y) \wedge (\overline{x} \vee z)$, e $x \wedge y$ sono in CNF
  - mentre $(x \vee y \vee z) \wedge (\overline{y} \vee \overline{z}) \vee (x \vee y \wedge z)$ non è in CNF
- Un espressione si dice in forma normale 3-congiuntiva (3-CNF) se è composta di clausole che hanno esattamente 3 letterali distinti

Il problema 3SAT

- Input: una formula Booleana in 3-CNF come

$$(a \vee b \vee c) \wedge (\overline{d} \vee b \vee \overline{c}) \wedge (\overline{a} \vee d \vee c)$$

- Output: Si, se è possibile assegnare dei valori booleani (Vero/Falso) alle variabili $a, b, c, \ldots$, in modo che il valore di verità della formula sia Vero; No altrimenti.

3SAT è NP-completo

- **3SAT** è in NP:

3SAT è NP-completo

- **3SAT** è in NP:
  il certificato è l'assegnamento di verità alle variabili $a, b, c, \ldots$

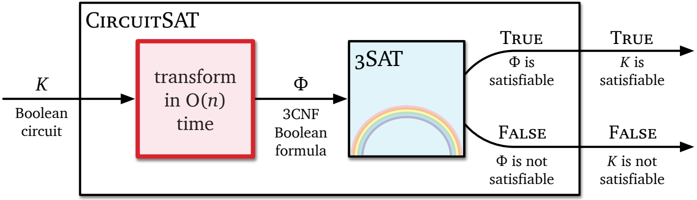
3SAT è NP-completo

Università degli Studi di Padova

- **3SAT** è in NP:
  il certificato è l'assegnamento di verità alle variabili $a, b, c, \ldots$
- **3SAT** è NP-hard:

3SAT è NP-completo

- **3SAT** è in NP:
  il certificato è l'assegnamento di verità alle variabili $a, b, c, \ldots$
- **3SAT** è NP-hard:
  dimostrazione per riduzione di **CircuitSAT** a **3SAT**



Riduzione di CircuitSAT a 3SAT

1. Fare in modo che ogni porta logica abbia al massimo due input
2. Trasformare il circuito in una formula booleana come abbiamo fatto per **SAT**
3. Trasformare la formula in CNF usando le regole seguenti:

$$a = b \land c \mapsto (a \lor \overline{b} \lor \overline{c}) \land (\overline{a} \lor b) \land (\overline{a} \lor c)$$

$$a = b \lor c \mapsto (\overline{a} \lor b \lor c) \land (a \lor \overline{b}) \land (a \lor \overline{c})$$

$$a = \overline{b} \mapsto (a \lor b) \land (\overline{a} \lor \overline{b})$$

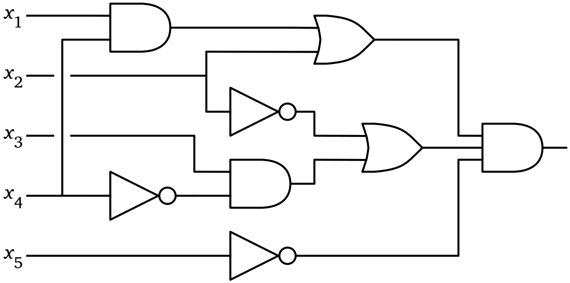
4. Trasformare la formula in 3CNF aggiungendo variabili alle clausole con meno di tre letterali:

$$a \lor b \mapsto (a \lor b \lor x) \land (a \lor b \lor \overline{x})$$

$$a \mapsto (a \lor x \lor y) \land (a \lor \overline{x} \lor y) \land (a \lor x \lor \overline{y}) \land (a \lor \overline{x} \lor \overline{y})$$

Esempio di Riduzione

- Dal circuito booleano ...



Esempio di Riduzione

- . . . alla formula in 3CNF

$$
\begin{gathered}
(y_1 \lor \overline{x_1} \lor \overline{x_4}) \land (\overline{y_1} \lor x_1 \lor z_1) \land (\overline{y_1} \lor x_1 \lor \overline{z_1}) \land (\overline{y_1} \lor x_4 \lor z_2) \land (\overline{y_1} \lor x_4 \lor \overline{z_2}) \\
\land (y_2 \lor x_4 \lor z_3) \land (y_2 \lor x_4 \lor \overline{z_3}) \land (\overline{y_2} \lor \overline{x_4} \lor z_4) \land (\overline{y_2} \lor \overline{x_4} \lor \overline{z_4}) \\
\land (y_3 \lor \overline{x_3} \lor \overline{y_2}) \land (\overline{y_3} \lor x_3 \lor z_5) \land (\overline{y_3} \lor x_3 \lor \overline{z_5}) \land (\overline{y_3} \lor y_2 \lor z_6) \land (\overline{y_3} \lor y_2 \lor \overline{z_6}) \\
\land (\overline{y_4} \lor y_1 \lor x_2) \land (y_4 \lor \overline{x_2} \lor z_7) \land (y_4 \lor \overline{x_2} \lor \overline{z_7}) \land (y_4 \lor \overline{y_1} \lor z_8) \land (y_4 \lor \overline{y_1} \lor \overline{z_8}) \\
\land (y_5 \lor x_2 \lor z_9) \land (y_5 \lor x_2 \lor \overline{z_9}) \land (\overline{y_5} \lor \overline{x_2} \lor z_{10}) \land (\overline{y_5} \lor \overline{x_2} \lor \overline{z_{10}}) \\
\land (y_6 \lor x_5 \lor z_{11}) \land (y_6 \lor x_5 \lor \overline{z_{11}}) \land (\overline{y_6} \lor \overline{x_5} \lor z_{12}) \land (\overline{y_6} \lor \overline{x_5} \lor \overline{z_{12}}) \\
\land (\overline{y_7} \lor y_3 \lor y_5) \land (y_7 \lor \overline{y_3} \lor z_{13}) \land (y_7 \lor \overline{y_3} \lor \overline{z_{13}}) \land (y_7 \lor \overline{y_5} \lor z_{14}) \land (y_7 \lor \overline{y_5} \lor \overline{z_{14}}) \\
\land (y_8 \lor \overline{y_4} \lor \overline{y_7}) \land (\overline{y_8} \lor y_4 \lor z_{15}) \land (\overline{y_8} \lor y_4 \lor \overline{z_{15}}) \land (\overline{y_8} \lor y_7 \lor z_{16}) \land (\overline{y_8} \lor y_7 \lor \overline{z_{16}}) \\
\land (y_9 \lor \overline{y_8} \lor \overline{y_6}) \land (\overline{y_9} \lor y_8 \lor z_{17}) \land (\overline{y_9} \lor y_6 \lor z_{18}) \land (\overline{y_9} \lor y_6 \lor \overline{z_{18}}) \land (\overline{y_9} \lor y_8 \lor \overline{z_{17}}) \\
\land (y_9 \lor z_{19} \lor z_{20}) \land (y_9 \lor \overline{z_{19}} \lor z_{20}) \land (y_9 \lor z_{19} \lor \overline{z_{20}}) \land (y_9 \lor \overline{z_{19}} \lor \overline{z_{20}})
\end{gathered}
$$

2SAT è in P!

Algoritmo polinomiale per **2SAT** (soddisfacibilità di formule in 2CNF):

- Prendiamo una variabile x e assegnamo valore 1 (vero)
- In ogni clausola con $\overline{x}$, l'altro letterale deve essere vero
  - **Esempio:** in $(\overline{x} \vee \overline{y})$, y deve essere falso (0)
- Continuiamo assegnado le variabili il cui valore è "forzato"

si continua finché. . .

- L'algoritmo assegna valori alle variabili finché non succede una di tre cose:
  1. una contraddizione: una variabile è forzata ad essere sia vera che falsa
  2. tutte le variabili con valore forzato sono state assegnate, ma ancora ci sono clausole non soddisfatte
  3. si ottiene un assegnamento che dà valore vero all'espressione

- L'algoritmo assegna valori alle variabili finché non succede una di tre cose:
  1. una contraddizione: una variabile è forzata ad essere sia vera che falsa
  2. tutte le variabili con valore forzato sono state assegnate, ma ancora ci sono clausole non soddisfatte
  3. si ottiene un assegnamento che dà valore vero all'espressione
- Di conseguenza:
  1. Ci può essere un assegnamento che dà valore vero solo valore falso per la variabile x. Ricominciamo assegnando x a 0 (falso).
  2. Ci sono ancora variabili e clausole che non sono assegnate. Eliminiamo le clausole soddisfatte, e ripartiamo.
  3. La risposta è **Si** (ho trovato un assegnamento che soddisfa l'espressione).

Esercizi

1. Utilizzare l'algoritmo polinomiale per **2SAT** per verificare se le seguenti formule in 2CNF sono soddisfacibili:
   - $\Phi_1 = (\overline{x} \vee y) \wedge (\overline{y} \vee z) \wedge (\overline{z} \vee \overline{x})$
   - $\Phi_2 = (x \vee y) \wedge (y \vee \overline{x}) \wedge (x \vee \overline{y}) \wedge (\overline{x} \vee \overline{y})$
2. Descrivere una riduzione polinomiale da **3SAT** a **4SAT**:
   - un algoritmo polinomiale per risolvere **4SAT** usando **3SAT** come subroutine