Manuale per la preparazione all'esame di complessità

Riccardo Torre

13 marzo 2025

Capitolo 1

Appunti delle lezioni

Capitolo 2

Riduzioni

2.1 Riduzioni

Definizione 1 (Riduzione polinomiale tra problemi (alla Karp))**.** *Un problema* $\mathbb{A}$ *si riduce polinomialmente (alla Karp) a* $\mathbb{B}$ *e si scrive* $\mathbb{A} \preceq \mathbb{B}$ *se* $\exists$ *un algoritmo polinomiale* f *tale che*

$$\forall x \in \mathbb{A} : \ \mathbb{A}(x) = yes \iff \mathbb{B}(f(x)) = yes$$

L'effetto di f *è*

$$\underbrace{|x|}_{\substack{istanza \\ di\ \mathbb{A}}} \xrightarrow{effetto\ di\ f} \underbrace{|y|}_{\substack{istanza \\ di\ \mathbb{B}}} \in O(|x|^c) \wedge y = f(x)$$

in tempo $O(|y|^d)$ *si risponde*

$$\boxed{\mathbb{B}(y) = yes \iff \mathbb{A}(x) = yes}$$

in tempo $O(|x|^c) + O(|y|^d)$*, che è polinomiale.*

2.1.1 Riduzione k-col a k+1-col

Un grafo G è propriamente colorato se

$$\forall (u, v) \in E(G) : \ u \neq v \ \wedge \ colore(v) \neq colore(u)$$

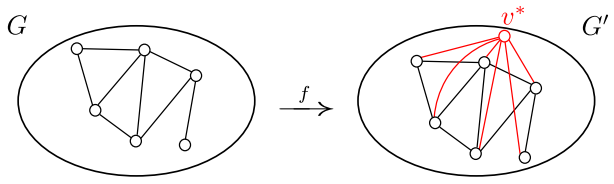

Figura 2.1: riduzione k-coloring a k+1-coloring.

Dimostrazione 1. *Fornire la funzione polytime f tale che se $\forall G$ è $k-col$ allora*

$$G \text{ è k-colorabile} \iff f(G) \text{ è k+1-colorabile}$$

f è polytime computable. Se G è $k-col$ allora G' è $k+1-col$.

- *quando viene colorato G' per i vertici che erano in G si tiene la k-colorazione che era presente e per v^* viene utilizzato il colore in più;*
- *se G' è k+1-col wlog*[1] *si imposta a $k+1$ il colore in v^*. Poiché $\forall v \neq v^*$ $(v, v^*) \in E(G')$ si ha che $colore(v) \neq k+1$ e $colore(v) \in \{1, \ldots, k\}$*
- *se $\forall v, w \in V(G) : e = (v, w) \wedge colore(v) \neq colore(w)$ allora la colorazione è propria per G.*

Per dimostrare l'implicazione inversa, basta partire da G' e mostrare che il risolutore restituisce yes per $k+1-col$, togliere v^* e gli archi che lo congiungono, togliere il $k+1$ colore e mostrare che anche il risolutore di $k-col$ restituisce yes. □

Da $k-col \preceq k+1-col$ segue che

$$\exists f : G \to G' = f(G) \wedge |G'| = |G|_f^c \quad [polytime]$$

Se esiste A per $k+1-col$ $A(G) = k+1-col$ e $T_A(G') = O(|G'|_a^c) \implies B(G) = A(f(G))$ B è un algoritmo *polytime* per $k-col$. Il tempo di B su G è

$$T_B(G) = O(|G|_d^c) + T_A(f(G)) = O(|G|_d^c) + O((|G|_d^c)_A^c) = O(|G|^{c_d \cdot c_A})$$

Alcune osservazioni

$$\mathbb{A} \preceq \mathbb{B} \wedge \mathbb{B} \in \mathrm{P} \implies \mathbb{A} \in \mathrm{P}$$
$$\mathbb{A} \preceq \mathbb{B} \wedge \mathbb{A} \notin \mathrm{P} \implies \mathbb{B} \notin \mathrm{P}$$

Se $k-col \notin$ P allora B non può esistere. Questo implica che A non può esistere e $k+1-col \notin$ P.

Si supponga che esista $\mathbb{B}$ tale che

$$\forall \mathbb{A} \in \mathrm{NP} \quad \mathbb{A} \preceq \mathbb{B}$$

- Se $\mathbb{B} \in$ P allora $\mathbb{A} \in$ P. Segue che $\forall \mathbb{A} \in$ NP risulta NP $\subseteq$ P $\implies$ P = NP
- Se P $\neq$ NP $\implies$ $B \notin$ P

Definizione 2 (NP-completezza)**.** *Si dice che $\mathbb{B}$ è NP-completo (NPC) se*

- $\mathbb{B} \in NP$
- $\forall \mathbb{A} \in NP : \mathbb{A} \preceq \mathbb{B}$ *ovvero $\mathbb{B}$ è NP-hard*

[1] without loss of generality.

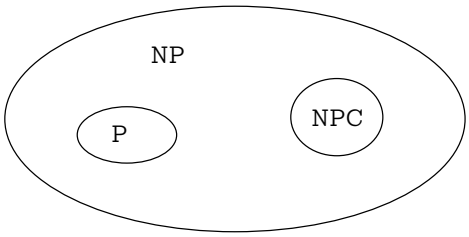

Figura 2.2: relazioni tra le classi di complessità.

2.1.2 Problemi NPC

Esistono problemi NP-completi. Se $\mathbb{A} \in$ NP se esiste un verificatore $B(\cdot,\cdot)$ ploytime per $\mathbb{A}$ tale che

$$\forall x \in \mathcal{I}(\mathbb{A}) : \mathbb{A}(x) = yes \iff \exists y : B(x,y) = yes$$

2.1.3 Problema SAT (Satisfiability)

SAT
Input: Formula CNF ϕ **Output:** *yes* $\iff$ ϕ è soddisfacibile

Definizione 3 (Forumla CNF)**.** *Una formula è CNF*[2] *se*

$$\phi(x_1 \dots x_n) = \underbrace{C^1 \wedge C^2 \wedge \dots \wedge C^n}_{congiunzione\ di\ clausole}$$

dove ogni $C^i : 1 \leq i \leq n \wedge l^i_j : 1 \leq j \leq k$

$$C^i = \underbrace{l^i_1 \vee l^i_2 \vee \dots \vee l^i_k}_{disgiunzione\ di\ letterali} \qquad l^i_j \in \{\underbrace{x_1, \dots, x_n, \overline{x_1}, \dots, \overline{x_n}}_{insieme\ di\ variabili}\}^3$$

Definizione 4 (Assegnamento)**.** *Un assegnamento a è definito come*

$$a = (a_1, \dots, a_n) \in \{T, F\}^n$$

Si dice che un assegnamento a soddisfa ϕ *se* $\boxed{\phi(a_1, \dots, a_n) = T}$

2.1.4 Riduzione k-col a SAT

Dato G è possibile costruire in *polytime* ϕ_G CNF tale che G è k-colorabile se e solo se ϕ_G è soddisfacibile.

Dimensioni delle istanze La taglia di G è

$$|G| = |(V,E)| = |V| + |E|$$

mentre quella di ϕ è

$$|\phi| = n$$

con n il numero di letterali che sono contenuti all'interno di ϕ.

[2] Conjunctive Normal Form.

Obbiettivo L'obbiettivo che si vuole raggiungere è definire una funzione f *polytime computable* che permetta di ridurre le istanze del problema $k - col$ a istanze del problema SAT. In altre parole, dare una definizione di $k - col$ ridefinendo le sue istanze in maniera tale da utilizzare le istanze di SAT. Si definisce ϕ_G come

$$\boxed{\phi_G = \bigwedge_{v \in V} (C^v \wedge D^v) \wedge \bigwedge_{e \in E} E^e} \tag{2.1}$$

tale che

$$\forall v \in V : \begin{cases} C^v = x_1^v \vee x_2^v \vee \ldots \vee x_k^v \\ D^v = \bigwedge_{1 \leq i \leq j \leq k} (\overline{x_i^v} \vee \overline{x_j^v}) \end{cases} \tag{2.2}$$

$$\forall e = (u, v) \in E : E^e = \bigwedge_{1 \leq i \leq k} (\overline{x_i^u} \vee \overline{x_i^v}) \tag{2.3}$$

dove eq. (2.2) contiene due condizioni:

1. ciascun vertice deve avere almeno un colore
2. ciascun vertice non può contenere più di un colore

che si traduce in "*ogni vertice deve avere esattamente un colore*". La eq. (2.3) garantisce la colorazione propria del grafo. La definizione scelta permette di specificare in posizione apice di una variabile il vertice, e in posizione pedice il colore. La dimensione di $|\phi_G|$ diventa

$$|\phi_G| = |V| \left(k + \binom{k}{2} 2 \right) + 2k|E|$$

ovvero ciascun vertice dell'insieme V ha una clausola C^v lunga k letterali, una clausola D^v lunga per ogni coppia scelta di colori $\binom{k}{2}$ si hanno due letterali $\overline{x_i^v}$ e $\overline{x_j^v}$ e ciascun arco nell'insieme E ha una clausola E^e lunga per ogni possibile colore (i colori sono k), due letterali.

Esempio

Si estraggono le informazioni dal grafo e si ricavano le equazioni

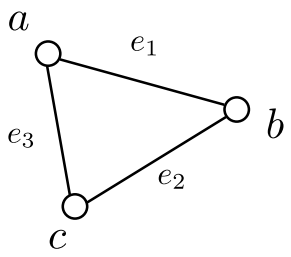

Figura 2.3: dal grafo si estraggono le informazioni per ricavare la formula CNF.

$$C^a = x_1^a \vee x_2^a \qquad C^b = x_1^b \vee x_2^b \qquad C^c = x_1^c \vee x_2^c$$
$$D^a = \overline{x_1^a} \vee \overline{x_2^a} \qquad D^b = \overline{x_1^b} \vee \overline{x_2^b} \qquad D^c = \overline{x_1^c} \vee \overline{x_2^c}$$
$$E^{e_1} = (\overline{x_1^a} \vee \overline{x_1^b}) \wedge (\overline{x_2^a} \vee \overline{x_2^b})$$
$$E^{e_2} = (\overline{x_1^c} \vee \overline{x_1^b}) \wedge (\overline{x_2^c} \vee \overline{x_2^b})$$
$$E^{e_3} = (\overline{x_1^a} \vee \overline{x_1^c}) \wedge (\overline{x_2^a} \vee \overline{x_2^c})$$

In questo caso i colori a disposizione sono 1 e 2 (pedice delle variabili). Il grafo non ha colorazione propria perché non si riesce a trovare una combinazione di colori sui vertici tale per cui eq. (2.3) è vera. Si dimostra la riduzione.

Dimostrazione 2 ($\implies$). *Si assume G k-colorabile. Sia $\{c(v) : v \in V\}$ una colorazione propria ovvero $\forall e = (u, v) \in E : c(u) \neq c(v)$ e con $c(v) \in \{1, \ldots, k\}$. Si definisce l'assegnamento $a = (a_1^{v_1}, \ldots, a_k^{v_1}, \ldots, a_1^{v_n}, \ldots, a_k^{v_n})$ e si assegna*

$$a_j^{v_i} = \begin{cases} T & c(v_i) = j \\ F & c(v_i) \neq j \end{cases} \tag{2.4}$$

Si mostra quindi che ogni gruppo di clausole è soddisfatto dall'assegnamento. Preso un $C^{|v|}(a)$[4] si ha che l'assegnamento rende vera ogni clausola della eq. (2.1)

- *$C^v(a) = T$ perché per ogni vertice, in particolare per v, una variabile associata ha valore T (esiste un legame tra il vertice e il colore per definizione di $a_j^{v_i}$);*
- *$D^v(a) = T$ perché nella definizione viene associato univocamente un valore alla variabile;*
- *$E^v(a) = T$ perché la colorazione è propria (poiché viene assunto che G sia $k - col$ e che la colorazione associata a G sia propria), ovvero $\forall e = (u, v) : c(u) \neq c(v)$. Se fosse $E^v(a) = F$ allora*

  $$\exists i : \overline{x_i^u} \vee \overline{x_i^v} = F \implies x_i^u = T \wedge x_i^v = T \implies a_i^u = T \wedge a_i^v = T$$

  ma per la eq. (2.4) *risulta $c(u) = i \wedge c(v) = i \implies c(u) = c(v)$ in contraddizione con l'assunzione che la colorazione è propria.*

concludendo che $\phi_G(a) = T$. □

Per dimostrare la coimplicazione si parte da un assegnamento che rende vera una formula ϕ senza conoscere il grafo.

Dimostrazione 3 ($\impliedby$). *Si assume che $\phi_G(a) = T$ e si mostra che G è $k-col$. L'assegnamento $a = (a_1^{v_1}, \ldots, a_n^{v_1}, \ldots, a_1^{v_n}, \ldots, a_n^{v_n})$ tale che $\phi_G(a) = T$. Si definisce una colorazione per G basata su a:*

$$\forall v \in V : c(v) = i \iff a_i^v = T$$

1. *ogni vertice ha un solo colore, infatti se $\phi(a) = T$ segue che:*
   - $C^v(a) = T \implies \exists i : a_i^v = T \implies c(v) = i$

[4] abuso di notazione, si immagini ci siano scritte tutte le variabili dell'assegnamento a.

- $D^v(a) = T \implies \nexists i, j : a_i^v = T \wedge a_j^v = T$

da cui

$$!\exists i : c(v) = i$$

2. *ogni arco* ***non è monocromatico*** *ovvero* $c(u) \neq c(v)$. *Poiché* $\phi(a) = T \implies E^e(a) = T \implies \nexists i : a_i^u = T \wedge a_i^v = T \implies c(u) \neq c(v)$.

Da cui G *è* $k - col$. □

Dunque $k - col \preceq SAT$.

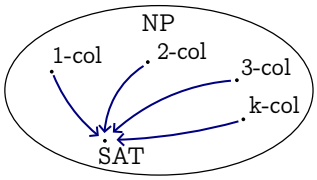

Figura 2.4: tutti i problemi k-col si riducono a SAT. Si vedrà che tutti i problemi si riducono a SAT.

2.1.5 Problema Circuit-SAT

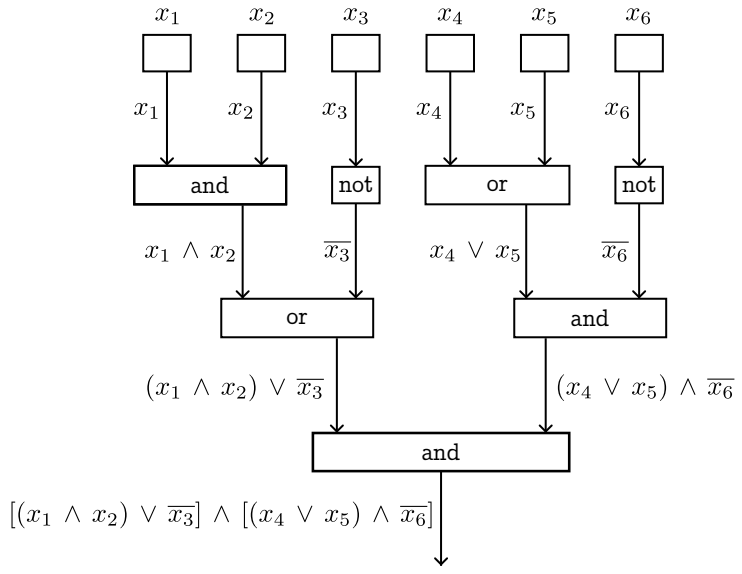

Figura 2.5: circuito booleano.

Circuit-SAT
Input: Circuito booleano C **Output:** *yes* $\iff$ C è soddisfacibile

Definizione 5 (Circuito booleano)**.** *È un grafo aciclico diretto in cui ogni vertice ha un in-degree=1,2 tranne i vertici di input che hanno un in-degree=0.*

Ogni vertice ha un out-degree=1 e ha associato un ***operatore booleano*** *and, or, not. In particolare not è associato a tutti i vertici con in-degree=1 mentre and e or a quelli con in-degree=2.*

Dato un assegnamento in input a $x_1, \ldots, x_n$ allora $C(x_1, \ldots, x_n) = v$ con v valore dell'arco in output. Nell'esempio in fig. 2.5 si ha $C(0, 1, 1, 1, 1, 1) = 0$.

2.1.6 Teorema di Cook-Levin

Teorema 1 (Cook-Levin)**.** *SAT è NP-completo*

Per dimostrare il teorema 1 si deve dimostrare che (✓ = già dimostrato)

- SAT $\in$ NP ✓
- Circuit-SAT $\preceq$ SAT
- Circuit-SAT è NP-completo
  - Circuit-SAT $\in$ NP ✓
  - Circuit-SAT è NP-hard

$$\forall \mathbb{A} \in \text{NP} : \ \mathbb{A} \preceq \text{Circuit-SAT}$$

Lemma 1. *Per ogni problema $\mathbb{B} \in P$ e per ogni $n \in \mathbb{N}$*

$$\exists C_n : \ \forall x \in \mathcal{I}(\mathbb{B}) \quad |x| = n \ \wedge \ C_n(x) = \mathbb{B}(x)$$

Inoltre C_n è computabile in tempo polytime ovvero $\exists c : \ O(|x|^c)$.

Il *lemma* 1 afferma che, fissato un input x con $|x| = n$, un algoritmo A può essere tradotto in un circuito booleano C_n tale che

$$A(x) = C_n(x)$$

Dimostrazione 4 (Circuit-SAT è NP-hard)**.** *Affermare*

$$\forall \mathbb{A} \in \mathit{NP} : \ \mathbb{A} \preceq \mathit{Circuit\text{-}SAT}$$

significa

$$x \in \mathcal{I}(\mathbb{A}) \xrightarrow[in \ x]{polytime} C_x^{\mathbb{A}} : \ \mathbb{A}(x) = yes \iff C_x^{\mathbb{A}} \text{ è soddisfacibile}$$

Sostenere che $\mathbb{A} \in NP$ implica che

$$\exists B(x, y) : \ \forall x \in \mathcal{I}(\mathbb{A}) \quad \mathcal{I}(\mathbb{A})(x) = yes \iff \exists y \in \{0, 1\}^{|x|^c}$$

Fissato x, il problema associato al calcolo di $B(x, y)$

Calcolo di B(x, y)
Input: y ***Output:*** $B(x, y)$

è in P. Seguendo quanto riportato dal lemma 1 si può affermare che

$$\exists C_n^{\mathbb{A}} : \forall y \quad |y| = n \quad C_n^{\mathbb{A}}(y) = B(x, y)$$

$C_n^{\mathbb{A}}$ *è calcolabile in polytime nell'input, quindi* $O(n^d) = O(|y|^d) = O(|x|^{c \cdot d})$.

$$\begin{aligned} x \in \mathcal{I}(\mathbb{A}) \text{ è } yes &\iff \exists y : B(x, y) = yes \\ &\iff \exists y : C_n^{\mathbb{A}}(y) = yes \\ &\iff C_n^{\mathbb{A}} \text{ è un'istanza } yes \text{ per Circuit-SAT} \\ &\iff \textit{Circuit-SAT}(C_n^{\mathbb{A}}) = yes \end{aligned}$$

da cui si ottiene

$$x \in \mathcal{I}(\mathbb{A}) \text{ è } yes \iff \textit{Circuit-SAT}(C_n^{\mathbb{A}}) = yes$$

Quindi Circuit-SAT $\preceq$ *SAT* □

Si dimostra che Circuit-SAT $\preceq$ SAT facendo riferimento ad un circuito booleano arbitrario in fig. 2.6

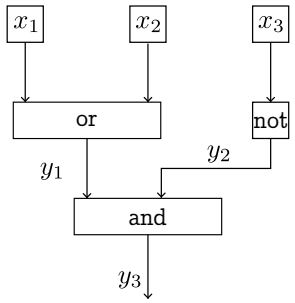

Figura 2.6: circuito booleano della dimostrazione.

Dimostrazione 5 (Circuit-SAT $\preceq$ SAT)**.**

$$\exists x : C(x) = T \iff \exists x' : \phi(x') = T$$

Si pone $x_1 \vee x_2 = y_1$, $\overline{x_3} = y_2$ *e* $y_3 = y_1 \wedge y_2$ *da cui si ottiene la formula* ϕ

$$\phi(x_1, x_2, x_3, y_1, y_2, y_3) = [(x_1 \vee x_2) = y_1] \wedge [\overline{x_3} = y_2] \wedge [y_3 = y_1 \wedge y_2] \wedge y_3$$

Osservazione $a = b \equiv (\overline{a} \vee b) \wedge (a \vee \overline{b})$

Quindi

$$\exists x : C(x) = T \iff \exists x, y : \phi(x, y) = T$$

Si sa che $\forall \mathbb{A} \in NP \quad \mathbb{A} \preceq \textit{Circuit-SAT} \preceq \textit{SAT}$

$$x \to C_x^{\mathbb{A}} \to \phi$$

e dunque $\mathbb{A} \preceq$ *SAT. SAT è NP-hard e appartiene alla classe NP. Dunque SAT è NP-completo.* □

Dimostrare che un problema è NP-completo

Per dimostrare che un problema $\mathbb{D}$ è NP-completo basta mostrare che

- $\mathbb{D} \in$ NP facile perché basta mostrare che una soluzione è verificabile in tempo polinomiale;
- $\forall \mathbb{A} \in$ NP $\quad \mathbb{A} \preceq \mathbb{D}$ la si dimostra usando la transitività delle riduzioni. Si sceglie un problema $\mathbb{B} \in$ NPC e si dimostra che $\mathbb{B} \preceq \mathbb{D}$. Poiché $\forall \mathbb{A} \quad \mathbb{A} \preceq \mathbb{B} \wedge \mathbb{B} \in \text{NPC} \implies \forall \mathbb{A} \quad \mathbb{A} \preceq \mathbb{D}$ (NP-hardness).

Definizione 6 (k-CNF)**.** *Una formula k-CNF è una CNF in cui tutte le clausole hanno al più k letterali.*

Si consideri la formula CNF

$$\phi = \underbrace{(x_1 \vee x_2 \vee x_3)}_{C^1} \wedge \underbrace{(x_1 \vee \overline{x_2} \vee \overline{x_4} \vee x_5)}_{C^2} \wedge \underbrace{(x_1 \vee \overline{x_3} \vee \overline{x_4} \vee x_5 \vee x_6)}_{C^3} \tag{2.5}$$

3-SAT
Input: 3-CNF ϕ **Output:** *yes* $\iff$ ϕ è soddisfacibile

Osservazione 2-SAT $\in$ NP. 3-SAT $\in$ NP come per SAT. Si vuole dimostrare che 3-SAT è NP-completo, mostrando che

$$\text{SAT} \preceq \text{3-SAT}$$
$$\underbrace{\phi}_{\text{CNF}} \rightarrow \underbrace{\phi'}_{\text{3-CNF}}$$

Data una clausola C che è formata da più di 3 letterali, si costruiscono le clausole $D_1, D_2, \ldots, D_t$ con 3 letterali tale che C è soddisfacibile $\iff$ $D_1 \wedge D_2 \wedge \ldots \wedge D_t$ è soddisfacibile. Si supponga che

$$C = l_1 \vee l_2 \vee l_3 \vee l_4 \vee \ldots \vee l_k \quad con \quad k > 3$$

dove $l_i \in \{x_1, \overline{x_1}, \ldots, x_n, \overline{x_n}\}$. La clausola CNF si può riscrivere come una 3-CNF

$$(l_1 \vee l_2 \vee z_1) \wedge (\overline{z_1} \vee l_3 \vee z_2) \wedge (\overline{z_2} \vee l_4 \vee z_3) \wedge \ldots \wedge (\overline{z_{k-3}} \vee l_{k-1} \vee l_k)$$

e considerando la ϕ e passandola in forma 3-CNF diventa

$$C' = (l_1 \vee l_2 \vee z_1) \wedge (\overline{z_1} \vee l_3 \vee z_2) \wedge (\overline{z_2} \vee l_4 \vee z_3) \wedge (\overline{z_3} \vee l_5 \vee l_6)$$

che contiene clausole di taglia 3. Alcune osservazioni:

- il numero dei letterali della nuova clausola C' è meno del doppio di quella originale C;
- se C è soddisfacibile, esiste un assegnamento che rende soddisfacibile anche la clausola C'. Quindi esiste un assegnamento che rende almeno un letterale vero. Usando la scelta che soddisfa C che la rende vera, allora si usa la stessa scelta su C' e conseguentemente deve risultare vera.

Quindi per verificare che C' è soddisfacibile se C è soddisfacibile, si mostra che, dato un letterale che rende vera la clausola, indipendentemente dagli altri letterali, si pongono i valori di z delle clausole vicine ad un valore che le rende vere. Questo genera un effetto "a catena" per cui si propaga la scelta a tutte le clausole tale che ciascuna risulterà vera.

Per mostrare la coimplicazione, si suppone che esista un assegnamento

$$\exists a_x, a_z : ((l_1 \wedge l_2 \wedge z_1) = T \implies C' = T) \wedge (a_x \textit{ rende vera } C)$$

Si suppone per assurdo che a_x, a_z soddisfa C' ma tutti gli l_i sono falsi. Se a_z rende C' vero ma a_x rende $C = F$, vuol dire che tutti gli l_i sono falsi. Il problema è che se non è presente almeno un letterale vero, i soli z non sono sufficienti a rendere vera C'. Per cui si arriva all'assurdo. Con il verde si indica il vero e con il rosso il falso.

$$(l_1 \vee l_2 \vee z_1) \wedge (\overline{z_1} \vee l_3 \vee z_2) \wedge (\overline{z_2} \vee l_4 \vee z_3) \wedge \ldots \wedge (\overline{z_{k-3}} \vee l_{k-1} \vee l_k)$$

Tornando a considerare l'eq. (2.5) la formula ϕ' diventa

$$\phi' = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee z_1) \wedge \underbrace{(\overline{z_1} \vee \overline{x_1} \vee x_5)}_{C^2} \wedge$$

$$\underbrace{(x_4 \vee \overline{x_3} \vee z_2) \wedge (\overline{z_2} \wedge \overline{x_4} \vee z_3) \wedge (\overline{z_3} \vee x_5 \vee x_6)}_{C^3}$$

Anche questa versione di 3-SAT è NP-completa.

3-SAT (esattamente 3)
Input: una 3-CNF in cui tutte le clausole hanno taglia 3 **Output:** *yes* $\iff$ ϕ è soddisfacibile

Dimostrazione 6 (3-SAT esattamente 3 è NP-completo)**.** *Data una CNF ϕ con clausole da ≤ 3 letterali, possiamo creare una CNF φ con clausole da 3 letterali tali che ϕ è soddisfacibile se e solo se φ è soddisfacibile.*

$$\phi = C^1 \wedge C^2 \wedge \ldots \wedge C^n$$

$$|C^1| = 1 \quad C^1 = l \implies (l \vee z_1 \vee z_2) \wedge (l \vee \overline{z_1} \vee z_2) \wedge (l \vee z_1 \vee \overline{z_2}) \wedge (l \vee \overline{z_1} \vee \overline{z_2})$$

$$|C^2 = 2| \quad C^2 = l_1 \vee l_2 \implies (l_1 \vee l_2 \vee z_1) \wedge (l_1 \vee l_2 \vee \overline{z_1})$$

2.1.7 NAE-k-SAT (Not all equal k-SAT)

NAE-k-SAT (Not all equal k-SAT)
Input: formula k-cnf ϕ **Output:** *yes* $\iff$ ϕ NAE-soddisfa k-SAT (esiste un assegnamento a tale che in ogni clausola C^i, a pone almeno un letterale a T e almeno uno a F)

Si consideri l'assegnamento ϕ

$$\phi(x_1, x_2, x_3) = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$$

con $a = (x_1 = T, x_2 = F, x_3 = T)$. Tale assegnamento non è NAE-soddisfacibile per ϕ. L'assegnamento $b = (x_1 = F, x_2 = F, x_3 = T)$ NAE-soddisfa ϕ.

Osservazione Data ϕ, se $a = (a_1, \ldots, a_n)$ NAE-soddisfa ϕ allora anche $\overline{a} = (\overline{a_1}, \ldots, \overline{a_n})$ NAE-soddisfa ϕ. Inoltre

$$\overline{a_i} = \begin{cases} F & a_i = T \\ T & a_i = F \end{cases}$$

NAE-3-SAT è NP-completo

1. NAE-3-SAT $\in$ NP: esiste un verificatore polinomiale ✓
2. NAE-3-SAT è NP-hard: 3-SAT $\preceq$ NAE-4-SAT $\preceq$ NAE-3-SAT. Si dà la dimostrazione in due parti.

Definizione 7 (Riduzione polinomiale)**.** $\mathbb{A}$ *è NP-completo e* $\mathbb{B} \in$ *NP e*

$$\underbrace{\mathbb{A} \preceq \mathbb{B}}_{\mathbb{B} \text{ è NP-hard}} \implies \mathbb{B} \text{ è NP-completo}$$

Una riduzione da $\mathbb{A}$ *a* $\mathbb{B}$ *(*$\mathbb{A} \preceq \mathbb{B}$*) permette di risolvere* $\mathbb{A}$ *in polytime se esiste un programma/algoritmo che risolve* $\mathbb{B}$ *in polytime.*

Si definisce un risolutore per $\mathbb{A}(x)$

Input: $x \in \mathcal{I}(\mathbb{A})$
$y \leftarrow Riduzione(x)$
return $Solutore - \mathbb{B}(y)$

e la subroutine di riduzione

Subroutine riduzione(x)
Input: $x \in \mathcal{I}(\mathbb{A})$
Output: $y \in \mathcal{I}(\mathbb{B})$

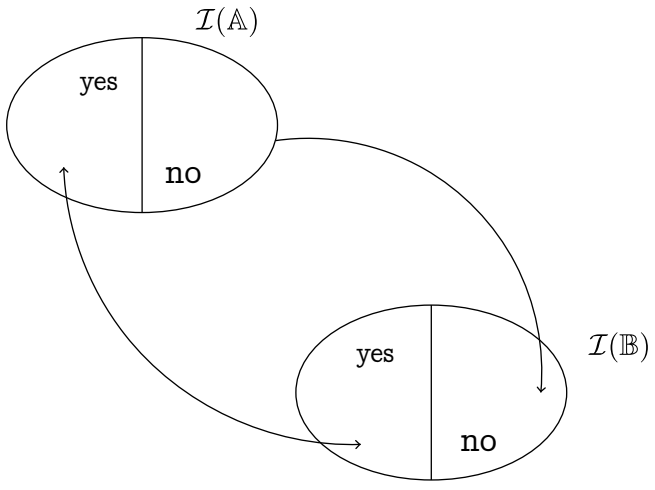

Figura 2.7:

Dimostrazione 7 (3-SAT $\preceq$ NAE-4-SAT)**.** *Per mostrare*