Lezione sulle Riduzioni in Informatica Teorica

Appunti da Trascrizione Automatica

30 giugno 2025

Indice

1 Introduzione al Concetto di Riduzione

Le riduzioni sono un concetto fondamentale in Informatica Teorica, utilizzate per dimostrare risultati di indecidibilità e irriconoscibilità. Esse permettono di correlare la difficoltà computazionale di diversi problemi.

1.1 Metafora e Intuizione della Riduzione

L'idea intuitiva di riduzione è quella di trasformare un problema da risolvere (problema di partenza) in un altro problema (problema di arrivo) per il quale si conosce già una soluzione.

Esempio 1.1 (Gestione Tornei di Calcetto)**.** *Si desidera creare un'applicazione web per la gestione di tornei di calcetto, in particolare un algoritmo per la generazione automatica del calendario delle partite (problema di decisione).*

Problema di Partenza (A): Generazione Calendario

- ***Input****: Una lista di squadre (es. Juventus, Milan, Bologna, Roma).*
- ***Output desiderato****: Un calendario di partite, ovvero una partizione dell'insieme di tutte le possibili partite in sottoinsiemi (giornate), tale che in ogni sottoinsieme le partite siano compatibili (nessuna squadra gioca più di una partita nella stessa giornata).*

Inizialmente, la soluzione diretta per questo problema risultava difficile.

Problema di Arrivo (B): K-Colorabilità di un Grafo *Si conosceva già un algoritmo funzionante per la K-colorabilità di un grafo.*

- ***Input****: Un grafo G e un numero intero K (numero di colori).*
- ***Output****: Una colorazione dei nodi di G con K colori tale che nodi adiacenti abbiano colori diversi, oppure un'indicazione che non è possibile.*

L'Idea della Riduzione*: Trasformare un'istanza del problema di generazione del calendario in un'istanza del problema di K-colorabilità, usare il risolutore della K-colorabilità e poi ritrasformare la soluzione ottenuta per risolvere il problema originale.*

Funzione di Riduzione (f):

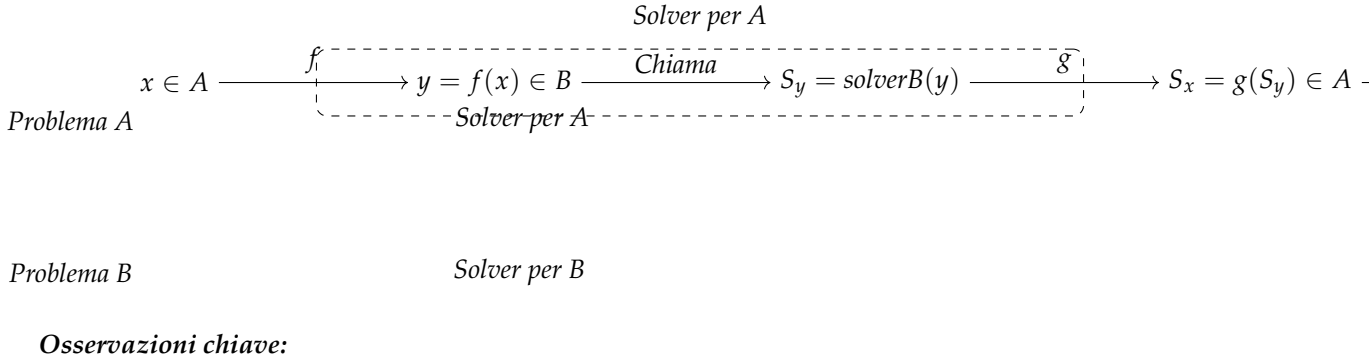
- ***Input di*** f*: Lista di squadre (istanza del problema A).*
- ***Output di*** f*: Coppia (G', K') (istanza del problema B).*
- ***Costruzione di*** G'*:*
  - *I nodi di G' corrispondono a tutte le possibili partite (coppie di squadre). Per N squadre, ci sono $\binom{N}{2}$ possibili partite/nodi.*
  - *Gli archi di G' collegano due nodi se le partite corrispondenti sono incompatibili (cioè, hanno una squadra in comune).*
- ***Determinazione di*** K'*: K' è il numero minimo di giornate necessarie. Per un campionato "all'italiana" con N squadre (ogni squadra gioca contro tutte le altre), $K' = N - 1$.*
- ***Esempio con 4 squadre (J, M, B, R)****:*
  - *Nodi: {JM, JB, JR, MB, MR, BR}*

- *Archi: (JM, JB) (J in comune), (JM, JR), (JB, JR), (MB, MR), (MB, BR), (MR, BR), (JM, MB) (M in comune), (JM, MR), (JB, BR), (JR, MB), (JR, MR), (JB, MR) etc. (ogni coppia di partite incompatibili è collegata da un arco).*
- $K' = 4 - 1 = 3$ *colori.*

***Workflow complessivo per risolvere il problema A**:*

1. *Si riceve un'istanza x (lista di squadre) del problema A.*
2. *Si applica la funzione di riduzione f per ottenere un'istanza $y = f(x) = (G', K')$ del problema B.*
3. *Si invia y a un risolutore (solver) del problema B, ottenendo una soluzione S_y (una colorazione del grafo G').*
4. *Si applica una funzione g per ritrasformare S_y in S_x (il calendario del torneo). La funzione g associa ad ogni colore una giornata, e tutte le partite/nodi con lo stesso colore appartengono alla stessa giornata.*
5. *Si restituisce S_x.*

Diagramma schematico:



Osservazioni chiave:

- *La riduzione è una **trasformazione di istanze** da un problema A a un problema B.*
- *La riduzione è **direzionale**: si riduce A a B ($A \leq B$), non viceversa.*
- *La funzione di riduzione f deve essere **progettata ad arte** (in modo "sensato") in modo che la soluzione ottenuta dal problema di arrivo possa essere utilmente riconvertita per il problema di partenza.*
- *Se A si riduce a B ($A \leq B$), intuitivamente A non è più difficile di B.*

2 Definizione Formale di Riduzione

Per i problemi di decisione (che hanno risposta sì/no), la definizione di riduzione è più semplice, poiché la "ritrasformazione" della soluzione non è necessaria (la risposta è già 0 o 1).

Definizione 2.1 (Riduzione tra Linguaggi/Problemi di Decisione)**.** *Siano A e B due linguaggi. Una funzione f è una **riduzione** da A a B, denotata $A \leq B$, se:*

1. *f è **calcolabile**.*

2. *Per ogni stringa w, $w \in A \iff f(w) \in B$.*

Spiegazione:

- La condizione $w \in A \iff f(w) \in B$ significa che f deve mappare istanze "sì" di A a istanze "sì" di B, e istanze "no" di A a istanze "no" di B.
- La calcolabilità di f è cruciale: f non può essere una "funzione magica" che risolve problemi difficili. Deve essere una funzione che può essere calcolata da una Macchina di Turing in tempo finito per ogni input.

2.1 Funzioni Calcolabili e Trasduttori

Per definire formalmente cosa significa che una funzione è "calcolabile", introduciamo il concetto di trasduttore.

Definizione 2.2 (Trasduttore)**.** *Un **trasduttore** è una Macchina di Turing caratterizzata da tre nastri:*

1. ***Nastro di Input**: Sola lettura. Contiene la stringa di input w.*
2. ***Nastro di Lavoro (Work Tape)**: Lettura e scrittura. Usato per i calcoli intermedi.*
3. ***Nastro di Output**: Sola scrittura. Viene scritto il risultato $f(w)$ della funzione.*

Definizione 2.3 (Funzione Calcolabile)**.** *Una funzione $f : \Sigma^* \to \Sigma^*$ è **calcolabile** se esiste un trasduttore M tale che per ogni stringa $w \in \Sigma^*$, M esegue su w, si arresta (termina) e lascia $f(w)$ sul nastro di output.*

Nota: Quando si specifica una riduzione, non è necessario mostrare esplicitamente il trasduttore. È sufficiente una descrizione algoritmica (pseudocodice o linguaggio naturale) sufficientemente precisa da far capire come l'output $f(w)$ viene generato dall'input w.

3 Proprietà delle Riduzioni e Indecidibilità

Le riduzioni sono uno strumento potente per dimostrare l'indecidibilità dei linguaggi, sfruttando la relazione di "non più difficile di".

Teorema 3.1 (Proprietà Fondamentali delle Riduzioni)**.** *Siano A e B due linguaggi tali che $A \leq B$.*

1. *Se $A \notin R$ (A non è ricorsivo/decidibile), allora $B \notin R$ (B non è ricorsivo/decidibile).*
2. *Se $A \notin RE$ (A non è ricorsivamente enumerabile/riconoscibile), allora $B \notin RE$ (B non è ricorsivamente enumerabile/riconoscibile).*

Dimostrazione. Procediamo per contraddizione.

Dimostrazione 1 (Decidibilità): Vogliamo dimostrare: $A \notin R \implies B \notin R$. Assumiamo per assurdo la negazione della tesi: $A \notin R \land B \in R$. Se $B \in R$, allora esiste una Macchina di Turing decisore M_B che decide B. Poiché $A \leq B$, esiste una funzione calcolabile f tale che $w \in A \iff f(w) \in B$. Consideriamo la seguente costruzione di una Macchina di Turing M_A che decide A:

```
M_A \text{ su input } w:
1.  Calcola } f(w). \text{ (Questo è possibile perché } f \text{ è calcolabile)}.
2.  Simula } M_B \text{ su } f(w).
3.  Se } M_B \text{ accetta, } M_A \text{ accetta.
4.  Se } M_B \text{ rifiuta, } M_A \text{ rifiuta.
```

Analisi di M_A:

- M_A decide A: Per definizione di riduzione, $w \in A \iff f(w) \in B$. Poiché M_B decide B, M_B accetta $f(w)$ se e solo se $f(w) \in B$. Dunque, M_A accetta w se e solo se $w \in A$.
- M_A si arresta sempre: f si arresta sempre (perché è calcolabile) e M_B si arresta sempre (perché è un decisore). Quindi M_A si arresta sempre su ogni input.

Questo significa che M_A è un decisore per A, quindi $A \in R$. Ma ciò contraddice la nostra assunzione iniziale che $A \notin R$. Pertanto, l'assunzione è falsa, e la tesi $A \notin R \implies B \notin R$ è vera. ■

Dimostrazione 2 (Riconoscibilità): Vogliamo dimostrare: $A \notin RE \implies B \notin RE$. Assumiamo per assurdo la negazione della tesi: $A \notin RE \land B \in RE$. Se $B \in RE$, allora esiste una Macchina di Turing accettore M'_B che riconosce B. Poiché $A \leq B$, esiste una funzione calcolabile f tale che $w \in A \iff f(w) \in B$. Consideriamo la seguente costruzione di una Macchina di Turing M'_A che riconosce A:

```
M_A' \text{ su input } w:
1.  Calcola } f(w). \text{ (Possibile perché } f \text{ è calcolabile)}.
2.  Simula } M_B' \text{ su } f(w).
3.  Se } M_B' \text{ accetta, } M_A' \text{ accetta.
```

Analisi di M'_A:

- M'_A riconosce A: Per definizione di riduzione, $w \in A \iff f(w) \in B$. Poiché M'_B riconosce B, M'_B accetta $f(w)$ se e solo se $f(w) \in B$. Dunque, M'_A accetta w se e solo se $w \in A$.
- M'_A non si arresta necessariamente su input non in A: Se $w \notin A$, allora $f(w) \notin B$. Poiché M'_B è un accettore (non un decisore), su input non in B può non terminare. Quindi, M'_A può non terminare su input non in A. Questo è consistente con la definizione di accettore.

Questo significa che M'_A è un accettore per A, quindi $A \in RE$. Ma ciò contraddice la nostra assunzione iniziale che $A \notin RE$. Pertanto, l'assunzione è falsa, e la tesi $A \notin RE \implies B \notin RE$ è vera. ■ □

Corollario 3.2 (Corollario delle Riduzioni)**.** *Siano A e B due linguaggi tali che $A \leq B$.*

1. *Se $B \in R$, allora $A \in R$.*
2. *Se $B \in RE$, allora $A \in RE$.*

Dimostrazione. Queste affermazioni sono le contronominali del teorema precedente. 1. Se $A \notin R \implies B \notin R$ (Teorema 1). La sua contronominale è $B \in R \implies A \in R$. 2. Se $A \notin RE \implies B \notin RE$ (Teorema 2). La sua contronominale è $B \in RE \implies A \in RE$. ■ □

Interpretazione del Corollario:

- Se il problema di arrivo (B) è decidibile/riconoscibile, allora anche il problema di partenza (A) lo è.
- Questo implica che se vogliamo dimostrare che un problema A è decidibile/riconoscibile, possiamo cercare di ridurlo a un problema B che sappiamo essere decidibile/riconoscibile.

4 Esempio: Indecidibilità di L_{NE} (Non-Empty Language)

Introduciamo un nuovo linguaggio e ne studiamo la decidibilità tramite riduzione.

Definizione 4.1 (Linguaggio L_{NE})**.** *Il linguaggio L_{NE} (Non-Empty) è l'insieme dei codici di Macchine di Turing M_i tali che il linguaggio accettato da M_i non è vuoto: $L_{NE} = \{\langle M_i \rangle \mid L(M_i) \neq \emptyset\}$ Intuitivamente, L_{NE} contiene le Macchine di Turing che accettano almeno una stringa.*

4.1 L_{NE} è Ricorsivamente Enumerabile ($L_{NE} \in RE$)

Proposizione 4.2. $L_{NE} \in RE$.

Dimostrazione. Possiamo costruire una Macchina di Turing M_{NE} che riconosce L_{NE}. M_{NE} prende in input il codice di una Macchina di Turing $\langle M \rangle$. L'obiettivo è verificare se M accetta almeno una stringa.

```
M_{NE} \text{ su input } \langle M \rangle:
1.  Per ogni possibile stringa } w \in \Sigma^* \text{ (in ordine lessicografico):
2.      Simula } M \text{ su } w.
3.      Se } M \text{ accetta } w \text{ (e si ferma in stato di accettazione), } M_{NE} \text{ ac
```

Questo approccio iterativo presenta un problema: se M non termina su una stringa w, M_{NE} si blocca e non può passare a testare altre stringhe. Per risolvere questo, si può usare una Macchina di Turing non deterministica, oppure una strategia di "dovetailing" (simulazione parallela):

```
M_{NE} \text{ (versione non deterministica) su input } \langle M \rangle:
1.  Indovina (non-deterministicamente) una stringa } w \in \Sigma^*.
2.  Simula la Macchina Universale } U \text{ su } (\langle M \rangle, w).
3.  Se } U \text{ accetta } (\langle M \rangle, w), \text{ allora } M_{NE} \text{ accetta } \langl
```

Analisi:

- Se $L(M) \neq \emptyset$, allora esiste almeno una stringa w_0 tale che M accetta w_0. In tal caso, M_{NE} può non-deterministicamente indovinare w_0, simulare M su w_0, e accettare. Quindi, se $\langle M \rangle \in L_{NE}$, M_{NE} accetta $\langle M \rangle$.
- Se $L(M) = \emptyset$, allora M non accetta alcuna stringa. Indipendentemente da quale w M_{NE} indovina, la simulazione di M su w non porterà mai all'accettazione. Quindi, M_{NE} non accetterà $\langle M \rangle$.

Poiché le Macchine di Turing non deterministiche sono equivalenti a quelle deterministiche per la classe RE, esiste una Macchina di Turing deterministica che riconosce L_{NE}. Quindi, $L_{NE} \in RE$. ■ □

4.2 L_{NE} non è Ricorsivo ($L_{NE} \notin R$)

Intuitivamente, è difficile decidere se $L(M) = \emptyset$. Per rispondere "no" (cioè $M \in L_E$, $L(M) = \emptyset$), bisognerebbe essere sicuri che M non accetta *nessuna* stringa, il che richiederebbe un tempo infinito per provare tutte le possibili stringhe.

Teorema 4.3. *$L_{NE} \notin R$.*

Dimostrazione. Dimostriamo che L_{NE} è indecidibile (non ricorsivo) tramite una riduzione dal linguaggio universale L_U, che sappiamo essere indecidibile ($L_U \notin R$). Vogliamo dimostrare che $L_U \leq L_{NE}$.

Problema di Partenza (A): $L_U = \{\langle M, w\rangle \mid M \text{ accetta } w\}$ **Problema di Arrivo (B)**: $L_{NE} = \{\langle M'\rangle \mid L(M') \neq \emptyset\}$

La nostra funzione di riduzione f deve prendere un input $\langle M, w\rangle$ (un'istanza di L_U) e produrre un output $\langle M'\rangle$ (un'istanza di L_{NE}) tale che:

$$\langle M, w\rangle \in L_U \iff f(\langle M, w\rangle) = \langle M'\rangle \in L_{NE}$$

Costruzione della funzione di riduzione f: La funzione f prende in input $\langle M, w\rangle$ e costruisce il codice di una nuova Macchina di Turing M', definita come segue:

```
M' \text{ su input } x:
1.  Ignora l'input } x.
2.  Scrive la stringa } w \text{ (ottenuta dall'input di } f) \text{ sul suo nastro.
3.  Simula } M \text{ (ottenuta dall'input di } f) \text{ su } w.
4.  Se } M \text{ accetta } w, \text{ allora } M' \text{ accetta } x.
5.  Se } M \text{ rifiuta } w \text{ o va in loop su } w, \text{ allora } M' \text{ rifiuta } x \t
```

La funzione f è calcolabile, in quanto è un algoritmo ben definito per costruire la descrizione di M' a partire da M e w.

Verifica della condizione $w' \in A \iff f(w') \in B$: Sia $w' = \langle M, w\rangle$.

Caso 1: $\langle M, w\rangle \in L_U$ (istanza "sì" di L_U) Questo significa che M accetta w. Secondo la costruzione di M', se M accetta w, allora M' accetterà *qualsiasi* input x le venga dato. Quindi, il linguaggio di M' è $L(M') = \Sigma^*$. Poiché $\Sigma^* \neq \emptyset$, ne consegue che $\langle M'\rangle \in L_{NE}$. La condizione è soddisfatta: un'istanza "sì" di L_U è mappata a un'istanza "sì" di L_{NE}.

Caso 2: $\langle M, w\rangle \notin L_U$ (istanza "no" di L_U) Questo significa che M non accetta w (cioè, M rifiuta w o M va in loop su w). Secondo la costruzione di M', se M non accetta w, allora M' non accetterà *alcun* input x le venga dato (o andrà in loop su qualsiasi x). Quindi, il linguaggio di M' è $L(M') = \emptyset$. Ne consegue che $\langle M'\rangle \notin L_{NE}$. La condizione è soddisfatta: un'istanza "no" di L_U è mappata a un'istanza "no" di L_{NE}.

Poiché abbiamo dimostrato che $L_U \leq L_{NE}$ e sappiamo che $L_U \notin R$, per il Teorema sulle Proprietà Fondamentali delle Riduzioni (punto 1), possiamo concludere che $L_{NE} \notin R$. ■ □

4.3 Linguaggio Complementare L_E

Consideriamo il linguaggio L_E, che è il complemento di L_{NE}.

Definizione 4.4 (Linguaggio L_E)**.** *Il linguaggio L_E (Empty) è l'insieme dei codici di Macchine di Turing M_i tali che il linguaggio accettato da M_i è vuoto: $L_E = \{\langle M_i\rangle \mid L(M_i) = \emptyset\}$*

Sappiamo che $L_{NE} \in RE$ ma $L_{NE} \notin R$. Un linguaggio L è decidibile ($L \in R$) se e solo se sia L che il suo complemento $\bar{L}$ sono ricorsivamente enumerabili ($L \in RE$ e $\bar{L} \in RE$). Poiché $L_{NE} \in RE$ ma $L_{NE} \notin R$, ne consegue che il suo complemento, $L_E = \overline{L_{NE}}$, non può essere ricorsivamente enumerabile. Quindi, $L_E \notin RE$.