MAIN COLACOLABILITÀ

Contents

Part 1. Automi e linguaggi

1. Convenzioni

Definizioni preliminari e prime proprietà

- un **alfabeto** è un insieme finito di simboli $\Sigma = \{a, b, c\}$
- la **stella di Kleene** $\Sigma^* = \{\varepsilon, a, b, c, aa, ab, ac, ...\}$ è il set di tutte le possibili parole (inclusa la parola vuota ε) sull'alfabeto Σ
- una **parola** è una sequenza di lettere $w = abbaac \in \Sigma^*$
  - $| w |$ è il numero dei simboli della parola `w`, dove $| \varepsilon |= 0$ e $| wa |=| w | +1$
  - $w \circ u = a_1a_2...a_nb_1b_2...b_m$ è la **concatenazione** della parola $w = a_1a_2...a_n$ e $u = b_1b_2...b_m$
  - $w^n = ww...w$ è la sua **potenza**, dove $w^0 = \varepsilon$ e $w^n = w^{n-1}w$ per $n \geq 0$
- un **linguaggio** è un set di parole su un certo alfabeto
  - il linguaggio `A` è **chiuso** rispetto ad un'operazione `f` se applicando `f` agli elementi di `A` si ottiene sempre un elemento di `A`
  - **unione** $A \cup B = \{w \mid w \in A \vee w \in B\}$
  - **concatenazione** $AB = \{uw \mid u \in A, w \in B\}$
  - **stella** $A^* = \{w_1w_2...w_k \mid k \geq 0 \wedge \forall i, w_i \in A\}$

2. Linguaggi regolari

2.1. DFA - automa a stati finiti deterministico.

Un **DFA** è un modello per computer con una quantità limitata di memoria.

Definition 2.1. Un **DFA** si indica formalmente come una 5-tupla $M = (Q, \Sigma, \delta, q_0, F)$, dove

- `Q` è l'insieme finito degli stati
- Σ è l'insieme finito dei simboli di input
- $\delta : Q \times \Sigma \to Q$ è la funzione di transizione
- $q_0 \in Q$ è lo stato iniziale
- $F \subseteq Q$ è l'insieme finito degli stati d'accettazione

Definition 2.2. Se `A` è l'insieme delle di parole che la macchina `M` accetta, allora si dice che $A = L(M)$ è il linguaggio di `M` e che `M` **riconosce** `A`. Si indica con $\mathcal{L}(DFA) = \{A \mid \exists M \in DFA \wedge A = L(M)\}$ la **classe dei linguaggi accettati** da un DFA.

Definition 2.3. Un linguaggio si dice **regolare** se è riconosciuto da un qualche DFA.

2.2. Problemi decisionali.

2.2.1. *Problema del vuoto.*

$$E_{DFA} = \{< A >\mid A \in DFA \wedge L(A) = \emptyset\}$$

Si riduce al problema di determinare se esiste un cammino tra due vertici di un grafo, ovvero $L(A) = \emptyset \iff$ esiste un cammino tra lo stato iniziale ed uno stato d'accettazione

2.2.2. *Problema dell'infinito.*

$$INF_{DFA} = \{< A >| \ A \in DFA \wedge L(A) \text{ infinito}\}$$

Si riduce al problema di determinare se esiste un ciclo nel grafo (eliminati gli stati irragiungibili), ovvero $L(A)$ infinito $\iff$ esiste un ciclo nel grafo

2.2.3. *Problema della totalità.*

$$TOT_{DFA} = \{< A >| \ A \in DFA \wedge L(A) = \Sigma^*\}$$

Si riduce al problema del vuoto, ovvero $L(A) = \Sigma^* \iff \neg L(A) = \emptyset$

2.3. **NFA - automa a stati finiti non-deterministico.** Quando una macchina è in un certo stato e legge il prossimo simbolo in input, sappiamo già quale sarà il suo prossimo stato: ciò si definisce come computazione **deterministica**. In una macchina **nondeterministica**, si possono avere a disposizione vari stati tra cui scegliere, oppure si può cambiare stato senza consumare input.

Definition 2.4. Un **NFA** si indica formalmente come una 5-tupla $N = (Q, \Sigma, \delta, q_0, F)$, dove

- `Q` è l'insieme finito degli stati
- Σ è l'insieme finito dei simboli di input
- $\delta : Q \times \Sigma_\varepsilon \to P(Q)$ è la funzione di transizione, dove $P(Q)$ è la collezione dei sottoinsiemi di Q e $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$
- $q_0 \in Q$ è lo stato iniziale
- $F \subseteq Q$ è l'insieme degli stati d'accettazione

2.3.1. *Equivalenza.* Due macchine si dicono **equivalenti** se riconoscono lo stesso linguaggio.

Theorem 2.1. *ogni NFA ha un DFA equivalente, ovvero $\mathcal{L}(NFA) = \mathcal{L}(DFA)$.*

Proof. Ogni DFA si può vedere come un caso particolare di NFA, quindi $\mathcal{L}(NFA) \supseteq \mathcal{L}(DFA)$, mentre si può dimostrare che per ogni NFA esiste un DFA equivalente, ovvero $\mathcal{L}(NFA) \subseteq \mathcal{L}(DFA)$, tramite algoritmo:

```
input: NFA N = (Q, Σ, δ, q0, F)
output: DFA M = (Q', Σ, δ', q0', F')
Q' = ε-closure(q0)
WHILE ∃T ∈ Q' | T non marcato DO
    marca T
    FOREACH a ∈ Σ DO
        V = ⋃_{q∈T} ε-closure(δ(q, a))
        IF V ∉ Q' THEN add(Q', V)
        δ'(T, a) = V
```

Infine si definiscono i restanti membri di M come

- $q_0' = \varepsilon$-closure(q_0)
- F' è l'insieme degli stati in Q' che contengono almeno uno stato d'accettazione di N

Se `k` è il numero di stati del NFA `N`, allora il DFA equivalente `M` ha 2^k stati, in quanto ogni sottoinsieme corrisponde ad una delle possibilità che `M` deve ricordare. □

Corollary 2.1. *Un linguaggio è regolare sse qualche NFA lo riconosce.*

Theorem 2.2. *La classe dei linguaggi regolari è chiusa rispetto alle operazioni di unione, intersezione, complemento, concatenazione e stella di kleene.*

2.4. **Espressioni Regolari.** Le espressioni regolari sono un modo per descrivere certi linguaggi.

Definition 2.5. Si dice che $re(\Sigma) = R$ è un'**espressione regolare** sull'alfabeto Σ se `R` è

- `a`, per qualche $a \in \Sigma$
- ε
- $\emptyset$
- $(R_1 \cup R_2)$, dove R_1 e R_2 sono due espressioni regolari
- $(R_1 R_2)$, dove R_1 e R_2 sono due espressioni regolari
- $(R_1)^*$, dove R_1 è un'espressione regolare

L'ordine di valutazione di una espressione regolare, senza le parentesi, è stella - concatenazione - unione.

Remark 2.1. Se `R` è un'espressione regolare, allora $L(R)$ è il linguaggio generato da `R`

2.5. **Teorema di Kleene.**

Theorem 2.3. *Il linguaggio A è regolare sse A ha un'espressione regolare*

$$\mathcal{L}(R) = \mathcal{L}(NFA) = \mathcal{L}(DFA)$$

Lemma 2.1. *Se un linguaggio A è regolare, allora ha un'espressione regolare R, ovvero*

$$\mathcal{L}(NFA) \subseteq \mathcal{L}(R)$$

Proof. Si dimostra tramite algoritmo:

1. si converte il NFA nel **GNFA** equivalente, in forma normale
   (a) ha un solo stato iniziale diverso dallo stato finale
   (b) lo stato finale non ha archi uscenti
   (c) per ogni coppia di nodi (i, j), c'è al più un'arco dal nodo `i` al nodo `j`
2. ripeti, finchè ci sono stati diversi da quello iniziale o finale, l'eliminazione dello stato `q`, per ogni coppia (i, j), per i quali `q` è uno stato intermedio
3. al termine rimane un solo arco dallo stato iniziale allo stato finale, che descrive l'espressione regolare

□

La correttezza dell'algoritmo si basa sul fatto che ogni passo di eliminazione di uno stato conserva il linguaggio accettato dal GNFA e la forma normale del grafo.

3. Linguaggi context-free

3.1. **Pumping lemma.** Il pumping lemma viene usato per provare che un certo linguaggio `A` non è regolare.

Theorem 3.1. *Se A è un linguaggio regolare, riconosciuto da un certo DFA con n stati, allora tutte le sue parole w, $\mid w \mid \geq n$, si possono ottenere come concatenazione di tre sottoparole $w = xyz$, tali che:*

(1) $| y |> 0$
(2) $\forall i \geq 0,\ xy^i z \in A$
(3) $| xy |\leq n$

Proof. Sia w una parola, $m =| w |$ e n il numero di stati dell'automa, allora servono $m+1$ stati per riconoscere w, e se $m \geq n$ allora $m+1 > n$, ovvero c'è uno stato che si ripete tra quelli lungo il cammino determinato da w.

(1) il ciclo è minimo di lunghezza 2, quindi $y \neq \varepsilon$ e $| y |> 0$
(2) percorrendo il ciclo si ottiene sempre una parola in A, quindi $w_i = xy^i z \in A$
(3) per provare il lemma basta prendere in considerazione una sola decomposizione, quella determinata dal primo ciclo incontrato sul cammino, quindi xy è lungo al più n

□

3.2. Context-free grammar.

Definition 3.1. una **CFG** viene definita formalmente come una 4-tupla $G = (V, \Sigma, R, S)$ dove

- `V` è il set finito delle variabili
- Σ è il set finito dei terminali
- `R` è il set finito delle regole di sostituzione, della forma $L \to X$, dove $L \in V$ e $X \in (V \cup \Sigma^*)$
- $S \in V$ è la variabile d'inizio

Definition 3.2. Dati $u, v, w \in (V \cup \Sigma^*)$ e $A \to w$, si dice che uAv **produce** uwv, scritto come

$$uAv \Rightarrow uwv$$

dove il simbolo $\Rightarrow$ indica un singolo passo di derivazione, che consiste nel sostituire una variabile con una stringa di variabili o terminali in base ad una regola di sostituzione. Mentre si dice che `u` **deriva** `v`, scritto come $u \Rightarrow^* v$ se

- $u = v$ oppure
- $\exists u_1, u_2, ..., u_k$ per qualche $k \geq 0$ tale che

$$u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow ... \Rightarrow u_k \Rightarrow v$$

dove il simbolo $\Rightarrow^*$ indica una sequenza ≥ 0 di singoli passi di derivazione.

Definition 3.3. Il linguaggio di una CFG $G = (V, \Sigma, R, S)$ è $L(G) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}$. Un qualsiasi linguaggio generato da una CFG si definisce **context-free**.

Theorem 3.2. *I CFL sono chiusi rispetto alle operazioni di unione, concatenazione e stella di kleene.*

3.2.1. *Problema del vuoto per CFL.*

Si può decidere tramite algoritmo se un CFL $L = \emptyset$

```
Input: CFG G = (V, Σ, R, S)
marca tutti i terminali in G
WHILE esistono nuove variabili marcabili DO
    marca ogni variabile a sinistra di una regola,
    la cui parte destra è marcata
IF variabile iniziale S non marcata RETURN true
RETURN false
```

3.3. **PDA - automa a pila.** Automa con un componente in più chiamato stack.

Definition 3.4. Un **PDA** si indica formalmente come una 6-tupla $M = (Q, \Sigma, \mathit{\Gamma}, \delta, q_0, F)$, dove

- `Q` è l'insieme finito degli stati
- Σ è l'insieme finito dei simboli di input
- $\mathit{\Gamma}$ è l'insieme finito dei simboli dello stack
- $\delta : Q \times \Sigma_\varepsilon \times \mathit{\Gamma}_\varepsilon \to P(Q \times \mathit{\Gamma}_\varepsilon)$ è la funzione di transizione, dove $P(Q)$ è la collezione dei sottoinsiemi di `Q`, $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$ e $\mathit{\Gamma}_\varepsilon = \mathit{\Gamma} \cup \{\varepsilon\}$
- $q_0 \in Q$ è lo stato iniziale
- $F \subseteq Q$ è l'insieme degli stati d'accettazione

Theorem 3.3. *Un linguaggio G è context-free sse un qualche PDA P lo riconosce.*

Part 2. Teoria della calcolabilità

4. Turing Machine

Una **Turing Machine** usa un nastro infinito come memoria, inzialemente contenente solo la parola in input e le parole $\sqcup$. La **TM** può sia leggere che scrivere sul nastro e muovere la testina (del nastro) sia a destra che a sinistra.

Definition 4.1. Una **turing machine** è formalmente una 7-tupla $M = (Q, \Sigma, \mathit{\Gamma}, \delta, q_0, q_A, q_R)$, dove

- `Q` è l'insieme finito degli stati
- Σ è l'alfabeto in input, con $\sqcup \notin \Sigma$
- $\mathit{\Gamma}$ è l'alfabeto del nastro, dove $\sqcup \in \Sigma \subseteq \mathit{\Gamma}$
- $\delta : Q \times \mathit{\Gamma} \to Q \times \mathit{\Gamma} \times \{L, R\}$ è la funzione di transizione
- $q_0 \in Q$ è lo stato iniziale
- $q_A \in Q$ è lo stato d'accettazione
- $q_R \in Q$ è lo stato di rifiuto, e $q_R \neq q_A$

Se la TM non entra in uno stato d'accettazione o di rifiuto, la sua esecuzione andrà avanti all'infinito.

Definition 4.2. Un linguaggio è **turing-riconoscibile** se una qualche TM lo riconosce

Definition 4.3. Un linguaggio è **decidibile** se una qualche TM lo decide, ovvero se la TM si arresta sempre per qualsiasi input.

Proposition. ***Tesi Church-Turing:*** *La classe dei linguaggi riconosciuti da una TM, $\mathcal{L}(TM) = \{A \mid \exists M \in TM \wedge L(M) = A\}$, corrisponde alla classe dei problemi effettivamente risolvibili; quindi possiamo identificare l'idea di **algoritmo** con una TM che si arresta sempre.*

4.1. Varianti TM.

Definition 4.4. Una turing machine a **k-nastri** è formalmente una 7-tupla $M = (Q, \Sigma, \mathit{\Gamma}, \delta, q_0, q_A, q_R)$, dove

- `Q` è l'insieme finito degli stati
- Σ è l'alfabeto in input, con $\sqcup \notin \Sigma$
- $\mathit{\Gamma}$ è l'alfabeto del nastro, dove $\sqcup \in \Sigma \subseteq \mathit{\Gamma}$
- $\delta : Q \times \mathit{\Gamma}^k \to Q \times \mathit{\Gamma}^k \times \{L, R\}^k$ è la funzione di transizione, dove $\mathit{\Gamma}^k = \mathit{\Gamma} \times \mathit{\Gamma} \times ... \times \mathit{\Gamma}$ k volte
- $q_0 \in Q$ è lo stato iniziale
- $q_A \in Q$ è lo stato d'accettazione
- $q_R \in Q$ è lo stato di rifiuto, e $q_R \neq q_A$

Theorem 4.1. *per ogni TM a k-nastri M_k, esiste una TM a nastro singolo M_S equivalente, ovvero $L(M_k) = L(M_S)$.*

Definition 4.5. Una turing machine **non-deterministica** è formalmente una 7-tupla $M = (Q, \Sigma, \mathit{\Gamma}, \delta, q_0, q_A, q_R)$, dove

- `Q` è l'insieme finito degli stati
- Σ è l'alfabeto in input, con $\sqcup \notin \Sigma$
- $\mathit{\Gamma}$ è l'alfabeto del nastro, dove $\sqcup \in \Sigma \subseteq \mathit{\Gamma}$
- $\delta : Q \times \mathit{\Gamma} \to P(Q \times \mathit{\Gamma} \times \{L, R\})$ è la funzione di transizione

- $q_0 \in Q$ è lo stato iniziale
- $q_A \in Q$ è lo stato d'accettazione
- $q_R \in Q$ è lo stato di rifiuto, e $q_R \neq q_A$

Theorem 4.2. *Ogni NTM N ha una TM equivalente M, ovvero* $L(N) = L(M)$

5. Decidibilità

Introduciamo una nuova TM denominata **universale**. La UTM può simulare l'esecuzione di ogni altra TM `M` ricevendo come input la sua **descrizione** $< M >$ e utilizza due nastri:

- il primo contiene la descrizione in input di una TM `M` da simulare. Una descrizione non è altro che la rappresentazione di una TM sotto forma di stringa
- il secondo contiene la configurazione corrente di `M`

5.1. Problema dell'accettazione per TM.

Si tratta di capire se esiste un algoritmo che risolve il problema dell'accettazione per TM, o equivalentemente se esiste una TM che decide il linguaggio $A_{TM} = \{< M, w >| \ M \in TM \wedge w \in L(M)\}$.

Theorem 5.1. A_{TM} *non è decidibile.*

Proof. Supponiamo per assurdo che la TM `M` decide il linguaggio A_{TM}, ovvero

$$M(< T, w >) = \begin{cases} \text{accetta} & w \in L(T) \\ \text{rifiuta} & \text{altrimenti} \end{cases}$$

consideriamo il suo complemento $\bar{M}$ su un input del tipo $< T, < T >>$ e costruiamo una tabella con i risultati del calcolo; a questo punto costruiamo una TM D che estrapola i risultati della diagonale, ovvero si comporta come $\bar{M}$ su input $< T, < T >>$:

$$D(< T >) = \begin{cases} \text{rifiuta} & < T > \in L(T) \\ \text{accetta} & \text{altrimenti} \end{cases}$$

eseguendo D per input $< D >$ otteniamo una contraddizione:

- se $< D > \in L(D)$ allora D rifiuta, ovvero $< D > \notin L(D)$
- se $< D > \notin L(D)$ allora D accetta, ovvero $< D > \in L(D)$

□

Corollary 5.1. *Alcuni linguaggi non sono Turing-riconoscibili*

1. *una TM è una stringa di lunghezza finita, quindi si ha un'infinità numerabile di TM*
2. *un linguaggio può essere messo in corrispondenza biunivoca con una sequenza binaria di lunghezza infinita, quindi si ha un'insieme non numerabile di linguaggi*
3. *visto che l'insieme non numerabile è più grande dell'insieme numerabile, esistono più linguaggi che TM in grado di riconoscerli*

Definition 5.1. Un linguaggio A è **coturing-riconoscibile** se il suo complemento $\bar{A}$ è turing-riconoscibile.

Theorem 5.2. *Un linguaggio è* ***decidibile*** *sse è turing-riconoscibile e co-turing-riconoscibile.*

Corollary 5.2. *Il complemento di un linguaggio indecidibile non è turing-riconoscibile.*

6. Riducibilità

Ridurre un problema `A` ad un problema `B` vuol dire che esiste una funzione calcolabile in grado di convertire le istanze del problema `A` in istanze del problema `B`. Se tale funzione esiste, chiamata semplicemente **riduzione**, allora si può risolvere il problema `A` risolvendo il problema `B`.

Definition 6.1. Una funzione $f : \Sigma^* \to \Sigma^*$ si dice **calcolabile** se esiste una TM che, per ogni input `w`, si arresta con $f(w)$ sul nastro.

Definition 6.2. Il linguaggio `A` è **riducibile** al linguaggio `B`, scritto come $A \leq_M B$, se esiste una funzione calcolabile $f : \Sigma^* \to \Sigma^*$ dove, per ogni `w`, si ha che

$$w \in A \Longleftrightarrow f(w) \in B$$

La funzione `f` viene denomitata **riduzione** da `A` verso `B`

Theorem 6.1. *Se $A \leq_M B$ e B è decidibile, allora A è decidibile.*

Corollary 6.1. *Se $A \leq_M B$ e A è indecidibile, allora B è indecidibile.*

Theorem 6.2. *Se $A \leq_M B$ e B è turing-riconoscibile, allora A è turing-riconoscibile.*

Corollary 6.2. *Se $A \leq_M B$ e A non è turing-riconoscibile, allora B non è turing-riconoscibile.*

Theorem 6.3. *Il linguaggio $HALT_{TM} = \{< M, w >| \ M \in TM \text{ e } M \text{ si ferma per } w\}$ è indecidibile.*

Proof. Supponiamo per assurdo che M sia una TM che decide $HALT_{TM}$:

$$M(< T, w >) = \begin{cases} \text{accetta} & \text{se T si ferma su w} \\ \text{rifiuta} & \text{altrimenti} \end{cases}$$

Possiamo utilizzare M per costruire una TM M' che decide A_{TM} (contraddizione):

```
M’: input <T,w>
    esegui M su <T,w>
         se M accetta esegui T su w
             se T accetta w, accetta
             se T rifiuta w, rifiuta
         se M rifiuta, rifiuta
```

□

Part 3. Teoria della Complessità

7. Complessità temporale

Definition 7.1. Siano $f : \Sigma^* \to \Sigma^*$ e $t : \mathbb{N} \to \mathbb{N}$ due funzioni. Diciamo che M calcola f in **tempo polinomiale** se, per ogni input w, al più esegue in tempo $O(t(n))$, dove $n =\mid w \mid$. Sia $TIME(t(n))$ la **classe delle complessità temporali**, ovvero l'insieme di tutti i linguaggi decidibili da una TM in al più tempo $O(t(n))$.

Remark 7.1. La restrizione $t(n) \geq n$ permette all'algoritmo di avere abbastanza tempo da leggere tutto l'input.

Theorem 7.1. *Sia $t(n)$ una funzione. Ogni NTM a nastro singolo con tempo $t(n)$ ha una TM equivalente a nastro singolo, con tempo $2^{O(t(n))}$.*

Proof. Sia N una NTM con tempo $t(n)$. Costruiamo una TM equivalente che simula N.

Per input di lunghezza n, ogni ramo dell'albero di calcolo non-deterministico ha lunghezza al più $t(n)$. Ogni nodo può avere al più b figli, dove b è il numero massimo di scelte legali in base alla funzione di transizione. Quindi il numero di foglie è $b^{t(n)}$ e il numero totale di nodi è al più $O(b^{t(n)}) = 2^{O(t(n))}$. □

Definition 7.2. Sia **P** la classe dei linguaggi decidibili in tempo polinomiale da una TM determinisitca, ovvero $P = \bigcup_{k \geq 1} TIME(n^k)$

Definition 7.3. Un **verificatore** per un linguaggio `A` è un algoritmo `V` (ovvero una TM), dove $A = \{w \mid V \text{ accetta } < w, c > \text{ per qualche parola c}\}$ e la `c` è detta certificato. Il verificatore `V` è detto polinomiale se esegue in tempo polinomiale, sulla lunghezza di `w`. Il linguaggio `A` è polinomialmente verificabile se ha un verificatore polinomiale.

Definition 7.4. **NP** è la classe dei linguaggi polinomialmente verificabili, ovvero si può verificare una potenziale soluzione in tempo polinomiale.

Corollary 7.1. *Equivalentemente possiamo dire che $NP = \bigcup_{k \in \mathbb{N}} NTIME(n^k)$, dove $NTIME(t(n)) = \{L \mid L \text{ è un linguaggio decidibile da una NTM in al più tempo } O(t(n))\}$.*

Remark 7.2. $P \subseteq NP$.

7.1. Problemi NP-Completi.

Alcuni problemi in NP hanno una complessità individuale relativa all'intera classe. Se un algoritmo in tempo polinomiale esistesse per uno di questi problemi, allora tutti i problemi in NP sarebbero risolvibili in tempo polinomiale. Questi problemi si chiamano **NP-completi**.

Definition 7.5. Un linguaggio A è **riducibile polinomialmente** ad un linguaggio B, ovvero $A \leq_P B$, se esiste una funzione calcolabile in tempo polinomiale $f : \Sigma^* \to \Sigma^*$, dove per ogni parola $w \in \Sigma^*$, si ha che

$$w \in A \iff f(w) \in B$$

Theorem 7.2. *Se $A \leq_P B$ e $B \in P$, allora $A \in P$.*

Definition 7.6. Un linguaggio B è **NP-completo** se soddisfa le seguenti condizioni:

(1) $B \in NP$
(2) B è **NP-hard**, ovvero $\forall A \in NP$, A è riducibile polinomialmente a B

Theorem 7.3. *Se $A \leq_P B$ e $A \in$NP-completo e $B \in NP$, allora $B \in$NP-completo.*

7.1.1. *SAT.*

Theorem 7.4. *Teorema Cook-Levin.* $SAT = \{\phi \mid \phi$ *è una formula booleana soddisfacibile*$\}$ *è NP-completo.*

Corollary 7.2. $SAT \leq_P 3SAT$.

7.1.2. *CLIQUE.* Dato un grafo non diretto, un clique è un sottografo completo indotto $3SAT \leq_p CLIQUE$.

7.1.3. *Vertex-cover.* Dato un grafo non diretto, un vertex-cover è un sottoinsieme dei vertici tale che ogni arco del grafo ne tocca uno. $3SAT \leq_p VC$.

8. Complessità spaziale

Definition 8.1. Sia $s : \mathbb{N} \to \mathbb{N}$ una funzione. Le due classi di **complessità spaziale** sono definite come:

- $SPACE(s(n)) = \{L \mid$ L è un linguaggio decidibile da una DTM in al più spazio $O(s(n))\}$
- $NSPACE(s(n)) = \{L \mid$ L è un linguaggio decidibile da una NTM in al più spazio $O(s(n))\}$

Theorem 8.1. *Sia* $s : \mathbb{N} \to \mathbb{N}$ *una funzione; si ha che*

$$TIME(s(n)) \subseteq SPACE(s(n)) \subseteq NSPACE(s(n)) \subseteq TIME(2^{O(s(n))})$$

Proof. Chiaramente $SPACE(s(n)) \subseteq NSPACE(s(n))$; mentre $TIME(s(n)) \subseteq SPACE(s(n))$ visto che una TM, ad ogni passo di calcolo, può accedere al più ad una sola cella del nastro.

Consideriamo una NTM che decide un certo linguaggio, con spazio $s(n)$. La macchina può avere $s(n)$ celle in cui la testina può essere, $g^{s(n)} =\mid \Sigma \mid^{s(n)}$ possibili parole sul nastro, $q =\mid Q \mid$ differenti stati in cui la NTM può essere. Questo vuol dire che ci sono al più $q \cdot s(n) \cdot g^{s(n)}$ differenti configurazioni; se la macchina esegue per più passi, allora ha visitato la stessa configurazione due volte. Sapendo che la NTM termina per qualsiasi input, la TM equivalente allora esegue per al più

$$q \cdot s(n) \cdot g^{s(n)} = q \cdot s(n) \cdot 2^{s(n) \cdot log_2 g} = 2^{O(s(n))}$$

passi, quindi $NSPACE(s(n)) \subseteq TIME(2^{O(s(n))})$. □

Theorem 8.2. ***Teorema di Savitch**: per una qualsiasi funzione* $s : \mathbb{N} \to \mathbb{N}$*, dove* $s(n) \geq n$*,* $NSPACE(s(n)) \subseteq SPACE(s^2(n))$*.*

Proof. Sia N una NTM che decide un linguaggio `A`, con complessità spaziale $s(n)$. Costruiamo una DTM `M` che decide il linguaggio `A`, utilizzando la subroutine $CANYIELD(c_1, c_2, t)$; la subroutine testa se la NTM `N`, per input `w`, può andare dalla configurazione c_1 alla configurazione c_2 in `t` o meno passi, lungo un cammino non-deterministico.

$$canyield(c_1, c_2, t) = \begin{cases} 1 & c_1 \Rightarrow_T^* c_2 \text{ in al più n passi} \\ 0 & altrimenti \end{cases}$$

Costruiamo la TM equivalente

M: `input x`
 `esegui` $canyield(c_1, c_n, 2^{k \cdot s(n)})$
 `se restituisce 1 allora accetta, altrimenti rifiuta`

Se la profondità della ricorsione è $k \cdot s(n)$, e ogni chiamata ricorsiva occupa $O(s(n))$ spazio sul nastro, M occupa $O(s^2(n))$ spazio.

Possiamo modificare l'algoritmo in modo che M non abbia bisogno di conoscere $s(n)$: per ogni valore $s(n) = i$, usiamo canyield per determinare se N utilizza almeno $i + 1$ spazio. Se raggiunge la configurazione d'accettazione, allora M accetta; se nessuna configurazione di lunghezza $i+1$ è raggiungibile, allora M rifiuta; altrimenti M continua per $s(n) = i + 1$. □

8.1. **La classe PSPACE.**

Definition 8.2. La classe dei linguaggi decidibili complessità spaziale polinomiale da una TM è denominato come

$$PSPACE = \bigcup_k SPACE(n^k)$$

La controparte non-deterministica NPSPACE, viene definita in termini di classe NSPACE.

Corollary 8.1. $P \subseteq NP \subseteq PSPACE = NPSPACE$

Definition 8.3. Un linguaggio `B` è **PSPACE-completo** se soddisfa le seguenti condizioni:

(1) $B \in PSPACE$
(2) B è **PSPACE-hard**, ovvero $\forall A \in PSPACE$, A è riducibile polinomialmente a B

Part 4. Esercizi

8.2. Prima parte.

Exercise 8.1. Sapendo che il problema del vuoto è decidibile, si dimostri che anche il problema dell'equivalenza tra DFA è decidibile.

Due insiemi X, Y sono equivalenti sse $X \subseteq Y$ e $X \supseteq Y$, e $X \subseteq Y \Leftrightarrow X \cap \neg Y = \emptyset$

Dati due DFA A, A', per le proprietà di chiusura, possiamo costruire un DFA B tale che

$$L(B) = (L(A) \cap \neg L(A')) \cup (L(A') \cap \neg L(A))$$

quindi $L(B) = \emptyset \Leftrightarrow L(A) = L(A')$ e applicando l'algoritmo per il problema del vuoto, si può decidere il problema dell'equivalenza $L(A) = L(A')$.

Exercise 8.2. Dimostrare che il linguaggio $L = \{0^n 1^m 2^{n \cdot m} \mid n, m \geq 0\}$ non è regolare.

Prendiamo in considerazione la parola $w_p = 0^p 1 0^p \in L$, $\forall p$ e applichiamo il pumping lemma:

poichè $\mid xy \mid \leq p$ e $\mid y \mid > 0$, le sottoparole xy contengono qualche 0, mentre z contiene i rimanenti 0 seguiti da 12^p, ovvero $x = 0^j$, $y = 0^k$, $z = 0^l 12^p$ dove $j, l \geq 0, k > 0, p = j + l + k$. Se poniamo $i = 0$ otteniamo $xy^0 z = 0^j (0^k)^0 0^l 12^p = 0^{j+l} 12^p \notin L$, quindi L non è regolare.

8.3. Riducibilità.

Exercise 8.3. si costruisca una riduzione da A_{TM} a $L_{TM} = \{< T_1, T_2 > \mid (T_1, T_2) \in TM \wedge \varepsilon \notin (L(T_1) - L(T_2))\}$. Perchè l'esistenza di questa funzione basta a dimostrare l'indecidibilità di L? Cosa possiamo dire del suo complemento, $\bar{L}_{TM}$?

```
R: input < M,w >
     output < T1,T2 >
     T1: qualsiasi input
           accetta
     T2: input x
           esegui M(w)
           se M accetta w, allora accetta
           se M rifiuta w, allora rifiuta
```

Correttezza:

- se $< M, w > \in A_{TM} \Rightarrow L(T_1) = \Sigma^* = L(T_2) \Rightarrow \varepsilon \notin (L(T_1) - L(T_2)) \Rightarrow < T_1, T_2 > \in L_{TM}$
- se $< M, w > \notin A_{TM} \Rightarrow L(T_1) = \emptyset,\ L(T_2) = \Sigma^* \Rightarrow \varepsilon \in (L(T_1) - L(T_2)) \Rightarrow < T_1, T_2 > \notin L_{TM}$

L'indecidibilità di L_{TM} deriva dalla riduzione, in quanto se esistesse una TM T che decide L_{TM} allora combinando la TM R che calcola la riduzione e la TM T otterremmo una TM che decide A_{TM}.

Mentre il complemento di L non è turing-riconoscibile, in quanto $\bar{A}_{TM} \leq_M \bar{L}_{TM}$: se $\bar{L}_{TM}$ fosse turing-riconoscibile allora si potrebbe provare che $\bar{A}_{TM}$ è turing-riconoscibile (contraddizione).

Exercise 8.4. Si dimostri che il linguaggio $L = \{0^n 1^m \mid n, m \geq 0\}$ non è turing-riconoscibile.

Bisogna dimostrare che $\bar{A}_{TM} \leq L$, o equivalentemente $A_{TM} \leq \bar{L}$:

```
R: input <M,w>
```

```
output <T>
T: input x
     esegui M(w)
         se M accetta w
           se x ≠
           0^n 1^m allora accetta, altrimenti rifiuta
         se M rifiuta w allora rifiuta
```

Correttezza:

- se $< M, w >\in A_{TM} \Rightarrow L(T) \neq 0^n 1^m \Rightarrow < T >\in \bar{L}$
- se $< M, w >\notin A_{TM} \Rightarrow L(T) = \emptyset$ oppure non si ferma $\Rightarrow < T >\notin \bar{L}$

Exercise 8.5. Si dimostri che il linguaggio $L = \{< T >\mid T \in TM \wedge \mid L(T) \mid = 2\}$ è indecidibile.

```
R: input <M,w>
    output: <T>
    T: input x
         se x == '00' allora accetta
         esegui M(w)
         se M accetta w
             se x == '01' allora accetta
             altrimenti rifiuta
```

Correttezza:

- se $< M, w >\in A_{TM} \Rightarrow L(T) = \{00, 01\},\ \mid L(T) \mid = 2 \Rightarrow < T >\in L$
- se $< M, w >\notin A_{TM} \Rightarrow L(T) = \{00\},\ \mid L(T) \mid \neq 2 \Rightarrow < T >\notin L$

8.4. **Calcolabilità.**

Exercise 8.6. Dimostrare che $NTIME(n^3) \subseteq SPACE(n^3)$.

Si costruisce una TM equivalente a 4 nastri:

1. contiene l'input e utilizza $O(n)$ spazio
2. è il nastro di lavoro e utilizza $O(n^3)$ spazio
3. contiene le stringhe che consentono di eseguire una alla volta le scelte non-deterministiche, utilizzando $O(n^3)$ spazio
4. contiene il numero di foglie dell'albero $2^{O(n^3)}$, che in binario occupano $O(n^3)$ spazio

Poichè passando da una TM a k-nastri ad una TM a nastro singolo la complessità spaziale non cambia, $NTIME(n^3) \subseteq SPACE(n^3)$.

Exercise 8.7. è vero che $NTIME(n^2) \subseteq NSPACE(2^{O(n^2)})$?

Si, ma possiamo dare un limite più preciso, in quanto $NTIME(n^2) \subseteq NSPACE(n^2)$.

Exercise 8.8. Se $A \leq_P B$, con A in $SPACE(n^3)$, allora anche B in $SPACE(n^3)$?

Se $A \leq_P B$ vuol dire che esiste una TM R che calcola in tempo polinomiale la riduzione, diciamo $O(n^k)$. Visto che il tempo limita superiormente lo spazio, si ha che R richiede $O(n^k)$ spazio. Mettendo in sequenza R con la TM che decide A otteniamo una TM che decide B, in $O(n^{max(k,3)})$; quindi B in $SPACE(n^3)$ se $k \leq 3$.

Exercise 8.9. Si dimostri che se $A \leq A_{TM}$ allora A è turing-riconoscibile. cosa possiamo concludere se $\bar{A}_{TM} \leq A$?

- La riduzione si può interpretare come limite superiore per la complessità di A, ovvero A è al più difficile quanto A_{TM} (turing-riconoscibile);
- La riduzione si può interpretare come limite inferiore per la complessità di A, ovvero A è almeno difficile quanto $\bar{A_{TM}}$ (non turing-riconoscibile).

Exercise 8.10. Si dimostri che ogni linguaggio PSPACE-hard è anche NP-hard.

Per definizione un linguaggio A è in PSPACE-hard se ogni linguaggio in PSPACE si riduce polinomialmente ad esso; stesso discorso per NP e NP-hard. Poichè il tempo limita lo spazio, $NP \subseteq NPSPACE$, e per il teorema di Savitch $PSPACE = NPSPACE$. Quindi ogni linguaggio in PSPACE è anche in NP, e se si riducono polinomialmente ad un linguaggio A, esso è sia PSPACE-hard che NP-hard.

Exercise 8.11. è vero che se A in $NSPACE(n^3)$ allora è in $TIME(2^{N^6})$?

Per il teorema di Savitch, $NSPACE(n^3) \subseteq SPACE(n^6)$, mentre $SPACE(n^6) \subseteq TIME(2^{O(n^6)})$.

Exercise 8.12. se $A \leq_M A_{TM}$, è possibile che $\bar{A}_{TM} \leq_M A$?

No, perchè l'ipotesi implica che A sia turing-riconoscibile, mentre per $\bar{A}_{TM} \leq_M A$ si ha che A non è turing-riconoscibile (contraddizione).

Exercise 8.13. Si dimostri che se A è turing-riconoscibile e $A \leq_M \bar{A}$, allora A è decidibile.

Se $A \leq_M \bar{A}$, allora $\bar{A} \leq_M A$. Essendo A turing-riconoscibile, per la riduzione si ha che $\bar{A}$ è turing-riconoscibile. Infine visto che sia A che il suo complemento sono turing-riconoscibili, A è decidibile.

Exercise 8.14. Se $L, L' \in NP$, allora anche $(L \cup L') \in NP$?

Si, in quanto metteno in seguenza le due macchine che decidono, rispettivamente, L e L', otteniamo una macchina che decide la loro unione impiegando lo stesso tempo, quindi $(L \cup L') \in NP$.