Appunti di Teoria descrittiva della complessità

Alessandro Achille, Giovanni Paolini

19 dicembre 2014

Indice

Capitolo 1

Definizioni

1.1 Macchine di Turing deterministiche

Una macchina di Turing deterministica, detta anche semplicemente macchina di Turing, può essere immaginata come un dispositivo libero di muoversi su un nastro infinito sul quale in ogni istante possono essere scritti finiti simboli appartenenti ad un alfabeto finito. La macchina di Turing è dotata di uno *stato*, e lavora nel modo seguente.

1. Legge il simbolo scritto sul nastro nella posizione in cui si trova attualmente.
2. Controlla in una tabella finita memorizzata al suo interno cosa deve fare quando legge quel simbolo nello stato in cui si trova.
3. Basandosi su quello che ha letto nella tabella scrive un simbolo sul nastro, si sposta a destra o a sinistra e passa in un nuovo stato.
4. Se lo stato in cui si trova ora è uno stato speciale, chiamato *terminale*, la macchina si ferma. Altrimenti il processo reinizia.

All'istante zero la macchina si trova in uno stato speciale detto *iniziale* e sul nastro può essere già scritta una stringa finita detta *input* della macchina. Diamo ora la definizione formale.

Definizione 1.1.1. Si dice macchina di Turing una tupla $M = \langle Q, \Gamma, b, \delta, q_0, g \rangle$ dove:

1. Q è un insieme finito non vuoto di *stati*;
2. Γ è un insieme finito di simboli, detto *alfabeto* della macchina;
3. $b \in \Gamma$ è un simbolo speciale, detto *blank*, che rappresenta una cella vuota;
4. $q_0 \in Q$ è lo *stato iniziale*;
5. $g : Q \to \{\exists, \texttt{accept}, \texttt{reject}\}$ indica per ogni stato se questo è *non terminale*, *terminale accettante* o *terminale rifiutante* (l'uso del simbolo "$\exists$" è per compatibilità con la Definizione 1.2.1).

6. $\delta : Q \times \Gamma \to Q \times (\Gamma \setminus \{b\}) \times \{\mathtt{L}, \mathtt{R}, \mathtt{S}\}$ è una funzione parziale detta *funzione di transizione*.

Osservazione 1.1.2. Con questa definizione abbiamo scelto di non permettere alla macchina di Turing di cancellare una cella (ovvero di scrivere il simbolo blank). Questo non modifica la potenza della macchina e permette di definire più facilmente lo spazio usato.

Definizione 1.1.3. Data una macchina di Turing $M = \langle Q, \Gamma, b, \delta, q_0, g\rangle$, chiamiamo configurazione della macchina una terna $\langle q, S, n\rangle$ dove:

1. $q \in Q$ rappresenta lo stato in cui si trova attualmente la macchina;

2. $S \in \Gamma^{\mathbb{Z}}$, con $S(k) = b$ per tutti tranne al più finiti $k \in \mathbb{Z}$, rappresenta lo stato attuale del nastro;

3. $n \in \mathbb{Z}$ rappresenta la posizione attuale della macchina di Turing sul nastro.

Nel seguito useremo la seguente notazione più compatta per descrivere la configurazione di una macchina di Turing.

Definizione 1.1.4. Data una macchina di Turing $M = \langle Q, \Gamma, b, \delta, q_0, g\rangle$, chiamiamo configurazione della macchina una funzione $S : \mathbb{Z} \to (\Gamma \sqcup \Gamma \times Q)$ tale che $S(k) = b$ per tutti tranne al più finiti $k \in \mathbb{Z}$ ed esiste unico $n \in \mathbb{Z}$ con $S(n) = (\gamma, q) \in \Gamma \times Q$. S rappresenta lo il contenuto attuale del nastro, n la posizione della macchina di Turing e q il suo stato nella configurazione S.

Definizione 1.1.5. Sia data una macchina di Turing $M = \langle Q, \Gamma, b, \delta, q_0, g\rangle$ e due sue configurazioni S e S'. Siano $n \in \mathbb{Z}$, $\gamma \in \Gamma$, $q \in Q$ gli unici tali che $S(n) = (\gamma, q)$ e supponiamo $\delta(\gamma, q) = (q', \gamma', \mathtt{R})$. Diciamo che M può fare la transizione da S a S', e lo indichiamo con $S \vdash_M S'$, se

1. $S'(n) = \gamma'$;

2. $S'(n+1) = (\sigma, q')$ per un qualche $\sigma \in \Gamma$;

3. $S(k) = S'(k)$ per ogni $k \in \mathbb{Z}$ diverso da n e $n+1$.

La definizione si adatta facilmente ai casi in cui $\delta(\gamma, q) = (q', \gamma', \mathtt{L}/\mathtt{S})$.

Osservazione 1.1.6. Per ogni configurazione S della macchina di Turing M, esiste al più una S' tale che $S \vdash_M S'$, da cui l'aggettivo *deterministica*.

Definizione 1.1.7. Una configurazione si dice accettante se lo stato corrispondente è accettante, oppure se può fare una transizione in una configurazione accettante. Viceversa diciamo che è rifiutante se lo stato corrispondente è rifiutante, oppure se può fare una transizione in una configurazione rifiutante.

Definizione 1.1.8. Data una macchina di Turing $M = \langle Q, \Gamma, b, \delta, q_0, g\rangle$, una sua configurazione S si dice *iniziale* se $S(0) = (\gamma, q_0)$ per un qualche γ in Γ. In tal caso, il contenuto del nastro si dice *input* della macchina di Turing e questo determina totalmente la configurazione iniziale.

Definizione 1.1.9. Sia S_w la configurazione iniziale con input w in Γ^*. Diciamo che una macchina di Turing M accetta l'input w, e lo indichiamo con $M(w)\downarrow_{\mathrm{Y}}$ se la configurazione S_w è accettante. Viceversa diciamo che M rifiuta l'input w, e lo indichiamo con $M(w)\downarrow_{\mathrm{N}}$, se la configurazione S_w è rifiutante.

Definizione 1.1.10. Dato un alfabeto Γ, chiamiamo *problema decisionale* un insieme di stringhe finite $\mathcal{A} \subseteq \Gamma^*$. Diciamo che il problema $\mathcal{A}$ è calcolabile dalla macchina di Turing M se, per ogni w in Γ^*, M accetta w se e solo se w è in $\mathcal{A}$ e M rifiuta w se e solo w non è in $\mathcal{A}$. Diciamo che il problema $\mathcal{A}$ è calcolabile se è calcolabile da una macchina di Turing.

Esistono varianti della definizione di macchina di Turing che, pur non cambiando l'insieme dei problemi calcolabili, sono più semplici da adoperare in alcuni casi. Una variante utile sono le *macchine di Turing a k nastri*. Queste funzionano esattamente come le macchine di Turing standard ma ad ogni passaggio leggono e scrivono contemporaneamente su tutti e k i nastri. La funzione di transizione sarà dunque del tipo $\delta : Q \times \Gamma^k \to Q \times ((\Gamma \setminus \{b\}) \times \{\mathtt{L}, \mathtt{R}, \mathtt{S}\})^k$. Una macchina di Turing a k nastri può chiaramente simulare una macchina di Turing a un solo nastro. Vale anche il viceversa, come descritto dal seguente risultato.

Fatto 1.1.11. Una macchina di Turing a un solo nastro può simulare una macchina di Turing a k nastri. In particolare tutti i problemi calcolabili da una macchina di Turing a k nastri sono calcolabili anche da una macchina di Turing a un solo nastro.

Il modello di macchina di Turing che useremo più spesso è quello di una macchina di Turing con un nastro di input e un nastro di lavoro. Questa segue le stesse regole di una macchina di Turing a due nastri, fatta eccezione del fatto che non può scrivere sul nastro di input, ma solo leggere, e che in una sua configurazione iniziale il nastro di lavoro è sempre vuoto. Se una macchina di Turing di questo tipo termina con input w, definiamo lo spazio di lavoro usato su input w come il numero di celle del nastro di lavoro non vuote nella configurazione terminale.

1.2 Macchine di Turing alternanti e non deterministiche

Una macchina di Turing alternate può essere pensata come una normale macchina di Turing che ad ogni transizione può "sdoppiarsi" e provare più strade contemporaneamente per risolvere il problema.

Definizione 1.2.1. Una macchina di Turing alternante è una tupla $M = \langle Q, \Gamma, b, \delta, q_0, g\rangle$ dove:

- Q è un insieme finito non vuoto di *stati*;
- Γ è un insieme finito di simboli, detto *alfabeto* della macchina;
- $b \in \Gamma$ è un simbolo speciale, detto *blank*, che rappresenta una cella vuota;
- $\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{\mathtt{L}, \mathtt{R}, \mathtt{S}\})$;
- $q_0 \in Q$ è lo *stato iniziale*;
- $g : Q \to \{\exists, \forall, \mathtt{accept}, \mathtt{reject}\}$ indica per ogni stato se questo è *esistenziale*, *universale*, *accettante* o *rifiutante*.

Per una macchina di Turing alternante valgono le stesse definizioni che abbiamo dato per le macchine di Turing deterministiche, fatta eccezione per la definizione di transizione e di configurazione accettante, che modifichiamo come segue.

Definizione 1.2.2. Sia data una macchina di Turing alternante $M = \langle Q, \Gamma, b, \delta, q_0, g\rangle$ e due sue configurazioni S e S'. Siano $n \in \mathbb{Z}$, $\gamma \in \Gamma$, $q \in Q$ gli unici tali che $S(n) = (\gamma, q)$ e supponiamo $(q', \gamma', \mathtt{R}) \in \delta(\gamma, q)$. Diciamo che M può fare la transizione da S a S', e lo indichiamo con $S \vdash_M S'$, se

1. $S'(n) = \gamma'$;
2. $S'(n+1) = (\sigma, q')$ per un qualche $\sigma \in \Gamma$;
3. $S(k) = S'(k)$ per ogni $k \in \mathbb{Z}$ diverso da n e $n+1$.

La definizione si adatta facilmente al caso $(q', \gamma', \mathtt{L}/\mathtt{S}) \in \delta(\gamma, q)$.

Osservazione 1.2.3. Per una generica macchina di Turing alternante non è più vero che per ogni configurazione S esiste al più una configurazione S' tale che $S \vdash_M S'$.

Definizione 1.2.4. Una configurazione di una macchina di Turing alternante si dice accettante se vale una delle seguenti:

- il suo stato è accettante;
- il suo stato è esistenziale e la configurazione può fare una transizione in una configurazione accettante;
- il suo stato è universale e ogni transizione che la configurazione può fare finisce in una configurazione accettante.

Viceversa si dice rifiutante se vale una delle seguenti:

- il suo stato è rifiutante;
- il suo stato è esistenziale e ogni transizione che la configurazione può fare finisce in una configurazione rifiutante;

- il suo stato è universale e la configurazione può fare una transizione in una configurazione rifiutante.

Possiamo infine definire cosa significa che una macchina di Turing alternante M accetti un input w allo stesso modo della Definizione 1.1.9.

Definizione 1.2.5. Chiamiamo *macchina di Turing non deterministica* una macchina di Turing alternante in cui tutti gli stati sono esistenziali, accettanti o rifiutanti.

Osservazione 1.2.6. Una macchina di Turing deterministica può essere vista come un caso particolare di una macchina di Turing non deterministica dove ogni stato può fare una transizione in al più un'altro stato.

Uno strumento molto utile nello studio delle macchine di Turing alternati è il grafo delle configurazioni G_M:

Definizione 1.2.7. Data una macchina di Turing alternante M con un unico stato accettante, il grafo delle configurazioni G_M è la struttura sul linguaggio $L = \langle E, G_\exists, G_\forall, t\rangle$ definita da:

- l'insieme V dei vertici del grafo coincide con l'insieme di tutte le possibili configurazioni di M;
- gli archi del grafo E rappresentano le possibili transizioni, ovvero vale $E(S, S') \iff S \vdash_M S'$;
- $G_\exists \subseteq V$ è un'etichetta dei nodi che indica le configurazioni con stato esistenziale;
- $G_\forall \subseteq V$ è un'etichetta dei nodi che indica le configurazioni con stato universale;
- $t \in V$ è una costante che indica l'unica configurazione avente nastro vuoto e stato accettante.

Questo grafo descrive completamente tutte le possibili computazioni di M con input qualsiasi. Spesso siamo interessati a una particolare computazione con input w fissato, per cui diamo anche la seguente definizione.

Definizione 1.2.8. Il grafo $G_{M,w}$ della computazione di M con input w è dato dalla struttura $(V, E, G_\exists, G_\forall, s, t)$ ottenuta estendendo G_M con una costante s, che denota la configurazione iniziale con input w. Se non c'è ambiguità al posto di $G_{M,w}$ scriveremo semplicemente G_w.

Dato G_w, è interessante trovare un modo per capire se una configurazione è accettante, in particolare se lo è la configurazione iniziale s, guardando solo il grafo.

Definizione 1.2.9. Dato $G_{M,w}$ il grafo di una computazione, definiamo $\mathrm{E}_{\mathrm{alt}}(x, y)$ essere la più piccola (rispetto all'inclusione) relazione binaria su $G_{M,w}$ tale che

$$\mathrm{E}_{\mathrm{alt}}(x, y) \longleftrightarrow (x = y) \vee \exists z \Big[E(x, z) \wedge \mathrm{E}_{\mathrm{alt}}(z, y) \wedge \Big(G_\forall(x) \to \forall z \left(E(x, z) \to \mathrm{E}_{\mathrm{alt}}(z, y) \right) \Big) \Big]$$

Che esista una minima tale relazione può essere verificato facilmente a mano, o lo si può dedurre dal più generale Corollario 3.1.4.

Definizione 1.2.10. Fissata una macchina di Turing M, l'insieme dei grafi G_w tali che M accetta l'input w verrà denotato da $\mathrm{Reach}_{\mathrm{alt}} := \{G_w | \, \mathrm{E}_{\mathrm{alt}}(s, t)\}$

1.3 Codifica e decodifica

Sia $L = \{R_1^{a_1}, \ldots, R_s^{a_s}, c_1, \ldots, c_l\}$ un linguaggio. Vogliamo fornire un modo standard di codificare una L-struttura finita A come stringa binaria, che denoteremo $\mathrm{bin}(A)$. Nel seguito supporremo sempre, senza perdita di generalità, che il dominio di una L-struttura finita sia un insieme del tipo $\{1, \ldots, n\} \subseteq \mathbb{N}$.

Definizione 1.3.1. Sia A una L-struttura, con $|A| = n < +\infty$. Se $R \subseteq A^k$ è una relazione, definiamo $\mathrm{bin}(R) : |A|^k \to \{0, 1\}$ come:

$$\mathrm{bin}(R)(a_1 + a_2 \cdot n + \ldots + a_k \cdot n^{k-1}) := \begin{cases} 1 \text{ se } R(a_1, \ldots, a_k) \\ 0 \text{ altrimenti} \end{cases}$$

Se a è un elemento di $A = \{1, \ldots, n\}$, definiamo $\mathrm{bin}(a)$ come la scrittura binaria del numero a, con eventualmente aggiunti degli zeri a sinistra affinché abbia lunghezza esattamente $\lceil \log_2(n) \rceil$. Definiamo infine

$$\mathrm{bin}(A) := \mathrm{bin}(R_1)\hat{} \ldots \hat{} \,\mathrm{bin}(R_s)\hat{}\,\mathrm{bin}(c_1)\hat{} \ldots \hat{} \,\mathrm{bin}(c_l).$$

Osservazione 1.3.2. Detta n la cardinalità di A, la lunghezza della codifica è

$$|\,\mathrm{bin}(A)| = n^{a_1} + \ldots + n^{a_s} + l \cdot \lceil \log_2(n) \rceil,$$

e in particolare è polinomiale in n. Inoltre, dato che il membro di destra è strettamente crescente in n, conoscendo solamente il linguaggio L e la codifica $\mathrm{bin}(A)$ possiamo ricavare n e più in generale possiamo ricostruire tutta la struttura A. Infatti avremo che vale $R_i^A(x_1, \ldots, x_{a_i})$ se e solo se $\mathrm{bin}(A)(n^{a_1} + \ldots + n^{a_{i-1}} + x_1 + x_2 \cdot n + \ldots + x_{a_i} \cdot n^{a_i - 1}) = 1$.

Useremo questa procedura principalmente per descrivere modelli all'interno di macchine di Turing. Infatti, come notato nell'Osservazione 1.3.2, una macchina di Turing può ricostruire il valore della relazione $R_i^A(x_1, \ldots, x_{a_i})$ calcolando il valore di n, spostando la testina e leggendo il valore nella posizione $n^{a_1} + \ldots + n^{a_{i-1}} + x_1 + x_2 \cdot n + \ldots + x_{a_i} \cdot n^{a_i - 1}$ del blocco di memoria corrispondente alla codifica. Nello pseudo-codice indicheremo questa procedura con $\mathrm{decode}_{R_i}(\mathrm{bin}(A), x_1, \ldots, x_{a_i})$. La corrispondente procedura per leggere il valore di una costante sarà indicata con $\mathrm{decode}_{c_i}(\mathrm{bin}(A))$.

Definizione 1.3.3. Fissato un linguaggio L ci riferiremo a un insieme $\mathcal{A}$ di L-strutture finite come a un *problema decisionale*, intendendo con questo che consideriamo il corrispondente problema $\mathrm{bin}(\mathcal{A}) = \{\mathrm{bin}(A) \mid A \in \mathcal{A}\}$ nel senso della Definizione 1.1.10.

Capitolo 2

Classi di complessità

2.1 Introduzione

Definizione 2.1.1. Sia $t : \mathbb{N} \to \mathbb{N}$ una funzione. Diciamo che una macchina di Turing alternante M lavora in tempo $t(n)$ se per ogni input w la macchina M accetta o rifiuta w e basta esaminare solo le configurazioni fino al tempo $t(|w|)$ per decidere se l'input verrà accettato o rifiutato.

Definizione 2.1.2. Sia $t : \mathbb{N} \to \mathbb{N}$ una funzione. Diciamo che una macchina di Turing alternante M lavora in spazio $s(n)$ se per ogni input w la macchina M accetta o rifiuta w e basta esaminare le configurazioni in cui la testina ha distanza minore di $s(|w|)$ dall'origine per decidere se l'input verrà accettato o rifiutato.

Osservazione 2.1.3. Se la macchina M lavora in tempo $t(n)$ o in spazio $s(n)$, allora, dato un qualunque input w, si ha che $G_{M,w} \in \text{Reach}_{\text{alt}}$ se e solo se un sottografo finito di $G_{M,w}$ appartiene a $\text{Reach}_{\text{alt}}$ (in particolare si può prendere il sottografo dato dalle configurazioni a distanza $t(n)$ dal nodo iniziale s nel primo caso e le configurazioni in cui la testina ha distanza minore di $s(n)$ dall'origine nel secondo). Sostituendo ad ogni grafo di una computazione il sottografo finito rilevante, otteniamo che $\text{Reach}_{\text{alt}}$ è un insieme di strutture finite, e quindi può essere considerato un problema decisionale nei sensi della Definizione 1.3.3.

Definizione 2.1.4. Sia $t : \mathbb{N} \to \mathbb{N}$ una funzione. Un problema decisionale $\mathcal{A}$ appartiene ad $\text{ATIME}(t(n))$ se esistono una macchina di Turing alternante M e una costante $\alpha > 0$ tale che M lavora in tempo $\alpha \cdot t(n)$ e M calcola il problema $\mathcal{A}$. Similmente diremo che il problema è in $\text{NTIME}(t(n))$ se la macchina M può essere presa non deterministica e in $\text{DTIME}(t(n))$ se può essere presa deterministica.

Definizione 2.1.5. Sia $s : \mathbb{N} \to \mathbb{N}$ una funzione. Un problema decisionale $\mathcal{A}$ appartiene ad $\text{ASPACE}(s(n))$ se eiste una macchina di Turing alternante M e una costante $\alpha > 0$ tale che M lavora in spazio $\alpha \cdot s(n)$ e M calcola il problema $\mathcal{A}$. Similmente diremo che il problema è in $\text{NSPACE}(s(n))$ se la macchina M può essere presa non deterministica e in $\text{DSPACE}(s(n))$ se può essere presa deterministica.

Risulta molto interessante il concetto di computabilità in tempo o spazio polinomiale. Per questo definiamo le classi di complessità DPTIME e DPSPACE nel seguente modo:

$$\mathrm{DPTIME} = \bigcup_{k\in\mathbb{N}} \mathrm{DTIME}(n^k), \quad \mathrm{DPSPACE} = \bigcup_{k\in\mathbb{N}} \mathrm{DSPACE}(n^k).$$

Analogamente si definiscono le classi NPTIME, APTIME, NPSPACE ed APSPACE. Particolarmente studiate sono le classi DPTIME e NPTIME, le quali vengono più comunemente denominate P ed NP, rispettivamente. Non è noto se P sia incluso strettamente in NP o se valga l'uguaglianza.

La classe NP ha una caratterizzazione equivalente: un problema decisionale $\mathcal{A}$ appartiene ad NP se e solo se esistono dei certificati che consentono di verificare in tempo polinomiale che una struttura A appartenga ad $\mathcal{A}$. Più formalmente, vale la seguente proposizione.

Proposizione 2.1.6. Sia $\mathcal{A}$ un problema decisionale su un linguaggio L. Si ha che $\mathcal{A} \in \mathrm{NP}$ se e solo se esistono una macchina di Turing deterministica M e un polinomio $p(n)$ tali che:

- per ogni stringa binaria c di lunghezza $\leq p(|\operatorname{bin}(A)|)$ (detta "certificato"), la macchina M su input $\operatorname{bin}(A)\hat{}c$ termina in tempo $p(|\operatorname{bin}(A)|)$;
- per ogni L-struttura A, si ha che $A \in \mathcal{A}$ se e solo se esiste una stringa binaria c di lunghezza $\leq p(|\operatorname{bin}(A)|)$ tale che $M(\operatorname{bin}(A)\hat{}c)\downarrow_{\mathrm{Y}}$.

Dimostrazione. Supponiamo che $\mathcal{A}$ appartenga ad NP. Allora esiste una macchina di Turing non deterministica N che determina in tempo polinomiale se una struttura A appartiene a S. Possiamo supporre senza perdita di generalità che tutte le scelte non deterministiche di N siano binarie. Definiamo una macchina deterministica M che, su input $\operatorname{bin}(A)\hat{}c$, simula l'esecuzione di N su input $\operatorname{bin}(A)$ sostituendo (per ogni i) la i-esima scelta non deterministica con la scelta deterministica descritta dall'i-esimo bit della stringa c. Nel caso in cui la stringa c non sia sufficientemente lunga, M termina in uno stato rifiutante. Notiamo che A appartiene a S se e solo se $N(\operatorname{bin}(A))\downarrow_{\mathrm{Y}}$, ovvero se e solo se esiste una sequenza di scelte (che possiamo codificare con la stringa binaria c) che portano in una configurazione finale accettante di N. In altre parole, A appartiene a S se e solo se $M(\operatorname{bin}(A)\hat{}c)\downarrow_{\mathrm{Y}}$. Il tempo di calcolo di M è polinomiale in $\operatorname{bin}(A)$, perché lo è quello di N. Definiamo allora $p(n)$ come un qualsiasi polinomio tale che M termini in tempo $p(|\operatorname{bin}(A)|)$.

Supponiamo ora che esista una macchina di Turing determinista M con le caratteristiche enunciate. Sia N una macchina non deterministica che, su input $\operatorname{bin}(A)$, simula M scegliendo non deterministicamente i simboli di c man mano che M li utilizza. Segue dalle definizioni che $N(\operatorname{bin}(A))\downarrow_{\mathrm{Y}}$ se e solo se esiste una stringa binaria c tale che $M(\operatorname{bin}(A)\hat{}c)\downarrow_{\mathrm{Y}}$. Il tempo di calcolo di N è polinomiale in $\operatorname{bin}(A)$, perché lo è quello di M (indipendentemente dalla lunghezza di c). □

Il seguente corollario è una semplice riscrittura della proposizione e ci sarà utile in seguito.

Corollario 2.1.7. Sia $\mathcal{A} \in$ NP un problema decisionale su un linguaggio L. Allora esiste un $k \in \mathbb{N}$ e una macchina di Turing deterministica M tale che, per ogni L-struttura finita A, si ha che A appartiene ad $\mathcal{A}$ se e solo se esiste una relazione k-aria $R \subseteq A^k$ per cui $M(\text{bin}(A)\hat{}\,\text{bin}(R))\downarrow_{\text{Y}}$.

Dimostrazione. Basta notare che, data una struttura $|A|$, ogni stringa binaria di lunghezza minore di $|A|^k$ può essere ottenuta come $\text{bin}(R)$, con R una relazione k-aria su A. Si conclude utilizzando la proposizione precedente. □

Per ogni classe di complessità $\mathcal{C}$ si può introdurre la classe dei problemi complementari, costituita per l'appunto dagli elementi $\mathcal{A}^c$ al variare di $\mathcal{A} \in \mathcal{C}$. Solitamente una tale classe viene indicata aggiungendo il prefisso "co" alla denominazione della classe $\mathcal{C}$. Per esempio, co-NP è la classe dei problemi decisionali i cui complementari appartengono a NP. Si può dare la seguente caratterizzazione di co-NP, analoga a quella che abbiamo appena dato di NP.

Proposizione 2.1.8. Sia $\mathcal{A}$ un problema decisionale su un linguaggio L. Si ha che $\mathcal{A} \in$ co-NP se e solo se esiste una macchina di Turing deterministica M tale che:

- la macchina M su input $\text{bin}(A)\hat{}\,c$ termina in tempo polinomiale rispetto a $|\,\text{bin}(A)|$, per ogni stringa binaria c;
- per ogni L-struttura A, si ha che $A \notin \mathcal{A}$ se e solo se esiste una stringa binaria c tale che $M(\text{bin}(A)\hat{}\,c)\downarrow_{\text{Y}}$.

Dimostrazione. Segue immediatamente applicando la Proposizione 2.1.6 ad $\mathcal{A}^c$. □

Osservazione 2.1.9. P = co-P. Di conseguenza, se fosse vero che P = NP, allora NP sarebbe anche uguale a co-NP. In altre parole, NP $\neq$ co-NP implica P $\neq$ NP.

2.2 Inclusioni tra le classi di complessità

Nel resto del capitolo dimostreremo diverse inclusioni interessanti tra le varie classi di complessità. Tali inclusioni possono essere riassunte nel seguente schema:

$$\begin{aligned} \text{DTIME}(t(n)) &\subseteq \text{NTIME}(t(n)) \subseteq \text{ATIME}(t(n)) \subseteq \\ \subseteq \text{DSPACE}(t(n)) &\subseteq \text{NSPACE}(t(n)) \subseteq \text{ASPACE}(t(n)) = \\ = \text{DTIME}(O(1)^{t(n)}) &\subseteq \quad \dots \end{aligned}$$

dove con la notazione $\text{DTIME}(O(1)^{t(n)})$ indichiamo la classe

$$\bigcup_{c>0} \text{DTIME}(c^{t(n)}).$$

Oltre a quelle precedentemente rappresentate, vale anche l'inclusione $\text{NSPACE}(t(n)) \subseteq \text{ATIME}(t(n)^2)$.

Le inclusioni presenti sulle righe dello schema precedente sono tutte ovvie: ciò che può essere fatto con una macchina deterministica può essere fatto anche con una macchina non deterministica, e ciò che può essere fatto con una macchina non deterministica può essere fatto anche con una macchina alternante. Nel resto di questo capitolo dimostreremo le inclusioni e le uguaglianze rimanenti, e ne trarremo poi alcune conseguenze.

2.3 Relazione tra ATIME e DSPACE

Teorema 2.3.1. $\mathrm{ATIME}(t(n)) \subseteq \mathrm{DSPACE}(t(n))$.

Dimostrazione. Data una macchina di Turing alternante M che lavora in $\mathrm{ATIME}(t(n))$, vogliamo trovare una macchina di Turing deterministica che lavori in $\mathrm{DSPACE}(t(n))$ e accetti gli stessi input w di M. Supponiamo per semplicità di notazione che ogni stato non deterministico di M possa fare una transizione in al più due stati. Sia G_w il grafo della computazione della macchina M su input w. Esibiamo una macchina di Turing N che lavora in $\mathrm{DSPACE}(t(n))$ la quale, preso in input un nodo del grafo, decide se quel nodo è accettante o meno. Ciò consente di concludere, poiché la macchina M accetta l'input w se e solo se il nodo iniziale di G_w è accettante.

La macchina N deve verificare se il nodo iniziale di G_w sia accettante o meno. L'idea per fare questo è di visitare ricorsivamente il grafo, il quale risulta tuttavia troppo grande per essere salvato interamente in memoria. La soluzione che adottiamo è di memorizzare solo il nodo iniziale c_0, il nodo c che stiamo attualmente visitando e il percorso che abbiamo fatto per raggiungerlo a partire dal nodo iniziale, ovvero una stringa binaria $P = \langle b_1, b_2, \ldots, b_n \rangle$ che ci dica per ogni nodo alternante che abbiamo incontrato se ci si è spostati sul primo o sul secondo figlio. Teniamo inoltre in memoria un simbolo $D \in \{\mathtt{S}, \mathtt{N}, \mathtt{?}\}$ che ci dice se abbiamo già verificato che quel nodo è accettante, non accettante oppure se ancora non lo sappiamo.

Si noti che, in particolare, non viene tenuto in memoria il percorso da c_0 a c. Di conseguenza, ogniqualvolta si debba passare da un nodo c al suo padre, risulta necessario percorrere nuovamente il cammino da c_0 a c (che può essere ricostruito grazie alla stringa binaria P).

Vediamo ora i dettagli dell'algoritmo di visita ricorsiva del grafo.

- Se D è uguale a `?`, non abbiamo ancora stabilito se il nodo c sia o meno accettante. Pertanto, se c ha dei figli modifichiamo lo stato nel modo seguente:

  $$P := \langle b_1, b_2, \ldots, b_n, 0 \rangle, \quad D := \mathtt{?}, \quad c := \text{figlio sinistro di } c.$$

  In caso contrario, c è una foglia. Di conseguenza corrisponde a uno stato in cui la macchina M ha terminato la computazione, per cui è possibile determinare se si tratti di un nodo accettante o non accettante. Modifichiamo allora lo stato in

  $$P := \langle b_1, b_2, \ldots, b_n \rangle, \quad D := \mathtt{S}/\mathtt{N}, \quad c := c,$$

  dove D assume il valore `S` se lo stato c è accettante, mentre assume il valore `N` se lo stato c è non accettante.

- Se D è in $\{\mathtt{S}, \mathtt{N}\}$ e $P = \langle\rangle$, ci troviamo nel nodo iniziale c_0 e sappiamo se tale nodo è accettante. Quindi la computazione di N può terminare, restituendo D come risultato.

- Se D è in $\{\mathtt{S}, \mathtt{N}\}$ e $P = \langle b_1, b_2, \ldots, b_n \rangle$ con $n \geq 1$, abbiamo determinato se il nodo corrente c è accettante e dobbiamo continuare la visita dell'albero di computazione.

1. Se c corrisponde ad uno stato esistenziale e $D = \mathtt{S}$, allora il padre di c non può che essere accettante, per cui passiamo nello stato

$$P := \langle b_1, b_2, \ldots, b_{n-1} \rangle, \quad D := \mathtt{S}, \quad c := \text{padre di } c.$$

2. Analogamente se c corrisponde ad uno stato universale e $D = \mathtt{N}$, il padre di c è sicuramente non accettante, dunque passiamo nello stato

$$P := \langle b_1, b_2, \ldots, b_{n-1} \rangle, \quad D := \mathtt{N}, \quad c := \text{padre di } c.$$

3. Se $b_n = 0$ (ovvero c è un figlio sinistro) e non ci troviamo in nessuno dei primi due casi, il padre di c è accettante se e solo se il figlio destro del padre di c è accettante. Passiamo pertanto nello stato

$$P := \langle b_1, b_2, \ldots, b_{n-1}, 1 \rangle, \quad D := \mathtt{?}, \quad c := \text{figlio destro del padre di } c.$$

4. Se $b_n = 1$ (ovvero c è un figlio destro) il padre di c è accettante se e solo se c è accettante. Di conseguenza passiamo nello stato

$$P := \langle b_1, b_2, \ldots, b_{n-1} \rangle, \quad D := D, \quad c := \text{padre di } c.$$

Esaminiamo infine la memoria utilizzata dalla macchina di Turing N. La macchina M lavora in tempo $O(t(n))$, per cui la codifica di un singolo nodo di G_w occupa spazio $O(t(n))$. La lunghezza della stringa binaria P è data al massimo dalla profondità del grafo G_w, che è sempre $O(t(n))$. Infine, memorizzare D richiede spazio $O(1)$. Possiamo quindi concludere che la macchina N lavora effettivamente in DSPACE($t(n)$).

□

2.4 Relazione tra ASPACE e DTIME

Teorema 2.4.1. ASPACE($t(n)$) = DTIME($O(1)^{t(n)}$).

Dimostrazione. L'inclusione ASPACE($t(n)$) $\subseteq$ DTIME($O(1)^{t(n)}$) può essere dimostrata in modo analogo al Teorema 2.3.1. Più nel dettaglio, data una macchina di Turing M che lavora in ASPACE($t(n)$), si costruisce una macchina di Turing N che visita il grafo G_w della computazione di M su input w e determina se w sia o meno accettante per M. La dimensione del grafo G_w è $O(1)^{t(n)}$, perché il numero di configurazioni accessibili da una singola configurazione è limitato da una costante che dipende solo da M, e non da w. Effettuiamo la visita del grafo in modo simile a quanto descritto nella dimostrazione del Teorema 2.3.1, tenendo però scritta in un apposito nastro la descrizione di tutti i nodi del percorso dalla radice c_0 al nodo corrente c. Passare da un nodo a uno dei figli richiede di scrivere sul nastro la descrizione del nodo figlio, mentre tornare da un nodo al suo padre richiede di cancellare la descrizione del nodo stesso dal nastro. Ogni procedura di scrittura o cancellazione di un nodo richiede tempo $O(t(n))$. Pertanto la macchina N lavora in tempo

$$O(1)^{t(n)} \cdot O(t(n)) = O(1)^{t(n)+O(\log t(n))} = O(1)^{t(n)}.$$

Dimostriamo ora l'inclusione DTIME$(O(1)^{t(n)}) \subseteq$ ASPACE$(t(n))$. Sia M una macchina di Turing che lavora in DTIME$(c^{t(n)})$ per qualche $c > 0$; vogliamo definire una macchina di Turing N che lavori in ASPACE$(t(n))$, accettando gli stessi input che accetta M. Possiamo assumere senza perdita di generalità che M abbia un unico nastro. L'idea è di definire una subroutine ricorsiva di N, che chiamiamo $C(t, p, a)$, al termine della quale N accetta se (nell'esecuzione di M su input w) al tempo t in posizione p vi è il simbolo a, altrimenti rifiuta. Seguiamo la convenzione che a possa appartenere a Σ (nel caso in cui non vi sia la testina) oppure a $\Sigma \times Q$ (se vi è la testina, nel qual caso l'elemento di Q indica lo stato attuale della macchina). Vediamo ora i dettagli sull'implementazione di questa subroutine.

L'osservazione principale è che l'esito della chiamata a $C(t, p, a)$ è univocamente determinato dall'esito delle chiamate a $C(t-1, p', a')$ al variare di $p' \in \{p-1,\ p,\ p+1\}$ ed $a' \in \Sigma \cup \Sigma \times Q$. Introduciamo uno stato $q_C^{\exists}$ in cui la macchina N deve iniziare ad eseguire la subroutine C. Supponiamo che N sia nello stato $q_C^{\exists}$, e che abbia in memoria (su un apposito nastro di lavoro dedicato all'inizializzazione della subroutine C) la terna (t, p, a). Se $t = 0$, N accetta se e solo se nella posizione p dell'input w (tenendo conto dell'eventuale presenza della testina) vi è il simbolo a. Se invece $t > 0$, N esegue le seguenti operazioni.

- Sceglie (non deterministicamente) tre elementi a_{-1}, a_0, a_1 in $\Sigma \cup \Sigma \times Q$.
- Verifica se la macchina M con i simboli a_{-1}, a_0, a_1 in posizioni $p-1$, p, $p+1$ (rispettivamente) avrebbe al passo successivo il simbolo a in posizione p. In caso affermativo continua, altrimenti rifiuta.
- Si porta in uno stato universale e sceglie (non deterministicamente) $i \in \{-1, 0, 1\}$.
- Scrive sul nastro dedicato all'inizializzazione di C la terna $(t-1, p+i, a_i)$ e si riporta nello stato $q_C^{\exists}$, in modo da chiamare ricorsivamente la subroutine C.

L'implementazione della subroutine C è sufficiente per determinare se un qualsiasi input w venga accettato da M: basta controllare che al tempo $c^{t(n)}$ la macchina M si trovi nello stato accettante. Osserviamo infine che la macchina N che abbiamo definito lavora effettivamente in ASPACE$(t(n))$, poiché oltre all'input w deve memorizzare solamente le seguenti variabili: un intero t (compreso tra 0 e $c^{t(n)}$, che quindi occupa $O(t(n))$ bit), un intero p (di modulo $\leq c^{t(n)}$), dei simboli a, a_{-1}, a_0, a_1 (che occupano $O(1)$ bit), un intero i compreso tra -1 e 1 (che occupa 2 bit). □

2.5 Relazione tra NSPACE e ATIME

Teorema 2.5.1. NSPACE$(t(n)) \subseteq$ ATIME$(t(n)^2)$.

Dimostrazione. Sia M una macchina di Turing che lavora in NSPACE$(t(n))$. Vogliamo definire una macchina N che lavori in ATIME$(t(n)^2)$ e che accetti gli stessi input che accetta M. Sia w un input, e sia G_w il grafo della computazione di M su input w. L'idea è di fare in modo che N implementi una subroutine ricorsiva $P(d, x, y)$, al termine

della quale N accetta se in G_w esiste un cammino lungo al più 2^d dal nodo x al nodo y, altrimenti rifiuta.

Entriamo ora maggiormente nei dettagli. Introduciamo uno stato esistenziale $q_P^{\exists}$, in cui N inizia l'esecuzione della subroutine P leggendo le variabili d, x, y da un apposito nastro. Se $d = 0$, la macchina N accetta nel caso in cui $x = y$ oppure $x \vdash_M y$, altrimenti rifiuta. Se invece $d > 0$, N esegue le seguenti operazioni.

- Si porta in uno stato esistenziale e sceglie (non deterministicamente) un nodo z di G_w. Tale z rappresenta un possibile nodo intermedio in un eventuale percorso da x a y.
- Si porta in uno stato universale e sceglie (non deterministicamente) un bit b. Tale bit determina se verificare l'esistenza di un percorso da x a z oppure se verificare l'esistenza di un percorso da z a y.
- Se $b = 0$, si porta nello stato $q_P^{\exists}$ inizializzando P con le variabili (d, x, z). Se invece $b = 1$, si porta nel medesimo stato ma inizializzando P con le variabili (d, z, y). Da questo momento in avanti, l'esecuzione continua ricorsivamente.

La procedura descritta ha l'effetto desiderato, perché l'esistenza di un percorso da x a y in $\leq 2^d$ passi è equivalente all'esistenza di un nodo intermedio z per cui si possa andare da x a z in $\leq 2^{d-1}$ passi e da z a y in $\leq 2^{d-1}$ passi.

La subroutine P è sufficiente per decidere se un input w venga accettato da M, poiché basta che N chiami P con i seguenti parametri:

- $x =$ configurazione iniziale di M su input w;
- $y =$ configurazione finale accettante di M;
- $d =$ logaritmo della profondità del grafo G_w.

Si noti che la profondità del grafo G_w è al massimo $c^{t(n)}$ per qualche costante $c > 0$ (che dipende da M ma non da w), quindi $d = O(t(n))$.

Verifichiamo infine che N lavori effettivamante in $\mathrm{ATIME}(t(n)^2)$. Per quanto appena osservato, il numero di chiamate alla subroutine P è $O(t(n))$. Ciascuna delle chiamate (compresa quella finale con $d = 0$) impiega tempo $O(t(n))$, perché la lunghezza della codifica di un qualsiasi nodo di G_w è $O(t(n))$. Quindi il tempo totale di esecuzione è effettivamente $O(t(n)^2)$. □

2.6 Alcune conseguenze

Esaminiamo infine alcuni risultati che si dimostrano a partire dalle inclusioni descritte nelle sezioni precedenti.

Corollario 2.6.1. $\mathrm{NPSPACE} \subseteq \mathrm{APTIME}$.

Dimostrazione. È sufficiente applicare il Teorema 2.5.1, osservando che il quadrato di un polinomio è ancora un polinomio. □

Corollario 2.6.2. NPSPACE = DPSPACE.

Dimostrazione. Abbiamo che NSPACE($t(n)$) $\subseteq$ ATIME($t(n)^2$) $\subseteq$ DSPACE($t(n)^2$). Dal momento che il quadrato di un polinomio è ancora un polinomio, ne deduciamo che NPSPACE = DPSPACE. □

Quindi il problema P vs NP è molto più semplice da risolvere rimpiazzando il tempo con lo spazio.

Capitolo 3

Caratterizzazione di P

3.1 Sintassi e semantica di $\mathrm{FO(LFP)}_{\leq,\cdot,+}$

Definizione 3.1.1. Sia $\phi(S^2, x, y)$ una formula. Diremo che S compare in *posizione positiva* in ϕ se tutte le volte in cui compare è preceduta da un numero pari di negazioni. Più precisamente, per induzione sulla complessità della formula:

- S è in posizione positiva in tutte le formule atomiche (che la contengano o no);
- se S compare in posizione positiva in ϕ e θ, allora compare in posizione positiva in $\phi \wedge \theta$, $\phi \vee \theta$, $\exists x \phi(x)$, $\forall x \phi(x)$;
- se S compare in posizione positiva in ϕ allora compare in posizione negativa in $\neg\psi$, $\psi \to \phi$ e in $\neg\psi \wedge \phi$;
- se S compare in posizione positiva in ϕ allora compare in posizione negativa in $\neg\phi$ e viceversa se S compare in posizione negativa in ϕ allora compare in posizione positiva in $\neg\phi$.

Definizione 3.1.2. Estendiamo la sintassi di FO introducendo un nuovo quantificatore, che indichiamo con LFP. Data una formula del primo ordine $\phi(S^2, x, y)$ in cui S compare in posizione positiva, l'espressione $\mathrm{LFP}_{S,x,y}\, \phi(S^2, x, y)$ avrà il tipo di una relazione binaria con variabili libere $\mathrm{FV}(\phi) \setminus \{S, x, y\}$. Chiameremo la sintassi così ottenuta $\mathrm{FO(LFP)}_{\leq,\cdot,+}$.

Semanticamente vogliamo interpretare $\mathrm{LFP}_{S,x,y}\, \phi(S, x, y)$ come la più piccola (rispetto all'inclusione) relazione binaria R tale che $R(x, y) \leftrightarrow \phi(R, x, y)$, ma dobbiamo prima verificare che una tale relazione esista.

Teorema 3.1.3 (Tarski-Knaster)**.** Sia A un insieme finito e sia $f : \mathcal{P}(A^k) \to \mathcal{P}(A^k)$ una funzione monotona rispetto all'inclusione (i.e. se $S \subseteq S'$ allora $fS \subseteq fS'$). Allora esiste un minimo punto fisso di f, che indicheremo con $\mathrm{LFP}\, f$, e per la precisione questo è dato da $f^{|A|^k}(\varnothing)$.

Dimostrazione. Dato che f è monotona e A è finito, deve esistere un $n \in \mathbb{N}$ tale che $f^n(\varnothing) = f^{n+1}(\varnothing)$, e inoltre tale n può essere preso minore di $|A|^k$. Indichiamo $f^n(\varnothing)$

con $\bar{S}$. Per definizione $\bar{S}$ è un punto fisso di f. Se S' è un altro punto fisso di f, dato che $\varnothing \subseteq S'$ e che f è monotona, avremo $\bar{S} = f^k(\varnothing) \subseteq f^k(S') = S'$. Quindi $\bar{S}$ è il più piccolo punto fisso. □

Corollario 3.1.4. Sia $\phi(S, x, y)$ una L-formula del primo ordine in cui S compare in posizione positiva, e sia A una L-struttura. Indichiamo con $f_\phi : \mathcal{P}(A \times A) \to \mathcal{P}(A \times A)$ la funzione

$$f_{\phi,A}(R) := \{(a, b) \in A \times A \mid [\![\phi(S, x, y)]\!]^{A,R,a,b}\}.$$

Questa funzione è monotona sulle relazione ordinate per inclusione. Il suo più piccolo punto fisso LFP $f_{\phi,A}$ è la più piccola relazione $R \subseteq A \times A$ tale che $R(a, b) \leftrightarrow [\![\phi(S, x, y)]\!]^{A,R,a,b}$.

Definizione 3.1.5. Usando le notazioni del Corollario 3.1.4, estendiamo la funzione semantica di FO ad FO(LFP)$_{\leq,\cdot,+}$ ponendo

$$[\![\text{LFP}_{S,x,y}\, \phi(S, x, y)]\!]^M = \text{LFP}\, f_{\phi,M}.$$

3.2 FO(LFP)$_{\leq,\cdot,+}$ equivale a P

Lemma 3.2.1. FO $\subseteq$ P. In altre parole, per ogni L-formula chiusa ϕ in FO esiste una macchina di Turing T_ϕ in P tale che, data una L-struttura A, si ha $A \models \phi$ se e solo se $T_\phi(\text{bin}(A))\downarrow_{\text{Y}}$. Inoltre le T_ϕ possono essere costruite in modo che siano logaritmiche nello spazio.

Dimostrazione. Procediamo per induzione strutturale sulla complessità della formula ϕ.

Sia ϕ atomica nel linguaggio $L = \{R^{a_1}, \ldots, R^{a_k}, c_1, \ldots, c_l\}$; allora ϕ deve essere nella forma $c_1 = c_2$ oppure nella forma $R_s(c_1, \ldots, c_k)$, dove i c_i sono simboli di costante. La macchina di Turing T_ϕ come prima cosa leggerà dal nastro di input i valori m_i corrispondenti alle costanti c_i. Se ϕ era nella forma $c_1 = c_2$ allora basta controllare l'uguaglianza di m_1 ed m_2. Se invece ϕ era nella forma $R_s(c_1, \ldots, c_k)$, leggiamo il bit in posizione $m_1 + m_2 \cdot n + \ldots + m_k \cdot n^{k-1}$ del blocco relativo alla relazione R_k e accettiamo nel caso abbia valore 1.

Se ϕ è combinazione booleana di formule ϕ_i, la macchina di Turing corrispondente si ottiene facilmente mettendo insieme le macchine di Turing delle singole ϕ_i.

Sia infine ϕ nella forma $\exists x\theta(x)$. Detto L' il linguaggio ottenuto aggiungendo a L un simbolo di costante c, la L'-formula $\theta(c)$ ha complessità minore di quella di ϕ e dunque, per ipotesi induttiva, esiste una macchina di Turing $T_{\theta(c)}$ che gli corrisponde. Notiamo che, dato $x \in \{0, \ldots, n-1\}$, possiamo trasformare A in una L' struttura (A, x) interpretando la costante c con x. Costruiamo allora la macchina di Turing T_ϕ in modo che $T_\phi(\text{bin}(A))$ accetti se e solo se $T_{\theta(c)}(\text{bin}(A)\hat{}\,\text{bin}(x))$ accetta per qualche x. Più concretamente, per ogni $x \in \{0, \ldots, n-1\}$ scriviamo su un nastro a parte $\text{bin}(A)\hat{}\,\text{bin}(x)$ e lanciamo la macchina $T_\theta(c)$ usando questo come nastro di input. Se $T_\theta(c)$ accetta, fermiamo la computazione e accettiamo. □

Proposizione 3.2.2. $\mathrm{FO(LFP)}_{\leq,\cdot,+} \subseteq \mathrm{P}$. In altre parole, per ogni L-formula chiusa ϕ in $\mathrm{FO(LFP)}_{\leq,\cdot,+}$ esiste una macchina di Turing T_ϕ in P tale che, data una L-struttura A, si ha $A \models \phi$ se e solo se $T_\phi(\mathrm{bin}(A))\downarrow_{\mathrm{Y}}$.

Dimostrazione. Rispetto al lemma precedente dobbiamo solo considerare in più il caso $\phi :\equiv \mathrm{LFP}_{S,x,y}\,\theta(S,x,y)(a,b)$, con a,b costanti. Notiamo che possiamo vedere $\theta(S,x,y)$ come una formula chiusa nel linguaggio $L' = L \cup \{S^2, x, y\}$, dunque, sfruttando l'ipotesi induttiva, ha senso considerare la macchina di Turing $T_{\theta(S,x,y)}$ ad essa corrispondente. Inoltre, usando le notazioni del Corollario 3.1.4, si ha $T_{\theta(S,x,y)}(\mathrm{bin}(A,R,a,b))\downarrow_{\mathrm{Y}}$ se e solo se $(a,b) \in f_{\theta,A}(R)$. Questa osservazione, insieme alla dimostrazione del Teorema 3.1.3, suggeriscono il seguente algoritmo per T_ϕ e ne dimostrano la correttezza.

function $T_\phi(bin)$
  $R \leftarrow 0$
  for $i < max^2$ **do**
    for $h,k = 0..max$ **do**
      if $T_{\theta(S,x,y)}(\mathrm{bin}(A)^\frown \mathrm{bin}(h)^\frown \mathrm{bin}(k))\downarrow_{\mathrm{Y}}$ **then**
        $R[a * max + b] \leftarrow 1$
      end if
    end for
  end for
  $a \leftarrow \mathrm{decode}_a(bin)$
  $b \leftarrow \mathrm{decode}_b(bin)$
  return $R[a * max + b]$
end function

□

Dimostriamo ora l'inclusione inversa, ovvero $\mathrm{P} \subseteq \mathrm{FO(LFP)}_{\leq,\cdot,+}$. La dimostrazione si basa sui seguenti claim.

Claim 1 $\mathrm{P} = \mathrm{ASPACE}(\log(n))$.

Claim 2 $\mathrm{Reach_{alt}} \in \mathrm{FO(LFP)}_{\leq,\cdot,+}$.

Claim 3 Se $\mathcal{A}$ è un problema in $\mathrm{ASPACE}(\log(n))$, allora $\mathcal{A} \leq_{\mathrm{FO}} \mathrm{Reach_{alt}}$.

Dimostrati questi tre claim, per concludere, basta notare che un problema che si riduce FO a un problema in $\mathrm{FO(LFP)}_{\leq,\cdot,+}$ è a sua volta in $\mathrm{FO(LFP)}_{\leq,\cdot,+}$. Il Claim 1 segue dal Teorema 2.4.1 prendendo $t(n) = \log(n)$. Dimostriamo dunque gli altri due.

Proposizione 3.2.3. $\mathrm{Reach_{alt}} \in \mathrm{FO(LFP)}_{\leq,\cdot,+}$;

Dimostrazione. La relazione $\mathrm{E_{alt}}(x,y)$ è più piccolo punto fisso della formula

$$\phi(S,x,y) :\equiv (x=y) \vee \exists z\Big[E(x,z) \wedge S(z,y) \wedge \Big(G_\forall(x) \rightarrow \forall z\,(E(x,z) \rightarrow S(z,y))\Big)\Big]$$

(cfr. Definizione 1.2.9). Dato che S compare in posizione positiva, possiamo scrivere $\mathrm{E_{alt}}(a,b) :\equiv (\mathrm{LFP}_{S,x,y}\,\phi(S,x,y))(a,b)$. In particolare $\mathrm{E_{alt}}$ è definibile in $\mathrm{FO(LFP)}_{\leq,\cdot,+}$, da cui segue facilmente la tesi. □

Proposizione 3.2.4. Se $\mathcal{A}$ è un problema in ASPACE($\log(n)$), allora $\mathcal{A} \leq_{\text{FO}} \text{Reach}_{\text{alt}}$. In particolare, $\text{Reach}_{\text{alt}}$ è un problema P-completo.

Dimostrazione. Sia $\sigma = \{R_1^{a_1}, \ldots, R_r^{a_r}, c_1, \ldots, c_s\}$ un linguaggio e M una macchina di Turing in ASPACE($c\log(n)$) che accetta come input la codifica binaria di una σ-struttura. Dobbiamo mostrare che esiste un modo per associare ad ogni σ-struttura A di cardinalità n un grafo alternante $I(A) = (V^A, G_\exists^A, G_\forall^a, E^a, s^A, t^A)$, definibile al primo ordine in A, tale che:

$$M(\text{bin}(A)) \iff I(A) \in \text{Reach}_{\text{alt}}\,.$$

L'idea è di associare alla struttura A il grafo della computazione di M su input $\text{bin}(A)$; in questo modo la nostra richiesta sarà automaticamente soddisfatta. A tal fine identifichiamo il dominio di A con l'insieme $\{1, 2, \ldots, n\}$ e scegliamo come insieme di vertici V^A l'insieme A^{4+a+c}, con $a = \max(a_1, \ldots, a_r)$. In questo modo un vertice ID del grafo può essere identificato con una tupla di numeri naturali minori di n:

$$\text{ID} = \langle p_1, \ldots, p_4, r_1, \ldots, r_a, w_1, \ldots, w_c\rangle$$

Ciascuna di queste tuple rappresenterà una possibile configurazione della macchina M. Per la precesione:

- $p_1 \in \{1, \ldots, r\}$ rappresenta la relazione che la macchina sta leggendo sul nastro di input, come verrà spiegato più tardi;
- $p_2 \in Q_M$ rappresenta lo stato in cui si trova la macchina di Turing;
- $p_3 \in \{0, \ldots, c\log(n)\}$ rappresenta la posizione della testina sul nastro di lavoro;
- $p_4 \in \{\forall, \exists\}$ indica se lo stato attuale è esistenziale o universale;
- $\langle w_1, \ldots, w_c\rangle \in n^c = 2^{c\log(n)}$ rappresenta una possibile configurazione del nastro di lavoro (stiamo usando il fatto che M è in ASPACE($c\log(n)$);
- $\langle r_1, \ldots, r_c\rangle \in n^c = r_1 + r_2 n + \ldots + r_a n^{a-1} < n^a$ rappresenta la posizione della testina sul nastro di input.

Ora che abbiamo definito il dominio V^A del grafo dobbiamo definire le relazioni E^A, $G_\exists^A$, $G_\forall^A$, s e t. Di queste, $G_\exists$ e $G_\forall$ sono immediate (basta guardare il valore di p_4), per quanto riguarda s, può essere definita come

$$s(\text{ID}) :\equiv (p_1 = 1 \wedge p_2 = q_0 \wedge p_3 = 0 \wedge \vec{r} = 0 \wedge \vec{w} = 0),$$

e anche t può essere definita in maniera simile. Concentriamoci dunque su E^A: dobbiamo trovare una relazione $E(\vec{x}, \vec{y})$ tale che valga $E(\text{ID}, \text{ID}')$ se e solo se nel grafo delle computazioni di M con input $\text{bin}(A)$ c'è un arco tra la configurazione corrisponente a ID quella corrispondente a ID′. Come prima cosa sia R una regola della macchina di Turing M nella forma

$$\langle q, \gamma_w, \gamma_i\rangle \vdash \langle q', \gamma_o, \delta_w, \delta_i\rangle,$$

dove q è lo stato attuale, γ_w il simbolo letto sul nastro di lavoro, γ_i il simbolo letto sul nastro di input, q' lo stato in cui si fa la transizione, γ_o il simbolo scritto sul nastro di lavoro, $\delta_w \in \{-1, 0, 1\}$ lo spostamento sul nastro di lavoro e $\delta_i \in \{-1, 0, 1\}$ lo spostamento sul nastro di input. Associamo a R la relazione

$$\begin{aligned} C_R(\text{ID}, \text{ID}') :\equiv \bigwedge_{1 \le i \le r} \Big[& p_1 = i \wedge p_2 = q \wedge p_4 = g(q) \wedge \text{BIT}(\vec{w}, p_3)^{\gamma_w} \wedge R_i(r_1, \ldots, r_{a_i})^{\gamma_i} \\ & \to p_2' = q' \wedge p_4' = g(q') \wedge p_3' = p_3 + \delta_w \wedge \text{BIT}(\vec{w}', p_3')^{\gamma_o} \\ & \wedge \forall p \neq p_3 (\text{BIT}(\vec{w}', p) \leftrightarrow \text{BIT}(\vec{w}, p)) \\ & \wedge \text{(spostamento testina di input)} \Big] \end{aligned}$$

dove $\text{BIT}(\vec{w}, p)$ è vera se e solo se il p-esimo bit del numero $w_1 + w_2 \cdot \max + \ldots + w_c \cdot \max^{c-1}$ è 1 (BIT è definibile in $\text{FO}_{\le,\cdot,+}$, si veda ad esempio [2, 1.17]). La parte della formula indicata con "spostamento della testina" descrive in che modo sono rapportate $\vec{r}$, $\vec{r}'$, p_1, p_1' e δ_i. Sebbene sia concettualmente simile alla parte già scritta, è molto più laboriosa, e abbiamo preferito tralasciarla. Notiamo che vale $C_R(\text{ID}, \text{ID}')$ se e solo se la configurazione corrispondente a ID può passare alla configurazione corrispondente a ID$'$ mediante la regola R. Per concludere definiamo allora

$$E(\text{ID}, \text{ID}') :\equiv \bigvee_R C_R(\text{ID}, \text{ID}').$$

È facile convincersi della correttezza di questa definizione. □

Osservazione 3.2.5. Nella dimostrazione precedente avremmo potuto memorizzare tutte le possibili configurazioni del nastro di lavoro all'interno del grafo grazie all'ipotesi $M \in \text{ASPACE}(c \log(n))$, ma non possiamo fare lo stesso per il nastro di input che ha lunghezza polinomiale in n (e quindi il numero di possibili configurazioni è esponenziale in n). Tuttavia, come visto, possiamo limitarci a memorizzare soltanto la posizione della testina sul nastro di input.

Corollario 3.2.6. $\text{P} \subseteq \text{FO(LFP)}_{\le,\cdot,+}$. In altre parole, per ogni macchina di Turing M che accetta o rifiuta una σ-struttura A in tempo polinomiale, esiste una σ-formula ϕ in $\text{FO(LFP)}_{\le,\cdot,+}$ tale che $A \models \phi$ se e solo se $M(\text{bin}(A))\downarrow_{\text{Y}}$

Capitolo 4

Caratterizzazione e proprietà di NP

4.1 NP equivale ad ESO

La dimostrazione seguente fa uso della caratterizzazione di NP data nel Corollario 2.1.7 e della caratterizzazione di P ottenuta nel capitolo precedente, nonché del fatto seguente. Per la dimostrazione svolta a lezione si veda invece [2, 7.2].

Fatto 4.1.1 ([1])**.** In strutture finite, ESO(LFP) senza punti fissi annidati equivale ad ESO.

Teorema 4.1.2 (Fagin)**.** NP = ESO.

Dimostrazione. Sia $\mathcal{A}$ un problema in NP. Per il Corollario 2.1.7 esistono una macchina di Turing M che lavora in tempo polinomiale e un $k \in \mathbb{N}$ tali che $A \in \mathcal{A}$ se e solo se $M(\mathrm{bin}(A)^\frown \mathrm{bin}(R))\downarrow_{\mathrm{Y}}$ per una qualche relazione R di arietà k su A. Per il Corollario 3.2.6, esiste una formula ϕ in $\mathrm{FO(LFP)}_{\leq,\cdot,+}$ tale che $M(\mathrm{bin}(A)^\frown \mathrm{bin}(R))\downarrow_{\mathrm{Y}}$ se e solo se $A, R \models \phi$. Dunque vale $A \in \mathcal{A}$ se e solo se $A \models \exists R^{(k)}\phi$. Concludiamo applicando il Fatto 4.1.1 alla formula $\exists R^{(k)}\phi$.

Viceversa, data una ϕ formula in ESO, mostriamo che esiste una macchina di Turing non deterministica N che lavora in tempo polinomiale e tale che $A \models \phi$ se e solo se $N(\mathrm{bin}(A))\downarrow_{\mathrm{Y}}$. Sia $\phi \equiv \exists \vec{R}\,\theta$, con θ in FO. Per la Proposizione 3.2.2 esiste una macchina di Turing M che lavora in tempo polinomiale tale che $A, \vec{R} \models \theta$ se e solo se $M(\mathrm{bin}(A)^\frown \mathrm{bin}(\vec{R}))\downarrow_{\mathrm{Y}}$. Notiamo che $\mathrm{bin}(A)^\frown \mathrm{bin}(\vec{R})$ ha lunghezza polinomiale rispetto a $|A|$, dunque M con input $\mathrm{bin}(A)^\frown \mathrm{bin}(\vec{R})$ termina ancora in tempo polinomiale rispetto a $|A|$. Costruiamo la macchina N in modo che scelga in modo non deterministico le relazioni $\vec{R}$ (equivalentemente i bit della stringa $\mathrm{bin}(\vec{R})$, che è polinomiale in $|A|$) e quindi simuli M con input $\mathrm{bin}(A)^\frown \mathrm{bin}(\vec{R})$. □

4.2 Giochi di Ehrenfeucht–Fraïssé

Definizione 4.2.1. Siano A, B due L-strutture. Descriviamo il gioco di Ehrenfeucht–Fraïssé $G_k(A, B)$ di durata $k \in \mathbb{N}$. Vi sono due giocatori indicati con i simboli "∃" e

"$\forall$". A ogni turno il giocatore $\forall$ sceglie un elemento da A o da B. Il giocatore $\exists$ cerca di imitarlo scegliendo un elemento corrispondente nell'altra struttura. Dopo k turni sono stati scelti degli elementi $a_1, \ldots, a_k \in A$ e $b_1, \ldots, b_k \in B$. Il giocatore $\exists$ vince il gioco se

$$A, a_1, \ldots, a_k \equiv_{QF} B, b_1, \ldots, b_k,$$

ovvero se, per ogni formula $\phi(x_1, \ldots, x_k)$ senza quantificatori, vale $A \models \phi(a_1, \ldots, a_k)$ se e solo se $B \models \phi(b_1, \ldots, b_k)$.

Definizione 4.2.2. Date due L-strutture A e B, diciamo che $A \sim_k^{EF} B$ se e solo se il giocatore $\exists$ ha una strategia vincente per il gioco di EF $G_k(A, B)$.

Definizione 4.2.3. Date due L-strutture A e B, diciamo che $A \equiv_k B$ se e solo se per ogni formula ϕ con rango di quantificazione $\mathrm{RQ}(\phi) \leq k$ vale $A \models \phi$ se e solo se $B \models \phi$.

Osservazione 4.2.4. Date due L-strutture A e B, vale $A \equiv_0 B$ se e solo se $A \equiv_{QF} B$ e vale $A \equiv B$ se e solo se per ogni $k \in \mathbb{N}$ vale $A \equiv_k B$.

Lemma 4.2.5. Ci sono solo un numero finito di classi di k-EF-equivalenza di n-uple; in particolare, se $C(n, k)$ è l'insieme di tali classi, si ha che $|C(n, k+1)| \leq s^{|C(n+1,k)|}$. Inoltre per ogni classe $H = [A, a_1, \ldots, a_n] \in C(n, k)$ esiste una formula ϕ_H con $\mathrm{RQ}(\phi_H) \leq k$ tale che:

1. $A \models \phi_H(a_1, \ldots, a_n)$;
2. Se $B \models \phi_H(b_1, \ldots, b_n)$, allora $(A, a_1, \ldots, a_n) \sim_k^{EF} (B, b_1, \ldots, b_k)$.

Dimostrazione. Per induzione su k. Supponiamo $k = 0$. Allora $A, a_1, \ldots, a_n \sim_0^{EF} B, b_1, \ldots, b_n$ se e solo se $A, \vec{a}$ verifica le stesse atomiche di $B, \vec{b}$. Ma, dato che il linguaggio è finito e non ci sono simboli di funzione, ci sono solo finite formule atomiche $R_1(\vec{x}), \ldots, R_r(\vec{x})$. Quindi H è caratterizzata dalla formula

$$\phi_H(x_1, \ldots, x_n) :\equiv \bigwedge_{i:A \models R_i(\vec{a})} R_i(\vec{x}) \wedge \bigwedge_{j:A \models \neg R_j(\vec{a})} \neg R_j(\vec{x})$$

Svolgiamo ora il passo induttivo. Per ipotesi induttiva sappiamo che una qualsiasi $H \in C(n+1, k)$ è caratterizzata da una formula $\phi(x_1, \ldots, x_{n+1})$. Ad ogni $\delta \subseteq C(n+1, k)$ associamo la formula

$$\phi_\delta(x_1, \ldots, x_n) :\equiv \bigwedge_{H \in \delta} \exists x_{n+1} \phi_H(x_1, \ldots, x_{n+1}) \wedge \bigwedge_{H \notin \delta} \neg \exists x_{n+1} \phi_H(x_1, \ldots, x_{n+1})$$

Osserviamo che $\mathrm{RQ}(\phi_\delta) \leq k+1$ poiché, per ipotesi induttiva, $\mathrm{RQ}(\phi_H) \leq k$. L'insieme di tutte le ϕ_δ coerenti (cioè aventi un modello) caratterizza le classi in $C(n, k+1)$. Infatti, sia data una struttura A e una n-upla $a_1, \ldots, a_n$. Definiamo $\delta \subseteq C(n+1, k)$ come l'insieme delle $H \in C(n+1, k)$ tali che $A \models \exists y \phi_H(a_1, \ldots, a_n, y)$. Per costruzione $A, a_1, \ldots, a_n \models \phi_\delta(x_1, \ldots, x_n)$. Mostriamo ora che se $B, b_1, \ldots, b_n \models \phi_\delta(x_1, \ldots, x_n)$

allora $B, b_1, \ldots, b_n \sim_{k+1}^{EF} A, a_1, \ldots, a_n$. A tal fine supponiamo, senza perdita di generalità, che il giocatore $\forall$ scelga al primo turno un elemento a in A. La classe $H = [A, a_1, \ldots, a_n, a]$ è in $C(n+1, k)$ e, dato che $B, b_1, \ldots, b_k \models \phi_\delta(x_1, \ldots, x_k)$, ne segue che $B \models \exists y \phi_H(b_1, \ldots, b_k, y)$. Sia allora b tale che $B \models \phi_H(b_1, \ldots, b_k, b)$. Per ipotesi induttiva $B, b_1, \ldots, b_k, b \sim_k^{EF} A, a_1, \ldots, a_k, a$ e quindi il giocatore $\exists$ ha una strategia vincente scegliendo b. □

Teorema 4.2.6. Date due L-strutture A e B, se $A \sim_k^{EF} B$ allora $A \equiv_k B$. Supponendo in più che L sia un linguaggio finito senza simboli di funzioni, vale anche il viceversa, ovvero $A \equiv_k B$ implica $A \sim_k^{EF} B$.

Dimostrazione. Supponiamo che valga $A \sim_k^{EF} B$. Per dimostrare che $A \equiv_k B$ procediamo per induzione su k. Il passo base $k = 0$ segue dall'osservazione precedente e dalle definizioni. Supponendo il risultato vero per k, dimostriamolo per $k+1$. Sia ϕ la più piccola formula di rango di quantificazione $k+1$ su cui i due modelli sono in disaccordo. Possiamo suppore senza perdita di generalità che $\phi \equiv \exists y \theta(y)$ e che esista $a \in A$ tale che $A \models \theta(a)$. Per la definizione di gioco di EF di durata $k+1$, esiste un $b \in B$ tale che $A, a \sim_k^{EF} B, b$. Per ipotesi induttiva questo implica che $A, a \equiv_k B, b$, e in particolare, essendo $\mathrm{RQ}(\theta(y)) = k$, che $B \models \theta(b)$, da cui $B \models \exists y \theta(y)$.

Viceversa, supponiamo $A, a_1, \ldots, a_n \equiv_k B, b_1, \ldots, b_n$, sia $H = [A, a_1, \ldots, a_n]$ e sia ϕ_H come nel Lemma 4.2.5. Dato che $\mathrm{RQ}(\phi_H) \leq k$, vale $B \models \phi_H(b_1, \ldots, b_n)$ e dunque per il lemma si ha $A, a_1, \ldots, a_n \sim_k^{EF} B, b_1, \ldots, b_n$. □

4.3 ESO$^{\text{MON}}$ non è uguale a co-ESO$^{\text{MON}}$

Sebbene non si sappia se NP $\neq$ co-NP, si può dimostrare un risultato simile: ESO$^{\text{MON}} \neq$ co-ESO$^{\text{MON}}$. Per fare questo utilizzeremo i giochi di EF, introdotti nella Sezione 4.2.

Definizione 4.3.1. Sia A una L-struttura. Il grafo di Gaifman di A è il grafo non orientato $G_A = (|A|, E^A)$ dove $G_A \models E(a, b)$ se e solo se a e b sono distinti e sono parte di una tupla $(c_1, \ldots, c_r)$ di elementi di A tali che $A \models R(c_1, \ldots, c_r)$ per una qualche relazione r-aria $R \in L$.

Osservazione 4.3.2. Se $L = \{E\}$ è il linguaggio dei grafi (con un'unica relazione E binaria) e A è un grafo non orientato, allora $G_A = A$.

Definizione 4.3.3. Data una L-struttura A, l'n-intorno $S_A(n, a)$ di un elemento $a \in A$ è la sottostruttura di A definita nel seguente modo:

$$S_A(n, a) = \{b \in A \mid d_{G_A}(a, b) \leq n\}.$$

Sia inoltre

$$S_A(n, a_1, \ldots, a_k) = \bigcup_{i=1}^{k} S_A(n, a_i).$$

Definizione 4.3.4. Siano (A, a) e (B, b) due L-strutture puntate. Diciamo che esse sono n-equivalenti secondo Gaifman se esiste un isomorfismo di strutture tra $S_A(n, a)$ ed $S_B(n, b)$ che manda a in b. La notazione che useremo per indicare la n-equivalenza secondo Gaifman è la seguente: $(A, a) \sim_n^G (B, b)$.

Definizione 4.3.5. Chiamiamo n-tipo di a in A la classe di equivalenza di (A, a) rispetto alla relazione $\sim_n^G$.

Definizione 4.3.6. Date due L-strutture A e B, diciamo che $A \sim_n^G B$ se per ogni n-tipo ι, A e B hanno lo stesso numero di elementi di n-tipo ι.

Osservazione 4.3.7. Se $A \sim_n^G B$, allora $A \sim_k^G B$ per ogni $k < n$.

Teorema 4.3.8. $A \sim_{3^n}^G B$ implica $A \sim_n^{EF} B$.

Dimostrazione. Mostriamo una strategia vincente per il giocatore $\exists$ nel gioco di Ehrenfeucht–Fraïssé $G_n(A, B)$. Dimostriamo per induzione sul numero k di mosse giocate (con $0 \leq k \leq n$) che il giocatore $\exists$ è in grado di preservare il seguente invariante:

$$\big\langle S_A(3^{n-k}, a_1, \ldots, a_k), a_1, \ldots, a_k \big\rangle \cong \big\langle S_B(3^{n-k}, b_1, \ldots, b_k), b_1, \ldots, b_k \big\rangle,$$

dove l'isomorfismo è di L-strutture. Chiamiamo π_k un tale isomorfismo (la cui esistenza verrà dimostrata induttivamente).

- Passo base ($k = 0$). È banalmente vero poiché entrambe le strutture sono vuote.
- Passo induttivo (k implica $k+1$). Supponiamo senza perdita di generalità che il giocatore $\forall$ scelga, come $(k+1)$-esima mossa, l'elemento $a = a_{k+1} \in A$. Distinguiamo due casi.
  - Primo caso: $a \in S_A(2 \cdot 3^{n-k-1}, a_i)$ per qualche $i \in \{1, \ldots, k\}$. Allora l'intorno $S_A(3^{n-k-1}, a)$ è completamente contenuto in $S_A(3^{n-k}, a_i)$. Di conseguenza è sufficiente che il giocatore $\exists$ giochi l'elemento $b = \pi_k(a)$: l'isomorfismo π_{k+1} si ottiene semplicemente restringendo π_k a $S_A(3^{n-k}, a_1, \ldots, a_k)$.
  - Secondo caso: $a \notin S_A(2 \cdot 3^{n-k-1}, a_1, \ldots, a_k)$. Sia ι il 3^{n-k-1}-tipo di a in A. Per ipotesi induttiva, gli insiemi $S_A(2 \cdot 3^{n-k-1}, a_1, \ldots, a_k)$ e $S_B(2 \cdot 3^{n-k-1}, b_1, \ldots, b_k)$ contengono lo stesso numero di elementi di tipo ι. Per differenza (usando l'ipotesi $A \sim_{3^n}^G B$), anche i complementari di questi due insiemi contengono lo stesso numero di elementi di tipo ι, quindi esiste $b \notin S_B(2 \cdot 3^{n-k-1}, b_1, \ldots, b_k)$ con lo stesso 3^{n-k-1} tipo di a. Osserviamo che $S_A(3^{n-k-1}, a)$ e $S_A(3^{n-k-1}, a_1, \ldots, a_k)$ sono disgiunti, e similmente anche $S_B(3^{n-k-1}, b)$ e $S_B(3^{n-k-1}, b_1, \ldots, b_k)$ sono disgiunti. Di conseguenza l'isomorfismo π_k, ristretto a $S_A(3^{n-k-1}, a_1, \ldots, a_k)$, può essere esteso ad un isomorfismo

    $$\pi_{k+1} \colon S_A(3^{n-k-1}, a_1, \ldots, a_k, a) \to S_B(3^{n-k-1}, b_1, \ldots, b_k, b)$$

    che manda a in b.

Per $k = n$ si ha in particolare che $A, a_1, \ldots, a_n \equiv_{QF} B, b_1, \ldots, b_n$. Pertanto quella che abbiamo mostrato è effettivamente una strategia vincente per il giocatore $\exists$. □

Corollario 4.3.9. $A \sim^G_{3^n} B$ implica $A \equiv_n B$.

Dimostrazione. Segue dai Teoremi 4.3.8 e 4.2.6. □

Nel resto di questa sezione utilizzeremo il linguaggio $L = \{E\}$ dei grafi orientati, dove E è una relazione binaria. Introduciamo il problema decisionale "Connected", al quale appartengono tutti e soli i grafi connessi. Dimostreremo che tale problema appartiene a co-ESO$^{\text{MON}}$ ma non ad ESO$^{\text{MON}}$.

Lemma 4.3.10. Connected $\in$ co-ESO$^{\text{MON}}$.

Dimostrazione. Dobbiamo esprimere la proprietà di essere sconnesso con una formula in ESO$^{\text{MON}}$. Una condizione equivalente ad essere sconnesso è che l'insieme dei vertici possa essere partizionato in due sottoinsiemi non vuoti per cui non esistono archi dal primo verso il secondo. Questa condizione può essere espressa mediante la seguente formula ESO$^{\text{MON}}$:

$$\exists U^{(1)}, W^{(1)}\Big[\exists x U(x) \wedge \exists x W(x) \wedge \forall x\big(U(x) \dot{\vee} W(x)\big) \wedge \forall x, y\big(U(x) \wedge W(y) \rightarrow \neg E(x,y)\big)\Big].$$

Quindi Connected $\in$ co-ESO$^{\text{MON}}$. □

Lemma 4.3.11. Connected $\notin$ ESO$^{\text{MON}}$.

Dimostrazione. Supponiamo per assurdo che esista una L-formula $\exists P_1^{(1)}, \ldots, P_r^{(1)}\varphi$, con $\varphi \in$ FO, tale che per ogni grafo orientato $G = (V^G, E^G)$ si abbia

$$G \models (\exists \vec{P})\varphi \iff G \text{ connesso.}$$

Sia $m = \text{RQ}(\varphi)$, e sia $\tau = \{E, P_1, \ldots, P_r\}$ un nuovo linguaggio che estende L in cui le P_i sono relazioni unarie. Osserviamo che una τ-struttura può essere considerata come un grafo colorato con 2^r colori, dove il colore del vertice x è codificato dai bit $P_1(x), \ldots, P_r(x)$.

Sia ℓ un numero naturale sufficientemente grande rispetto a m ed r (sarà chiaro in seguito cosa si intende con "sufficientemente grande"). Sia $G = (V^G, E^G)$ un grafo ciclico su ℓ vertici, con gli archi orientati tutti nello stesso verso. Essendo G connesso, soddisfa la formula $(\exists \vec{P})\varphi$. In altre parole esiste una τ-struttura $A = (|A|, E^A, P_1^A, \ldots, P_r^A)$, con $|A| = V^G$ ed $E^A = E^G$, tale che $A \models \varphi$. Rappresentiamo A come un grafo colorato con 2^r colori (vedi Figura 4.1).

Avendo preso ℓ sufficientemente grande, esistono sicuramente due vertici $a, b \in |A|$ dello stesso 3^m-tipo e con $S_A(3^m, a) \cap S_B(3^m, b) = \varnothing$. Siano a^- e b^- i vertici del grafo che vengono immediatamente prima di a e b, come nella Figura 4.1. Sia B la τ-struttura ottenuta da A mantenendo le stesse relazioni unarie (cioè ponendo $P_i^B = P_i^A$ per ogni $i = 1, \ldots, r$) ma cambiando alcuni archi: rimuoviamo gli archi da a^- ad a e da b^- a b, ed aggiungiamo gli archi da a^- a b e da b^- ad a. Come L-struttura, B è un grafo costituito da due cicli disgiunti.

L'idea di questa costruzione è che, localmente, i grafi colorati A e B sono indistinguibili fino a distanza 3^m; più formalmente, si ha che $A \sim^G_{3^m} B$. Allora, per il Corollario 4.3.9, possiamo dedurre che $A \equiv_m B$. Dato che $A \models \varphi$ e che $\mathrm{RQ}(\varphi) = m$, si ha anche che $B \models \varphi$. Conseguentemente il grafo $H = (|B|, E^B)$ soddisfa la formula $(\exists \vec{P})\varphi$, dunque deve essere connesso. Ma questa è una contraddizione, perché H è costituito da due cicli disginti.

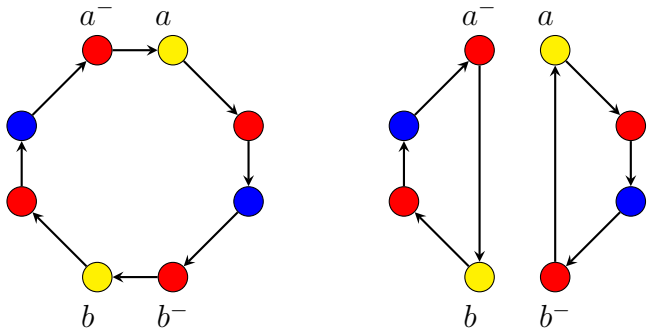

Figura 4.1: Esempio delle strutture A (sulla sinistra) e B (sulla destra), per $\ell = 8$.

□

Teorema 4.3.12. $\mathrm{ESO}^{\mathrm{MON}} \neq \text{co-}\mathrm{ESO}^{\mathrm{MON}}$.

Dimostrazione. Segue immediatamente dai Lemmi 4.3.10 e 4.3.11. □

4.4 NP-completezza di SAT

Il Teorema 4.1.2 consente di dimostrare in modo particolarmente semplice l'NP-completezza del problema di soddisfacibilità booleana di formule proposizionali, denominato comunemente "SAT".

Teorema 4.4.1. SAT è NP-completo.

Dimostrazione. SAT appartiene a NP, perché un certificato di soddisfacibilità di una formula è dato dal valore da assegnare alle variabili per fare in modo che la formula risulti vera.

Dobbiamo dimostrare che un qualsiasi problema $\mathcal{A} \in \mathrm{NP}$ si riduce polinomialmente a SAT. Supponiamo che $\mathcal{A}$ sia un insieme di strutture nel linguaggio $L = \{R_1, \ldots, R_t\}$, dove $R_1, \ldots, R_t$ sono relazioni di arietà $b_1, \ldots, b_t$, rispettivamente. Per il Teorema 4.1.2 esiste una formula γ della forma

$$\gamma :\equiv \exists\, S_1, \ldots, S_r\, \forall \underbrace{x_1, \ldots, x_n}_{\text{prim'ordine}}\ \underbrace{\varphi(x_1, \ldots, x_k)}_{\text{senza quantificatori}}$$

tale che $\mathcal{A}$ sia dato dai modelli di γ.

Sia A una L-struttura, e sia $n = |A|$. Abbiamo che A appartiene a $\mathcal{A}$ se e solo se soddisfa la formula γ, ovvero se e solo se esistono delle relazioni $S_1^A \subseteq n^{a_1}, \ldots, S_r^A \subseteq n^{a_r}$ tali che sia soddisfatta $\forall x_1, \ldots, x_n \, \varphi(x_1, \ldots, x_k)$. Questo è equivalente a trovare un'assegnazione che renda vera la formula proposizionale

$$\psi :\equiv \bigwedge_{0 \leq c_1, \ldots, c_k < n} \varphi(c_1, \ldots, c_k)$$

nelle varabili booleane $S_i(\vec{y})$ ed $R_j(\vec{z})$, al variare di $i \in \{1, \ldots, r\}$, $j \in \{1, \ldots, t\}$, $\vec{y} \in n^{a_i}$, $\vec{z} \in n^{b_j}$. Sostituiamo a ciascuna variabile del tipo $R_j(\vec{z})$ il valore vero o falso, a seconda che valga o meno $A \models R_i(\vec{z})$; chiamiamo $\tilde{\psi}$ la formula così ottenuta, che dipende ora solamente dalle variabili $S_i(\vec{y})$ al variare di i e di $\vec{y} \in n^{a_i}$. In conclusione, A appartiene a $\mathcal{A}$ se e solo se esiste un'assegnazione delle variabili $S_i(\vec{y})$ che rende vera la ψ. Il numero di tali variabili è $n^{a_1} + \ldots + n^{a_r}$, che è un polinomio in n. Abbiamo quindi mostrato una riduzione polinomiale di $\mathcal{A}$ a SAT. □

Bibliografia

[1] Jan Van den Bussche, *Introduction to finite model theory*, `http://dtai.cs.kuleuven.be/krr/files/seminars/IntroToFMT-janvdbussche.pdf`, consultato il 18 dicembre 2014.

[2] N. Immerman, *Descriptive complexity*, Graduate texts in computer science, Springer New York, 1999.