Disclaimer

Questi appunti nascono ad uso e consumo degli autori, che li hanno TEXati in diretta e successivamente risistemati durante il corso di Teoria Descrittiva della Complessità tenuto dal professor Berarducci presso l'Università di Pisa durante il primo semestre dell'anno accademico 2014/2015. Come conseguenza possono essere *molto* poco chiari, difettare di qualcosa, eccetera eccetera, e in particolare *non* sono gli appunti ufficiali del corso. Sulla pagina del corso (vedi bibliografia) ci sono altri ottimi appunti presi da altri studenti e diverso materiale fornito dal docente, che consigliamo vivamente di consultare. Sentitevi liberi di mandare insulti, segnalare sviste, errori, imprecisioni, carenze eccetera presso `mennuni@mail.dm.unipi.it`. Ogni contributo è bene accetto.

L'ultima versione di questi appunti e il relativo sorgente sono disponibili presso `poisson.phc.unipi.it/~mennuni/`. Questa versione è stata compilata in data 12 gennaio 2015. La maniera migliore cui riesca a pensare per sapere cosa cambia rispetto alla versione precedente che avete scaricato è scaricarsi anche i sorgenti e tiraci sopra un `diff`. Se non avete il sorgente della versione precedente potete provare a usare `diffpdf`, ma sinceramente io non ci ho mai provato.

Rosario "Mufasa" Mennuni

P.S.: Se sapete usare git potete recuperare i sorgenti di questi appunti anche all'indirizzo `http://git.phc.unipi.it/logica/teoria-descrittiva-della-complessita` o direttamente usando

```
git clone git://git.phc.unipi.it/logica/teoria-descrittiva-della-complessita.git
```

GiorgioMossa.

SPAM: ascoltate `www.radiocicletta.it`

Indice

[Questa pagina è (quasi) bianca per motivi di parità.]

1 25/09

Nota: Berarducci va in congedo il 15 gennaio, quindi sarebbe cosa buona e giusta fare l'esame prima. I nomi di riferimento (libri, web, eccetera) sono Väänänen, Immerman e Ebbinghaus-Flum. Ci sono degli appunti di uno studente sulla pagina vecchia del corso, tipo del 2008. Vedi Bibliografia in fondo.

Notazione 1.1. Le frecce "maiuscole" $\Rightarrow$, $\Leftrightarrow$ "vivono" nella metateoria, quelle del tipo $\rightarrow$, $\leftrightarrow$ sono parte delle formule. Ad esempio $\mathcal{A} \vDash \varphi \leftrightarrow \psi \Leftrightarrow \mathcal{B} \vDash \psi \rightarrow \varphi$ vuol dire che in $\mathcal{A}$ è vera $\varphi \leftrightarrow \psi$ se e solo se in $\mathcal{B}$ è vera $\psi \rightarrow \varphi$.

1.1 Motivazioni

Scopo del corso è studiare la Teoria della Complessità usando metodi di Teoria dei Modelli. Essenzialmente sarà un corso di modelli finiti.

Definizione 1.2. Quella di modello. Vedi appunti di Logica Matematica[1].

Esempio 1.3. Un grafo $G = (V, E^G)$, con $E^G \subseteq V^2$ nel linguaggio $L = \{E\}$.

Si può dare come assioma il fatto che G non sia orientato chiedendo la transitività di E, cioè

$$G \vDash \forall x, y\ [E(x, y) \leftrightarrow E(y, x)]$$

e che sia senza loops con

$$G \vDash \forall x\ \neg E(x, x)$$

Al contrario che nel corso di Teoria dei Modelli, ci concentreremo prevalentemente su strutture finite, cioè con dominio finito. Come ulteriore esempio vediamo come si fa a dire che un grafo è 3-colorabile, cioè partizionabile con tre "colori" in maniera che nessuna coppia di vertici adiacenti, cioè collegati da un arco, abbia lo stesso colore.

Svarione 1.4. [2]Esempi di grafi planari e non, e citazione del Teorema dei 4 colori (non riportato).

La 3-colorabilità non è esprimibile al primo ordine nel linguaggio $L = \{E\}$, ma al secondo sì. Le variabili maiuscole varieranno su relazioni, e l'apice indica l'arietà.

$$\begin{aligned} G \vDash \exists R^1, B^1, G^1\ &\big[\forall x\ [(Rx \vee Bx \vee Gx) \wedge \neg(Rx \wedge Bx) \wedge \neg(Rx \wedge Gx) \wedge \neg(Bx \wedge Gx)]\big] \\ &\wedge \big[\forall x, y\ Exy \rightarrow [\neg(Rx \wedge Ry) \wedge \neg(Bx \wedge By) \wedge \neg(Gx \wedge Gy)]\big] \end{aligned}$$

questa è una formula *esistenziale monadica del second'ordine* ($\exists$SO-MON), dove il *monadica* vuol dire che i quantificatori spaziano su variabili di arietà 1. Il primo legame fra complessità e modelli finiti è dato da risultati del tipo

[1]Disponibili dove avete trovato questi appunti; vedi prima pagina.

[2]Etichetterò con "Svarione" i fuori programma, le digressioni, eccetera. Chi non lo gradisse può leggerlo come "digressione".

Teorema (Fagin, '74)**.** $\mathsf{NP} = \exists\mathsf{SO}$.

Vediamo di cosa stiamo parlando.

1.2 Complessità Computazionale

Sia Σ un alfabeto finito, ad esempio $\Sigma = \{A, B, C, \ldots, Z\}$ o più comodamente (per lo studio teorico) $\Sigma = \{0, 1\}$. Indichiamo l'insieme delle parole (stringhe) con

$$\Sigma^* = \bigcup_{n \in \omega} \Sigma^n$$

ad esempio per il primo alfabeto $ABA \in \Sigma^*$. Indichiamo la lunghezza di x con $|x|$.

Definizione 1.5. Un *problema* S è un sottoinsieme di Σ^*.

Ad esempio vedremo che si può scrivere come problema la 3-colorabilità codificando i grafi come stringhe.

Definizione 1.6. Sia $S \subseteq \Sigma^*$. Diciamo che $S \in \mathsf{P}(= \mathsf{PTIME})$ se esistono una macchina di Turing[3] M e un polinomio $p(x) \in \mathbb{Z}[x]$ tali che per ogni $x \in \Sigma^*$ si ha che $M(x)$ si ferma in $\leq p(|x|)$ passi e risponde "sì" se $x \in S$ e "no" se $x \notin S$.

In altre parole i problemi P sono quelli risolvibili da una macchina di Turing in tempo polinomiale nell'input.

Esempio 1.7. L'insieme dei primi è un problema in (sottoinsieme di) $\mathbb{N} \subseteq \Sigma^*$, dove $\Sigma = \{0, 1\}$ o[4] $\Sigma = \{0, 1, \ldots, 9\}$.

L'algoritmo "stupido", provare tutti i divisori *sembra* polinomiale: per vedere se 1000 è primo si prova a dividerlo per i numeri fino a 1000. In realtà l'input è lungo 4, e in generale logaritmico nel numero, per cui l'algoritmo stupido è in realtà esponenziale, cioè ci mette circa $2^{|x|}$ passi. È stato mostrato[5] che questo problema è polinomiale:

Teorema 1.8 (Algoritmo AKS, 2002)**.** Primi $\in \mathsf{PTIME}$.

Come dicevamo prima la complessità varia se usiamo l'alfabeto unario[6]. Inoltre bisognerebbe anche controllare se l'input codifica qualcosa (ad esempio le stringhe che iniziano per 0 non codificano un numero), ma se questo controllo è fattibile in tempo polinomiale l'analisi che faremo non cambia[7].

[3] Si dà per buono che il lettore sappia cos'è, ma lo rivedremo più avanti.

[4] Ma non $\{0\}$, perché la complessità cambierebbe troppo. La rappresentazione in base 1 è significativamente più lunga (lineare) che nelle altre basi (logaritmica).

[5] Di recente e con molta fatica.

[6] "Con gli uomini delle caverne era lineare, poi è stato inventato l'alfabeto binario ed è stato un regresso, ora è esponenziale."

[7] Ma a seconda della codifica o del modello di calcolo lo stesso problema può essere lineare, quadratico...

Un grafo può essere codificato, ad esempio, con la sua matrice di adiacenza scrivendo le righe una dopo l'altra, oppure scrivendo le coppie collegate nell'alfabeto con le virgole, parentesi, eccetera. La seconda è più comoda per grafi sparsi. Comunque questi dettagli non ci interessano, e daremo per buono che le codifiche siano fatte in maniera ragionevole.

Problema Aperto 1.9. 3-colorabili $\in \mathsf{P}$?

Sicuramente *se qualcuno ci fornisce una colorazione*, controllare se questa è una colorazione valida o meno (se non colora vertici adiacenti con lo stesso colore) è fattibile in tempo polinomiale. Se $|V| = n$ però le possibili partizioni di V con 3 classi di equivalenza sono 3^n e le possibili "colorazioni cattive" (cioè dove non è detto che ogni vertice abbia esattamente un colore) sono 2^{3n}, e controllarle tutte richiede tempo esponenziale in n. Invece essere 2-colorabile è P, infatti colorato un vertice gli altri sono forzati.

1.3 La Classe NP

Una maniera per definirla è

$$\mathsf{NP} = \exists^{\text{poly}}\mathsf{P}$$

Ad esempio

$$(G, V) \in \text{3-col} \Leftrightarrow \exists f \colon V \to 3[\ldots]$$

la funzione f (che pensiamo come colorazione) è codificabile in maniera lineare in n (basta fare la "tabella"), e quindi polinomiale; in altre parole esiste $p \in \mathbb{Z}[x]$ tale che se $G = (V, E)$ e f è una colorazione[8] $|f| \leq p(|G|)$. Questa f, e in generale la variabile che stiamo quantificando, la chiamiamo *certificato*.

Definizione 1.10. Un problema S è in NP se e solo se esistono un problema $Q \in \mathsf{P}$ e un polinomio $p(x) \in \mathbb{Z}[x]$ tale che per ogni stringa y si ha che $y \in S$ se e solo se esiste un certificato x di lunghezza $|x| \leq p(|y|)$ tale che $(x, y) \in Q$.

Chiaramente i possibili certificati per y (contando anche quelli "falsi", per cui $(x, y) \notin Q$) sono $|\Sigma|^{p(|y|)}$. Notiamo che *non* stiamo chiedendo nulla sulla "reperibilità" dei certificati, ma solo che siano abbastanza corti e che fare la verifica sia un processo breve. Il nome NP sta per Nondeterministico-Polinomiale. Sono i problemi risolvibili in tempo polinomiale da macchine di Turing nondeterministiche, cioè che possono "tirare a caso"[9]. Comunque, visto che c'è un numero esponenziale di certificati, abbiamo

$$\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{EXP}$$

[8] Con $|f|$ e $|G|$ indichiamo la lunghezza della codifica.

[9] O che hanno la testina guidata da un automa nondeterministico. Si può pensare alla sua computazione come ad un albero; in questo caso la macchina proverebbe contemporaneamente tutti i certificati, e quindi sa rispondere in tempo polinomiale. Info più precise più avanti.

dove con EXP indichiamo $\bigcup_{n\in\omega} \mathsf{TIME}\left(2^{|x|^n}\right)$. Si sa che $\mathsf{P} \neq \mathsf{EXP}$, se le altre inclusioni sono strette è un problema aperto[10].

1.4 Cosa C'entrano i Modelli Finiti

Avevamo preannunciato che $\mathsf{NP} = \exists\mathsf{SO}$. Dovrebbe essere chiaro a questo punto che $\exists\mathsf{SO}$ vuol dire che la formula ha quantificatori esistenziali del second'ordine all'inizio seguiti da una formula del prim'ordine nel linguaggio espanso con i simboli di predicato appena introdotti dalla quantificazione.

In questi termini dunque $S \in \mathsf{NP}$ se e solo se esiste una formula $\varphi \in \exists\mathsf{SO}$ tale che per ogni grafo finito G

$$G \in S \Leftrightarrow G \vDash \varphi$$

Quello che fa la formula sui modelli infiniti è irrilevante.

Indichiamo con FO le formule del prim'ordine e con SO quelle del secondo. Chiaramente $\mathsf{FO} \subseteq \exists\mathsf{SO} \subseteq \mathsf{SO}$. Anticipiamo che SO è incluso in PSPACE, mentre $\mathsf{FO} \subseteq \mathsf{L} \subseteq \mathsf{P}$ (vedremo le definizioni precise più avanti). Nei modelli finiti la logica del second'ordine è molto importante, perché quasi tutti gli strumenti del prim'ordine (Löwenheim-Skolem, per dirne uno) falliscono. L'unica cosa che sopravvive sono i giochi di Ehrenfeucht-Fraïssé.

Ripetiamo che, mentre il significato del lato destro dell'uguaglianza nel Teorema di Fagin non ha troppe ambiguità, quello sinistro potrebbe dipendere dalla codifica, ma c'è un modo abbastanza standard di codificare una struttura in una stringa, ma per il momento non staremo troppo a insistere su questi aspetti.

Definizione 1.11. La classe $\mathsf{co\text{-}NP}$ è quella dei problemi il cui complementare è in NP.

Dunque per Fagin coincide con $\forall\mathsf{SO}$[11]. Notiamo che, dato che P è chiusa per complemento, da $\mathsf{P} = \mathsf{NP}$ seguirebbe $\mathsf{NP} = \mathsf{co\text{-}NP}$. In particolare

Corollario 1.12. $\mathsf{NP} \neq \mathsf{co\text{-}NP} \Rightarrow \mathsf{P} \neq \mathsf{NP}$.

Alcuni risultati coinvolgono le formule in cui i quantificatori sono ammessi solo per predicati unari ("monadici"):

Teorema. $\exists^{\mathsf{MON}}\mathsf{SO} \neq \forall^{\mathsf{MON}}\mathsf{SO}$.

La 3-colorabilità è un problema *NP-completo*. Questo vuol dire che se è P allora[12] $\mathsf{P} = \mathsf{NP}$. Inoltre questo problema è monadico. Si sarebbe tentati di dire che il Teorema precedente conclude $\mathsf{P} = \mathsf{NP}$. Chiaramente questo discorso non funziona.

[10] Ed è uno dei "problemi del millennio" del Clay Institute. Ci si vince un milione di dollari americani.

[11] Il significato è analogo a quello di $\exists\mathsf{SO}$. Una notazione meno ambigua, che chiarisce che la formula dopo il quantificatore è del prim'ordine, è $\forall^{\mathsf{SO}}\mathsf{FO}$.

[12] Questa non è la definizione "vera", la vedremo più avanti.

2 29/09

2.1 Esercitazione

Notazione 2.1. ESO indicherà le formule che nella scorsa lezione indicavamo con $\exists$SO.

Esercizio 2.2. PARI $\in$ ESO

In altre parole abbiamo in input una struttura $\mathcal{A}$ e vogliamo una formula φ tale che $\mathcal{A} \vDash \varphi$ se e solo se $|A|$ è pari. Osserviamo che PARI $\in$ P. È vero inoltre che PARI $\notin$ FO.

Soluzione. La formula sarebbe "esiste una funzione bigettiva con dominio e codominio che partizionano A":

$$\begin{aligned}\exists C^1\ \exists D^1\ \exists F^2\ \big[&(\forall x\ C(x) \leftrightarrow \neg D(x))\\ &\wedge(F(x,y) \wedge F(x,z) \to y = z)\\ &\wedge(F(x,u) \wedge F(x',u) \to x = x')\\ &\wedge((\forall x : Cx)\ (\exists y : Dy)\ F(x,y)\\ &\wedge((\forall y : Dy)(\exists x : Cx)F(x,y)\big]\end{aligned}$$

□

Notazione 2.3. $\forall y : Dy$ vuol dire $\forall y\ Dy \to$ e $\exists y : Cy$ vuol dire $\exists yCy\wedge$.

Soluzione (alternativa). Un'altra maniera è

$$(\exists F^2\ F \text{ funzione iniettiva definita ovunque})(\exists S^1)\ F(x,y) \to [S(x) \leftrightarrow \neg S(y)]$$

Infatti l'iterazione di una tale funzione crea tanti "cicli" della forma $S, \neg S, S, \neg S, \ldots$ che per iniettività devono avere ognuno cardinalità pari. □

Svarione 2.4. Pari si può fare monadico?[13]

Esercizio 2.5. La chiusura transitiva TC. Data R binaria possiamo indicare con R^+ la sua chiusura transitiva[14]

$$R^+(x,y) \leftrightarrow \big[\exists n \in \omega\backslash 1\ \exists\langle x_i \mid i < \omega\rangle\ (x_0 = x)\wedge(x_n = y)\wedge(\forall i < n\ R(x_i, x_{i+1}))\big]$$

Se usiamo ω invece di di $\omega \setminus 1$ diventa la chiusura sia transitiva che riflessiva (basta prendere $n = 0$).

[13] "Bella domanda; credo di no perché sennò ci sarebbe venuto in mente, visto che siamo tutti molto preparati."

[14] La notazione è quella dello Jech: $\langle x_i \mid i < \omega\rangle$ indica una successione, mentre $\{x_i \mid i < \omega\}$ indica un insieme indicizzato. Se non è chiara la differenza si pensi a cosa succede quando gli x_i sono tutti uguali.

Osservazione 2.6. La chiusura transitiva può anche essere definita ricorsivamente come la più piccola relazione binaria che verifica $R^+(x,y) \leftarrow R(x,y)$ e $R^+(x,z) \leftarrow (R(x,y) \wedge R^+(y,z))$.

Vogliamo una $\{R^2\}$-formula $\varphi(x,y) \in \mathsf{ESO}$ tale che

$$(A,R) \vDash \varphi(a,b) \Leftrightarrow R^+(a,b)$$

Dire "è la più piccola tale che ..." è $\exists\forall$ e non va bene. Una cosa più furba è dire "esiste un cammino P da a a b" in questo modo:

$$\begin{aligned}
\exists P^2 \big[& \forall x,y\ P(x,y) \to R(x,y) \\
& \wedge \forall x,y,z\ P(x,y) \wedge P(x,z) \to y = z \\
& \wedge \exists x P(a,x) \\
& \wedge \exists x P(x,b) \\
& \wedge \neg\exists x P(x,a) \\
& \wedge \neg\exists x P(b,y) \\
& \wedge \forall x(\exists y\ P(y,x) \to \forall z(P(z,x) \to y = z)) \\
& \wedge \forall x(\exists y\ P(x,y) \to \forall z(P(x,z) \to y = z)) \\
& \wedge (\forall x \neq b)\exists y\ P(y,x) \to \exists z P(x,z) \\
& \wedge (\forall x \neq a,b)\exists y P(x,y) \leftrightarrow \exists z P(z,x) \\
& \vee (a = b \wedge P(a,a)) \big]
\end{aligned}$$

(in realtà con questi assiomi P contiene un cammino e magari è più grande, ma questo non dà problemi).

Osservazione 2.7. La chiusura transitiva si riesce a scrivere con una formula del second'ordine anche per strutture infinite (con la $\exists\forall$), ma questo trucco "furbo" funziona solo su strutture finite; il problema sono cose del tipo

$$\begin{aligned}
& a \to \circ \to \circ \to \circ \to \circ \to \dots \\
& \dots \to \circ \to \circ \to \circ \to \circ \to b
\end{aligned}$$

Anche $\mathsf{TC} \in \mathsf{P}$. Supponiamo infatti che $A = n$ e ci chiediamo $(A,R,a,b) \overset{?}{\to} R^+(a,b)$. Prendiamo una variabile[15] x che rappresenta una lista dei vertici raggiungibili da a, partiamo da $x = \{a\}$ e ad ogni passo visitiamo i vicini dei nodi nella lista, cioè visitiamo il grafo in ampiezza. Quando la lista si stabilizza (o dopo n passi) possiamo controllare se nella lista c'è b o meno. È facile vedere che questo algoritmo è polinomiale. Dunque $\mathsf{TC} \in \mathsf{P} \subseteq \mathsf{NP} \cap \mathsf{coNP}$. Non è noto se l'altra inclusione è vera, ma notiamo l'analogia con Teorema di Post. Comunque

[15] "Che varia per davvero, non come quelle dei matematici che non variano."

Esercizio 2.8. Dato che il problema è in coNP deve essere risolubile anche con una formula universale. Trovarla.

Soluzione. "Per ogni partizione con due classi, una con a e una con b, c'è una freccia che va dalla classe di a a quella di b". □

Soluzione (alternativa). $\forall S^2\ (S^2 \supseteq R \wedge S^2 \text{ transitiva} \to S^2(a,b))$. □

2.2 Least Fixed Point

Vogliamo formalizzare l'idea che R^+ è il minimo punto fisso delle equazioni $R^+(x,y) \leftarrow R(x,y)$ e $R^+(x,z) \leftarrow (R(x,y) \wedge R^+(y,z))$. La via che prendiamo è aggiungere alla logica del prim'ordine la possibilità di usare costruzioni induttive aggiungendo un operatore LFP di minimo punto fisso ottenendo un sistema che chiameremo FO(LFP). Vogliamo quindi scrivere qualcosa del tipo $R^+ = \min S(\varphi([S]))$.

Vogliamo fare in modo che gli oggetti a sinistra siano più grandi di quelli a destra, cioè che la freccia corrisponda ad un "accrescimento" della relazione. Vedremo meglio la prossima volta. In ogni caso $\mathsf{TC} \in \mathsf{FO(LFP)}$. Vedremo che

Spoiler 2.9. $\mathsf{P} = \mathsf{FO(LFP)} + (\leq, \mathsf{BIT})$.

L'ultimo pezzo è necessario, infatti $\mathsf{PARI} \notin \mathsf{FO}$. Il $\leq$ vuol dire che imponiamo che $L \ni \leq$ e che il $\leq$ sia interpretato come ordine totale. L'idea è che a un computer non possiamo dare in input un grafo, ma solo un *grafo ordinato*, nel senso che i vertici vanno scritti in qualche ordine, e questo aggiunge struttura al problema, che può essere usata per risolverlo in meno tempo. Ad esempio se vogliamo vedere se un grafo ha almeno grado 2 il tempo di calcolo può dipendere da come è memorizzato il grafo, ma a noi piacerebbe che proprietà invarianti per isomorfismo non dipendessero "dall'implementazione"... BIT vuol dire che si possono interpretare gli elementi della struttura come numeri e fare operazioni bitwise. Questo suggerisce che sia una soluzione molto più sporca rispetto alla caratterizzazione di NP.

Esempio 2.10 (Successioni binarie)**.** Possiamo vedere le successioni binarie come una struttura nel linguaggio $\leq, P^1$, dove $\leq$ è un ordine totale. Infatti possiamo scrivere $A = \{0, 1, \ldots, n-1\}$ e usare la P per dire 1, cioè $P(i) \Leftrightarrow a_i = 1$.

Questo ci permette di fare a meno di parlare di computer e codifiche.

Esercizio 2.11. Provare a scrivere una formula del prim'ordine che dica che una successione binaria ha almeno cinque uni usando solamente due variabili.

Svarione 2.12. In generale ogni cosa sulle successioni binarie che si riesce a dire al prim'ordine si riesce a dire usando solo tre variabili. Il numero di variabili per le formule è un po' come lo spazio in memoria per i computer.

3 02/10

3.1 Ancora sul Least Fixed Point

Sistemiamo bene questa questione del minimo punto fisso, studiando la chiusura transitiva e riflessiva R^*, per una relazione binaria R. Vogliamo che valga:

1. $R^*(x, x)$
2. $R^*(x, y) \leftarrow R(x, y)$
3. $R^*(x, y) \leftarrow \exists z\, [R(x, z) \wedge R^*(z, y)]$

La chiusura transitiva e riflessiva R^* è la minima[16] relazione che verifica le condizioni precedenti. Consideriamo la formula

$$\varphi(S, x, y) \equiv \big[x = y \vee \exists z\, (Rxz \wedge Szy)\big]$$

che è *positiva in* S, cioè S compare all'interno di un numero pari di negazioni. Più formalmente

Definizione 3.1. Se S è positiva in ψ e θ, allora lo è in $\psi \wedge \theta$, $\psi \vee \theta$, $\exists x\, \psi$, $\forall x\, \psi$; è *negativa* in $\neg\psi$ né in $\psi \to \varphi$. Se S è negativa in θ è positiva in $\neg\theta$. Le formule atomiche sono positive.

Una formula positiva definisce una funzione monotona, in questo senso: assumendo $\varphi(S, x, y)$ positiva in S (ad esempio la φ sopra) abbiamo la monotonia degli insiemi definiti, nel senso che se $S \subset S'$ vale

$$\{(x, y) \mid \varphi(S, x, y)\} \subset \{(x, y) \mid \varphi(S', x, y)\}$$

in ogni struttura $\mathcal{A}$. Questo può agilmente essere dimostrato per induzione sulla complessità di φ. Abbiamo quindi una funzione

$$f_\varphi(S) = \{(x, y) \mid \varphi(S, x, y)\}$$

che è monotona crescente, nel senso che $S \subset S' \Rightarrow f_\varphi(S) \subseteq f_\varphi(S')$.

Teorema 3.2 (Tarski-Knaster)**.** Se $f\colon \mathscr{P}(A) \to \mathscr{P}(A)$ è crescente, allora ha un minimo punto fisso S, cioè tale che $f(S) = S$ e S è minimo[17].

Dimostrazione. Definiamo $f^0(x) = x$, $f^1(x) = f(x)$, $f^{n+1}(x) = f(f^n(x))$. Il minimo punto fisso di f è $\bigcup_{n\in\omega} f^n(\emptyset)$. Le ipotesi fanno funzionare le ovvie verifiche; l'unica cosa da notare è che esiste k tale che $f^k(\emptyset) = f^{k+1}(\emptyset)$ perché stiamo assumendo che il dominio sia finito, e quindi $k \leq \{A\}$. □

[16] Ordinando le relazioni per inclusione.

[17] Non solo minimale: è proprio incluso in ogni punto fisso.

Svarione 3.3. Se la struttura fosse infinita l'unione va presa su tutti gli ordinali. Chiaramente per λ limite $f^\lambda(x) = \bigcup_{\alpha<\lambda} f^\alpha(x)$. Chiaramente l'unione è "su tutti gli ordinali" per finta, nel senso che anche qui per ragioni di cardinalità ci si deve stabilizzare. Notiamo che non richiediamo la continuità; se ce l'avessimo potremmo fermarci al passo ω.

Osservazione 3.4. Nel caso finito, se il singolo passo è calcolabile in tempo polinomiale, il punto fisso è calcolabile in tempo polinomiale; se partiamo da $\mathscr{P}(A^2)\ldots$

Tornando al caso che ci interessa, f_φ è crescente, quindi ha un minimo punto fisso $S = f_\varphi(S) = \text{LFP}_{S,x,y}\,\varphi(S,x,y)$ che coincide con R^*. La scrittura S, x, y a pedice serve a indicare quell'oggetto è "una specie di quantificatore", nel senso che φ dipendeva da S, x, y, ma $\text{LFP}_{S,x,y}\,\varphi(S,x,y)$ non più (chiaramente se contenva altre variabili libere queste restano tali). Questo ci pemette di creare nuovi simboli di predicato; sempre nel nostro esempio

$$R^*(a,b) \Leftrightarrow \big[\text{LFP}_{S,x,y}\,\varphi(S,x,y)\big](a,b)$$

Ora possiamo definire precisamente

Definizione 3.5. FO(LFP) è la chiusura delle formule atomiche per $\wedge, \vee, \neg, \forall, \exists, \text{LFP}$.

Proposizione 3.6. FO(LFP) $\subset$ SO.

Nel senso che se $\varphi \in$ FO(LFP) esiste $\psi \in$ SO tale che φ e ψ hanno gli stessi modelli *finiti*.

Teorema. Se K è una classe di L-strutture finite e $K = \text{Mod}(\varphi)$, con $\varphi \in$ FO(LFP), allora esiste un algoritmo PTIME che data una L-struttura $\mathcal{A}$ stabilisce se $A \in K$.

La dimostrazione per ora non la vediamo, l'idea è che il minimo punto fisso è calcolabile in tempo polinomiale. FO(LFP) è effettivamente più espressiva di FO: un esempio di classe descrivibile con una formula che usa il minimo punto fisso ma *non* con una semplice formula del prim'ordine è quella dei grafi connessi. È anche più espressiva di FO(TC), cioè della logica del prim'ordine più l'operatore di chiusura transitiva. Tuttavia

Teorema 3.7. Per strutture ordinate con BIT[18] vale FO(LFP) = PTIME.

Come già detto l'inclusione $\subseteq$ vale sempre. Invece la $\supseteq$ vale solo per certi linguaggi tali che $L \supseteq \{\leq, \text{BIT}\}$. Invece NP = ESO vale sempre.

[18] Definizione poi.

3.2 Giochi di Ehrenfeucht-Fraïssé

Siano $\mathcal{A}$, $\mathcal{B}$ due L-strutture[19]. Un gioco di Ehrenfeucht-Fraïssé $G_k(\mathcal{A}, \mathcal{B})$ di durata $k \in \mathbb{N}$ è definito come segue. Ci sono due giocatori $\forall$belardo ed $\exists$loisa[20]. Pensiamo ad $\mathcal{A}$ e $\mathcal{B}$ come a due scacchiere. Ad ogni mossa $\forall$ sceglie una delle due scacchiere e ci sceglie anche un punto, e poi $\exists$ sceglie un punto sull'*altra* scacchiera. Rimarchiamo che $\forall$ a ogni mossa può scegliere dove giocare, mentre, $\exists$ è vincolato a rispondere sulla scacchiera (struttura) opposta. Il turno consiste delle mosse di entrambi i giocatori, e il gioco finisce dopo k turni, e saranno stati scelti $a_1, \ldots, a_k \in A$ e $b_1, \ldots, b_k \in B$, dove al turno i sono stati scelti a_i e b_i (indipendentemente da chi ha scelto cosa). Vince $\exists$ se per ogni formula $\varphi(x_1, \ldots, x_k)$ senza quantificatori vale

$$\mathcal{A} \vDash \varphi(a_1, \ldots, a_k) \Leftrightarrow \mathcal{B} \vDash \varphi(b_1, \ldots, b_k)$$

Per chi ha fatto Teoria dei Modelli, questo è equivalente a dire che $a_i \mapsto b_i$ è un isomorfismo fra le sottostrutture generate $\langle a_1, \ldots, a_n \rangle^{\mathcal{A}}$ e $\langle b_1, \ldots, b_k \rangle^{\mathcal{B}}$. Ad esempio in $G_3((\mathbb{Q}, <), (\mathbb{Z}, <))$ vince il $\forall$, mentre in $G_3((\mathbb{Q}, <), (\mathbb{R}, <))$ vince $\exists$. Nel primo caso basta che $\forall$ scelga prima $0 \in \mathbb{Z}$, poi $1 \in \mathbb{Z}$ e poi, se $\exists$ ha scelto $a, b \in \mathbb{Q}$, scelga $\frac{a+b}{2} \in \mathbb{Q}$. Ci deve essere dunque una formula con 3 quantificatori che distingue le due strutture: in questo caso

$$\mathbb{Q} \vDash \forall a, b\ \exists c\ (a < c \wedge c < b)$$

mentre in $\mathbb{Z}$ è chiaramente falsa. In realtà quello che conta non è il numero di quantificatori ma

Definizione 3.8. Il *rango* di quantificazione è definito induttivamente come

- $\mathrm{rk}(\varphi \wedge \psi) = \max(\mathrm{rk}\,\varphi, \mathrm{rk}\,\psi)$
- $\mathrm{rk}(\varphi \vee \psi) = \max(\mathrm{rk}\,\varphi, \mathrm{rk}\,\psi)$
- $\mathrm{rk}(\neg\varphi) = \mathrm{rk}(\varphi)$
- $\mathrm{rk}(\forall x\ \varphi) = 1 + \mathrm{rk}(\varphi)$
- $\mathrm{rk}(\exists x\ \varphi) = 1 + \mathrm{rk}(\varphi)$
- $\mathrm{rk}(\text{atomiche}) = 0$

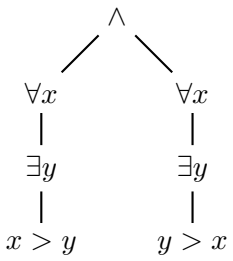
Se invece del massimo avessimo messo la somma sarebbe stato esattamente il numero di quantificatori. Il rango conta il massimo numero di quantificatori *annidati*, non *paralleli*. Ad esempio il rango di

$$\forall x\ \exists y\ (x < y) \wedge \forall x\ \exists y\ (y < x)$$

è 2 e non 4. È il massimo numero di quantificatori presenti in un ramo del parsing tree della formula:

[19] Notazione quasi standard: A è il dominio di $\mathcal{A}$.

[20] Con tanti nomi che iniziano per A ed E...



Definizione 3.9. Se $\mathcal{A}$, $\mathcal{B}$ sono L-strutture, $a_1, \ldots, a_\ell \in A$, $b_1, \ldots, b_\ell \in B$, definiamo la k-equivalenza secondo Ehrenfeucht-Fraïssé $\mathcal{A}, a_1, \ldots, a_\ell \overset{\text{EF}}{\sim}_k \mathcal{B}, b_1, \ldots, b_\ell$ se $\exists$ ha una strategia per resistere k mosse, ovvero $\exists$ ha una strategia vincente per il gioco

$$G_k((\mathcal{A}, a_1, \ldots, a_\ell), (\mathcal{B}, b_1, \ldots, b_\ell))$$

Spoiler 3.10. C'è anche una variante del gioco con infinite mosse ma fissate pedine che però possono essere spostate quando finiscono di essere piazzate. Vedremo più avanti.

Definizione 3.11. $\mathcal{A}, a_1, \ldots, a_\ell \equiv_k \mathcal{B}, b_1, \ldots, b_\ell$ se per ogni formula $\varphi(x_1, \ldots, x_\ell) \in \mathsf{FO}$ di rango di quantificazione $\leq k$ si ha $\mathcal{A} \vDash \varphi(a_1, \ldots, a_\ell) \Leftrightarrow \mathcal{B} \vDash \varphi(b_1, \ldots, b_\ell)$.

Teorema 3.12 (Ehrenfeucht-Fraïssé)**.** Se L contiene solo simboli di relazione, allora $(\mathcal{A}, \vec{a}) \overset{\text{EF}}{\sim}_k (\mathcal{B}, \vec{b}) \Leftrightarrow (\mathcal{A}, \vec{a}) \equiv_k (\mathcal{B}, \vec{b})$.

Osservazione 3.13. $\mathcal{A} \equiv \mathcal{B}$ se e solo se per ogni k vale $\mathcal{A} \equiv_k \mathcal{B}$.

In realtà una direzione (quella "dai giochi alle formule") del Teorema vale sempre, anche per linguaggi non relazionali. In ogni caso questi giochi si generalizzano anche a linguaggi del second'ordine. Questo si può usare, ad esempio, per mostrare che la 3-colorabilità non è di complessità FO. Basta mostrare che ci sono due grafi, uno 3-colorabile l'altro no, elementarmente equivalenti, e per fare questo si può usare un gioco di Ehrenfeucht-Fraïssé.

Notazione 3.14. $\mathcal{A} \equiv_\varphi \mathcal{B}$ se $A \vDash \varphi \Leftrightarrow B \vDash \varphi$.

Osservazione 3.15. Se $\mathcal{A} \equiv_\varphi \mathcal{B}$ e $\mathcal{A} \equiv_\varphi \mathcal{B}$, allora $\mathcal{A} \equiv_{\varphi \wedge \psi} \mathcal{B}$, $\mathcal{A} \equiv_{\varphi \vee \psi} \mathcal{B}$, $\mathcal{A} \equiv_{\neg\varphi} \mathcal{B}$. In particolare[21]

$$\mathcal{A} \equiv_{\text{atomiche}} \mathcal{B} \Leftrightarrow \mathcal{A} \equiv_{\text{QF}} \mathcal{B}$$

Dimostriamo ora il verso "facile" del Teorema, cioè che $(\mathcal{A}, \vec{a}) \overset{\text{EF}}{\sim}_k (\mathcal{B}, \vec{b}) \Rightarrow (\mathcal{A}, \vec{a}) \equiv_k (\mathcal{B}, \vec{b})$. Non staremo a insistere troppo su dettagli ovvi[22].

[21] QF sta per quantifier-free, si intende formule senza quantificatori.

[22] "Tipo che se due cose sono 10-equivalenti sono anche 5 equivalenti. Se uno non ci pensa non sbaglia."

Dimostrazione del Teorema 3.12, prima parte. Se $k = 0$ è ovvio dalle definizioni. Induttivamente sia $\varphi(x_1, \ldots, x_\ell)$ di rango $k+1$. Vogliamo

$$(\mathcal{A}, a_1, \ldots, a_\ell) \equiv_{\varphi(x_1,\ldots,x_\ell)} (\mathcal{B}, b_1, \ldots, b_\ell)$$

possiamo assumere WLOG[23] che $\varphi(x_1, \ldots, x_\ell) \equiv \exists y\ \theta(x_1, \ldots, x_\ell, y)$, con $\mathrm{rk}(\theta) \leq k$. La dimostrazione è routine: supponiamo $\mathcal{A} \vDash \exists y\ \theta(\vec{a}, y)$, prendiamo un testimone a' tale che $\mathcal{A} \vDash \theta(\vec{a}, a')$ e diciamo che $\forall$ gioca $a' \in A$. Chiaramente $\mathcal{A}\vec{a} \overset{\text{EF}}{\sim}_{k+1} \mathcal{B}\vec{b}$ se e solo se

$$\forall a' \in A\ \exists b' \in B\ \mathcal{A}\vec{a}a' \overset{\text{EF}}{\sim}_k \mathcal{B}\vec{b}b' \qquad \text{e} \qquad \forall b' \in B\ \exists a' \in A\ \mathcal{B}\vec{b}b' \overset{\text{EF}}{\sim}_k \mathcal{A}\vec{a}a'$$

cioè se e solo per per ogni mossa di $\forall$ c'è una mossa di $\exists$ che gli permette di resistere altre k mosse. Dicevamo, $\forall$ gioca $a' \in A$. Per ipotesi deve esistere $b' \in B$ tale che $\mathcal{A}, \vec{a}, a' \overset{\text{EF}}{\sim}_k \mathcal{B}, \vec{b}, b'$. Per ipotesi induttiva da $\mathcal{A} \vDash \theta(\vec{a}, a')$ si ha $\mathcal{B} \vDash \theta(\vec{b}, b')$, e quindi $\mathcal{B} \vDash \exists y\ \theta(\vec{b}, y)$. □

Per l'altro verso avremo bisogno del fatto —non completamente ovvio— che a meno di equivalenza esiste solo un numero finito di formule di rango k (se L è finito e relazionale, come nelle ipotesi).

4 06/10

4.1 Ancora sui Giochi di Ehrenfeucht-Fraïssé

Senza parlare di giochi si può definire $\overset{\text{EF}}{\sim}_k$ come

Definizione 4.1. $\mathcal{A}, a_1, \ldots, a_n \overset{\text{EF}}{\sim}_0 \mathcal{B}, b_1, \ldots b_n$ sse per ogni $\varphi(x_1, \ldots, x_n)$ atomica si ha $\mathcal{A} \vDash \varphi(a_1, \ldots, a_n) \Leftrightarrow \mathcal{B} \vDash \varphi(b_1, \ldots, b_n)$, mentre $\mathcal{A}, a_1, \ldots, a_n \overset{\text{EF}}{\sim}_{k+1} \mathcal{B}, b_1, \ldots b_n$ sse per ogni $a' \in A$ esiste $b' \in B$ tale che $\mathcal{A}, a_1, \ldots, a_n, a' \overset{\text{EF}}{\sim}_k \mathcal{B}, b_1, \ldots, b_n, b'$ e per ogni $b' \in B$ esiste $a' \in A$ tale che $\mathcal{A}, a_1, \ldots, a_n, a' \overset{\text{EF}}{\sim}_k \mathcal{B}, b_1, \ldots, b_n, b'$.

Avevamo già mostrato che $\overset{\text{EF}}{\sim}_k \Rightarrow \equiv_k$. Oggi mostriamo il viceversa, assumendo L finito e con soli simboli di relazione.

Definizione 4.2. Se $\mathcal{A}, a_1, \ldots, a_n \overset{\text{EF}}{\sim}_k \mathcal{B}, b_1, \ldots, b_n$ diciamo che $(\mathcal{A}, a_1, \ldots, a_n)$ e $(\mathcal{B}, b_1, \ldots, b_n)$ appartengono alla stessa $\overset{\text{EF}}{\sim}_k$-classe di equivalenza di n-uple.

Lemma 4.3. Se L è finito e relazionale e $C(n, k+1)$ è l'insieme di delle classi di $\overset{\text{EF}}{\sim}_{k+1}$-equivalenza di n-uple si ha $|C(n, k+1)| \leq 2^{C(n+1,k)}$. Inoltre per ogni classe $H = [\mathcal{A}, a_1, \ldots, a_n] \in C(n, k)$ esiste una formula $\varphi_H(x_1, \ldots, x_n)$ di rango di quantificazione $\leq k$ che caratterizza la classe, nel senso che

1. Se $H = [\mathcal{A}, a_1, \ldots, a_n]$, allora $\mathcal{A} \vDash \varphi(a_1, \ldots, a_n)$

[23] Fatto questo, negli altri casi è ovvio.

2. Se $\mathcal{B} \vDash \varphi(b_1, \ldots, b_n)$, allora $\mathcal{B} \in H$.

Osservazione 4.4. $\mathcal{A}$ e $\mathcal{B}$ possono anche essere infinite, ma comunque la stima non dipende dalla loro cardinalità.

Esempio 4.5. Sia $L = \{<\}$. Per $\operatorname{rk}\varphi(x, y) = 0$ le uniche possibilità per gli elementi in $C(2, 0)$ sono $x < y$, $y < x$, $x = y$ e le loro negazioni[24] e combinazioni booleane e questo è facile da vedere mettendo φ in forma normale disgiuntiva. In $C(2, 1)$ ci possiamo avere anche cose come $\exists z\ (x < z \wedge z < y)$.

Il Lemma 4.3 permette di concludere la dimostrazione del Teorema di Ehrenfeucht-Fraïssé molto velocemente:

Dimostrazione del Teorema 3.12, seconda parte. Se $H = [\mathcal{A}, a_1, \ldots, a_n] \in C(n, k)$ vale $\mathcal{A} \vDash \varphi_H(a_1, \ldots, a_n)$, e dato che $\operatorname{rk}(\varphi_H) \leq k$ si ha anche $\mathcal{B} \vDash \varphi_H(b_1, \ldots, b_n)$, per cui $\mathcal{B}, b_1, \ldots, b_n \in H$, che è la tesi. □

Concludiamo quindi con la

Dimostrazione del Lemma 4.3. Per induzione su k. Se $k = 0$ sappiamo che $\mathcal{A}, \vec{a} \overset{\text{EF}}{\sim}_0 \mathcal{B}, \vec{b}$ se e solo se le due strutture verificano le stesse formule atomiche. Dato che L non contiene simboli di funzione, di formule atomiche ψ_i ce ne sono solo una quantità finita r. Congiungendo quelle vere per $\mathcal{A}$ e la negazione di quelle false per $\mathcal{A}$ otteniamo la nostra φ_H, cioè

$$\varphi_H(\bar{x}) \equiv \bigwedge_{i:\mathcal{A}\vDash\psi_i(\vec{a})} \psi_i(\vec{x}) \wedge \bigwedge_{j:\mathcal{A}\vDash\neg\psi_j(\vec{a})} \neg\psi_j(\vec{x})$$

Notando che n è arbitrario facciamo il passo induttivo passando da $(n+1, k)$ a $(n, k+1)$. Per ogni $H \in C(n+1, k)$, che sappiamo essere finito, esiste per ipotesi induttiva $\varphi_H(x_1, \ldots, x_{n+1})$ che la caratterizza. Le formule che caratterizzano la $k+1$-equivalenza sono fatte così: per ogni $\delta \subseteq C(n+1, k)$ definiamo[25]

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

Scartando le φ_δ incoerenti, quelle che rimangono caratterizzano le classi in $C(n, k+1)$ (e la stima sulla cardinalità è ovvia). Infatti, fissata $(\mathcal{A}, a_1, \ldots, a_n)$, definiamo $\delta \subset C(n+1, k)$ in questa maniera:

- Se $H \in C(n+1, x)$ e $\mathcal{A} \vDash \exists y\ \varphi_H(a_1, \ldots, a_n, y)$ allora $H \in \delta$
- Altrimenti, $H \notin \delta$

Per costruzione $\mathcal{A}, a_1, \ldots, a_n \vDash \varphi_\delta(x_1, \ldots, x_n)$[26]. Viceversa mostriamo che, se $\mathcal{B} \vDash \varphi_\delta(b_1, \ldots, b_n)$, allora $\mathcal{B}, b_1, \ldots, b_n \overset{\text{EF}}{\sim}_{k+1} \mathcal{A}, a_1, \ldots, a_n$. Sia[27] $a' \in A$

[24] Non richiedendo che $<$ sia interpretato come ordine le negazioni ci vanno messe.

[25] Quantificare solo x_{n+1} non è restrittivo, è più facile capirlo che scriverlo.

[26] Con questa grafia intendiamo $\mathcal{A} \vDash \varphi_\delta(a_1, \ldots, a_n)$.

[27] L'altra metà è simmetrica.

e $H = [\mathcal{A}, a_1, \ldots, a_n, a'] \in C(n+1, k)$. Da $\mathcal{A} \vDash \varphi_H(a_1, \ldots, a_n, a')$ segue $\mathcal{A} \vDash \exists y\ \varphi_H(a_1, \ldots, a_n, y)$, per cui $H \in \delta$. Per ipotesi $\mathcal{B} \vDash \varphi_\delta(b_1, \ldots, b_n)$, e quindi $\mathcal{B} \vDash \exists y\ \varphi_H(b_1, \ldots, b_n, y)$, per cui esiste un b' che lo testimonia. Per ipotesi induttiva $\mathcal{B}, b_1, \ldots, b_n, b' \overset{\text{EF}}{\sim}_k \mathcal{A}, a_1, \ldots, a_n, a'$. □

Svarione 4.6. Chiedere che $\exists$ abbia una strategia vincente anche senza fissare in anticipo la lunghezza del gioco, la cosiddetta $\overset{\text{EF}}{\sim}_\infty$ equivalenza, vuol dire che fra le strutture esiste un sistema di back-and-forth. Nel corso di Teoria dei Modelli abbiamo usato questa, e non la $\overset{\text{EF}}{\sim}_\omega$-equivalenza, perché tanto potevamo permetterci di usare modelli saturi...

Esempio 4.7. $\mathbb{Z} \overset{\text{EF}}{\not\sim}_\infty \mathbb{Z} \oplus \mathbb{Z}$, ma $\mathbb{Z} \overset{\text{EF}}{\sim}_\omega \mathbb{Z} \oplus \mathbb{Z}$.

Teorema 4.8. Se $(A, \leq)$, $(B, \leq)$ sono ordini lineari di cardinalità $\geq 2^n$, allora $(A, \leq) \overset{\text{EF}}{\sim}_n (B, \leq)$.

Dimostrazione. Basta mostrare che $\exists$ può resistere n mosse. Dopo k passi abbiamo $\mathcal{A}, a_1, \ldots, a_k$ e $\mathcal{B}, b_1, \ldots, b_k$, e supponiamo induttivamente che $\exists$ riesca a fare in modo che:

1. $a_i < a_j \Leftrightarrow b_i < b_j$ e $a_i = a_j \Leftrightarrow b_i = b_j$
2. $\operatorname{dist}(a_i, a_j) = \operatorname{dist}(b_i, b_j)$ a meno che siano entrambe $> 2^{n-k}$
3. in questo caso $\operatorname{dist}(a_i, a_j) > 2^{n-k} \Leftrightarrow \operatorname{dist}(b_i, b_j) > 2^{n-k}$
4. $a_i = \max / \min \Leftrightarrow b_i = \max / \min$[28]
5. Forse qualche altra condizione sulla distanza fra minimo e massimo, o fissarli all'inizio
6. Il tutto sopra è suscettibile di off-by-one o altri errorini simili, ma s'aggiusta

È banale (ma noioso) verificare che se $\exists$ riesce a soddisfare le precedenti per k mosse riesce a farlo anche per $k+1$, e questo conclude. □

Questo mostra che[29]

Corollario 4.9. $\mathsf{EVEN} \notin \mathsf{FO}$.

Dimostrazione. Se esistesse una tale formula, sia r il suo rango. Basta prendere un ordine lineare di cardinalità 2^r e uno di cardinalità $2^r + 1$. □

[28] Questa potrebbe essere ridondante.

[29] Non è chiarissimo in che senso, sicuramente funziona se intendiamo che non esiste nessuna formula del prim'ordine nel linguaggio vuoto tale che... Tra l'altro per mostrare questo non c'è nemmeno bisogno di scomodare gli ordini, si può fare direttamente il gioco col linguaggio vuoto. Se invece si intende che il linguaggio lo si può scegliere basta fare la stessa cosa fra due strutture un cui tutte le relazioni sono interpretate col vuoto, entrambe col dominio grande almeno quanto il rango di quantificazione della formula, uno pari e uno dispari.

4.2 Qualche "Digressione"

Osservazione 4.10. Nel corollario precedente non possiamo assumere ipotesi sull'interpretazione del linguaggio: se avessimo una relazione PLUS definita come $\text{PLUS}(i,j,k) \Leftrightarrow i+j=k$, che ha senso quando c'è il $\leq$ nel linguaggio ed è interpretato come un ordine totale a patto di identificare coi naturali, è facile scrivere una formula vera solo per le strutture di cardinalità pari.

Abbiamo già anticipato che $\mathsf{NP} = \mathsf{ESO} = \Sigma^1_1$. Quello che vedremo è che, definendo un TIMES come il PLUS di sopra ma per la moltiplicazione, si ha

$$\mathsf{P} = \mathsf{FO}(\leq, \text{PLUS}, \text{TIMES})(\mathsf{LFP})$$

Avevamo anticipato qualcosa del genere con un certo BIT, che sarebbe una funzione che dice se l'i-esimo bit è 0 o 1, ma PLUS e TIMES sono più maneggevoli.

5 09/10

5.1 PebbleGames

Vediamo la variante dei giochi di Ehrenfeucht-Fraïssé con p pedine (per giocatore). Indicheremo l'equivalenza con

$$\mathcal{A}, a_1, \dots, a_n \overset{\text{EF}^p}{\sim}_k \mathcal{B}, b_1, \dots, b_n$$

Finché $n \leq p$ si gioca come prima, partendo con $n = 0$ (invece di "scegliere elementi" si "piazzano pedine"). Vediamo il caso $n = p$: il giocatore $\forall$ sceglie una delle due strutture e "muove una pedina", diciamo la i-esima, cioè cambia la scelta di uno degli elementi, e poi il giocatore $\exists$ muove l'i-esima pedina nell'altra struttura. Diciamo che vale

$$\mathcal{A}, a_1, \dots, a_s \overset{\text{EF}^p}{\sim}_k \mathcal{B}, b_1, \dots, b_s$$

se per ogni scelta di $a_i \mapsto a$ (risp. $b_i \mapsto b$), con $i \leq p$, esiste b (risp. a) tale che

$$\mathcal{A}, a_1, \dots, \underbrace{\not{a}_i}_{\mapsto a}, \dots, a_s \overset{\text{EF}^p}{\sim}_{k-1} \mathcal{B}, b_1, \dots, \underbrace{\not{b}_i}_{\mapsto b}, \dots, b_s$$

e per $k = 0$ definiamo $\overset{\text{EF}^p}{\sim}_0 = \overset{\text{EF}}{\sim}_0$.

Esempio 5.1. Consideriamo il grafo fatto da due triangoli e quello fatto da un esagono[30]. Qui con due pedine $\exists$ vince sempre, a prescindere da k (facile).

[30] Entrambi non orientati e senza loop.

Le pedine corrispondono alle variabili; vuol dire che le strutture verificano le stesse formule con rango di quantificazione k e p variabili. Nell'Esempio precedente la formula che differenzia le due strutture è

$$\exists x, y, z\ (Exy \wedge Eyz \wedge Ezx)$$

Esercizio 5.2. Formalizzare e dimostrare il verso "facile" dell'equivalenza sopra: se $\exists$ resiste k mosse allora le due strutture verificano le stesse formule di rango di quantificazione $\leq k$ con $\leq p$ variabili legate[31].

Ricordiamo che pensiamo le stringhe binarie come strutture con dominio la lunghezza $n = \{0, 1, \ldots, n-1\}$, un predicato 1-ario (che dice se nel posto j c'è un 1) e l'ordine $\leq$ naturale sul dominio. Ad esempio 01001 va a finire nella struttura $(5, S, \leq)$, dove vale

$$\neg S(0), S(1), \neg S(2), \neg S(3), S(4)$$

Soluzione dell'Esercizio 2.11. La formula è

$$\exists x\ \big(S(x) \wedge (\exists y\ y > x \wedge S(y) \wedge \exists x\ x > y \wedge S(x))\big)$$

Notiamo che il rango di quantificazione è 3 anche se le variabili usate sono 2. Ne segue che $01001 \overset{\text{EF}}{\sim}_3^2 01101$. □

Esercizio 5.3. [molto difficile[32]] Provare a scrivere una formula che esprima la transitività di una relazione R e una per l'associatività del prodotto usando una sola R per la prima e un solo simbolo di prodotto per la seconda.

5.2 Macchine di Turing

Hanno uno o più nastri[33] divisi in caselle in cui si possono scrivere simboli di un alfabeto Σ (tipicamente $\{0, 1\}$). La macchina ha un insieme Q di *stati*, e le cose che può fare sono muoversi sul nastro e leggere o scrivere sulle caselle. Un programma è quindi una funzione

$$\delta \colon \Sigma \times Q \to \Sigma \times Q \times \{-1, 0, 1\}$$

in sostanza prende in input cosa leggo e in che stato sono e restituisce cosa scrivo, in che stato vado e se e dove muovo la testina. Si può anche prevedere un carattere speciale che sta per "vuoto". La cosa importante è saper

[31] Le variabili possono essere riutilizzate: in $\exists x\ \varphi(x) \wedge \exists x\ \psi(x)$ contiamo una sola variabile legata. È permesso anche ripetere variabili annidandole: vedi subito dopo

[32] "Non so se ci riuscirete mai, ve lo lancio come sfida."

[33] In genere se ne mettono almeno due, uno per l'input e uno per fare i conti. Comunque a meno di trucchi ne dovrebbe bastare 1; l'efficienza peggiora quadraticamente, vedi sul Papadimitriou. Se andiamo su tempo/spazi di calcolo sublineari la differenza diventa però importante, tipo potrebbe avere senso contare solo spazio usato per i conti, che magari è meno della lunghezza dell'input...

riconoscere l'output, ad esempio si potrebbe convenire di non poter mai scrivere il carattere vuoto (bianco), di cui la stringa all'inizio (prima dell'input) è piena, togliendolo quindi dall'alfabeto di scrittura. Si possono fare altre convenzioni sul nastro, tipo che è infinito da un lato o da entrambi, che è finito ma può essere creato all'occorrenza...

Esempio 5.4. Vediamo un programma δ che somma due numeri positivi scritti in base 1.

Abbiamo $\delta\colon \{0,1,\sqcup,\textvisiblespace\} \times Q \to \{0,1,\sqcup\} \times Q \times \{-1,0,1\}$. Gli stati sono, q_0, q_1 e q_2. Il primo funge da stato di partenza e da "sto leggendo il primo numero". Il progamma è

$$\begin{aligned} q_0, 1 &\mapsto q_0, 1, +1 \\ q_0, 0 &\mapsto q_1, 1, +1 \\ q_1, 1 &\mapsto q_1, 1, +1 \\ q_1, \textvisiblespace &\mapsto q_2, \sqcup, -1 \\ q_2, 1 &\mapsto q_{\text{fine}}, \sqcup, 0 \end{aligned}$$

q_1 vuol dire "sto leggendo il secondo numero". Abbiamo indicato con$\textvisiblespace$ il vuoto (quello che non può essere scritto), con $\sqcup$ un "vuoto falso" che può essere scritto e con q_{fine} uno stato di arresto. L'input va dato come i due numeri scritti in base 1 separati da uno 0 (provare a "eseguire" a mano il programma fa capire bene cosa fa).

Definizione 5.5. Data $f\colon \Sigma^* \to \Sigma^*$ diciamo $f \in \mathsf{TIME}(g)$ se esiste una macchina di Turing δ che per ogni input $x \in \Sigma^*$ tale che $|x| = n$ si ferma in tempo $\leq g(n)$ con output $f(x)$. Idem per $\mathsf{SPACE}(g)$ se contiamo il numero di caselle utilizzate invece del tempo[34]. Se c'è un nastro di input non conta nel calcolo dello spazio (ma si può solo leggere).

Cambiare il modello di calcolo potrebbe cambiare g, ma "poco", nel senso che tutti i modelli sensati si simulano a vicenda in tempo polinomiale. Ci sono anche classi di complessità miste, tipo $\mathsf{TIME}(n^2)$-$\mathsf{SPACE}(n)$.

5.3 Varianti delle Macchine di Turing

5.3.1 Macchine di Turing Non-Deterministiche

Vuol dire che $\delta\colon \Sigma \times Q \to \mathscr{P}(\Sigma \times Q \times \{-1,0,1\})$. Ad esempio da $1, q_0$ andiamo "contemporaneamente" in $0, q_1, +1$ e in $1, q_2, -1$.

Per parlare di calcolo non-deterministico bisogna parlare di *configurazioni*; senza dare la definizione precisa, ad esempio

$$001\langle 1, q_3\rangle 100$$

[34] Qui è importante che non si possa scrivere il vuoto "vero".

vuol dire

$$\llcorner\llcorner 0011100 \lrcorner$$

con la testina sul secondo 1 e la macchina in stato q_3. Tipo una "foto" della macchina in quel momento. Data una configurazione c_1 ha senso chiedersi se posso passare tramite δ a una configurazione c_2 (in un passo solo). Nel caso deterministico basta applicare δ, qui la situazione è più complessa. Ad esempio dato un problema booleano $L\colon \Sigma^* \to \{V, F\}$ (o, se ci piace di più, $L \subseteq \Sigma^*$) vorremmo dire cosa vuol dire $L \in \mathsf{NTIME}(g)$, cioè che esiste una macchina di Turing non-deterministica δ tale che per ogni fissato input questo sta in L se e solo se δ accetta x e lo fa in tempo $g(|x|)$. C'è da definire ancora "accetta x". Conviene parlare direttamente anche di

5.3.2 Macchine di Turing Alternanti

Diviamo gli stati Q in

$$Q = \underbrace{\underbrace{\{q_{\text{sì}}, q_{\text{no}}\} \cup Q_\exists}_{\text{non-deterministiche}} \cup Q_\forall}_{\text{alternanti}}$$

La funzione di transizione è

$$\delta\colon \Sigma \times (Q_\exists \times Q_\forall) \to \mathscr{P}(\Sigma \times Q \times \{-1, 0, 1\})$$

e diciamo $c \mapsto c'$ se esiste un elemento di $\delta(c)$ che porta c in c'.

Definizione 5.6. Sia c una configurazione. Se c è nello stato $q_{\text{sì}}$ è accettante, se è in q_{no} è rifiutante. Se c è in uno stato $q \in Q_\exists$ è accettante se $\exists c'\ c \mapsto c'$ e c' è accettante. Se c è in uno stato $q \in Q_\forall$ è accettante se esiste c' tale che che $c \mapsto c'$ e ogni tale c' è accettante.

Visto che ci sono un numero finito di possibilità, si denota anche $Q_\forall = Q_\wedge$ e $Q_\exists = Q_\vee$. Possiamo pensare di stare sparando tante "copie" della macchina e poi congiungere/disgiungere gli output. Il tempo va pensato in parallelo, come se un processore potesse creare altri processori, dunque per le alternanti va preso il massimo del tempo sui vari rami e per le non-deterministiche il minimo (+1).

Abbiamo quindi le classi

$$\mathsf{DTIME}(g) \subseteq \mathsf{NTIME}(g) \subseteq \mathsf{ATIME}(g)$$
$$\mathsf{DSPACE}(g) \subseteq \mathsf{NSPACE}(g) \subseteq \mathsf{ASPACE}(g)$$

Le prime e le ultime sono chiuse per complemento (risposta "no" al problema"), cioè $\mathsf{DTIME}(g) = \mathsf{coDTIME}(g)$ e $\mathsf{ATIME}(g) = \mathsf{coATIME}(g)$. Per le deterministiche basta complementare l'output. Per le alternanti il motivo è che basta cambiare gli stati da esistenziali in universali e viceversa (cambia il "cosa accettano") e complementare nella maniera giusta ($\neg\exists\neg \simeq \forall$).

Definizione 5.7. Una funzione f è *time-costruibile* se esiste un programma "orologio" che con input lungo n termina in $f(n)$ passi. Definizione analoga per *space-costruibile*[35].

Usando questo ci si può limitare senza troppa perdita di generalità a macchine i cui cammini si fermano tutti dopo lo stesso tempo. Così $\mathsf{ATIME}(g) = \mathsf{coATIME}(g)$ è abbastanza evidente. Alcuni teoremi importanti sono che

Teorema. $\mathsf{NPSPACE} = \mathsf{DPSPACE} = \mathsf{APTIME}$

La prima uguaglianza dovrebbe essere di Savitch, l'altra di Immerman. In altre parole l'alternanza è una buona definizione di calcolo parallelo.

6 13/10

6.1 Chiarimenti sulle Macchine Alternanti

Per evitare dubbi su come si calcola il tempo nelle macchine alternanti conveniamo che la macchina è dotata di un orologio (clock) e che ogni computazione con input lungo n si ferma entro $f(n)$ passi (per $\mathsf{ATIME}(f(n))$). Se passa troppo tempo la macchina rifiuta. Se f è time-costruibile l'orologio può essere implementato. Prima o poi dimostreremo i seguenti Teoremi:

Teorema. Se $t(n) \geq n$ allora $\bigcup_{k=1}^{\infty} \mathsf{ATIME}(t(n)^k) = \bigcup_{k=1}^{\infty} \mathsf{DSPACE}(t(n)^k)$. In particolare per $t(n) = n$ si ha $\mathsf{APTIME} = \mathsf{DPSPACE}$.

Svarione 6.1. La gerarchia polinomiale[36] è inclusa in APTIME.

Teorema. $\mathsf{ASPACE}(s(n)) = \bigcup_{k=1}^{\infty} \mathsf{DTIME}(k^{s(n)})$. In particolare $\mathsf{APSPACE} = \mathsf{EXP\text{-}DTIME}$ e $\mathsf{ASPACE}(\log n) = \mathsf{P}$.

Osservazione 6.2 (Ovvia)**.** $\mathsf{TIME}(f) \subseteq \mathsf{SPACE}(f)$.

Teorema. $\mathsf{NSPACE}(s(n)) \subseteq \mathsf{ATIME}(s(n)^2) \subseteq \mathsf{DSPACE}(s(n)^2)$

L'inclusione $\mathsf{NSPACE}(s(n)) \subseteq \mathsf{DSPACE}(s(n)^2)$ è di Savitch, il Teorema intero dovrebbe essere di Immerman.

	D	N	A
TIME	$k^{s(n)}$		$\circ$
SPACE	$\circ$	$\circ$	$s(n)$

[35] Altrove la definizione è "esiste una macchina di Turing che, con input una stringa di n uni, in tempo $O(f(n))$ scrive una stringa di $f(n)$ uni". Più avanti, forse implicitamente, useremo questa definizione.

[36] Non ancora definita, è una risposta a una domanda di uno studente. Comparirà più avanti.

6.2 Completezza

Definizione 6.3. Se S_1 e S_2 sono due problemi binari, $S_1 \leq_{\mathsf{P}} S_2$ se esiste $f\colon \Sigma^* \to \Sigma^*$ di classe PTIME tale che $\sigma \in S_1 \Leftrightarrow f(\sigma) \in S_2$. In questo caso f viene detta *riduzione polinomiale*.

Dunque $S_2 \in \mathsf{P} \Rightarrow S_1 \in \mathsf{P}$. In altre parole il problema S_2 è "più difficile" del problema S_1. Si può anche definire, ad esempio, $\leq_{\mathsf{FO}}$, chiedendo che f sia FO invece che PTIME. Vale $S_1 \leq_{\mathsf{FO}} S_2 \Rightarrow S_1 \leq_{\mathsf{P}} S_2$.

Definizione 6.4. Un problema $S \subset \Sigma^*$ è *NP-completo* (rispetto a $\leq_{\mathsf{P}}$) se $S \in \mathsf{NP}$ e per ogni $S' \in \mathsf{NP}$ si ha $S' \leq_{\mathsf{P}} S$.

Osservazione 6.5. Se S è NP-completo allora $S \in \mathsf{P} \Rightarrow \mathsf{NP} = \mathsf{P}$.

Esempio 6.6. Il problema della 3 colorabilità è NP-completo.

Vedremo "a mano" che un certo problema di nome SAT è NP-completo. Poi le dimostrazioni di NP-completezza si fanno essenzialmente riconducendo problemi a SAT o altri che è noto essere NP-completi.

Definizione 6.7. Dati L, L' linguaggi del prim'ordine, una *FO-query* è una funzione FO dalla classe delle L'-strutture a quella delle L-strutture.

Perché abbia senso chiedersi se una *FO-query* è calcolabile, polinomiale, FO, ecc. bisogna codificare le strutture (ordinate) con stringhe binarie. Le stringhe sono codificate in strutture con $([n], P^1)$ come già visto[37].

Esempio 6.8. Sia $L = \{E^2\}$. Si codifica[38] $\mathrm{bin}\colon (n, E^2) \mapsto (n^2, P^1)$ interpretando n^2 come prodotto cartesiano, cioè deve valere $E(x, y) \Leftrightarrow P(nx + y)$.

Esempio 6.9. $\mathrm{bin}\colon (n, E^2, R^3, c) \mapsto \mathrm{bin}(E^2)\,\mathrm{bin}(R^3)\,\mathrm{bin}(c)$. Si intende che le stringhe sono concatenate e che la lunghezza è nota (se il dominio è n il primo pezzo sarà lungo n^2, il secondo n^3, il terzo $\lceil \log_2(n) \rceil$. In questo caso $R(x, y, z) \Leftrightarrow P(n^2x + ny + z)$.

Esempio 6.10 (Somma di due numeri in binario)**.** Consideriamo la somma senza ultimo riporto, che prende in input due numeri A, B di lunghezza n e restituisce un numero S sempre di lunghezza n. Questa è una funzione $+\colon (n, A^1, B^1) \to (n, S^1)$.

È abbastanza ovvio che questo si fa in tempo polinomiale (addirittura lineare), ma è fastidioso da scrivere. Mostriamo la cosa più forte che si

[37] L'altra volta avevamo usato $n = \{0, \ldots, n-1\}$. Useremo anche $[n] = \{1, \ldots, n\} = n + 1 \setminus 1$.

[38] Qui usiamo n e non $[n]$ per eleganza della bigezione.

riesce a fare al primo ordine, che è meno ovvio ma più facile (da scrivere). L'x-esimo bit di S si ottiene dall'$\oplus$[39] più il "riporto" carry(x).

$$S(x) = A(x) \oplus B(x) \oplus \text{carry}(x)$$

dove il riporto può essere definito come

$$\text{carry}(x) \equiv (\exists y < x)\big(Ay \wedge By \wedge (\forall z : x < z < y)(Az \vee Bz)\big)$$

dunque $S(x) \in \mathsf{FO}$ con $L = \{A, B, \leq\}$, e dato che $\mathsf{FO} \subset \mathsf{PTIME}$ allora la somma è PTIME.

In generale una FO-query sarà del tipo

$$(n, R_1, \ldots, R_\ell, \leq) \mapsto (\underbrace{\text{Dominio}}_{n \times n \times \ldots \times n}, \text{Relazioni}, \leq_{\text{lex}})$$

Usando questi strumenti mostreremo che

Proposizione 6.11. SAT $\leq_{\mathsf{FO}}$ CLIQUE.

Iniziamo con le definizioni:

Definizione 6.12. SAT è l'insieme delle formule proposizionali in forma normale congiuntiva (CNF) *soddisfacibili*. Una formula $\varphi(x_1, \ldots, x_n)$ è soddisfacibile se esiste un'assegnazione $v\colon \{x_1, \ldots, x_n\} \to 2$ che renda la formula vera.

Osservazione 6.13. SAT $\in \mathsf{NP}$: un certificato è un'assegnazione...

Definizione 6.14. Le *clausole* di una formula in CNF sono gli oggetti congiunti. Gli oggetti disgiunti dentro le clausole (variabili proposizionali o negazioni di variabili proposizionali) si chiamano *letterali*.

2-SAT, la variante in cui ogni clausola ha al più due variabili, è polinomiale.

Definizione 6.15. CLIQUE è l'insieme delle coppie (G, k) dove $G = (V, E^2)$ è un grafo non orientato che contiene una k-cricca, cioè un sottografo completo su k vertici.

Vediamo come codificare le formule in CNF in strutture nell linguaggio $L = \{P^2, N^2\}$. Sia $\varphi = \bigwedge_{i=1}^{m} c_i$, con i c_i del tipo[40] $\bigvee_{j=1}^{k} \pm x_{i,j}$. Sia n il massimo fra il numero di clausole e il numero di variabili. Definiamo

$$A_\varphi = \{n, P, N\}$$

dove $P(c, v)$ se la variabile v compare non negata (positivamente) nella clausola c, e $N(c, v)$ vuol dire che v compare negata nella clausola c.

[39] Dove $\oplus$ denota lo XOR, la "o" esclusiva, $\varphi \oplus \psi \equiv \neg(\varphi \leftrightarrow \psi)$.

[40] Con $\pm x$ si intende x o $\neg x$, dove x è una variabile proposizionale.

Esempio 6.16. La formula $\varphi = c_1 \wedge c_2 \wedge c_3$, dove

$$c_1 = x \vee y \vee \neg z$$
$$c_2 = \neg x \vee y \vee z$$
$$c_3 = \neg x \vee \neg y \vee z$$

viene codificata in una struttura che conterrà $P(1,2)$, $N(2,1)$,...

CLIQUE non ha bisogno di codifica, dato che i grafi sono già strutture[41]. Prima della Proposizione 6.11 mostriamo che

Teorema 6.17. SAT $\leq_{\mathsf{P}}$ CLIQUE

Dimostrazione. Bisogna trovare $f\colon \varphi \to A_\varphi$ funzione PTIME tale che

$$A_\varphi = (n, P, N) \in \text{SAT} \Leftrightarrow \underbrace{f(A_\varphi)}_{(G,n)} \in (\text{CLIQUE}, n+1)$$

Se C sono le clausole e L i letterali di φ, il grafo $f(A_\varphi)$ ha come vertici $C \times L \cup \{w_0\}$ e $n+1$ come costante per il k. La relazione E è data da

$$E = \{((c_1,\ell_1),(c_2,\ell_2)) \mid c_1 \neq c_2 \wedge \ell_1 \neq \bar{\ell}_2\} \cup \{(w_0,(c,\ell)),((c,\ell),w_0) \mid \ell \text{ compare in } c\}$$

dove $\bar{x} = \neg x$ e $\overline{\neg x} = x$. È facile convincersi che funziona è che la traduzione è polinomiale[42]. Però, al solito, è fastidioso da scrivere. Questo è il motivo per cui mostreremo direttamente la Proposizione 6.11. □

7 16/10

7.1 Inclusioni fra Classi di Complessità

Ricordiamo brevemente alcune delle inclusioni tra le varie classe di complessità che abbiamo già visto: per funzioni time/space costruibili vale

$$\mathsf{DTIME}(t(n)) \subseteq \mathsf{NTIME}(t(n)) \subseteq \mathsf{ATIME}(t(n))$$
$$\mathsf{DSPACE}(s(n)) \subseteq \mathsf{NSPACE}(s(n)) \subseteq \mathsf{ASPACE}(s(n))$$

Una prima inclusione non banale è data dal seguente

Teorema 7.1. $\mathsf{ATIME}(t(n)) \subseteq \mathsf{DSPACE}(t(n))$

[41] È un po' fastidioso dire come codificare k, ma la questione è abbastanza irrilevante. Si può fare ad esempio con una costante che "punta" il vertice appropriato, o rappresentata in base 2.

[42] Non trascritto.

Prima di dimostrare detto teorema è opportuno fare un breve digressione per definire alcuni costrutti e ridare in maniera più dettagliata alcune definizioni.

Definizione 7.2. Dato un predicato P gli associamo due funzioni best_P e worse_P come quelle funzioni tali che valgano le seguenti coimplicazioni:

$$\exists x P(x, y) \leftrightarrow P(\text{best}_P(y), y)$$
$$\forall x P(x, y) \leftrightarrow P(\text{worse}_P(y), y) \ .$$

Piccola osservazione collaterale: è possibile definire worse_P in termini di best_P (e viceversa) semplicemente osservando che

$$\forall x P(x, y) \leftrightarrow \neg \exists x \neg P(x, y)$$

e quindi

$$P(\text{worse}_P(y), y) \leftrightarrow P(\text{best}_{\neg P}(y), y)$$

Queste funzioni ci serviranno nel seguito, per ora mettiamole da parte.

Definizione 7.3 (Configurazione)**.** Data una qualsivoglia macchina di Turing M (deterministica, non-deterministica, alternante) una configurazione è una tripla ordinata $(mem, state, pos) \in \Sigma^\omega \times Q \times \mathbb{N}$ dove

- *mem* rapprensenta il contenuto del nastro della macchina M in un dato momento di una computazione
- *state* è lo stato della macchina
- *pos* è una posizione della testina di M.

Definizione 7.4. Date due configurazioni c_1 e c_2 di una macchina di Turing M scriviamo $c_1 \vdash c_2$ se e solo se la macchina M ha una transizione che permette di passare dalla configurazione c_1 alla configurazione c_2. Se $c_1 \vdash c_2$ allora si dice che c_1 *riduce a* c_2.

Data una macchina di Turing è possibile costruire il cosidetto *grafo di computazione* o semplicemente *computazione*.

Definizione 7.5. Una computazione di una macchina di Turing M su input $w \in \Sigma^*$ è un grafo orientato i cui vertici sono configurazioni per la macchina M e la relazione (diretta) del grafo R è la restrizione della relazione $\vdash$ all'insieme dei vertici considerato.

Definizione 7.6 (Configurazione accettante)**.** Una configurazione c per una macchina M si dice accettante se e solo se:

- $c = (mem, q, p)$ e $q = q_{\text{sì}}$
- $c = (mem, q, p)$, q è uno stato $\forall$, esiste una configurazione c' tale che $c \vdash c'$ e per ogni configurazione c' tale che $c \vdash c'$ allora c' è accettante
- $c = (mem, q, p)$ con q stato di tipo $\exists$ ed esiste una configurazione c' accettante tale che $c \vdash c'$.

Osservazione 7.7. Una *configurazione* è accettante se e solo se una computazione che parta da quella configurazione termina in uno stato accettante, i.e. se la relativa macchina di Turing (diciamo alternate, così da includere anche gli altri casi), con input w, passa per quella configurazione allora la macchina termina restituendo *sì* come risposta per l'input w.

È possibile dare una caratterizzazione della proprietà di essere una configurazione accettante in termini delle funzioni best e worse.

Osservazione 7.8. Definiamo la relazione figlio come

$$\text{figlio}(c_2, c_1) \equiv c_1 \vdash c_2$$

allora una configurazione $c = (mem, q, pos)$ è accettante se e solo se

- q è lo stato $q_{\text{sì}}$, oppure
- q è uno stato di tipo $\exists$ e $\text{best}_{\text{figlio}}(c)$ è accettante, o anche
- q è uno stato di tipo $\forall$ che ammette almeno un *figlio* e $\text{worse}_{\text{figlio}}(c)$ accetta.

Il vantaggio di questa caratterizzazione è che essa permette di verificare se uno stato è accettante semplicemente controllando che un suo particolare discendente sia accettante, anziché dover guardare potenzialmente tutti i suoi discenti (per esempio nel caso degli stati $\forall$).

Siamo adesso pronti a dimostrare il nostro Teorema 7.1.

Teorema.

$$\mathsf{ATIME}(t(n)) \subseteq \mathsf{DSPACE}(t(n))$$

Dimostrazione. Sia $S \in \mathsf{ATIME}(t(n))$; vogliamo far vedere che $S \in \mathsf{DSPACE}(t(n))$. Sia A una macchina di Turing alternante che preso come input $w \in \Sigma^*$ termina dicendo se w appartiene o meno a S in tempo $t(n)$. Vogliamo costruire una macchina B deterministica che accetti o rifiuti stringhe per lo stesso insieme S usando uno spazio di memoria[43] che al massimo sia un multiplo di $t(n)$, ovvero che sia $O(t(n))$. La macchina B cercata vorrebbe[44] fare la seguente cosa:

[43] Numero di celle del nastro su cui la macchina scrive.
[44] Nel senso che così non funziona; vedi dopo.

- B prende in input la stringa w
- B costruisce nel nastro una rappresentazione del grafo della computazione della macchina A su input w, d'ora in avanti indicheremo questo grafo con G_w: questo lo può fare in tempo arbitrario ma lo può fare perché per ogni macchina alternante esiste una deterministica che è in grado di rappresentare la stessa computazione[45]. Dato che la macchina A termina in al più $t(n)$ passi sappiamo che il numero di configurazioni di un ramo della *computazione A con input w* sono in numero $O(t(n))$ e che lo spazio occupato da una configurazione è al più $O(t(n))$.

  Per semplificare il resto della dimostrazione supponiamo che il grafo G_w sia fatto in modo tale che da ogni configurazione ci siano due discendenti: i.e. per ogni c vertice ci siano esattamente due configurazioni c_1 e c_2 in G_w tali che $c \vdash c_1$ e $c \vdash c_2$ e quindi supponiamo di avere un ordinamento su tali figli (i.e. una biiezione con l'insieme $\{0, 1\}$). Questa semplificazione non mina alla generalità della dimostrazione perché è sempre possibile trasformare un grafo di computazione in un grafo di computazione di questo tipo rimanendo, a meno di costanti moltiplicative, nello stesso spazio. In generale per dire che una configurazione c' è l'i-esimo figlio di c in G_w scriveremo $c \vdash_i c'$.
- a questo punto la macchina B "visita" il grafo memorizzato per sapere se la macchina A accetta il dato input, ovvero se la configurazione iniziale è accettante.

Questo, in realtà, richiede troppo spazio: $O(2^{ct(n)})$. Dunque, per quanto a noi torni comodo pensare al grafo G_w, la nostra macchina deterministica B *non può* scriverselo per intero in spazio $O(t(n))$. Tuttavia questa difficoltà può essere aggirata: per la visita del grafo che stiamo per descrivere basta salvarsi solo le scelte alternanti fatte (quale dei due figli scegliere), cosa fattibile in spazio $O(t(n))$, e quando servono informazioni (ad esempio se il padre del nodo corrente era esistenziale o universale) ricalcolarsele; questo non è un problema perché abbiamo la stringa delle scelte e basta risimulare tutto su spazio di memoria a parte, di cui ne basta $O(t(n))$ perché tutta la computazione di A dura $O(t(n))$.

B userà delle coppie ordinate $\langle b, s\rangle$ dove $b = \langle b_1, \ldots, b_k\rangle$ è una stringa di elementi $b_i \in \{0, 1\}$ e $s \in \{\text{sì}, \text{no}, ?\}$.

La sequenza dei b_i *traccia* una configurazione, quindi vertice, nel grafo G_w, infatti ad ogni stringa b possiamo associare univocamente la configurazione c' ottenuta come estremo del cammino $\langle c_0, c_1, \ldots, c_n\rangle$ in G_w fatto così:

- c_0 è la configurazione iniziale di G_w

[45] Altrimenti la tesi Turing-Church sarebbe falsificata :P

- per ogni $i \geq 1$ il vertice c_i è il b_i-esimo figlio di c_i.

Si può vedere che questa associazione è biunivoca. L'algoritmo che funziona impiega dello spazio per scriversi la configurazione che sta attualmente visitando (fattibile con $O(t(n))$ celle di memoria) e la coppia $(\langle b_1, \ldots, b_r \rangle, s)$ (anche questo fattibile senza sforare lo spazio $O(t(n))$). La macchina B lavora come segue, dove con "cancellare b_r" si intende sostituire $\langle b_1, \ldots, b_r \rangle$ con $\langle b_1, \ldots, b_{r-1} \rangle$. Se non menzioniamo cosa succede ad s vuol dire che rimane invariato:

- Scrive $\langle [], ? \rangle$[46] e come c la configurazione iniziale di A su input w.
- inizia il seguente ciclo: legge la configurazione in cui si trova $c = (mem, q, pos)$ e la stringa $(\langle b_1, \ldots, b_r \rangle, s)$

  se c **è una configurazione finale (cioè non ha figli)** allora la macchina vede se la configurazione è accettante o meno, nel primo caso pone $s =$ sì, nel secondo $s =$ no. Dopodiché sostituisce c con suo padre[47] e cancella b_r. Altrimenti

  se $s =?$ $c \vdash_0 c'$ allora la macchina scrive $(\langle b_1, \ldots, b_r, 0 \rangle, ?)$ e sostituisce c con c'

  se $b_r = 0$ **e** $s =$ **sì** allora B si ricalcola il padre di c. Se questi era in uno stato esistenziale cancella b_r e sostituisce c con suo padre. Se invece era in uno stato universale pone $b_r = 1$ ed $s =?$ e sostituisce q con suo fratello.

  se $b_r = 0$**,** $s =$ **no** allora B fa il duale di quanto sopra: si ricalcola il padre di c; se questi era in uno stato universale cancella b_r e sostituisce c con suo padre. Se invece era in uno stato esistenziale pone $b_r = 1$ ed $s =?$ e sostituisce q con suo fratello.

  se $b_r = 1$ a B basta cancellare b_r.

- reitera il ciclo finché non arriva allo stato $([], \text{sì})$ o nello stato $([], \text{no})$.

Osserviamo che nella dimostrazione precedente abbiamo usato degli stati della forma

$$(\langle b_1, \ldots, b_n \rangle, s)$$

dove non abbiamo posto delle limitazioni sulla stringa $\langle b_1, \ldots, b_n \rangle$, questo potrebbe implicare che con questa costruzione potremmo aver costruito una macchina con infiniti stati, tuttavia è possibile dimostrare che ogni stringa ottenuta (dovendo rappresentare le mosse che bisogna compiere per muoversi dentro G_w per raggiungere uno stato terminale) non può avere lunghezza

[46] Qui [] denota la stringa vuota.
[47] Ricalcolandosi chi è grazie ai b_i.

maggiore di $t(n)$ (il tempo di una computazione). Quindi le nostre stringhe sono di lunghezza limitata e lo spazio impiegato da B è effettivamente $O(t(n))$, ergo il programma di sopra può essere davvero implementato da una macchina di Turing.

Abbiamo così ottenuto una macchina deterministica B che effettivamente lavora in spazio $t(n)$, e che accetta tutte e sole le stringhe in S.

□

In realtà usando la precedente dimostrazione con piccoli adattamenti[48] si può dimostrare che vale anche l'inclusione $\mathsf{ASPACE}(t(n)) \subseteq \mathsf{DTIME}(O(1)^{t(n)})$.

8 20/10

8.1 Precisazioni sull'Ultima Lezione

L'altra volta abbiamo visto $\mathsf{ATIME} \subseteq \mathsf{DSPACE}$. Questo rende "lineare" la tabella dell'altra volta. Per vedere questo supponevamo di avere una macchina A che lavora in $\mathsf{ATIME}(t(n))$ e su input w consideravamo il grafo $G_w = (V, \vdash)$ con vertici V le configurazioni e archi $c \vdash c'$ quando la macchina può passare dalla configurazione c a quella c'.

Definizione 8.1. Sia (V, E) un grafo. Definiamo (V, E_{alt}) come[49]

$$E_{\text{alt}}(x,y) \equiv \Big[(x=y) \vee \exists z \ \big[E(x,z) \wedge E_{\text{alt}}(z,y) \wedge [G_\forall(x) \to \forall z \ (E(x,z) \to E_{\text{alt}}(z,y))]\big]\Big]$$

Questa va ottenuta via minimo punto fisso. In altre parole si pone

$$\varphi(R,x,y) \equiv \Big[(x=y) \vee \exists z \ \big[E(x,z) \wedge R(z,y) \wedge [G_\forall(x) \to \forall z \ (E(x,z) \to R(z,y))]\big]\Big]$$

che è positiva in R e quindi induce una funzione monotona. Per il Teorema di Tarski-Knaster dunque è ben definito il suo minimo punto fisso $\text{LFP}_{R,x,y}(\varphi(R,x,y)) \equiv E_{\text{alt}}$.

Corollario 8.2. $E_{\text{alt}}(a,b) \in \mathsf{FO(LFP)}$

Abbiamo già anticipato che $\mathsf{FO(LFP)} \subseteq \mathsf{P}$. Purtroppo l'input che diamo a E_{alt} è esponenziale in w perché sarebbero tutte le configurazioni della macchina A... Come anticipato, essenzialmente la stessa dimostrazione prova che

Teorema 8.3. $\mathsf{ASPACE}(s(n)) \subseteq \mathsf{DTIME}(O(1)^{s(n)})$

[48] Che essenzialmente sono salvarsi tutto il grafo e visitarlo senza doversi ricalcolare il padre ogni volta.

[49] $G_\forall(x)$ vuol dire che x è un nodo universale.

Basta rifare lo stesso grafo e le opportune stime. Basta stimare la lunghezza dei percorsi e osservare che non serve visitare i nodi più di una volta. Alternativamente basta verificare se vale E_{alt} fra la configurazione iniziale e quella accettante, e per il Corollario sopra più il fatto che la dimensione del grafo è esponenziale in $s(n)$... Nota: alla macchina $B \in \mathsf{DTIME}(O(1)^{s(n)})$ non viene dato il grafo come input, le si dà la configurazione iniziale e poi se lo calcola.

8.2 Altre Inclusioni fra Classi di Complessità

In conclusione l'ordine della tabella vista prima è lineare e dall'ultimo punto (ASPACE) si può tornare al primo (DTIME) con un salto esponenziale[50]. Si può dire di più, cioè che tre classi intermedie sono più o meno uguali[51]:

Teorema 8.4. $\mathsf{NSPACE}(s(n)) \subseteq \mathsf{ATIME}(s(n)^2)$

Dimostrazione. Sia $A \in \mathsf{NSPACE}(s(n))$ e costruiamo $B \in \mathsf{ATIME}(s(n)^2)$ che la simula. Su input w il grafo $(V, E) = G_w^A$ ha $O(1)^{s(n)}$ vertici, dove $E(x, y)$ se e solo se A può andare in un passo dalla configurazione x alla configurazione y. Creiamo una subroutine $P(d, x, y)$, dove $d \in \mathbb{N}$ e x, y sono configurazioni, che risponde "sì" se e solo se esiste un cammino[52] lungo al più 2^d passi da x a y. La definiamo induttivamente; l'idea è "cercare di indovinare nondeterministicamente il punto di mezzo":

$$P(0, x, y) \equiv (x = y \vee x \overset{A}{\vdash} y)$$
$$P(d+1, x, y) \equiv \exists z[P(d, x, z) \wedge P(d, z, y)]$$

La macchina B ha in memoria $d+1, x, y$ ed è in uno stato esistenziale $q_P^\exists$ che dice "devo implementare la subroutine". Quello che fa qui B è scegliere nondeterministicamente z e scriverselo, cosa fattibile in $s(n)$ passi; poi si porta in uno stato $q'^\forall$ in cui sceglie uno fra $P(d, x, z)$ e $P(d, z, y)$, cioè sceglie un bit, se fa 0 si mette nella configurazione d, x, z con stato $q_P^\exists$, se fa 1 fa la stessa cosa con d, z, y e $q_P^\exists$. Si ferma quando $d = 0$, dove abbiamo il caso base che è fattibile in tempo lineare. Inizializza con x uguale alla configurazione iniziale su input w, y configurazione finale accettante e $d = s(n)$ e parte subito con d, x, y, q_P. Quanto tempo ci mette? Bisogna calcolare il tempo

[50] Vedremo fra poco che vale proprio l'uguaglianza.

[51] Qui ci sono ipotesi di sensatezza su $s(n)$ che sono state omesse: sicuramente che sia space-costruibile, e poi che sia almeno logaritmica; nel corso non è stato detto, ma è vero che una funzione space-costruibile che sia $o(\log(n))$ e definitivamente dominata da una costante. Per saperne di più cercare `sublogarithmic space constructible function` su internet.

[52] Il cammino è semplice, non alternante, tanto la macchina è non-deterministica e quindi ha nodi tutti esistenziali.

$T(d)$ che la subrountine ci mette su input $P(d, x, y)$, con x, y qualunque. Chiaramente $T(0) = O(1) \cdot s(n)$. Mentre per $T(d+1)$ ci mette il tempo per scrivere z, più un passo per scegliere un bit, più il passo ricorsivo $T(d)$, cioè

$$T(d+1) = s(n) + 1 + T(d)$$

la cosa complicata è scrivere z, che è lungo $s(n)$ e va fatto $O(s(n))$ volte[53], quindi a meno di costanti $T(s(n)) = s(n)^2$. □

Corollario 8.5. $\mathsf{NSPACE}(s(n)) \subseteq \mathsf{ATIME}(s(n)^2) \subseteq \mathsf{DSPACE}(s(n)^2)$

Dunque in termini di spazio il problema $\mathsf{P} = \mathsf{NP}$ non si pone:

Corollario 8.6. $\mathsf{NPSPACE} = \mathsf{DPSPACE}$.

Nota: con $\mathsf{DTIME}(O(1)^{s(n)})$ si intende $\bigcup_{c\in\mathbb{N}} \mathsf{DTIME}(c^{s(n)})$. Vediamo ora l'altra inclusione "col salto esponenziale":

Teorema 8.7. $\mathsf{DTIME}(O(1)^{s(n)}) \subseteq \mathsf{ASPACE}(s(n))$.

Dimostrazione. Al solito simuliamo una macchina A nella classe a sinistra con una B in quella a destra. Dato che A questa volta è deterministica il grafo G_w^A è lineare. Mettiamo il grafo[54] in una tabella di dimensioni $c^{s(n)} \times c^{s(n)}$, dove una coordinata p rappresena lo spazio e l'altra t il tempo. In posizione (p, t) ci scriviamo cosa c'è in posizione di memoria p al tempo t. Consideriamo la subroutine $C(t, p, a)$ che significa "al tempo t in posizione p c'è il simbolo a". Conveniamo che se al tempo t la testina è in posizione p allora nel simbolo presente in posizione (p, t) sia incorporato lo stato della macchina. Quindi in uno slot di memoria in un caso a sarà un elemento di $\Sigma \times Q$, e non di Σ, e negli altri un semplice elemento di Σ. Ora l'idea fondamentale per la dimostrazione è che $C(t+1, p, a)$ è determinato da $C(t, p', a')$, con $p' \in \{p-1, p, p+1\}$ (dipende da dov'era la testina al passo prima, che si può muovere solo di uno slot). Formalmente

$$C(t+1,p,a) \equiv \exists \underbrace{a_{-1}, a_0, a_1}_{\in \Sigma \cup (\Sigma\times Q)} \big[(a_{-1}, a_0, a_1) \overset{A}{\vdash} a \wedge \forall i \in \{-1, 0, 1\}\ C(t, p+i, a_i)\big]$$

Ora $(a_{-1}, a_0, a_1) \overset{A}{\vdash} a$ si può calcolare in tempo costante (basta vedere nelle istruzioni di macchina). Se B ha in memoria $t+1, p, a$ ed è nello stato $q_C^{\exists}$ in cui deve implementare C; sceglie a_{-1}, a_0, a_1 nondeterministicamente

[53] Infatti un percorso nel grafo è lungo al più $2^{O(s(n))}$: le configurazioni sono quelle che sono e il percorso può essere scelto in modo che non ripassi mai dalla stessa configurazione. Infatti visto che per ipotesi la computazione finisce in uno stato accettante/rifiutante, fissata una configurazione ci sarà un'ultima volta che la si visita. Basta cancellare tutti i cicli e fare subito la scelta che avremmo fatto l'ultima volta.

[54] Noi, non la macchina B, perché è troppo grosso.

e controlla $(a_{-1}, a_0, a_1) \overset{A}{\vdash} a$ in tempo costante. Poi con in memoria $t+1, p, a, a_{-1}, a_0, a_1, i$ si porta in uno stato $q'^{\forall}$ dove sceglie i e controlla $C(t, p+1, a_i)$, cioè si porta in $t, p+1, a_i, q_C$. Se $t = 0$ deve controllare che quella è la configurazione iniziale, cioè

$$C(0, p, a) \equiv a \text{ è il } p\text{-esimo simbolo dell'input } w, \text{ o } p \text{ è } 1 \text{ e } a = (q_1, w_1)$$

In memoria B deve tenere solo gli input delle subroutine, ma può ogni volta cancellare gli input precedenti; in altre parole la tabella di cui parlavamo prima non va tenuta tutta in memoria, ma solo un pezzetto per volta. Lo spazio necessario è quello per scrivere $t+1, p, a, a_{-1}, a_0, a_1, i$ che si stima con (sempre a meno di costanti)

$$\text{spazio} \leq s(n) + \log(s(n)) + \text{ costante}$$

e se $s(n)$ era almeno logaritmico abbiamo la tesi. □

9 23/10

9.1 Riassunto delle Puntate Precedenti

Rivediamo le inclusioni viste; etichettiamo con dei numeri le "idee" per le inclusioni non banali:

$$\begin{aligned} &\mathsf{DTIME}(t) \subseteq \mathsf{NTIME}(t) \subseteq \mathsf{ATIME}(t) \\ \overset{(1)}{\subseteq} &\mathsf{DSPACE}(t) \subseteq \underbrace{\mathsf{NSPACE}(t)}_{\overset{(2)}{\subseteq}\mathsf{ATIME}(t^2)} \subseteq \mathsf{ASPACE}(t) \\ &\underbrace{=}_{\substack{(1) \\ \subseteq}\quad\substack{(3) \\ \supseteq}} \mathsf{DTIME}(O(1)^t) \end{aligned}$$

dove stiamo supponendo[55] $(t) = (O(1) \cdot t + O(1))$. Poniamo[56] $\mathsf{L} = \mathsf{LOGSPACE} = \mathsf{DSPACE}(\log n)$, $\mathsf{NL} = \mathsf{NSPACE}(\log n)$ e $\mathsf{AL} = \mathsf{ASPACE}(\log n)$ e per i Teoremi visti abbiamo

$$\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{AL} = \mathsf{P}$$

Ricapitoliamo quali erano le idee usate

(1) Esiste un grafo G_w dove si può fare backtracking cui ci possiamo appoggiare per fare le stime.

[55] Questo si può prendere per definizione, o si può usare un Teorema che prende il nome di Linear Speed-Up, che vedremo in futuro.

[56] Con la convenzione di non contare nello spazio l'input, ad esempio di metterlo su un nastro separato.

(2) "Guess middle": usavamo il nondeterminismo per "indovinare" una tappa intermedia.

(3) "Località delle computazioni": passando dal tempo t al tempo $t+1$ può cambiare al più una cella di memoria, quindi fuori da una "palla" che contiene tre celle (a seconda di dove si sposta la testina) il nastro è sicuramente uguale a com'era al tempo prima.

9.2 Primo Ordine vs. Tempo Polinomiale

Teorema 9.1. $\mathsf{FO} \subseteq \mathsf{L}$, e in particolare $\mathsf{FO} \subseteq \mathsf{P}$.

Dimostrazione. Data $\varphi \in \mathsf{FO}$ vogliamo una macchina di Turing $T_\varphi \in \mathsf{L}$ che accetta le stesse strutture accettate da φ, cioè per ogni L-struttura[57] A

$$A \vDash \varphi \Leftrightarrow T_\varphi(\mathrm{bin}(A)) \downarrow$$

dove $\mathrm{bin}(A)$ è un'opportuna codifica di A, fatta ad esempio come visto le altre volte[58]. La lunghezza della codifica $\mathrm{bin}(A)$, se

$$A = (n, R^{a_1}, \dots, R^{a_k}, c_1, \dots, c_\ell)$$

sarà

$$|\mathrm{bin}(A)| = n^{a_1} + \ldots + n^{a_k} + \lceil \log n \rceil + \ldots + \lceil \log n \rceil$$

Qui stiamo assumendo che la macchina conosca L, e quindi si può ricavare n da $\mathrm{bin}(A)$ nella maniera ovvia. Poi può controllare se in una certa posizione il bit è acceso o no. La dimostrazione dell'esistenza di T_φ si fa per induzione su φ, e come al solito l'unico caso non banale sono i quantificatori; studiamo quindi la costruzione di $T_{\exists x \varphi(x)}$ supponendo induttivamente di avere $T_{\varphi(c)}$ (dove $\varphi \in L \sqcup \{c\}$) tale che, posto $B = (A, c^B)$,

$$T_{\varphi(c)}(\mathrm{bin}(A)\,\mathrm{bin}(c^B)) \downarrow \iff B \vDash \varphi(c)$$

Quello che fa $T_{\exists x \varphi(x)}$ è guardarsi tutti gli $x \in n = \{0, \dots, n-1\}$ e controllare se $T_{\varphi(c)}$ accetta $(\mathrm{bin}(A), x)$. Appena ne trova uno si ferma e accetta; se non ne trova e arriva in fondo rifiuta. Dato che $|x| \leq \lceil \log n \rceil$ e che lo spazio può essere riutilizzato, abbiamo la tesi. □

Chiaramente, data l'arbitrarietà di L, la tesi è in particolare vera quando contiene $\leq, +, \cdot$, che è il caso che ci interesserà.

Teorema 9.2. $\mathsf{FO(LFP)} \subseteq \mathsf{P}$.

[57] Finita, chiaramente.

[58] Qui c'è stato un rinfresco della memoria, non trascritto.

Dimostrazione. La strategia di dimostrazione è la stessa della dimostrazione precedente, e ci basta esaminare il caso

$$\varphi(\vec{a}) = \text{LFP}_{R,\vec{x}}(\varphi(R,\vec{x}))(\vec{a})$$

Dove la notazione è la stessa vista in passato[59]. In realtà questa dimostrazione l'abbiamo praticamente già fatta, e l'unica cosa da osservare è che il LFP si può calcolare in tempo polinomiale.

Supponendo per induzione di avere $T_{\varphi(R,c_1,c_2)}$, l'algoritmo per $T_{\text{LFP}_{R,x,y}(\varphi(R,x,y))(c_1,c_2)}$ ha input $\text{bin}(A)_L\,\text{bin}(a)\,\text{bin}(b)$, inizializza $R = \emptyset$ e scorre tutte le coppie (a,b), che sono n^2, facendo quanto segue:

- Controlla se $(A,R,a,b) \vDash \varphi(\underline{R},x,y)$ usando $T_{\varphi(R,c_1,c_2)(\text{bin}(A,R,a,b))}$ data dall'ipotesi induttiva. Se sì mette in R' il bit 1 in posizione $na+b$.
- Alla fine del ciclo abbiamo $\text{bin}(R')$ (che ha lunghezza n^2).
- Ripartiamo con R' al posto di R.

Dopo n^2 iterazioni abbiamo[60] il punto fisso[61]. □

Formalmente in FO(LFP) bisogna aggiungere alle regole di FO per la formazione di formule la regola

$$\frac{\varphi(R,\vec{x}) \in L \cup \{R\}}{\text{LFP}_{R,x,y}(\varphi(x,y))(a,b) \in L}$$

La prossima volta vedremo l'altra inclusione.

10 27/10

10.1 Caratterizzazione di P

Come promesso vediamo l'inclusione che ci manca. **Importante:** in quanto segue assumeremo che il linguaggio contenga $\leq$, PLUS, TIMES, (oppure $\leq$, BIT, sono mutualmente definibili), e che vengano interpretati come ci si aspetta, quindi in realtà invece di

Teorema 10.1. $\mathsf{P} \subseteq \mathsf{FO(LFP)}$.

[59] Data ad esempio $\varphi(\underline{R},x,y)$ positiva in $\underline{R}$ in $L \cup \{\underline{R}\}$, e una L-struttura A possiamo definire la funzione $R \mapsto R'$ dove per ogni $a,b \in A$ si ha $R'(a,b) \underset{\text{def}}{\equiv} (A,R) \vDash \varphi(\underline{R},a,b)$. Poi si applica Tarski-Knaster...

[60] —Scusi, ma qui perché [roba]?
—Perché [motivo], torna?
—Mi sto convincendo.
—Sì, queste sono dimostrazioni che si fanno "a maggioranza".

[61] Vedi dimostrazione di Tarski-Knaster.

Dovremmo scrivere qualcosa come $\mathsf{P} \subseteq \mathsf{FO(LFP)}_{\leq,\mathrm{BIT}}$. Per evitare di appesantire la notazione assumiamo quanto sopra implicitamente. La dimostrazione procede come segue:

1. $\mathsf{P} = \mathsf{ASPACE}(O(\log n)) = \mathsf{AL}$ (lo sappiamo già).
2. $\mathrm{Reach}_{\mathrm{alt}} \in \mathsf{FO(LFP)}$, dove $\mathrm{Reach}_{\mathrm{alt}}$ è decidere se in un grafo alternante si può andare da un nodo all'altro (già visto, anche se non gli avevamo dato un nome).
3. $\forall S \in \mathsf{AL}\ S \leq_{\mathsf{FO}} \mathrm{Reach}_{\mathrm{alt}}$. Qui serve che la riduzione sia FO; farla P non basta perché non sappiamo se $\mathsf{FO(LFP)}$ è chiusa per riduzioni polinomiali, mentre per riduzioni FO è ovvio.

Per sicurezza definiamo formalmente $\mathrm{Reach}_{\mathrm{alt}}$. Sia $G = (V, G^1_\exists, G^1_\forall, E^2, s, t)$, dove s e t sono costanti che rappresentano un *source* e un *target*, e gli altri simboli sono interpretati come suggerito dalla notazione. Poniamo $G \in \mathrm{Reach}_{\mathrm{alt}} \Leftrightarrow E_{\mathrm{alt}}(s,t)$, dove E_{alt} è la formula già scritta qualche lezione fa. Come esercizio ridimostriamo direttamente che $\mathrm{Reach}_{\mathrm{alt}} \in \mathsf{P}$, senza usare i "cannoni" visti ma esibendo direttamente un algoritmo polinomiale.

- `Input` $G = (V, G^1_\exists, G^1_\forall, E^2, s, t)$
- `Declare queue` Q
- `Make` $Q = \emptyset$; `Mark`(t); `insert` t `in` Q
- `while` $Q \neq \emptyset$ `do`
  - `remove1st` x `from` Q
  - `for each unmarked` $y : E(y,x)$ `do`
    - `delete` (y,x) `from` E
    - `if` $G_\exists(y) \vee y$ `has no outgoing edges then mark` (y); `insert` y `in` Q
- `if` s `is marked accept`

(segue esempio di esecuzione, non trascritto). È facile vedere che l'algoritmo è polinomiale[62].

Facciamo un esempio di riduzione FO. Construiamo una funzione da grafi ordinati non orientati ($E(x,y) \leftrightarrow E(y,x)$) a grafi ordinati, non orientati e puntati

$$I\colon \mathrm{Grafi}_\leq \to (\mathrm{Grafi}, 0, \max)_\leq$$

tale che G è connesso se e solo se

$$I((V,E,0,\max)) \in (\mathrm{Reach}, 0, \max)$$

[62] La stima stupida dovrebbe dire "quadratico", ma in realtà dovrebbe essere lineare.

e con Reach indichiamo essenzialmente la stessa cosa di $\text{Reach}_{\text{alt}}$ ma con in nodi tutti esistenziali (quindi è proprio vedere se esiste un cammino fra 0 e max. Esibiamo una $I \in \mathsf{FO}$ che lo fa.

Sia $G = (n, E^G)$ e poniamo $I(G) = (V, E, 0, \max)$, dove

- $V = (n \times n)$
- $E(x, y, x', y') \equiv \Big[(x = x' \wedge E^G(y, y')) \vee (\text{succ}(x, y) \wedge x' = y' = y)\Big]$, dove succ rappresenta il successore, che ha senso perché abbiamo un ordine discreto.

Questa riduzione si dice *binaria* perché mappa $n \mapsto n^2$. Abbiamo chiaramente $I \in \mathsf{FO}$. Il libro (Immerman) la denota come

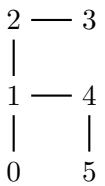
$$I = \lambda xyx'y'.\langle \texttt{true}, E(x, y, x', y'), \langle 0, 0\rangle, \langle \max, \max\rangle\rangle$$

dove quel `true` sarebbe una formula che dice quali coppie stanno nel dominio: nel nostro caso tutte. Abbiamo

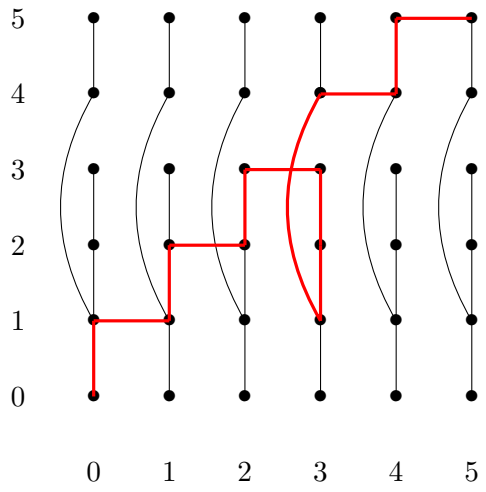
$$I\colon \text{bin}(G) \underset{\mathsf{PTIME}}{\mapsto} \text{bin}(I(G))$$

Infatti, dato che $E^G \in 2^{n \times n}$ la sua codifica è lunga n^2. La nuova E invece ha codifica lunga n^4. Come lunghezza dunque la trasformazione è polinomiale. Basta dunque scorrere tutto il dominio e calcolare una formula del prim'ordine per ogni coppia di punti, ma le formule del prim'ordine si calcolano in tempo polinomiale.

Esempio 10.2. Vediamo cosa succede al grafo



Succede questa cosa (in grassetto rosso un (l'unico) percorso)



Dunque Connesso $\leq_{\mathsf{FO}}$ Reach$_\exists$. Conlcudiamo ora mostrando che

Teorema 10.3. Se $S \in \mathsf{AL}$ allora $S \leq_{\mathsf{FO}} \text{Reach}_{\text{alt}}$.

Inizio della Dimostrazione. Sia M una macchina di Turing in $\mathsf{ASPACE}(c \cdot \log n)$. Sia poi $\mathcal{A}$ un σ-struttura, dove

$$\sigma = \{R_1^{a_1}, \ldots, R_r^{a_r}, c_1, \ldots, c_s\}$$

Vogliamo I tale che $M(\text{bin}(\mathcal{A})) \downarrow \Leftrightarrow I(\mathcal{A}) \in \text{Reach}_{\text{alt}}$. Costruiamo

$$I\colon \sigma\text{-strutture} \to \underbrace{\text{Grafi} = (V, G_\exists, G_\forall, E, s, t)}_{\tau\text{-strutture con } \tau=(G_\exists, G_\forall, E, s, t)}$$

Iniziamo a dire chi è V: se A è il dominio della σ-struttura argomento di I, $a = \max_{j=1}^{r} a_j$ e c è la c di $c \log n$, allora

$$V = A^{4+a+c}$$

L'idea è che $I(\mathcal{A})$ è il grafo delle computazioni di M con input bin$(\mathcal{A})$. La formula E dipende da M e da σ ma non da $\mathcal{A}$. Dato che si parla di spazio logaritmico la M ha un nastro di input in sola lettura e dei nastri di lavoro (che sono quelli rilevanti per il calcolo dello spazio), avremo $E(\text{ID}, \text{ID}')$ con $\text{ID} \overset{m}{\vdash} \text{ID}'$, dove

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

Fissata $\mathcal{A}$, ID varia su n^{4+a+c}. Vediamo il ruolo delle variabili. $\langle w_1, \ldots, w_c \rangle \in n^c$ codifica il contenuto del nastro di lavoro. Infatti n^c lo possiamo pensare come $2^{c \log n}$, e quindi come la codifica di una stringa binaria lunga $c \log n$, e quindi può codificare il contenuto del nastro di lavoro. Invece $\langle r_1, \ldots, r_a \rangle$, se facessimo le cose nella prima maniera che ci viene in mente, dovrebbe codificare il contenuto del nastro di input, che però è troppo grosso; quindi ci codifichiamo solo la posizione della testina sul nastro di input, senza dunque segnare cosa c'è scritto. [finiamo la prossima volta] □

11 30/10

La prossima settimana niente lezioni.

11.1 Reach-alt è AL-completo

Finiamo la dimostrazione lasciata in sospeso l'ultima volta. Ricordiamo che vogliamo mostrare che

Teorema. $\text{Reach}_{\text{alt}}$ è AL-completo rispetto a $\leq_{\mathsf{FO}}$.

Cioè $\text{Reach}_{\text{alt}} \in \mathsf{AL}$ e ogni problema[63] in AL si riduce a $\text{Reach}_{\text{alt}}$ (questa seconda condizione da sola si riassume dicendo che $\text{Reach}_{\text{alt}}$ è *P-hard*, ed è quello che ci manca da mostrare). Fatto questo potremo affermare a pieno titolo di aver dimostrato che $\mathsf{P} \subseteq \mathsf{FO(LFP)}$, perché allora per ogni $S \in \mathsf{P}$ abbiamo $S \leq_{\mathsf{FO}} \text{Reach}_{\text{alt}} \in \mathsf{FO(LFP)}$. Finiamo la dimostrazione dell'altra volta.

Dimostrazione del Teorema 10.3, seconda parte. La query I sarà una stringa di formule fatta così:

$$I = \langle \delta(\vec{x}), E(\vec{x}, \vec{y}), G_\forall(\vec{x}), G_\exists(\vec{x}), s, t \rangle$$

dove δ rappresenta il dominio. Basta prendere la formula sempre vera, cioè prendere come dominio dell'interpretazione tutto A^{4+a+c}. Invece di $p_1, \ldots, p_4$, che sono piccoli, uno potrebbe usare una sola variabile, ma non cambia nulla se non la (minore) chiarezza espositiva. Poniamo $|\vec{x}| = |\vec{y}| = 4 + a + c$, e se

$$\tau = \{E, G_\forall, G_\exists, s, t\}$$

$I(=I(\mathcal{A}))$ sarà una τ-struttura. Vogliamo

$$\forall \mathcal{A} \in \text{Struct}(\sigma)\ \mathcal{A} \in S \Leftrightarrow I(\mathcal{A}) \in \text{Reach}_{\text{alt}}$$

Questo per definizione vuol dire $S \overset{I}{\leq_{\mathsf{FO}}} \text{Reach}_{\text{alt}}$. Se

$$\mathcal{A} = (A, \underbrace{R_1^{\mathcal{A}}}_{\subset A^{a_1}}, \ldots)$$

allora $I(\mathcal{A})$ sarà

$$(A^k, E^{I(\mathcal{A})}, G_\forall^{I(\mathcal{A})}, \ldots)$$

dove

$$E^{I(\mathcal{A})} \subseteq A^k \times A^k \qquad I(\mathcal{A}) \vDash E(\vec{a}, \vec{b}) \Leftrightarrow \mathcal{A} \vDash E(a_1, \ldots, a_k, b_1, \ldots, b_k)$$

[63] Ricordiamo che per noi un problema è una classe di strutture in un certo linguaggio.

Notiamo che la E a sinistra del $\Leftrightarrow$ è un simbolo di τ, mentre quella a destra è una σ-formula, che definiremo fra poco. Come troviamo I? Come anticipato δ sarà la formula sempre vera, cioè il dominio di $I(\mathcal{A})$ è tutto $|A|^k$. Dato che $S \in \mathsf{AL} = \mathsf{ASPACE}(c \log n)$ c'è una macchina di Turing $M \in \mathsf{ASPACE}(c \log n)$ che riconosce S, cioè per ogni σ-struttura $\mathcal{A}$ abbiamo $\mathcal{A} \in S \Leftrightarrow M(\text{bin}(\mathcal{A})) \downarrow$. La I dipende da M, ma non da $\mathcal{A}$ perché M dipende solo dal problema S e non dalla singola istanza $\mathcal{A}$. L'idea è che $I(\mathcal{A})$ sia il grafo delle computazioni di M su input $\text{bin}(\mathcal{A})$. Quanto è lunga la codifica $\text{bin}(\mathcal{A})$? Se ad esempio $R \subseteq A^3$, allora $R \in 2^{A^3}$, la cui codifica binaria è lunga n^3. In definitiva la codifica di $\mathcal{A}$ sarà polinomiale in $|A| = n$, e a meno di moltiplicare c per quello che esce dal logaritmo (in sostanza il grado del polinomio che esprime la lunghezza della codifica) possiamo supporre che sul nastro di lavoro sia usato spazio $\leq c \log(n)$. Il nastro di input però è troppo lungo per essere scritto in $I(\mathcal{A})$ senza sforare il tempo polinomiale.

$E(\vec{x}, \vec{y})$ deve dire che possiamo passare dalla configurazione $\vec{x}$ alla $\vec{y}$ in un passo di M. Vediamo quante variabili ci servono(ID sta per "Instantaneous Description"):

$$\vec{x} = \text{ID} = \langle p_1, p_2, p_2, p_4, r_1, \ldots, r_a, w_1, \ldots, w_c \rangle$$

Dove[64][65]

$p_1 \in \{1, \ldots, r\}$	(quale relazione stiamo leggendo)
$p_2 \in Q = \text{Stati di } M = \{1, \ldots, \lvert Q \rvert\}$	
$p_3 \in \{0, \ldots, c \log(\max)\}$	(posizione sul nastro di lavoro)
$p_4 \in \{0, 1\}$	$= \{\forall, \exists\}$
$\langle w_1, \ldots, w_c \rangle \in \max^c = 2^{c \log n}$	(contenuto del nastro di lavoro)
$\langle r_1, \ldots, r_a \rangle \in \max^a$	(posizione sul p_1-esimo pezzo del nastro di input)

per i w_i intendiamo che scelti $w_i \in \{0, \ldots, n-1\}$, sarà

$$\langle w_1, \ldots, w_c \rangle = w_1 + nw_2 + n^2 w_3 + \ldots + n^{c-1} w_c < n^c = 2^{c \log n}$$

e analogamente per gli r_j. Inoltre notiamo che per ogni i vale[66] $|\text{bin}\, R_i| < n^{a_i} \leq n^a$. Con "posizione sul p_1-esimo pezzo del nastro di input" si intende "posizione della testina sulla porzione del nastro di input contentente

[64] Ricordiamo che stiamo supponendo che in tutti i linguaggi ci siano $\leq, \min, \max, +, \cdot$, e che nelle rispettive strutture $\leq$ venga interpretato come ordine totale, min come minimo di quest'ordine, max come massimo e $+$, $\cdot$ come le usuali operazioni una volta identificato grazie all'ordine il dominio con $\{1, \ldots, \max\}$. Con questi simboli si riesce anche a definire il log, da intendersi come parte intera superiore del logaritmo in base 2. La costruzione che stiamo facendo funziona solo per strutture con almeno r elementi, ma le altre sono finite e quindi possono essere gestite a mano in qualche maniera.

[65] Forse in p_1 invece di r bisogna metterci $r+s$ per non dimenticarsi le costanti, comunque la sostanza della dimostrazione non cambia.

[66] Infatti ricordiamo che la codifica è $r_1 + r_2 n + \ldots + r_{a_i} n^{a_i - 1}$ dove per sapere se vale $R_i(r_1, \ldots, r_{a_i})$ basta vedere il bit in posizione opportuna.

bin(R_i)", dove chi è R_i ce lo dice p_1. Siamo finalmente in grado di scrivere

$$E(\text{ID}, \text{ID}') = \bigvee_R C_R(\text{ID}, \text{ID}')$$

dove R varia fra le regole della macchina di Turing M e e C_R vuol dire che R permette di passare da ID a ID$'$. Ad esempio R può essere

$$R = \langle q_0, \exists, \underbrace{0}_{\text{input}}, \underbrace{1}_{\text{lavoro}} \rangle \mapsto \langle q_1, \forall, \underbrace{0}_{\text{lavoro}}, \leftarrow, \rightarrow \rangle$$

e in tal caso la σ-formula C_R sarà[67]

$$C_R(\text{ID}, \text{ID}) \equiv \bigwedge_{1 \le i \le r} \Big[\big[(p_1 = i) \wedge (p_2 = q_0) \wedge (p_4 = \exists) \wedge \text{BIT}(w_1 + \max \cdot w_2 + \ldots, +\max^{c-1}, p_3) \\ \wedge \neg R_i(r_1, \ldots, r_{a_i}) \big] \to \big[(p_2' = q_1) \wedge (p_4' = \forall) \wedge \neg\, \text{BIT}(w_1' + \max w_2' + \ldots, p_3) \wedge \\ (\forall p \neq p_3 \;\; \text{BIT}(w_1+, \ldots, p) \leftrightarrow \text{BIT}(w_1', \ldots p)) \wedge (\cdots) \big] \Big]$$

dove ricordiamo che BIT $\subseteq \mathbb{N}^2$ è la relazione definita da BIT(n, i) sse l'i-esimo bit di n in binario è 1. Questa è definibile da $+$ e $\cdot$ e viceversa, quindi non ci sono problemi viste le ipotesi sul linguaggio. Quel $\neg R_i(r_1, \ldots, r_{a_i})$ vuol dire "leggo 0 sul nastro di input". I puntini sono lì perché la formula non finisce lì ma più o meno ci dovremmo essere capiti. Tipo ci dovrebbe essere anche

$$\langle r_1', \ldots, r_a' \rangle = \{r_1, \ldots, r_a\} - 1$$

a meno di casi limite con overflow o brutte bestie del genere, e manca anche qualcosa del tipo $p_3' = p_3 + 1 \ldots$

Ci mancano ancora $G_\forall(\text{ID})$, che però può essere definita come "$p_4 = \forall'$", ed s e t che introduciamo come predicati 1-ari che devono dire roba tipo "la testina è all'inizio, lo stato è quello iniziale..." Ora è ovvio[68] che

$$\mathcal{A} \in S \Leftrightarrow M(\text{bin}(\mathcal{A})) \downarrow \Leftrightarrow I(\mathcal{A}) \in \text{Reach}_{\text{alt}}$$

□

12 10/11

12.1 Dimostrazione del Teorema di Fagin - Parte Difficile

Scopo di oggi è dimostrare il

Teorema 12.1 (Fagin, '74)**.** NP = ∃SO

[67] Nota: ∃ e ∀ sono superflui, è un'informazione già contenuta nel sapere chi è lo stato. Stanno lì per comodità.

[68] "O uno dice che è ovvio o non lo dimostra."

La $\supseteq$ è facile, per cui ci occupiamo prima della $\subseteq$. Sia $S \subseteq \mathrm{Struct}(\sigma)$ un problema in NP e sia N una macchina di Turing in $\mathsf{NTIME}(n^k - 1)$ che risolve il problema. Vogliamo una formula del tipo

$$\exists \vec{Q}\ \Phi \qquad \Phi \in \mathsf{FO}(\sigma, \vec{Q})$$

tale che per ogni $\mathcal{A} \in \mathrm{Struct}(\sigma)$ ($n = |A|$ o, equivalentemente a meno di costanti, la lunghezza della sua codifica) valga

$$N(\mathrm{bin}\,\mathcal{A}) \downarrow \iff \mathcal{A} \vDash \exists \vec{Q}\ \Phi$$

L'idea dietro la costruzione della formula è che questa dica "esiste una computazione accettante di N". La parte $\exists \vec{Q}$ è quella che dice "esiste una computazione" e la Φ è quella che dice "$\vec{Q}$ è una computazione accettante di N". Passiamo ai dettagli.

Supponiamo che N lavori con simboli da Σ e stati da Q, e come al solito pensiamo ad una configurazione come a una cosa del tipo

$$\underline{\overline{\mid \sigma_1 \mid \sigma_2 \mid \ldots \mid \langle \sigma_i, q \rangle \mid \sigma_{i+1} \mid \ldots \mid}}$$

dove intendiamo che il nastro è

$$\underline{\overline{\mid \sigma_1 \mid \sigma_2 \mid \ldots \mid \sigma_i \mid \sigma_{i+1} \mid \ldots \mid}}$$

e la testina è su σ_i nello stato q. Dunque posto $\Gamma = \Sigma \cup (\Sigma \times Q)$ una configurazione è una stringa di simboli di Γ (tutti in Σ, tranne uno che è in $\Sigma \times Q$). Dato che Γ è finito lo possiamo identificare con $\{1, \ldots, g\}$. Probabilmente quello che faremo funzionerà solo con le strutture di cardinalità almeno g, comunque il resto si sistema, l'importante è che funzioni definitivamente. La formula è:

$$\left(\exists C_1^{(2k)}, \ldots, C_g^{(2k)}, \Delta^{(k)}\right)\Phi$$

L'idea è che

$$C_i(\underbrace{s_1, \ldots, s_k}_{spazio}, \underbrace{t_1, \ldots, t_k}_{\text{tempo}})$$

dice "al tempo $t = t_1 + t_2 n + \ldots + t_k n^{k-1}$ la cella di memoria $s = s_1 + s_2 n + \ldots + s_k n^{k-1}$ contiene il simbolo i" (quell'n sarebbe l'interpretazione di max, cioè la cardinalità della struttura). Invece il predicato

$$\Delta(t_1, \ldots, t_k)$$

dice "al tempo t, N ha scelto 0". Questo perché WLOG possiamo assumere che le scelte nondeterministiche di N siano tutte binarie. L'ipotesi che N lavori in tempo polinomiale qui è essenziale a causa di come stiamo codificando t ed s. Resta da vedere com'è fatta Φ, che poi è la vera responsabile del funzionamento della baracca; abbiamo già anticipato che Φ esprime il fatto che $(C_1, \ldots, C_g, \Delta)$ codifica una computazione corretta.

Esempio 12.2. Lavoriamo con $\sigma = \{E^{(2)}, R^{(1)}\}$. Il caso generale è uguale a meno di autismi con le notazioni.

Tanto per cominciare Φ deve dire che l'input è codificato correttamente. Una σ-struttura è del tipo $\mathcal{A} = (n, E^{\mathcal{A}}, R^{\mathcal{A}})$, e come anticipato supponiamo $n \geq g$. Ricordiamo che $\text{bin}(\mathcal{A}) = \text{bin}(E^{\mathcal{A}}) \frown \text{bin}(R^{\mathcal{A}})$, dove $\text{bin}(E^{\mathcal{A}})_{an+b} = 1 \Leftrightarrow \mathcal{A} \vDash E(a,b)$ e $\text{bin}(R^{\mathcal{A}})_a = 1 \Leftrightarrow \mathcal{A} \vDash R(a)$. Ad esempio prendiamo $E(2,3)$ ed $R(5)$ e viene una cosa tipo

$$\underline{\overline{|\ \langle\ ,q\rangle\ |\qquad\qquad |\ 1\ |\qquad\qquad}|}_{n^2-1}\ \underline{\overline{|\qquad\qquad |\ 1\ |\qquad\qquad}|}$$

dove il primo 1 è in posizione $2n+3$ e il secondo in posizione n^2+5, e q è il q iniziale. Dunque $E(2,3), R(5)$ viene codificato con $C_1(3,2,0,\vec{0},\vec{0})$ e $C_1(5,0,1,\vec{0},\vec{0})$.

La formula Φ contiene un congiunto del tipo

$$\forall a,b,c\ \Big(E(a,b) \wedge R(c) \to C_1(a,b,\vec{0}) \wedge C_1(c,0,1,\vec{0})\Big)$$

Il caso $a,b = 0,0$ va trattato a parte: ci vuole $C_{\langle 1,\text{stato}\rangle}(a,b,\vec{0})$ e qualcosa del genere $\neg E(a,b) \to C_0(a,b,0)$ eccetera. Denotiamo $P^0 = P$ e $P^1 = P$, e indicando con δ la scelta in $\{0,1\}$ non deterministica, per ogni regola[69] di N un altro congiunto di Φ conterrà quella regola, che dipenderà da δ. Tipo, ad

$$R\colon (\sigma_0, q_3, \underbrace{1}_{\delta}) \mapsto (\sigma_4, q_5, -1)$$

corrisponde la transizione

$$\underline{\overline{|\qquad |\ \sigma_{-1}\ |\ \langle\sigma_0,q_3\rangle\ |\ \sigma_1\ |\qquad}|} \mapsto \underline{\overline{|\qquad |\ \langle\sigma_{-1},q_5\rangle\ |\ \sigma_4\ |\ \sigma_1\ |\qquad}|}$$

ma possiamo anche riscriverla come

$$(\sigma_{-1}, \langle\sigma_0,q_3\rangle, \sigma_1, 1) \xrightarrow{N} \sigma_4$$

$$(\sigma_{-2}, \sigma_{-1}\langle\sigma_0,q_3\rangle_{,1}\ , 1) \xrightarrow{N} \langle\sigma_{-1},q_5\rangle$$

Riscriviamo quindi le regole di N nella forma

$$\langle\gamma_{-1},\gamma_0,\gamma_1,\delta\rangle \xrightarrow[N]{R} b$$

con $\gamma_{-1},\gamma_0,\gamma_1,b \in \Gamma$. Per ogni regola R, Φ contiene il congiunto

$$\Phi_R \equiv \forall\vec{t},\vec{s}\Big[\Delta^{\delta}(\vec{t}) \wedge C_{\gamma_{-1}}(\vec{s}-1,t) \wedge C_{\gamma_0}(\vec{s},\vec{t}) \wedge C_{\gamma_1}(\vec{s}+1,\vec{t}) \to C_b(\vec{s},\vec{t}+1)\Big]$$

[69] Una regola non deterministica $R(\sigma,q)$ che può fare due cose a seconda della scelta nondeterministica δ può essere scritta come una regola deterministica $R(\sigma,q,\delta)$.

Tra l'altro questa, se non ci fosse quel $\Delta^\delta(\vec{t})$ (cioè se non ci fosse il nondeterminismo) sarebbe una formula del tipo Horn, cioè della forma "congiunzione di atomiche $\to$ atomica". Queste formule sono di una certa importanza in informatica, perché modellano le definizioni induttive. Tipo, $(A \wedge B) \to (C \vee D)$ non è Horn (possiamo pensarla come una cosa induttiva ma non deterministica). Equivalentemente una Horn è del tipo

$$\text{Atomica} \vee \bigvee \neg\text{Atomica}$$

Tornando alla dimostrazione, ci sarebbe da sistemare il fatto che con quei -1 si rischia di generare roba negativa, ma si dovrebbe sistemare. Comunque l'ultima cosa da aggiungere a Φ è un pezzo che dica "l'output accetta". Se ad esempio conveniamo che la macchina, quando accetta, scrive 1 all'inizio del nastro, aggiungiamo il congiunto

$$C_{\langle q_{\text{finale}},1\rangle}(\vec{0}, \overrightarrow{\max})$$

(qui stiamo sempre usando che n^k è time-costruibile e quindi possiamo suppore che la macchina si arresti sempre esattamente al tempo max). Ricapitolando abbiamo quindi

$$\Phi \equiv \bigwedge_i \forall \vec{s}\vec{t}(\psi_i)$$

con le ψ_i senza quantificatori[70] e per costruzione vale

$$A \vDash \exists(C_1, \dots, C_g, \Delta)\Phi$$

se e solo se esiste una computazione nondeterministica di N su input $\text{bin}(A)$ che accetta in tempo $n^k - 1$, dove $n = |A|$. □

12.2 Conseguenze ed Esempi

Corollario 12.3. $\mathsf{P} \subseteq \exists\mathsf{SO}\text{Horn}$

Dimostrazione. Se non ci fosse quel Δ^δ che può comparire negato (cioè se la macchina fosse deterministica) le ψ_i sarebbero tutte Horn[71]. □

Esempio 12.4. GrafiSconnessi $\in \exists\mathsf{SO}(= \mathsf{ESO})$.

[70] E che somigliano molto a delle Horn, vedi Corollario seguente. Tra l'altro, visto che somigliano a delle Horn clauses, si potrebbe essere tentati di chiamarle "Horny clauses". Googlare le parole fra virgolette è NSFW. La mail per gli insulti è in prima pagina.

[71] Grädel ha mostrato che se il linguaggio include una relazione "successore" allora vale anche l'altra inclusione.

Ad esempio se $G(V, E)$ la formula che esprime la sconnessione è

$$\exists A^{(1)}, B^{(1)} \Big(\forall x \ (A(x) \vee B(x)) \wedge \forall x, y \big[E(x, y) \to [A(x) \leftrightarrow \neg B(x)] \big] \Big)$$

Avevamo comunque visto che GrafiConnessi $\in \mathsf{P}$, e dunque, siccome $\mathsf{P} \subseteq \mathsf{NP} \cap \mathsf{coNP}$, dovremmo riuscire a dire "G connesso" con una ESO. Comunque mentre "sconnesso" si può dire con una ESO monadica, vedremo che "connesso" non si riesce a dire con una monadica. Per dire "G connesso" l'idea è "G è connesso se e solo se esiste uno *spanning tree*", cioè un sottografo che è un albero. La cosa che scriviamo non è proprio questa, ma "esiste un ordine parziale R sui vertici di G con un unico elemento minimale e tale che se y è un successore immediato di x in R allora x e y sono collegati da un arco del grafo", anche questa equivalente alla connessione (sotto l'ipotesi che G sia finito), ed è ESO. Funziona perché ogni elemento o è il minimo oppure ha predecessore immediato. Viceversa se un grafo è connesso ha uno spanning tree, e uno spanning tree va bene per le richieste di sopra.

13 13/11

13.1 Dimostrazione del Teorema di Fagin - Parte Facile

Vediamo la freccia facile $\mathsf{ESO} \subset \mathsf{NP}$.

Dimostrazione. Prendiamo $(\exists R_1, \ldots, R_k)\varphi$, con $\varphi \in \mathsf{FO}(\tau \cup \{R_1, \ldots, R_k\})$, e costruiamo M macchina di Turing in $\mathsf{NTIME}(n^\ell)$, per un qualche ℓ, tale che

$$\mathcal{A} \vDash \exists \vec{R} \varphi \iff M(\operatorname{bin} \mathcal{A}) \downarrow$$

Per definizione, per ogni τ-struttura $\mathcal{A}$, vale

$$\mathcal{A} \vDash \exists \vec{R} \varphi \iff \exists R_1^{\mathcal{A}}, \ldots, R_k^{\mathcal{A}} \ (\mathcal{A}, R_1^{\mathcal{A}}, \ldots, R_k^{\mathcal{A}}) \vDash \varphi$$

Fissati $R_1^{\mathcal{A}}, \ldots, R_k^{\mathcal{A}}$, la verifica che $(\mathcal{A}, R_1^{\mathcal{A}}, \ldots, R_k^{\mathcal{A}}) \vDash \varphi$ è fattibile in tempo polinomiale, perché φ è una formula del prim'ordine. Per controllare nondeterministicamente se $\mathcal{A} \vDash \exists \vec{R} \varphi$, scegliamo nondeterministicamente i bit $\operatorname{bin}(R_1^{\mathcal{A}}), \ldots, \operatorname{bin}(R_k^{\mathcal{A}})$, ricordando che la codifica è fatta come

$$R_i^A(a_1, \ldots, a_{n_i}) \iff \operatorname{bin}(R_i^{\mathcal{A}})_{a_1 + |A| a_2 + \ldots + |A|^{n-1} a_{n_i}} = 1$$

e quindi ci bastano $|A|^{n_i}$ bit per la singola relazione. Scegliamo nondeterministicamente $|A|^{n_1} + \ldots + |A|^{n_k}$ bit e poi facciamo il controllo di cui sopra in tempo polinomiale. □

13.2 Quantificatori Monadici

Abbiamo visto che la classe di grafi sconnessi è di tipo $\exists^{\mathsf{MON}}\mathsf{SO}$ (si intende monadico), mentre per i grafi connessi ce l'avevamo fatta con $\exists\mathsf{SO}$ ma quantificando su cose binarie[72]. Vogliamo vedere ora che

Teorema 13.1. $\exists^{\mathsf{MON}}\mathsf{SO} \neq \mathsf{co}\text{-}\exists^{\mathsf{MON}}\mathsf{SO}$

Prima della dimostrazione notiamo che se riuscissimo a levare il MON avremmo $\mathsf{NP} \neq \mathsf{coNP}$, e quindi $\mathsf{P} \neq \mathsf{NP}$ perché P è chiuso per complemento. Ci serve un risultato del tipo

Teorema. Se abbiamo una $\overset{\mathrm{G}}{\sim}_m$ (per tutti gli m) tali che

- $\overset{\mathrm{G}}{\sim}_0 \Rightarrow \equiv_0$
- $\overset{\mathrm{G}}{\sim}_{n+1}$ se e solo se $\forall a \exists b$ (etc.) $\overset{\mathrm{G}}{\sim}_n$

allora $\overset{\mathrm{G}}{\sim}_m \Rightarrow \equiv_m$

Diamo per ora per buono questo risultato[73].

Dimostrazione del Teorema 13.1. Basta mostrare che GrafiConnessi $\notin \exists^{\mathsf{MON}}\mathsf{SO}$, dato che sappiamo già che sta in $\mathsf{co}\text{-}\exists^{\mathsf{MON}}\mathsf{SO}$. Supponiamo per assurdo che esista

$$\exists \underbrace{P_1, \ldots, P_r}_{\mathsf{MON}} \ \varphi$$

con $\varphi \in \mathsf{FO}$ tale che per ogni grafo G

$$G \vDash \exists \vec{P} \ \varphi \iff G \text{ è connesso}$$

e sia m il rango di quantificazione di φ. Ricordando che un grafo è una $\{E^{(2)}\}$-struttura, poniamo

$$\tau = \{E^{(2)}, P_1^{(1)}, \ldots, P_r^{(1)}\}$$

Ora una τ-struttura la possiamo pensare come un grafo colorato con 2^r colori, ognuno dato da un sottoinsieme di $\{P_1, \ldots, P_r\}$: ogni vertice è colorato con l'insieme dei P_i che soddisfa. Sia

$$\mathcal{A} = (A, E^{\mathcal{A}}, P_1^{\mathcal{A}}, \ldots, P_r^{\mathcal{A}})$$

una τ-struttura tale che $G = (A, E^{\mathcal{A}})$ sia un grafo ciclico con $\ell + 1$ vertici, dove $\ell \ggg m$[74]. Definiamo una τ-struttura della forma

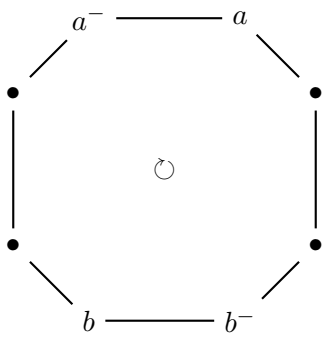
$$\mathcal{B} = (A, E^{\mathcal{B}}, P_1^{\mathcal{A}}, \ldots, P_r^{\mathcal{A}})$$

[72] In realtà avevamo anche detto che sono entrambi in P.

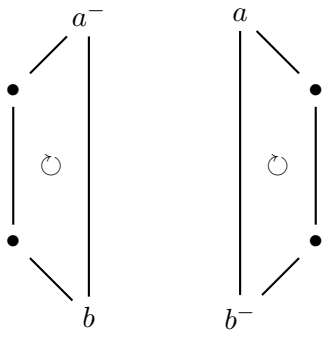
[73] Tutto sistemato nel Teorema 20.1.

[74] Sarà chiaro in seguito.

cioè con gli stessi vertici e gli stessi colori di $\mathcal{A}$, cambiamo solo gli archi. Quello che facciamo è "spezzarlo in due" fissando degli opportuni vertici a, b, a^-, b^-, togliendo gli archi $E(a, a^-)$ e $E(b, b^-)$ e aggiungendo $E(a, b^-)$ e $E(a^-, b)$. Ad esempio se $(A, E^{\mathcal{A}})$ è il grafo[75]



il grafo $(A, E^{\mathcal{B}})$ sarà qualcosa del tipo



Per una scelta opportuna di a, b, abbiamo $\mathcal{B} \equiv_m \mathcal{A}$. Questo genera un assurdo perché la restrizione di B all'opportuno linguaggio è sconnesso ma verifica φ. Definiamo la palla di raggio s centrata in a come

$$B_s(a) = \{x \in A \mid d(x, a) \leq s\}$$

Per $\ell \ggg m, r$, se $|A| = \ell + 1$ troviamo $a, b \in A$ tali che

- $B_{3^m}(a) \cong B_{3^m}(b)$ come grafi colorati,
- $B_{3^m}(a) \cap B_{3^m}(b) = \emptyset$.

Questo è facile da vedere: $|B_s(a)| = 2s + 1$, le possibili s-palle colorate sono $(2^r)^{2s+1}$, quindi per ℓ abbastanza grande due isomorfe le troviamo, e aumentandolo ancora si riescono a trovare disgiunte. Definiamo $\mathcal{B}$ spezzando $\mathcal{A}$ in due cicli disgiunti come nella figura sopra: per fare questo basta definire

$$E^{\mathcal{B}} = \big(E^{\mathcal{A}} \backslash \{(a^-, a), (a, a^-), (b^-, b), (b, b^-)\}\big) \cup \{(a^-, b), (b, a^-), (a, b^-), (b^-, a)\}$$

[75] La $\circlearrowleft$ è per dare un senso a quel a^- come "predecessore" di a.

Ora mostriamo $\mathcal{A} \equiv_m \mathcal{B}$ usando i giochi di Ehrenfeucht-Fraïssé. L'idea è che se uno guarda localmente in a non riesce a distinguere a^- del primo grafo da b^- del secondo (guardare le frecce circolari nel disegno potrebbe dare un'idea di cosa stiamo cercando di dire[76]).

Definizione 13.2. Diciamo che

$$\mathcal{A}, a_1, \ldots, a_k \overset{\text{G}}{\sim}_j \mathcal{B}, b_1, \ldots, b_k$$

se e solo se

1. esiste un isomorfismo di grafi colorati

   $$f \colon B(3^j, a_1) \cup \ldots \cup B(3^j, a_k) \to B(3^j, b_1) \cup \ldots \cup B(3^j, b_k)$$

   che associa $a_i \mapsto b_i$

2. per ogni tipo di isomorfismo t di 3^j-palle[77] $\mathcal{A}$ e $\mathcal{B}$ hanno lo stesso numero di vertici di tipo t, e questo è testimoniato da una bigezione $h \colon A \to B$.

Vediamo un po' di cose:

1. La relazione $\overset{\text{G}}{\sim}_j$ ha il va e vieni: se $\mathcal{A}, \bar{a} \overset{\text{G}}{\sim}_{j+1} \mathcal{B}, \bar{b}$, per ogni $a \in A$ esiste $b \in B$ tale che $\mathcal{A}, \bar{a}, a \overset{\text{G}}{\sim}_j \mathcal{B}, \bar{b}, b$, e similmente per ogni $b \in B$ esiste $a \in A$ tale che (stessa cosa di prima). Questo è il punto cruciale e lo verichiamo alla fine.

2. $A, x \overset{\text{G}}{\sim}_m \mathcal{B}, x$, $\ell \ggg m$, $B_{3^m}(x)^{\mathcal{A}} \cong B_{3^m}(x)^{\mathcal{B}}$.

3. $\overset{\text{G}}{\sim}_0$ implica $\equiv_0$ per la clausola $a_i \mapsto b_i$ nella definizione di $\overset{\text{G}}{\sim}_j$, almeno per $l(\bar{a}) > 0$; ma tanto basta partire fissando un vertice[78]. Comunque questo è vero perché dato che $j \geq 0$ allora le palle hanno raggio almeno $3^j \geq 3^0 \geq 1$ e quindi i "vicini" del punto ci sono e le formule atomiche sono preservate.

Notiamo che se $i < j$ allora $\overset{\text{G}}{\sim}_j \Rightarrow \overset{\text{G}}{\sim}_i$: il caso $\overset{\text{G}}{\sim}_0$ vuol dire "esiste una bigezione che preserva i colori", e per $j > 0$ stiamo dicendo che questa bigezione preserva anche gli archi in una certa porzione del grafo (le palle di raggio 3^j centrate nei punti $\bar{a}$ e $\bar{b}$.).

Vediamo come fare il va e vieni. Supponiamo di avere $\mathcal{A}, a_1, \ldots, a_k \overset{\text{G}}{\sim}_{j+1} , \mathcal{B}, b_1, \ldots, b_k$. Abbiamo due casi:

[76] Ma anche no. Comunque è stato tutto sistemato nel Teorema 20.1 e discussione successiva.

[77] $x \in A$, $y \in B$ hanno lo stesso tipo di 3^j-palla se $B_{3^j}(x)^{\mathcal{A}} \overset{x \mapsto y}{\cong} B_{3^j}(y)^{\mathcal{B}}$, dove l'isomorfismo è fra grafi colorati.

[78] O autisticare su convenzioni nelle definizioni, tipo mettere $\perp$ di default nelle formule...

- Siamo fortunati: esiste i tale che il nuovo a è nella palla di raggio $B_{2\cdot 3^j(a_i)}$. Dato che
$$B_{3^j}(a) \subset B_{3^{j+1}}(a_i)$$
basta mandare b in $f(a)$, dove f è l'isomorfismo garantito da $\overset{\mathrm{G}}{\sim}_{j+1}$. Chiaramente funziona tutto perché $\mathcal{A}, a_1, \ldots, a_k, a \overset{\mathrm{G}}{\sim}_j \mathcal{B}, b_1, \ldots, b_k, b$ tramite la restrizione di f.

- Siamo meno fortunati[79]: $a \notin \bigcup_{u=1}^{k} B_{2\cdot 3^j}(a_u)$. Chiaramente ci serve $b \in B$ fuori dalle palle corrispondenti e con lo stesso tipo di 3^j-palle di a, cioè deve esistere un isomorfismo $g\colon B_{3^j}(b) \cong B_{3^j}(a)$. Mostriamo che un tale b esiste. Per l'ipotesi fatta su a in questo caso la palla di a e disgiunta da tutte quelle degli a_i, cioè $B_{3^j}(a) \cap B_{3^j}(a_i) = \emptyset$. Quindi un b che va bene esiste per questioni di cardinalità (dentro ce ne sono lo stesso numero, in totale anche, e quindi pure fuori; meditare su questo passaggio). Abbiamo quindi $\mathcal{A}, a_1, \ldots a_k, a \overset{\mathrm{G}}{\sim}_j \mathcal{B}, b_1, \ldots, b_k, b$, dove l'isomorfismo si ottiene unendo l'isomorfismo vecchio a g; questo si può fare senza problemi perché i loro domini sono disgiunti.

Ora vediamo cosa succede ai nostri $\mathcal{A}$, $\mathcal{B}$ di prima. La mappa "identità" $\mathcal{A} \to \mathcal{B}$ preserva i tipi dei 3^j-intorni: come grafi non colorati le palle sono isomorfe perché sono grafi lineari (per la scelta di $\ell \ggg m$ il ciclo non si può chiudere). I colori sono rispettati per scelta[80] di a e b.

Ora per concludere basta mostrare che $\mathcal{B} \vDash \exists \vec{P} \varphi$, che è assurdo perché $\mathcal{B}$ è sconnesso. Dato che per ipotesei $\mathcal{A} \vDash \exists \vec{P} \varphi$, esistono $P_1, \ldots, P_r$ tali che $\mathcal{A} \vDash \varphi(P_1, \ldots, P_r)$. Dato che φ ha rango m e $\mathcal{B} \equiv_m \mathcal{A}$, allora $\mathcal{B} \vDash \varphi(P_1, \ldots, P_r)$ e quindi $\mathcal{B} \vDash \exists \vec{P}\ \varphi$. □

14 17/11

14.1 Fine Dimostrazione Precedente e Commenti

Risistemata direttamente sulla lezione scorsa.

Dicevamo che avevamo un isomorfismo di grafi colorati $f\colon \bigcup_{i=1}^{k} B(3^j, a_i) \to \bigcup_{i=1}^{k} B(3^j, b_i)$ e una bigezione $h\colon A \to B$ tale che per ogni elemento $a \in A$ valga $B(3^j, a) \cong B(3^j, h(a))$ come grafi colorati, il che non vuol dire che A, B siano isomorfi; si pensi a una bigezione che non rispetta la E fra i due grafi disegnati prima colorati con un solo colore; non è dunque detto che $E(x, y) \to E(hx, hy)$. In ogni caso possiamo assumere $h(a_i) = b_i = f(a_i)$, ma non dovrebbe servire.

[79] Il punto è che f preserva i tipi di 3^j-palle solo in una zona interna del dominio, su "bordo" le 3^j-palle possono uscire dal dominio e lì fuori f non dice niente.

[80] Bisogna un attimo mettersi a controllare.

14.2 Precisazioni

Rienunciamo il Teorema che non abbiamo dimostrato più precisamente:

Teorema (Giochi)**.** Supponiamo di avere una famiglia di relazioni $(\overset{*}{\sim}_j)_j$ tale che per ogni coppia di σ-strutture $\mathcal{A}$, $\mathcal{B}$ valga

1. $\mathcal{A}, a_1, \ldots, a_k \overset{*}{\sim}_j \mathcal{B}, b_1, \ldots, b_k \Rightarrow \mathcal{A}, a_1, \ldots, a_k \equiv_0 \mathcal{B}, b_1, \ldots, b_k$
2. $\mathcal{A}, a_1, \ldots, a_k \overset{*}{\sim}_{j+1} \mathcal{B}, b_1, \ldots, b_k \Rightarrow \forall a \exists b\, \mathcal{A}, a_1, \ldots, a_k, a \equiv_0 \mathcal{B}, b_1, \ldots, b_k, b$
   e analogamente con $\forall b \exists a \ldots$

La dimostrazione è in rete sulla pagina di Berarducci, ma in realtà l'abbiamo già fatta: è metà della dimostrazione del risultato analogo ma con i se e solo se e con $\overset{*}{\sim}_j = \overset{\text{EF}}{\sim}_j$.

Se abbiamo una classe di strutture K, quanto visto si può adattare per dimostrare che $K \notin \text{Mon-}\Sigma^1_1$. Ci sono anche ovvie varianti per $K \notin \Sigma^1_1 = \mathsf{NP}$. Ad esempio se $\neg$3-col $\notin \mathsf{NP}$ questa metodologia deve funzionare. Ne riparleremo[81]. Comunque moar info in fondo agli appunti di uno studente di 5 anni fa (ci sono due pagine aggiunte) disponibili sulla pagina del corso.

14.3 NP-Completezza di SAT

Il fatto che SAT sia NP è ovvio perché un'assegnazione è un certificato, oppure, nella codifica vista subito dopo la Definizione 6.15, passando da Fagin con la formula

$$\exists S^1\, \forall c \exists v\, [P(c,v) \wedge S(v)] \vee [N(c,v) \wedge \neg S(v)]$$

che si legge come "esiste un'assegnazione che rende ogni clausola soddisfatta". Mostriamo ora che SAT è NP-hard:

Teorema 14.1. $\mathscr{B} \in \mathsf{NP} \Rightarrow \mathscr{B} \leq_{\mathsf{P}} \text{SAT}$

Dimostrazione. Per noi $\mathscr{B} \subseteq \text{Struct}(\tau)$. Dato che[82] $\mathscr{B} \in \mathsf{NP} = \Sigma^1_1$ per il Teorema di Fagin, esiste una formula $\gamma \in \Sigma^1_1$ tale che $\mathscr{B} = \text{Mod}(\gamma)$; inoltre (vedi dimostrazione di Fagin) γ la possiamo prendere della forma

$$\gamma = \exists S_1, \ldots, S_r\, \forall x_1, \ldots, x_k \quad \underbrace{\varphi(x_1, \ldots, x_k)}_{\text{senza quantificatori e in CNF}}$$

Dunque, se $L = \{R_1, \ldots, R_t\}$ e $\mathcal{A}$ è una L-struttura, abbiamo $\mathcal{A} \in \mathscr{B} = \text{Mod}(\exists \vec{S}\, \forall \bar{x}\, \varphi)$ se e solo se

$$\mathcal{A} \vDash \exists S_1, \ldots, S_r\, \forall x_1, \ldots, x_k\, \varphi(x_1, \ldots, x_k)$$

[81] Vedi Teorema 18.1.

[82] Σ^1_1 vuol dire ESO. Per ora la si pensi come abbreviazione, ma ricomparirà.

se e solo se, posto $n = |A|$,

$$\exists S_1 \subset n^{a_1}, \dots, S_r \subset n^{a_r} \bigwedge_{0 \le a_1, \dots, a_k < n} \varphi(a_1, \dots, a_k)$$

Dato che φ era senza quantificatori e tutte le variabili sono state istanziate, la possiamo vedere come formula proposizionale nelle variabili proposizionali $S_i(\vec{a}), R_j(\vec{a})$. Ad esempio

$$\mathcal{A} \vDash \exists S \, \forall x \, (R(x) \vee S(x)) \iff \exists S \subset n \Big[\bigwedge_{0 \le a < n} R(a) \vee S(a) \Big]$$

dove pensiamo le $S_i(\vec{a})$, come variabili proposizionali, mentre in $\mathcal{A}$ le R hanno già un valore preciso, e quindi le sostituiamo direttamente con "vero" o "falso". Dunque

$$\exists S_1 \subset n^{a_1}, \dots, S_r \subset n^{a_r} \bigwedge_{0 \le a_1, \dots, a_k < n} \varphi(a_1, \dots, a_k) \iff \bigwedge_{0 \le a_1, \dots, a_k < n} \varphi(a_1, \dots, a_k) \in \text{SAT}$$

Dunque $\mathcal{A} \in \mathscr{B} \Leftrightarrow \gamma(\mathcal{A}) \in \text{SAT}$, per cui $\mathscr{B} \le \text{SAT}$ tramite γ. Bisogna vedere che la riduzione è polinomiale, ma φ è fissata e bisogna fare circa n^k operazioni (guardare quanti sono i congiunti di γ!). □

In realtà la riduzione che abbiamo fatto, scrivendo SAT con la solita codifica, è al prim'ordine, per cui abbiamo in realtà mostrato che

$$\mathscr{B} \in \mathsf{NP} \Rightarrow \mathscr{B} \le_{\mathsf{FO}} \text{SAT}$$

15 20/11

L'ultima data buona per l'esame prima che Berarducci parta è il 13 gennaio. Altrimenti ci si inventerà qualcosa.

15.1 Il Teorema di Gerarchia

Ricordiamo che le funzioni usate come bound per il tempo/spazio sono sempre assunte time/space-costruibili.

Teorema 15.1 (di Gerarchia)**.** Se f è time-costruibile, $\mathsf{DTIME}(f(n)) \subsetneq \mathsf{DTIME}(f(2n+1)^3)$.

In realtà invece del cubo dovrebbe bastare una cosa tipo $t(n)\log(t(n))$. Comunque ne segue che

Corollario 15.2. $\mathsf{P} \subsetneq \mathsf{EXPTIME}$

Dimostrazione. Abbiamo

$$\mathsf{P} = \bigcup_k \mathsf{TIME}(n^k) \subseteq \mathsf{TIME}(2^n) \subsetneq \bigcup_k \mathsf{TIME}(2^{n^k}) = \mathsf{EXPTIME}$$

□

L'idea per dimostrare il Teorema di Gerarchia è la seguente. Ricordiamo il problema della fermata: l'insieme $K_0 = \{M \mid \varphi_M(M) \downarrow\}$ è ricorsivamente enumerabile ma non ricorsivo. Infatti se lo fosse stato, anche il suo complemento $\neg K_0 = D = \{M \mid \varphi_M(M) \uparrow\}$ lo sarebbe, e in particolare sarebbe ricorsivamente enumerabile. Dunque esiste una macchina di Turing e tale che $M \in D \Leftrightarrow \varphi_e(M) \downarrow$. Ma allora si ha l'assurdo $\varphi_e(e) \uparrow \Leftrightarrow e \in D \Leftrightarrow \varphi_e(e) \downarrow$. L'idea per il Teorema di Gerarchia è simile. Analogamente si definisce $K = \{(M, x) \mid \varphi_M(x) \downarrow\}$, che non è ricorsivo perché sennò lo sarebbe pure K_0. Definiamo[83]

$$K_{t(n)} = \{(M, x) \mid \varphi_M(x) \downarrow_{t(|(M,x)|)}\}$$

Le Macchine di Turing usate nella definizione delle classi ci complessità sono in generale a più nastri, dunque della forma (Q, Σ, δ, s), con $s \in Q$ stato iniziale e, se ha k nastri,

$$\delta \colon Q \times \Sigma^k \to (Q \cup \{\text{sì}, \text{no}\}) \times (\Sigma \times 3)^k$$

dove $3 = \{\leftarrow, -, \rightarrow\}$, "sì" e "no" sono stati finali che accettano/rifiutano e WLOG $\Sigma = \{0, 1\}$. La macchina universale U, su input M, x simula $M(x)$; diciamo che U ha tre nastri, uno in cui scrive le istruzioni di M (codificate in qualche maniera), uno che usa come "orologio"[84] per f, e uno in cui all'inizio scrive[85] una cosa tipo

$$\underbrace{\underbrace{\triangleright x \triangleleft}_{\text{blocco}} \underbrace{\triangleright \triangleleft}_{\text{blocco}} \cdots \underbrace{\triangleright \triangleleft}_{\text{blocco}}}_{k \text{ blocchi}} \triangleleft'$$

dove $\Sigma_U = \Sigma \cup \{\triangleright, \triangleleft, \triangleleft'\}$, e quei simboli significano "inizio nastro", "fine nastro" e "fine dei nastri". In realtà avevamo detto che gli alfabeti sarebbero stati binari, ma si può simulare l'alfabeto esteso con codifiche tipo

$$\begin{aligned} 0 &\mapsto 000 \\ 1 &\mapsto 010 \\ \triangleleft &\mapsto 100 \\ \triangleright &\mapsto 110 \\ \triangleleft' &\mapsto 111 \end{aligned}$$

[83] Il fatto che t dipenda anche da (una codifica di) M è perché bisogna "simularla" tramite una macchina universale.

[84] Ad esempio scrivendo 1 per $f(|x|)$ volte, poi cancellandone uno ad ogni passaggio e fermandosi quando arriva a 0. Ricordiamo che f è time-costruibile.

[85] È inefficiente: fare questo passaggio meglio rafforza il Teorema di Gerarchia come detto sopra.

In un momento generico il secondo nastro di U contiene qualcosa del tipo

$$\triangleright x \triangleleft \triangleright w_1 \triangleleft \ldots \triangleright w_{k-1} \triangleleft \triangleleft'$$

che avrà quindi lunghezza, se M lavora in tempo $t(|x|)$,

$$\leq |x| + \sum |w_i| + 2k + 1 \leq 2k + 1 + kt(|x|)$$

Per simulare un passo di M, U fa avanti e dietro sul suo nastro, nel senso che lo percorre due volte, e ci mette tempo[86] $kt(n) \cdot c(k, M)$, dove quella c è lineare[87]. Dunque, dato che $t(n)$ per essere time-costruibile deve soddisfare[88] $t(n) \geq n$, per fare $t(n)$ passi ci mette dunque $O(t(n)^3)$, dove $n = |x|$. Questo prova metà del Teorema, il resto lo vedremo nella prossima lezione.

15.2 Gerarchia Polinomiale

Se $\Sigma_1^1 = \mathsf{ESO} = \mathsf{NP}$, la logica del second'ordine "full" corrisponde alla *gerarchia polinomiale* PH, che si può definire come segue.

Considieriamo le *macchine di Turing con oracolo per* $B \subset \Sigma^*$. Questo vuol dire che M ha un nastro "oracolo", su cui può comunque scrivere, e uno stato speciale q_{or} che le dice che se è in quello stato deve consultare l'oracolo. Se M raggiunge q_{or} e sul suo nastro oracolo c'è la stringa x, allora passa in uno stato q_1 se $x \in B$ e in uno stato q_2 se $x \notin B$ (B può fare schifo a piacere, ad esempio può non essere ricorsivamente enumerabile). In genere le macchine con oracolo si indicano con l'oracolo ad apice, tipo $\varphi_M^B(w)$.

Definizione 15.3. Un problema A si dice *Turing-riducibile a* B, denotato con $A \leq_T B$, se A è calcolabile da una macchina con oracolo per B.

Ad esempio $A \leq_T \neg A$, mentre con le riduzioni viste in precedenza[89] in genere non era vero. Uno poi può definire $A \equiv_T B$ quando $A \leq_T B$ e $B \leq_T A$, e ci sono teoremi tipo che esistono A, B, ricorsivamente enumerabili[90] tali che $A \not\leq_T B$ e $B \not\leq_T A$. Comunque possiamo definire induttivamente $\mathsf{PH}_1 = \mathsf{NP}$ e $\mathsf{PH}_{n+1} = \mathsf{NP}^{\mathsf{PH}_n}$, dove con $B \in \mathsf{NP}^{\mathsf{PH}_n}$ intendiamo che esiste $A \in \mathsf{PH}_n$ (oracolo) e una macchina di Turing nondeterministica in NP con oracolo A che riconosce B, cioè

$$x \in B \Leftrightarrow M^A(X) \downarrow_{\text{poly}} = \text{sì}$$

e poi possiamo dire

[86] Mi sa che è $2kt(n)$, ma tanto dopo va tutto nell'$O(\ldots)$.

[87] Bisogna riguardarsi le istruzioni di M...

[88] Sennò non c'è nemmeno tempo di leggere l'input.

[89] Che si chiamano anche "many-one", quelle con $f\colon A \to B$ ricorsiva tale che $x \in A \Leftrightarrow f(x) \in B$.

[90] Quelli decidibili sono riconducibili a tutto, visto che non serve nemmeno l'oracolo.

Definizione 15.4. $\mathsf{PH} = \bigcup \mathsf{PH}_i$.

Ora $\mathsf{P}^{\mathsf{P}} = \mathsf{P}$ perché essenzialmente incorporare l'oracolo nella macchina di partenza fa restare il tutto polinomiale. Dunque se $\mathsf{P} = \mathsf{NP}$ allora anche $\mathsf{PH} = \mathsf{P}$. Si può dimostrare che $\mathsf{SO} = \mathsf{PH}$, e anzi noi mostreremo direttamente il risultato più preciso

Teorema. $\mathsf{PH}_k = \Sigma_k^1$

Dove con Σ_k^1 indichiamo k alternanze di quantificatori e in fondo qualcosa P, (i quantificatori sono del second'ordine)[91].

16 24/11

16.1 Gerarchia Revisited

Finiamo di dimostrare il

Teorema (di Gerarchia)**.** Se f è time-costruibile[92],

$$\mathsf{DTIME}(f(n)) \subsetneq \mathsf{DTIME}(f(2n+1)^3)$$

Dimostrazione. Definiamo

$$H_f = \{([M], x) \mid M \text{ accetta } x \text{ in } 2f(|x|) \text{ passi}\}$$

Questo insieme sarà proprio quello che testimonia l'inclusione stretta. Poniamo m pari alla lunghezza di $([M], x)$, cioè $|[M]| + |x| + 1$ (l'1 è per codificare la virgola). Intanto che H_f è calcolabile è chiaro: basta simulare M su input x per $f(|x|)$ passi, e abbiamo già visto che ce la si fa in tempo $O(f(2n+1)^3)$.

Vediamo che $H_f \notin \mathsf{DTIME}(f(\lfloor m/2 \rfloor))$. Sia per assurdo K una macchina che decide se un generico input $([M], x) \in H_f$ in $f(\lfloor m/2 \rfloor)$ passi. In particolare K decide se $([M], [M]) \in H_f$ in un numero di passi minore o uguale a

$$f\left(\left\lfloor \frac{m}{2} \right\rfloor\right) = f\left(\left\lfloor \frac{2|[M]| + 1}{2} \right\rfloor\right) = f(|[M]|)$$

Usiamo ora K per costruire una macchina N che su input $[M]$ fornisce $([M], [M])$ a K e accetta se e solo se K rifiuta. In altre parole

$$N([M]) = \neg K([M], [M])$$

Il tempo di N è $|[M]|$, che serve a scrivere una seconda copia di $[M]$, più il tempo di K. In totale

$$t(N) = |[M]| + f([M]) \leq 2f([M])$$

[91] Vedi dopo

[92] Quest'ipotesi è fondamentale; provate a cercare il `Gap Theorem` di Borodin-Trakhtenbrot...

perché una funzione time-costruibile è sempre grande almeno quanto l'identità (uno deve almeno leggere l'input[93]). Ora, sviscerando le definizioni, N accetta $[M]$ (in tempo $2f(|[M]|)$) se e solo se M non accetta $[M]$ in $2f(|[M]|)$ passi. Basta ora porre $M = N$ per avere un assurdo. □

16.2 Linear Speed Up

Giustifichiamo il nostro non curarci delle costanti.

Teorema 16.1. Siano $\varepsilon > 0$ ed f time-costruibile. Allora $\mathsf{TIME}(f(n)) \subset \mathsf{TIME}(\varepsilon f(n) + n + 2)$.

Dimostrazione. Sia M che riconosce un certo linguaggio in $f(n)$ passi. Costruiamo M' che ce la fa in $\varepsilon \cdot f(n) + n + 2$. Se Σ_M è l'alfabeto di M, allora per un certo $m \in \mathbb{N}$ che fisseremo dopo compattiamo m simboli in un simbolo solo: $\Sigma_{M'} = \Sigma_M \sqcup (\Sigma_M)^m$. Poi M' "aggrega" i simboli sul nastro di m di conseguenza (e codifica l'informazione su quale simbolo stesse puntando M negli stati). Per simulare m passi di M ad M' ne bastano 6 (a prescindere da m). Dettagli per esercizio[94]. Poi basta prendere $6/m = \varepsilon$. Quell' $n+2$ serve a passare dal linguaggio "normale" a quello "compattato" (l'input andrà pur letto). □

16.3 Alcuni Risultati di Solovay

Come l'altra volta uno può definire NP^B e P^B, dove B è un oracolo.

Teorema 16.2 (Solovay)**.** Esistono oracoli B e C tali che $\mathsf{NP}^B = \mathsf{P}^B$ e $\mathsf{NP}^C \neq \mathsf{P}^C$.

Quindi servono cose un po' più furbe delle solite[95] dimostrazioni via oracolo. Se B è un oracolo casuale, uno può calcolare la probabilità che $\mathsf{NP}^B = \mathsf{P}^B$ e fa 0. In parole più suggestive "NP è diverso da P con probabilità 1". In realtà non è così liscia perché ci sono classi che sono uguali, ma sono diverse con oracolo random.

17 27/11

17.1 Gerarchie

Vogliamo mostrare che

[93] Questa è essenzialmente una convenzione. Stiamo dicendo che non ci interessano linguaggi tipo "le stringhe che cominciano per 1".

[94] Sul *Computational Complexity* di Papadimitriou c'è qualche dettaglio in più. Comunque quel 6 sta lì perché in m passi possono cambiare solo il "blocco da m" corrente e i due adiacenti. Quindi per simulare m passi di M ad M' basta fare quattro passi (destra-sinistra-sinistra-destra) per leggere le celle influenzate e due passi per modificare i due blocchi modificati.

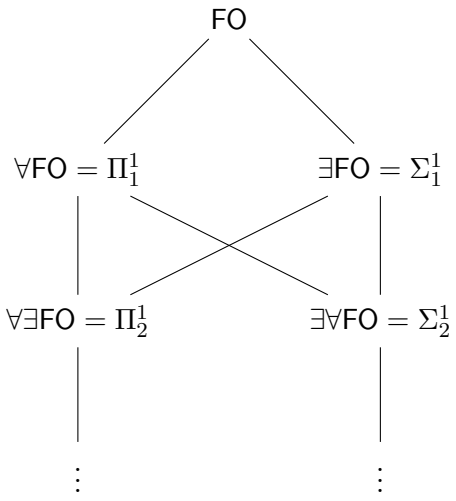
[95] Solite per altri corsi, tipo quello di ASD.

Teorema 17.1. $\mathsf{NP}^{\mathsf{NP}} = \Sigma_2^{\text{poly}} = \Sigma_2^1$

ma prima definiamo un po' di cose. Avevamo detto che $\mathsf{SO} = \bigcup_n \Sigma_n^1$, ma non avevamo definito bene tutto. Avevamo detto che $\Sigma_1^1 = \exists\mathsf{SO}$. In generale uno parte dalle formule del primo ordine $\mathsf{FO} = \Sigma_0^1 = \Pi_0^1$ e definisce (i quantificatori si intendono del second'ordine)

$$\Sigma_{n+1}^1 = \exists\Pi_n^1 \qquad \Pi_{n+1}^1 = \forall\Sigma_n^1$$

dove intendiamo che di $\exists$ e $\forall$ ne possiamo mettere quanti vogliamo senza alternare. In altre parole $\exists\Sigma$ è ancora Σ e $\forall\Pi$ è ancora Π. Il quadro è più o meno questo



analogamente uno parte da $\mathsf{PTIME} = \Sigma_0^{\text{poly}} = \Pi_0^{\text{poly}}$ e definisce

$$\Sigma_{n+1}^{\text{poly}} = \exists^{\text{poly}}\Pi_n^{\text{poly}} \qquad \Pi_{n+1}^{\text{poly}} = \forall^{\text{poly}}\Sigma_n^{\text{poly}}$$

il significato di "poly" è che, se $|x|$ è la lunghezza della rappresentanzione (quindi se x è un numero è il suo logaritmo in base 2), la variabile quantificata ha lunghezza controllata polinomialmente:

$$\forall^{\text{poly}} := (\forall x : |x| \leq \text{poly}|y|)Q(x,y) \qquad \forall^{\text{poly}} := (\exists x : |x| \leq \text{poly}|y|)Q(x,y)$$

con Q in PTIME e "poly" da sostituire con un opportuno polinomio, e viene una gerarchia come quella sopra. Comunque un esempio è

$$\Sigma_2^{\text{poly}} \ni \exists|x| \leq |y|^k \, \forall|z| \leq |u|^{k'} \, Q(x,y,z,u)$$

Esercizio 17.2. A meno di equivalenza logica, ogni Σ_n^1 e Π_n^1 è chiusa per quantificatori del prim'ordine e per i connettivi $\wedge$, $\vee$ (per la negazione no).

Soluzione. Supponiamo di avere $\forall x\ \exists R\ \varphi(R, x, y)$. Per com'è scritta R dipenderà da x, e il trucco è far diventare la R, supponiamo diciamo n-aria, una Q relazione $n+1$-aria

$$R(z) = R_x(z) = Q(x, z)$$

e quantificarla all'inizio[96]. □

Dunque la definizione ha senso e possiamo pensare le formule tutte in forma prenessa con i quantificatori "importanti" (quelli del second'ordine) tutti all'inizio.

17.2 Equivalenze fra Gerarchie

In un certo senso $\Sigma_n^{\text{poly}} \sim \Sigma_n^1$. Vediamo in che senso. Prendiamo un *linguaggio*[97] $S \subset \{0,1\}^*$, ad esempio

$$S = \{\text{bin}(G) \mid G \text{ 3-colorabile}\}$$

che sappiamo appartenere ad NP, e che possiamo quindi vedere come $S = \{G \mid G \vDash \varphi\}$, con $\varphi \in \Sigma_1^1$. Il senso è che questo potrà anche essere espresso come

$$S = \{w \in \{0,1\}^* \mid \psi(w)\} \qquad \psi \in \Sigma_1^{\text{poly}}$$

Teorema 17.3. Valgono

1. $S \in \mathsf{NP}$ se e solo se esiste $\psi(-) \in \Sigma_1^{\text{poly}}$ tale che $S = \{w \mid \psi(w)\}$
2. Più in generale $S \in \Sigma_n^1$ se e solo se esiste $\psi(-) \in \Sigma_n^{\text{poly}}$ tale che $S = \{w \mid \psi(w)\}$ (e vale il risultato analogo con Π al posto di Σ).

Nota: ψ non è una formula da "dare in pasto" a una struttura per vedere se la accetta, ma una formula verificata dalla codifica binaria della struttura. In altre parole non ha senso dire $\mathcal{A} \vDash \psi$, ma $\psi(\text{bin}(\mathcal{A}))$ sì.

Dimostrazione.

1. $\psi(-) \in \Sigma_1^{\text{poly}}$ se e solo se esiste $Q(-,-) \in \mathsf{PTIME}$ ed esiste k tale che

   $$\psi(w) \Leftrightarrow (\exists c : |c| \leq |w|^k)\ Q(c, w)$$

   e si conclude con la definizione di NP coi certificati[98]. Finire i dettagli per esercizio.

[96] Domanda: ma quindi per $\exists^{\mathsf{MON}}$ che succede? Risposta: che esplode (non è chiusa per quantificazione del prim'ordine). Guardare sull'Immerman, Corollary 8.10.

[97] Nel senso "insieme di parole". Equivalentemente possiamo pensarlo come problema (quello di decidere se una stringa appartiene al linguaggio).

[98] Che, ricordiamo, è equivalente all'altra perché la lista delle scelte nondeterministiche di una computazione che accetta è un certificato.

2. Fatto il punto precedente come passo base, il resto è induttivo. Per semplicità notazionale facciamo il caso Π^1_2 e mostriamo che è equivalente a Π^{poly}_2 (per fare le altre si passa ai complementi). Sappiamo già che

$$\Sigma^1_1 \underset{\text{Fagin}}{\equiv} \mathsf{NP} \equiv \Sigma^{\text{poly}}_1$$

Prendiamo $\varphi \in \Pi^1_2$, ad esempio nel linguaggio τ, della forma

$$\varphi \equiv \forall R\ \exists Q\ \theta$$

con θ del primo ordine nel linguaggio $\tau \cup \{R, Q\}$. Vogliamo

$$\mathcal{A} \vDash \varphi \Leftrightarrow \psi(\text{bin}(\mathcal{A}))$$

con $\psi(-) \in \Pi^{\text{poly}}_2$. Ora abbiamo, usando l'ipotesi induttiva per far comparire ψ,

$$\begin{aligned}
&\mathcal{A} \vDash \varphi \\
\Leftrightarrow &\mathcal{A} \vDash \forall R\ \exists Q\ \theta \\
\Leftrightarrow &\forall R \subseteq A^k\ (\mathcal{A}, R) \vDash \exists Q\ \theta \\
\Leftrightarrow &\forall |\text{bin}(R)| \leq |\text{poly}(\text{bin}(\mathcal{A}))|\ \psi(\text{bin}(\mathcal{A}) \frown \text{bin}(R)) \qquad \psi \in \Sigma^{\text{poly}}_1 \\
\Leftrightarrow &\forall |c| \leq \text{poly}(|\text{bin}(\mathcal{A})|)\ \psi(\text{bin}(\mathcal{A}), c)
\end{aligned}$$

Sostanzialmente quantificare sulle relazioni è come quantificare sulle stringhe che le codificano, che sono limitate polinomialmente[99].

□

Come si connettono queste gerarchie con quella polinomiale?

Teorema 17.4. $\mathsf{PH}_k \equiv \Sigma^{\text{poly}}_k$.

Il caso $k = 1$ è gratis dalle definizioni e da quanto già visto. Vediamo il caso $k = 2$, gli altri si fanno alla stessa maniera.

Teorema 17.5. $\mathsf{PH}_2 \equiv \Sigma^{\text{poly}}_2$, cioè $\mathsf{NP}^{\mathsf{NP}} = \Sigma^{\text{poly}}_2$.

Dimostrazione. Sia $S \in \mathsf{NP}^{\mathsf{NP}}$, dove $S \subset \{0,1\}^*$. Vorremmo scrivere $\mathcal{A} \in S$ nella forma $\exists^{\text{poly}} \forall^{\text{poly}}$. Per definizione sappiamo che esiste un oracolo $B \in \mathsf{NP}$ tale che[100] $\mathcal{A} \in S \Leftrightarrow M^B(\mathcal{A})$ accetta in[101] $\text{poly}|\mathcal{A}|$, dove M è nondeterministica con oracolo. L'oracolo B lo possiamo vedere come problema

[99] Stiamo supponendo che il linguaggio di $\mathcal{A}$ contenga almeno una relazione oltre alle $\leq, +, \cdot$, altrimenti la codifica di $\mathcal{A}$ potrebbe essere troppo corta e far saltare la polinomialità. Comunque basta aggiungere ad ogni linguaggio la relazione vuota di default.

[100] Qui in realtà dovremmo aggiungere un po' di bin in giro, ma manteniamo la notazione un po' più snella.

[101] Qui $|\mathcal{A}|$ è indifferentemente la cardinalità del dominio di $\mathcal{A}$ o la lunghezza della sua codifica, tanto stiamo supponendo che nel linguaggio ci sia almeno una relazione e il "polinomiale" fa sparire le differenze fra cose come n ed n^2.

$\subseteq \{0,1\}^*$, ma anche come una funzione $B\colon \{0,1\}^* \to \{0,1\}$ passando alla funzione caratteristica. Per brevità scriviamo $M^B(\mathcal{A}) \downarrow$ al posto di tutto il papocchio sopra, includendo nella grafia il bound temporale; inoltre per lo stesso motivo scriveremo i quantificatori illimitati, senza il bound, cioè ad esempio scriveremo $\forall x$ invece di $\forall |x| \leq |\dots|$ eccetera, e ci cureremo dei bound alla fine.

Abbiamo $M^B(\mathcal{A}) \downarrow$ se e solo se esistono un sottoinsieme finito[102] $f \subset B$ ed una computazione c tali che valga $\psi(c, \mathcal{A}, f)$, per l'opportuna $\psi \in \mathsf{PTIME}$. Scriviamo tutto come

$$(\exists f)(\exists c)(\psi(c, \mathcal{A}, f) \wedge f \subset B)$$

Se ora riusciamo a scrivere $f \subset B$ nella forma $\exists\forall$ abbiamo finito. Dato che[103] $B \in \mathsf{NP}$ abbiamo

$$u \in B \Leftrightarrow \exists w\ P(u, w)$$

con P predicato in PTIME, w certificato e dove $\exists w$ vuol dire $\exists |w| \leq \text{poly}|u|$. Ora dire $f \subset B$, dato che B la stiamo pensando come funzione, vuol dire che f è una restrizione di B, cioè

$$(\forall u \in \text{dom}(f))(f(u) = 1 \to B(u) = 1 \wedge f(u) = 0 \to B(u) = 0)$$

e dato che $B(u) = 1$ vuol dire $u \in B$ e per quanto detto sopra, diventa

$$(\forall u \in \text{dom}(f))(f(u) = 1 \to \exists w\ P(u, w) \wedge fu = 0 \to \forall w \neg P(u, w))$$

che è a sua volta equivalente a

$$(\exists (w(u))_{u \in \text{dom}\, f})(\forall u \in \text{dom}\, f)((fu = 1 \to P(u, w(u))) \wedge (fu = 0 \to \forall w \neg P(u, w)))$$

dove quella $w(u)$ è una funzione. Questa è una formula $\exists\forall$ a meno di metterla in forma prenessa[104]. Resta da vedere che tutti i quantificatori sono limitati polinomialmente.

- c era una computazione della macchina di Turing, che lavorava in tempo polinomiale in $\mathcal{A}$ e quindi $|c| \leq \text{poly}|\mathcal{A}|$.
- L'oracolo può essere consultato al massimo una volta per ogni passo della computazione, quindi[105] $|\text{dom}\, f| \leq \text{poly}|\mathcal{A}|$.

[102] L'idea è che visto che la computazione termina l'oracolo è stato consultato un numero finito di volte.

[103] Qui se invece di NP scriviamo PH_{100} abbiamo il passo induttivo numero 101.

[104] Il secondo $\forall$ è nella parte positiva dell'implicazione e quindi torna tutto.

[105] Non stimiamo direttamente con $|c|$ perché magari invece di codificare in c tutta la computazione vogliamo codificarci solo il certificato (le scelte nondeterministiche fatte).

- $|u| \leq \text{poly}|\mathcal{A}|$, perché prima di poter invocare l'oracolo su u bisogna scriverlo, e quindi è sempre soggetto a un bound dato dalla lunghezza della computazione.

- $|w| \leq \text{poly}|u|$ perché $u \in B$ se e solo se $\exists^{\text{poly}}|w|P(u,w)$ e quindi il bound polinomiale è praticamente già nella definizione

- $|w(u)| \leq \text{poly}|\mathcal{A}|$ perché l'intera successione[106] dei $w(u)$ è una successione di lunghezza $\operatorname{dom} f$, e quindi polinomiale in $\mathcal{A}$, e in cui ogni oggetto ha lunghezza polinomiale in u, e quindi in $\mathcal{A}$.

□

Nel caso più generale

Dimostrazione del Teorema 17.4. Come prima salvo che $u \in B$ è $\Sigma_{k-1}^{\text{poly}}$ e $\mathsf{PH}_k = \mathsf{NP}^{\mathsf{PH}_{k-1}}$. Usando l'ipotesi induttiva l'oracolo è $\Sigma_{k-1}^{\text{poly}}$ e la P di prima invece di essere in P è in Π_{k-2}^{poly}. Di conseguenza $f \subseteq B$ è Σ_k^{poly}. □

Corollario 17.6. $\mathsf{PH} = \mathsf{SO}$.

Dimostrazione.

$$\mathsf{PH} = \bigcup_{k\in\mathbb{N}} \mathsf{PH}_k = \bigcup_{k\in\mathbb{N}} \Sigma_k^{\text{poly}} = \bigcup_{k\in\mathbb{N}} \Sigma_k^1 = \mathsf{SO}$$

□

C'è un'ulteriore equivalenza, che è col numero di alternanze, la quale è abbastanza ovvia perché è il numero di alternanze di quantificatori, cioè

$$\mathsf{NP} \subseteq \mathsf{PH} \subseteq \mathsf{APTIME}$$

e, intendendo con ALT che la macchina può cambiare da uno stato $\exists$ a uno $\forall$ o viceversa c volte,

$$\mathsf{PH} = \bigcup_{k,c} \mathsf{ATIME\text{-}ALT}(n^k, c)$$

18 01/12

Importante: la data di gennaio per l'esame è il 12. Ore 9:30 nello studio di Berarducci.

[106] Per qualche perverso motivo si è finiti per denotare con w un elemento e con $w(u)$ una funzione, e non viceversa.

18.1 Il Teorema di Metodologia

Questo sull'Immerman è a pagina 127. Il succo è "come mostrare che una classe K di strutture non è Σ^1_1-monadica[107]". Per i grafi connessi l'avevamo già visto usando i giochi di Ehrenfeucht-Fraïssé; questa tecnica si generalizza.

Teorema 18.1 (Aytai-Fagin)**.** Una classe K di strutture non è Σ^1_1-monadica se e solo se il giocatore $\exists$ vince il seguente gioco; le regole sono:

- $\forall$ sceglie due numeri naturali c, m che rappresentano la complessità della formula che dovrebbe attestare la Σ^1_1-monadicità, nel senso che dovrebbe avere l'aspetto[108] $\varphi \equiv \exists R^{(1)}_1, \ldots, R^{(1)}_c \theta$, con $\mathrm{rk}(\theta) = m$.
- $\exists$ sceglie $\mathcal{A} \in K$
- $\forall$ sceglie relazioni 1-arie $R^{\mathcal{A}}_1, \ldots, R^{\mathcal{A}}_c \subset A$
- $\exists$ sceglie $\mathcal{B} \notin K$ e $R^{\mathcal{B}}_1, \ldots, R^{\mathcal{B}}_c \subset B$
- ora comincia il gioco G_m di Ehrenfeucht-Fraïssé a m mosse fra le strutture $\mathcal{A}$, $\mathcal{B}$ espanse con le nuove relazioni. Come sappiamo $\exists$ vince G_n se e solo se
$$(\mathcal{A}, R^{\mathcal{A}}_1, \ldots, R^{\mathcal{A}}_c) \sim_m (\mathcal{B}, R^{\mathcal{B}}_1, \ldots, R^{\mathcal{B}}_c)$$

Dimostrazione. [109] Mostriamo il "$\Leftarrow$". Supponiamo che $K \in \Sigma^1_1\text{-}\mathsf{MON}$ e mostriamo che $\forall$ vince. Per ipotesi abbiamo

$$K = \mathrm{Mod}(\exists R_1, \ldots, R_c\, \theta)$$

con θ di rango di quantificazione m. Chiaramente $\forall$ sceglierà c ed m come i c ed m di questa formula. A questo punto $\exists$ sceglie $\mathcal{A} \in K$ (se non può farlo ha perso), e $\forall$ sceglie $R^{\mathcal{A}}_1, \ldots, R^{\mathcal{A}}_c \subset A$ tali che

$$(\mathcal{A}, R^{\mathcal{A}}_1, \ldots, R^{\mathcal{A}}_c) \vDash \theta(R_1, \ldots, R_c)$$

cosa possibile perché $\mathcal{A} \in K$. Ora $\exists$ sceglie $\mathcal{B} \notin K$ e $R^{\mathcal{B}}_1, \ldots, R^{\mathcal{B}}_c \subset B$, dopodiché comincia il gioco di Ehrenfeucht-Fraïssé "normale". Ma qui sappiamo già che vince $\forall$, altrimenti per costruzione avremmo l'assurdo $\mathcal{B} \in K$.

Vediamo il "$\Rightarrow$". Supponiamo che $K \notin \Sigma^1_1\text{-}\mathsf{MON}(\tau)$ e mostriamo che $\exists$ vince. Di nuovo, $\forall$ sceglie c, m. Per descrivere la mossa di $\exists$ serve qualche considerazione preliminare. Sia Φ un insieme finito massimale di

[107] Visti i problemi con la forma prenessa quando diciamo "monadico" intendiamo che il quantificatore è all'inizio.

[108] Se invece vogliamo mostrare che una cosa non è Σ^1_1 tout-court bisogna aggiungere qui altri numeri che indicano l'arietà dei quantificatori.

[109] Nota: c'è un typo negli appunti di Berarducci, le frecce di questa dimostrazione sono al contrario.

formule del primo ordine di rango di quantificazione m nel linguaggio $\sigma = \tau \cup \{R_1, \ldots, R_c\}$ a due a due non equivalenti[110]. Definiamo

$$\exists\Phi = \{\exists R_1, \ldots, R_c\ \varphi \mid \varphi \in \Phi\} \qquad \Gamma = \{\gamma \in \exists\Phi \mid \forall \mathcal{B} \notin K\ \mathcal{B} \vDash \neg\gamma\}$$

(se Γ è vuoto funziona tutto lo stesso) e poi poniamo[111] $\theta = \bigvee \Gamma \in \Sigma^1_1\text{-MON}$. Per ipotesi $K \neq \mathrm{Mod}(\theta)$, ma $\mathrm{Mod}(\theta) \subset K$, perché se $\mathcal{B} \vDash \theta$ esiste $\gamma \in \Gamma$ tale che $\mathcal{B} \vDash \gamma$, e per definizione di Γ abbiamo $\mathcal{B} \in K$. Dunque $\mathrm{Mod}(\theta) \subsetneq K$. Finito questo preambolo, $\exists$ sceglie $\mathcal{A} \in K$ tale che $\mathcal{A} \nvDash \theta$. Poi $\forall$ gioca $R_1^{\mathcal{A}}, \ldots, R_c^{\mathcal{A}} \subset A$. Consideriamo $\varphi_0 \in \mathsf{FO}$ "completa"[112] fra quelle di rango di quantificazione m tale che $(\mathcal{A}, R_1^{\mathcal{A}}, \ldots, R_c^{\mathcal{A}}) \vDash \varphi_0$. Ora, dato che $\mathcal{A} \nvDash \theta \equiv \bigvee \Gamma$, allora $\mathcal{A}$ non verifica nessuna delle γ. D'altra parte $\mathcal{A} \vDash \exists R_1, \ldots, R_c\ \varphi_0$, per cui questa formula non sta in Γ. Per definizione allora esiste $\mathcal{B} \notin K$ tale che $\mathcal{B} \vDash \exists R_1, \ldots, R_c\ \varphi_0$. Chiaramente $\exists$ sceglie questa $\mathcal{B}$ e relazioni $R_1^{\mathcal{B}}, \ldots, R_c^{\mathcal{B}}$ che testimoniano che $\mathcal{B}$ verifica la formula sopra. Dato che φ_0 è "completa", abbiamo

$$(\mathcal{B}, R_1^{\mathcal{B}}, \ldots, R_c^{\mathcal{B}}) \sim_m (\mathcal{A}, R_1^{\mathcal{A}}, \ldots, R_c^{\mathcal{A}})$$

Infatti se così non fosse ci sarebbe φ' di rango di quantificazione m che le distingue, e dato che $\varphi_0 \to \varphi'$ oppure $\varphi_0 \to \neg\varphi'$ anche φ_0 le distinguerebbe. □

Dunque se ad esempio $\mathsf{co\text{-}NP} \neq \mathsf{NP}$, una maniera per dimostrarlo potrebbe essere con i giochi di Ehrenfeucht-Fraïssé, ad esempio mostrando che co-SAT non sta in NP.

18.2 QSAT

Definizione 18.2. QSAT è l'insieme delle formule proposizionali con quantificatori su proposizioni soddisfacibili (vere[113]).

Ad esempio una formula in QSAT è $\forall A\ \exists B\ (A \to \neg B)$.

Teorema 18.3. QSAT è PSPACE-completo rispetto a riduzioni polinomiali (addirittura rispetto a riduzioni FO).

È facile ricondurre QSAT a un sacco di roba, come ad esempio il Go, che quindi è PSPACE-completo. Un altro è Geography: si gioca su un grafo, si

[110] Cfr. Lemma 4.3.

[111] Dato che $\vee$ distribuisce su $\exists$ rimaniamo in $\Sigma^1_1\text{-MON}$.

[112] Nel senso che, dato che di formule di rango fissato sono finite, prendiamo quella che dà più informazioni sulla struttura, cioè la congiunzione di tutte quelle in Φ vere per $(\mathcal{A}, R_1^{\mathcal{A}}, \ldots, R_c^{\mathcal{A}})$.

[113] In questo caso non c'è molta differenza: basta quantificare esistenzialmente all'inizio. Ad esempio dalla formula soddisfacibile $A \to \neg B$ si ottiene la formula vera $\exists A, B\ (A \to \neg B)$

parte con una pedina su un vertice, a turno si sposta la pedina su un vertice adiacente alla posizione attuale della pedina, e perde il primo che muove su un vertice già visitato[114].

Nel contesto "formule proposizionali quantificate" come connettivo basta $\to$. Ad esempio possiamo definire

$$\begin{gathered}\bot \equiv (\forall B)B \\ \neg A \equiv \forall B(A \to B) \\ A \vee B \equiv \forall C((A \to C) \to ((B \to C) \to C))\end{gathered}$$

Dimostrazione del Teorema 18.3. Sia $K \in \mathsf{PSPACE}$ ed esibiamo $f \in \mathsf{PTIME}$ tale che $\mathcal{A} \in K$ se e solo se $f(\mathcal{A}) \in$ QSAT. Per ipotesi esistono k ed M tali che $M \in \mathsf{SPACE}(n^k)$ riconosce K, cioè $M(\mathcal{A})$ accetta se e solo se $\mathcal{A} \in K$. Sia $G_{\mathcal{A}}$ il grafo delle computazioni di M su input $\mathcal{A}$, che è lineare. Dato che la memoria utilizzata è limitata da n^k ci sono al più $O(1)^{n^k}$ configurazioni che scriviamo come 2^{n^k+S} (quell'S è perché in una configurazione bisogna anche segnarsi stato e posizione della testina). Scriviamo[115] $A \vdash B$ per indicare che dalla configurazione A passiamo alla B in un passo. Definiamo

$$\psi_0(A, B) \equiv (A \vdash B \vee A = B)$$

che, contando che la configurazione la codifichiamo in base 2, sarebbe

$$\psi_0(A_1, \ldots, A_{n^k+S}, B_1, \ldots, B_{n^k+S})$$

Chiaramente come $A = B$ intendiamo $\bigwedge A_i = B_i$, mentre $A \vdash B$ dipende da M ma è comunque scrivibile[116]. Vogliamo definire induttivamente un'opportuna $\psi_i(X, Y)$ che esprima il fatto che $X \vdash_{\leq 2^i} Y$. Fatto questo basta dire che $M(\mathcal{A})$ accetta se e solo se

$$\psi_{n^k}(\text{Confg. Iniziale}(\mathcal{A}), \text{Config. Accettante})$$

e vogliamo scrivere questo con una formula proposizionale quantificata *corta* (tempo polinomiale). Costruiamo le ψ_i per induzione, dove il passo base l'abbiamo già fatto. La prima idea sarebbe ripassare dall' "indovinare la configurazione intermedia" come abbiamo già fatto in passato[117]:

$$\psi_{i+1}(X, Y) \equiv \exists Z\ \psi_i(X, Z) \wedge \psi_i(Z, Y)$$

è facile vedere che questo genera un albero binario di altezza n^k, con quindi 2^{n^k} foglie: decisamente troppo grosso. La cosa furba è fare in modo che

[114] Chicca: il nome Geography viene da quel vecchio gioco per bambini da farsi tipicamente in macchina in cui bisogna elencare nomi di città che comincino con l'ultima lettera del nome prima.

[115] In questo caso A non è più necessariamente il dominio di $\mathcal{A}$.

[116] L'abbiamo già fatto (o spiegato come si fa) in passato.

[117] Cfr. Teorema 8.4.

ψ_i compaia una volta sola[118] nella definizione induttiva, in maniera da non avere biforcazioni. A meno di equivalenza infatti

$$\psi_{i+1}(X,Y) \equiv \exists Z\, \forall U,V[((U = X \wedge V = Z) \vee (U = Z \wedge V = Y)) \to \psi_i(U,V)]$$

che è lunga $\text{poly}(n^k) + |\psi_i|$. Ora non c'è più nessun albero che biforca, e induttivamente la lunghezza di ψ_{n^k} e controllata da $\text{poly}(n^k) \cdot n^k = \text{poly}'(n^k)$. In conclusione basta porre

$$\theta_{\mathcal{A}} \equiv \psi_{n^k}(\text{Confg. Iniziale}(\mathcal{A}), \text{Config. Accettante})$$

e abbiamo $\mathcal{A} \in K \Leftrightarrow \theta_{\mathcal{A}} \in \text{QSAT}$, per cui $K \leq_f \text{QSAT}$, dove $f(\mathcal{A}) = \theta_{\mathcal{A}}$, ed $f \in \mathsf{PTIME}$ perché θ è "corta e facile da costruire". □

19 04/12

19.1 Ripasso

Nel Teorema 8.4 avevamo visto che $\mathsf{NSPACE}(s(n)) \subseteq \mathsf{ATIME}(s(n)^2)$. [ripasso della dimostrazione, non trascritto] Lo "scrivere $s(n)$, che è la cosa che fa perdere tempo, serve a chiamare ricorsivamente la subroutine P (bisogna scriversi su cosa la si chiama). Altra nota: Lo spazio usato da una subroutine viene riutilizzato appena sono stati chiamati i suoi figli.

Nota: uno può usare anche le *macchine a registri* in alternativa alle macchine di Turing, ma forse ci vuole un po' di cautela con i tempi (se diciamo che un registro può essere copiato in un altro in un passo a prescindere dalla sua dimensione...).

Rivediamo anche $\mathsf{ATIME}(t(n)) \subseteq \mathsf{DSPACE}(t(n))$, con una dimostrazione diversa.

Dimostrazione. Sia A una macchina di Turing alternante che lavora in tempo $t(n)$ (con le dovute ipotesi su $t(n)$, tipo time-costruibile o quello che è). Su input w formiamo il grafo alternante[119] delle configurazioni di A con input w. Questo ha come vertici le configurazioni e c'è un arco da un vertice all'altro se si può passare da una configurazione all'altra in un passo. Ci sono anche dei predicati per dire di che tipo è la configurazione (esistenziale, universale), un nodo iniziale I_w e degli stati finali (se sono tanti può essere comodo fare con un predicato invece che con una costante) accettanti/rifiutanti. Quindi è una cosa del tipo

$$G_w = (V, E, G_\forall, G_\exists, I_w, \text{sì}, \text{no})$$

[118] Cfr. Esercizio 5.3.

[119] Nel senso che ha i nodi etichettati con "$\exists$" o "$\forall$".

I nodi esistenziali accettano se almeno uno dei figli accetta, mentre un nodo universale accetta se tutti i figli accettano e ha almeno un figlio. Per definizione A accetta w se e solo se $E_{\text{alt}}(I_w, \text{sì})$, dove definiamo E_{alt} come la minima, nel senso di minimo punto fisso, che verifica

$$\begin{aligned} E_{\text{alt}}(x,y) \equiv &\big[x = y \vee \exists z[E(x,z) \wedge G_\exists(x) \wedge E_{\text{alt}}(z,y)\big] \\ &\vee \big[\exists z\ E(x,z) \wedge G_\forall(x) \wedge \forall z'[E(x,z') \to E_{\text{alt}}(z',y)\big] \end{aligned}$$

Nota: G_w è un DAG, cioè non ha cicli (altrimenti la definizione di E_{alt} sarebbe circolare). DAG sta per *Directed Acyclic Graph*. Questo non è scontato perché la macchina può andare in loop, ed è qui che serve la time-costruibilità, per avere un contatore che a un certo punto termina la macchina (e quindi, visto che c'è il contatore che va giù, non si ripassa mai dalla stessa computazione). Qua va un attimo sistemato, visto che in realtà la macchina è in ASPACE non in ATIME, comunque è vero il fatto che il grafo è un DAG. Il punto è che diciamo che la macchina lavora in spazio $s(n)$, ma comunque lavora in un tempo $t(n)$ finito.

[non ho trascritto tutti i commenti]

Dato che abbiamo limitazioni sullo spazio, non possiamo segnarci tutta la computazione generata da E_{alt}; ci segnamo solo le scelte non deterministiche (e ci ricalcoliamo le cose all'uopo, tanto non abbiamo limiti di tempo). [fine ripasso non riportato] □

Avevamo anche detto che con lo stesso algoritmo si mostra $\mathsf{ASPACE}(s(n)) \subseteq \mathsf{DTIME}(O(1)^{s(n)})$. Per fare la stima si osserva che $X \in G_w \Rightarrow |X| \leq s(n)$, $|G_w| = O(1)^{s(n)}$, e che nessun nodo viene esplorato due volte.

20 11/12

20.1 Moar Ripasso e una Lieve Generalizzazione

Quanto segue è una lieve generalizzazione/precisazione di quanto visto nella Lezione 13.

Se $\mathcal{A}$ è una struttura relazionale gli associamo il suo *grafo di Gaifman* $G_{\mathcal{A}}$. Se $\mathcal{A}$ era già un grafo $G_{\mathcal{A}} = \mathcal{A}$. In ogni caso è definito così il dominio è lo stesso di $\mathcal{A}$

$$G_{\mathcal{A}} = (A, E^{\mathcal{A}} = A)$$

e c'è una relazione binaria E tale che $G_{\mathcal{A}} \vDash E(a,b)$ se e solo se a e b sono distinti e sono componenti una tupla $c_1, \ldots, c_r$ di A ed esiste $R \in L$ tale che $\mathcal{A} \vDash R(c_1, \ldots, c_r)$. Quindi in realtà se partiamo da un grafo orientato ne otteniamo uno non orientato (a e b possono essere scambiati per definizione). Definiamo poi

$$S(n,a) = \{b \in A \mid d_{G_{\mathcal{A}}}(a,b) \leq n\}$$

che abbiamo definito essenzialmente come una palla nel grafo di Gaifman, ma la pensiamo come sottostruttura di $\mathcal{A}$.

Ora supponiamo di avere due L-strutture "puntate" $\mathcal{A}, a$ e $\mathcal{B}, b$. Diciamo che

$$\mathcal{A}, a \overset{\text{G}}{\sim}_n \mathcal{B}, b$$

se e solo se

$$\langle S^{\mathcal{A}}(n, a), a\rangle \overset{L}{\cong} \langle S^{\mathcal{B}}(n, b), b\rangle$$

dove si intende che l'isomorfismo è di strutture puntate, cioè manda a in b. L'isomorfismo non è dei grafi di Gaifman ma della struttura[120]. Ora possiamo considerare la classe di equivalenza

$$\iota = [\mathcal{A}, a]_{\overset{\text{G}}{\sim}_n}$$

e la chiamiamo *n-tipo* di a in $\mathcal{A}$. Diciamo che $\mathcal{A} \sim_{3^n} \mathcal{B}$ se $\mathcal{A}$ e $\mathcal{B}$ hanno lo stesso numero di 3^n-tipi, nel senso che per ogni 3^n-tipo ι il numero di elementi di tipo ι in $\mathcal{A}$ è lo stesso che in $\mathcal{B}$; equivalentemente se esiste una bigezione f fra A e B che preserva i 3^n-tipi, cioè se il 3^n-tipo di $f(a)$ è lo stesso di a:

$$\langle S(3^n, a), a\rangle \cong \langle S(3^n, f(a)), f(a)\rangle$$

Occhio: l'isomorfismo non è detto che sia f; lei si occupa solo di testimoniare che c'è lo stesso numero di tipi.

Teorema 20.1 (Hanf)**.** Se $\mathcal{A} \sim_{3^n} \mathcal{B}$ allora $\mathcal{A} \equiv_n \mathcal{B}$.

In altre parole il rango di quantificazione esprime in un certo senso "quanto è locale" la formula, nel senso che che una formula di rango n "vede" solo le palle di raggio 3^n. Quello che avevamo visto nella Lezione 13 era il caso particolare in cui la struttura era già un grafo.

Dimostrazione. Intanto osserviamo che se $k < n$ allora $\mathcal{A} \sim_{3^n} \mathcal{B}$ implica $\mathcal{A} \sim_{3^k} \mathcal{B}$. Questo segue banalmente dal fatto che gli isomorfismi fra 3^n-palle possono essere ristretti e vanno anche bene come isomorfismi fra 3^k-palle. Poniamo

$$\langle S(3^j, a_1, \ldots, a_k)\rangle := \bigcup_{i=1}^{k} S(3^j, a_i)$$

Il giocatore "$\exists$" vuole preservare l'invariante che

$$\langle S(3^j, a_1, \ldots, a_k)\rangle \cong \langle S(3^j, b_1, \ldots, b_k)\rangle$$

Dopo k turni il raggio delle palle si è ristretto ma sono aumentati i centri e la costante è che $j + k = n$. All'inizio del gioco $k = 0$, e per convenzione le palle sono senza centro, cioè sono tutte il vuoto, e la tesi è vera per

[120] Infatti avevamo detto che S la pensavamo come sottostruttura di $\mathcal{A}$.

convenzione perché[121] $\emptyset \cong \emptyset$. Dato che l'isomorfismo è nel linguaggio L il gioco lo possiamo interpretare come partita di back-and-forth, quindi se $\exists$ resiste n mosse abbiamo la tesi $\mathcal{A} \equiv_n \mathcal{B}$; bisogna solo dimostrare che $\exists$ ci riesce.

Senza perdita di generalità[122] $\forall$ gioca $a_{k+1} = a \in A$.

Caso 1: $a \in S(2 \cdot 3^{j-1}, a_1, \ldots, a_k)$. Allora $S(3^{j-1}, a) \subseteq S(3^j, a_1, \ldots, a_k)$. Se π era l'isomorfismo fra le palle del passo prima allora ad $\exists$ basta scegliere $b = \pi(a)$. Ne segue[123] che

$$S(3^{j-1}, a_1, \ldots, a_k, a) \cong S(3^{j-1}, b_1, \ldots, b_k, b)$$

Caso 2: $a \notin S(2 \cdot 3^{j-1}, a_1, \ldots, a_k)$. Allora $S(3^{j-1}, a) \cap S(3^{j-1}, a_1, \ldots, a_k) = \emptyset$. Per l'ipotesi induttiva le palle $S(2 \cdot 3^{j-1}, a_1, \ldots, a_k)$ e $S(2 \cdot 3^{j-1}, b_1, \ldots, b_k)$ contengono lo stesso numero di elementi di tipo ι, dove ι è il 3^{j-1}-tipo di a. Quindi[124] esiste $b \notin S(2 \cdot 3^{j-1}, b_1, \ldots, b_k)$ con lo stesso 3^{j-1}-tipo di a. Questo perché globalmente c'era lo stesso numero di 3^{j-1}-tipi (addirittura di 3^n-tipi). Dato che dentro $S(2 \cdot 3^{j-1}, \ldots)$ ce n'è lo stesso numero deve essere lo stesso anche fuori. Abbiamo quindi

$$S(3^{j-1}, a_1, \ldots, a_k, a), \vec{a}, a \overset{\pi \cup \pi'}{\cong} S(3^{j-1}, b_1, \ldots, b_k, b), \vec{b}, b$$

dove con $\pi \cup \pi'$ si intende che π è il vecchio isomorfismo ristretto a $S(3^{j-1}, a_1, \ldots, a_k)$ e π' è un testimone del fatto che $\langle S(3^{j-1}, a), a \rangle$ e $\langle S(3^{j-1}, b), b \rangle$ sono isomorfe, cioè che a e b hanno lo stesso tipo di 3^{j-1}-palle. Dato che π, π' hanno domini disgiunti l'unione è ancora un isomorfismo.

□

Sul libro (Ebbinghaus-Flum, Finite Model Theory) ci sono ulteriori generalizzazioni. Comunque l'Immerman fa tutto con le palle di raggio 2^j invece che 3^j. Non è chiarissimo se funzioni.

Da questa cosa seguiva il Teorema 13.1. Rivediamo la

Dimostrazione. Supponiamo che $\varphi \in \Sigma^1_1\text{-}\mathsf{MON}$ sia tale che $\varphi = \exists P_1, \ldots, P_r\ \psi$ con ψ del primo ordine di rango di quantificazione m. Prendiamo $D_\ell = (V, E)$ grafo ciclico orientato[125] con $\ell \ggg m, r$ elementi. Consideriamo $(V, E, P_1, \ldots, P_r)$ e pensiamo l'insieme delle P_i soddisfatte da un nodo come colore del nodo (ci sono 2^r colori). Per ℓ sufficientemente grande[126] è

[121] Qui serve il fatto che il linguaggio non contenga simboli di costante.
[122] Il caso in cui gioca $b \in B$ è chiaramente simmetrico.
[123] "Ne segue che torna."
[124] "Questo "Quindi" va un attimo meditato."
[125] Possiamo prendere $E(i, i+1)$ con $i \in \mathbb{Z}/\ell\mathbb{Z}$.
[126] Che dipende da m e da r ma non da $P_1, \ldots, P_r$.

abbastanza evidente che esistono $a, b \in V$ con lo stesso 3^m-tipo e tali che $S(3^m, a) \cap S(3^m, b) = \emptyset$ e $\langle S(3^m, a), a\rangle \cong \langle S(3^m, b), b\rangle$. Ora si spezza il grafo[127] e si mostra che fra i due grafi vale $\equiv_m$. Ma allora φ non è vera in tutti e soli i grafi connessi. □

[segue una chiacchierata sul LFP e inizio ripasso di $\mathsf{DTIME}(O(1)^{s(n)}) = \mathsf{ASPACE}(s(n))$, non riportati]

21 15/12

21.1 Ripasso

[fine ripasso di $\mathsf{DTIME}(O(1)^{s(n)}) = \mathsf{ASPACE}(s(n))$] Sia M una macchina di Turing in $\mathsf{DTIME}(k^{s(n)})$

$$M\colon Q \times \Sigma \to Q \times \Sigma \times \{-1, 0, 1\}$$

Su input w di lunghezza n, consideriamo il grafo G_w delle computazioni di M su input w. Dato che M è determistica questo è in realtà è una matrice $k^{s(n)} \times k^{s(n)}$. Introduciamo il predicato $C(t, p, a)$, che vuol dire "al tempo t in posizione p c'è a", dove $a \in \Sigma$ oppure $a = \langle q, \sigma\rangle \in Q \times \Sigma$, a indicare che al tempo t la testina di M su input w si trova in posizione p nello stato q e legge $\sigma \in \Sigma$. Ora pensiamo le istruzioni di M come relazione $(a_{-1}, a_0, a_1) \vdash_M a$ (cosa succede ad a dipende solo da cosa c'è al passo prima nelle caselle adiacenti). Allora possiamo definire C induttivamente come

$$\begin{aligned} C(t+1, p, a) \Leftrightarrow \exists a_{-1}, a_0, a_1 \big[&(a_{-1}, a_0, a_1) \vdash_M a \\ &\wedge C(t, p-1, a_{-1}) \wedge C(t, p, a_0) \wedge C(t, p+1, a_1)\big] \end{aligned}$$

ovviamente bisogna anche dare il passo base: definiamo $C(0, p, a)$ se e solo se $p > 1$ e a è il p-esimo simbolo di w, oppure $p = 0$ e $a = \langle q_0, w_1\rangle$.

Ora simuliamo M con una macchina alternante. Questa essenzialmente calcolerà il predicato C. Usa un C-nastro (un nastro che chiamiamo C) dove ci scriviamo gli input del predicato C, e lo inizializza con quello che ci sarà sul nastro alla fine, cioè, supponendo che la testina alla fine si riporti sempre all'inizio del nastro e la macchina accetti, $\langle k^{s(n)}, 1, \langle \text{accetto}, \text{simbolo}\rangle\rangle$. La macchina alternante esegue il seguente ciclo. Supponiamo che sul C-nastro ci sia $\langle t+1, p, a\rangle$. Allora la macchina si mette in uno stato esistenziale dove "indovina" tre simboli che verificano $(a_{-1}, a_0, a_1) \vdash_M a$, dopodiché entra in uno stato universale in cui sceglie nondeterministicamente $i \in \{-1, 0, 1\}$ e fa il controllo di $C(t, p+i, a_i)$ e un quarto i in cui si controlla $(a_{-1}, a_0, a_1) \vdash_M a$. Sul C-nastro scrive $\langle t, p+i, a_i\rangle$ e torna al ciclo. Quando sul C-nastro c'è $\langle 0, p, a\rangle$ controlla se a è il p-esimo simbolo di w (o la coppia $\langle q_0, w_1\rangle$ se $p = 1$).

[127] Vedi vecchia dimostrazione, ci sono pure i disegni, evito di ricopiare tutto qui.

Vediamo quanto spazio abbiamo usato (ricordiamo che lo spazio va contato per un percorso accettante nella computazione; negli stati universali comunque va contato lo spazio solo per uno dei figli). Stimiamo la lunghezza di $\langle t, p, a\rangle$. Dato che $t \leq k^{s(n)}$ abbiamo $|t| \in O(s(n))$. Anche $p \leq k^{s(n)}$ e quindi anche $|p| \in O(s(n))$, mentre $|a|$ è controllato da una costante, quindi in tutto abbiamo $|\langle t, p, a\rangle| \in O(s(n))$.

Una piccola precisazione: $-1, 0, 1$ sono da intendersi in $\mathbb{Z}/k^{s(n)}$ (nel senso che non "sforiamo" dalla matrice presentata prima).

21.2 Riduzione First-Order di SAT a CLIQUE

Avevamo visto che SAT si riduce polinomialmente a CLIQUE. Vedremo ora che in realtà la riduzione è first-order. Ricordiamo che SAT sono le formule proposizionali in forma normale congiuntiva soddisfacibili, cioè come congiunzione di clausole, nel senso che $\varphi = c_1 \wedge \ldots \wedge c_n$ dove $c_i = \ell_1 \vee \ldots \vee \ell_k$ e ℓ_j può essere una variabile o la negazione di una variabile. Avevamo codificato φ in una struttura A_φ nel linguaggio $\{P^{(2)}, N^{(2)}\}$. Senza perdita di generalità possiamo supporre[128] che il numero di clausole sia lo stesso delle variabili. Dunque $A_\varphi = \langle n, P, N\rangle$, dove $P(c, v)$ vuol dire "v appare positivamente in c" e $N(c, v)$ vuol dire "v appare negata in c". Ad esempio se

$$\varphi = (x_1 \vee \neg x_2) \wedge (\neg x_2 \vee x_2)$$

allora vale $P(1, 1)$ e $N(1, 2)$. Dunque pensiamo SAT come $\{A_\varphi \mid \varphi \text{ soddisfacibile}\}$. Ora associamo a φ un grafo G_φ. Il dominio $|G_\varphi|$ è un sottoinsieme di $n \times n \times n$, quindi un insieme di triple di naturali: $\langle c, v, s\rangle \in n \times n \times n$, a indicare rispettivamente clausola, variabile e segno (quest'ultimo è chiaramente binario, diciamo 1 o 2). Quindi in realtà il dominio è $|G_\varphi| = n \times n \times \{1, 2\}$, cui aggiungiamo d'ufficio un punto "speciale", diciamo $\langle 1, 1, 3\rangle$, che battezziamo $\circledast$. Dunque, riscrivendo di nuovo chi è $|G_\varphi|$ per chiarezza,

$$|G_\varphi| = n \times n \times \{1, 2\} \cup \{\langle 1, 1, 3\rangle\}$$

Vediamo chi sono gli archi di G_φ. Abbiamo $E \subseteq n^6$, quindi è una formula in sei variabili, che però pensiamo a gruppi di tre[129]: $E(\langle c, v, s\rangle, \langle c', v', s'\rangle)$. Iniziamo a definire

$$E(\underbrace{\langle 1, 1, 3\rangle}_{\circledast}, \langle c, v, s\rangle) \Leftrightarrow [(s = + \wedge P(c, v)) \vee (s = - \wedge N(c, v))]$$

In altre parole $E(\circledast, -)$ dice com'è fatta φ. Quella a destra è una formula in $\mathsf{FO}(P, N)_\leq$, (il $\leq$ serve per 1, 2, cioè $+, -$) quindi la E la stiamo definendo

[128] A meno di aggiungere variabili fittizie o cose del genere.

[129] Le $\langle\rangle$ stanno lì solo per bellezza/comodità psicologica, ma neanche troppo: in fondo a CLIQUE dobbiamo dare in pasto un grafo, quindi un oggetto con una relazione binaria. Però i nodi del grafo (gli oggetti) sono triple.

dentro la struttura. Per gli altri nodi definiamo

$$E(\langle c, v, s\rangle, \langle c', v', s'\rangle) \Leftrightarrow \big[s, s' \leq 2 \wedge c \neq c' \wedge (v = v' \to s = s')\big] \\ \vee \big[s = 3 \wedge [(s' = + \wedge P(c', v')) \vee (s' = - \wedge N(c', v'))]\big]$$

Dopodiché simmetrizziamo chiedendo $E(\vec{x}, \vec{y}) \leftrightarrow E(\vec{y}, \vec{x})$ (basta scrivere una formula lunga il doppio con la seconda metà uguale ma a parti invertite). L'idea è che colleghiamo due cose se possiamo dare alle rispettive variabili un valore di verità coerente, mentre se una delle due è il nodo speciale allora colleghiamo se l'altro nodo compare veramente in φ.

Chiariamo questa storia della relazione binaria piuttosto che 6-aria: avremo $G_\varphi = (V, E)$, dove $L = \{E^{(2)}\}$. Abbiamo $(\operatorname{def} E) \in \mathsf{FO}(P, N)$, che è una formula in 6 variabili libere. $(\operatorname{def} V)$ è una formula con 3 variabili in $\mathsf{FO}(P, N)_\leq$ (in realtà in $\mathsf{FO}_\leq$), e le due definizioni non dipendono da φ (cioè vanno bene per tutte le φ) e vengono date in A_φ. Fissata φ poi si ha

$$G_\varphi = (V, E) \cong \langle (\operatorname{def} V)^{A_\varphi}, \operatorname{def} E^{A_\varphi}\rangle$$

Ora è abbastanza chiaro che

Teorema 21.1. $\varphi \in$ SAT se e solo se G_φ ha una $n+1$-cricca[130].

Cioè se e solo se $\langle G_\varphi, n+1\rangle \in$ CLIQUE. (quindi in realtà bisogna anche interpretare la costante che sarà $n+1$, ma basta aggiungere un predicato 1-ario[131] verificato solo da $n+1$.) C'è qualche dettaglio per il $\leq$, forse può semplificare le cose definire $\operatorname{def} V = n^3$, tanto tutti i nodi che non ci interessano non sono collegati a niente. Comunque mostriamo il Teorema:

Dimostrazione. Basta mostrare che

$$\{v \colon n \to \{0, 1\} \mid \varphi^v = \text{true}\}$$

cioè le valutazioni che rendono φ vera, è non vuoto se e solo se

$$\{n+1\text{-cricche in } G_\varphi\}$$

è non vuoto. Innanzitutto notiamo che qualunque $n+1$-cricca deve contenere $\circledast$, perché fra gli altri nodi ci sono collegamenti solo fra clausole diverse. Vediamo prima come passare da una cricca a un'assegnazione: è facile vedere che per costruzione, se ad esempio nella cricca c'è $\langle c', x, +\rangle$, nella valutazione si assegna "vero" a x. Per costruzione una variabile riceve uno e un solo valore, e sempre per costruzione φ è soddisfatta. Viceversa una valutazione che rende vera φ rende vero almeno un letterale per ogni clausola, e scegliendone uno è facile da qui produrre una cricca. □

[130] O "clicca", a seconda dei gusti.

[131] Questo è abbastanza overkill, nel senso che con un predicati 1-ari ci possiamo codificare i sottoinsiemi e quindi le successioni; nel libro è fatto in maniera più furba.

In altre parole abbiamo mostrato che la funzione $\mathrm{Struct}(P, N) \to \mathrm{Struct}(E, c)$ che associa $A_\varphi \mapsto (G_\varphi, n+1)$ (cioè E viene interpretato tramite la definizione di E nell'altra struttura) è una riduzione FO, perché $A_\varphi \in$ SAT se e solo se $(G_\varphi, n+1) \in$ CLIQUE. Dunque SAT si riduce a CLIQUE, che quindi ne eredita la NP-completezza.

Nota: che NP è chiuso per riduzioni first-order si può vedere ad esempio usando Fagin, oppure notando che le riduzioni FO sono polinomiali e che NP è chiuso per riduzioni polinomiali.

Per chi fa l'esame a dicembre la data è venerdì prossimo alle 15.

Riferimenti bibliografici

[1] Pagina del corso: `http://www.dm.unipi.it/~berardu/Didattica/2014-15TDC/`

[2] Heinz-Dieter Ebbinghaus, Jörg Flum, *Finite Model Theory*, Spinger Monographs in Mathematics,

[3] Neil Immerman, *Descriptive Complexity*, Graduate Text in Computer Science, Springer

[4] Christos Harilaos Papadimitriou, *Computational Complexity*, Addison Wesley Longman