Lezione14 – riducibilità polinomiale

Lezione del 23/04/2024

Relazioni interessanti, ma…

- La maggior parte delle relazioni fra classi complessità che abbiamo visto, fino ad ora, sono **inclusioni improprie**
- A parte P $\subset$ EXPTIME e PSPACE = NPSPACE
- Ossia, a parte queste due ultime relazioni, per ciascuna delle rimanenti relazioni non siamo in grado di dimostrare né l'inclusione propria né la coincidenza delle due classi che la costituiscono.
- Ad esempio, sappiamo che
  - tutti i linguaggi che sono in PSPACE sono anche in EXPTIME (PSPACE $\subseteq$ EXPTIME)
  - tutti i linguaggi che sono in P sono anche in NP (P $\subseteq$ NP)
- Ma non sappiamo rispondere alle seguenti domande
  - non sarà forse che tutti i linguaggi in EXPTIME sono anche in PSPACE? Ossia, che PSPACE **=** EXPTIME?
  - Oppure, esiste almeno un linguaggio in NP che **non** può essere deciso in tempo **deterministico** polinomiale? Ossia: è P $\subset$ NP oppure P = NP ????
- Le relazioni che conosciamo sono, in massima parte, relazioni deboli

Relazioni interessanti, ma…

- La maggior parte delle relazioni fra classi complessità che abbiamo visto, fino ad ora, sono **inclusioni improprie**
- A parte P ⊂ EXPTIME e PSPACE = NPSPACE
- le relazioni che conosciamo sono, in massima parte, relazioni deboli

- E, inoltre, pur riuscendo a dimostrare che una certa classe di complessità $\mathcal{C}_1$ è contenuta propriamente in un'altra classe di complessità $\mathcal{C}_2$ (ossia, $\mathcal{C}_1 \subset \mathcal{C}_2$)
- anche in questo caso, seppure dimostriamo che un certo linguaggio L appartiene a $\mathcal{C}_2$
- come facciamo a sapere se quel linguaggio è anche in $\mathcal{C}_1$ oppure se, invece, è un *linguaggio separatore* fra $\mathcal{C}_1$ e $\mathcal{C}_2$, ossia è contenuto in $\mathcal{C}_2$ - $\mathcal{C}_1$?

- Certo sarebbe utile se disponessimo di uno strumento che permettesse di individuare i ***linguaggi separatori*** fra due classi di complessità $\mathcal{C}_1$ e $\mathcal{C}_2$ …
  - ossia i linguaggi che, **nell'ipotesi $\mathcal{C}_1 \subset \mathcal{C}_2$**, appartengono a $\mathcal{C}_2$ ma non a $\mathcal{C}_1$

Una vecchia conoscenza…

- Ve le ricordate le care, vecchie, riduzioni? (paragrafo 5.5)
- Dati due linguaggi, $L_1 \subseteq \Sigma_1^*$ e $L_2 \subseteq \Sigma_2^*$, diciamo che **L_1 è *riducibile* a L_2** e scriviamo **$L_1 \leqslant L_2$** se
- Esiste una funzione $f : \Sigma_1^* \rightarrow \Sigma_2^*$ tale che
- 1) f è totale e calcolabile – ossia,
  - è definita per ogni parola $x \in \Sigma_1^*$ e, inoltre,
  - esiste una macchina di Turing di tipo trasduttore T_f tale che, per ogni parola $x \in \Sigma_1^*$, la computazione $T_f(x)$ termina con la parola $f(x) \in \Sigma_2^*$ scritta sul nastro di output
- 2) per ogni $x \in \Sigma_1^*$ vale che: $x \in L_1$ se e solo se $f(x) \in L_2$
  - **$\forall x \in \Sigma_1^* [x \in L_1 \leftrightarrow f(x) \in L_2]$**
- Ora, aggiungiamo una *piccola* richiesta alla funzione di riduzione f

... rivisitata

- Sia π un predicato definito sull'insieme delle funzioni totali e calcolabili – ossia, una proprietà, che deve essere posseduta da una funzione ad esempio:
  - per ogni $x \in \Sigma_1^*$, $|f(x)| = |x|$
  - per ogni $x \in \Sigma_1^*$, f è calcolabile in tempo polinomiale in $|x|$
- Dati due linguaggi, $L_1 \subseteq \Sigma_1^*$ e $L_2 \subseteq \Sigma_2^*$, diciamo che **L_1 è π-riducibile a L_2** e scriviamo **$L_1 \preccurlyeq_\pi L_2$** se esiste una funzione $f : \Sigma_1^* \rightarrow \Sigma_2^*$ tale che
  - 1) f è totale e calcolabile, ossia
    - è definita per ogni parola $x \in \Sigma_1^*$ e, inoltre,
    - esiste una macchina di Turing di tipo trasduttore T_f tale che, per ogni parola $x \in \Sigma_1^*$, la computazione $T_f(x)$ termina con la parola $f(x) \in \Sigma_2^*$ scritta sul nastro di output
  - 2) per ogni $x \in \Sigma_1^*$ vale che: $x \in L_1$ se e solo se $f(x) \in L_2$
  - 3) f soddisfa π

Chiusura di una classe rispetto a $\preccurlyeq_\pi$

- Lo strumento che potrebbe permettere di individuare i linguaggi separatori fra due classi di complessità è basato sui seguenti due concetti che si riferiscono alle π-riduzioni
  - **chiusura** di una classe rispetto a una π-riduzione
  - **completezza** di un linguaggio per una classe rispetto a una π-riduzione

- **Definizione 6.4**: Una classe di complessità C è **chiusa** rispetto ad una generica π-riduzione se, *per ogni coppia di linguaggi* L_1 *ed* L_2 *tali che* $L_1 \preccurlyeq_\pi L_2$ *e* $L_2 \in C$, *si ha che* $L_1 \in C$.
- **La chiusura di una classe C rispetto ad una π-riduzione può essere utilizzata per dimostrare l'appartenenza di un linguaggio L a C**:
  - segue direttamente dalla definizione che, se sappiamo che una classe di complessità C è chiusa rispetto ad una π-riduzione e che un certo linguaggio L_0 appartiene a C, allora, se dimostriamo che $L \preccurlyeq_\pi L_0$, possiamo dedurre che anche L appartiene a C.

Completezza di un linguaggio per una classe rispetto a $\preccurlyeq_\pi$

- **Definizione 6.3**: Sia C una classe di complessità di linguaggi e sia $\preccurlyeq_\pi$ una generica π-riduzione.

  Un linguaggio $L \subseteq \Sigma^*$ e` **C-completo rispetto alla π-riducibilità** se:

  a) $L \in C$

  e

  b) per ogni altro $L_0 \in C$, vale che $L_0 \preccurlyeq_\pi L$.

- Le nozioni di
  - completezza di un linguaggio per una classe rispetto ad una π-riduzione
  - chiusura di una classe rispetto alla π-riduzione
- sono gli strumenti che ci permettono di arrivare al concetto di linguaggio ''più difficile'' in una classe

Il linguaggio ‘‘più difficile’’ di una classe

- Abbiamo due classi di complessità C_1 e C_2 tali che $\mathbf{C_1 \subseteq C_2}$,
- e sappiamo che **C_1 è chiusa** rispetto ad una qualche π-riduzione:
  - allora, per ogni coppia di linguaggi L_1 ed L_2 tali che $\mathbf{L_1 \preccurlyeq_\pi L_2}$ **e** $\mathbf{L_2 \in C_1}$, si ha che $L_1 \in C_1$.
- Se per caso troviamo un linguaggio **L C_2–completo** rispetto a $\preccurlyeq_\pi$
  - ossia, $\mathbf{L \in C_2}$ **e per ogni altro** $\mathbf{L_0 \in C_2}$**, vale che** $\mathbf{L_0 \preccurlyeq_\pi L}$
- e **se dimostriamo che** $\mathbf{L \in C_1}$
- abbiamo che: **per ogni altro** $\mathbf{L_0 \in C_2}$**, vale che** $\mathbf{L_0 \preccurlyeq_\pi L}$ e inoltre $\mathbf{L \in C_1}$
- allora, in virtù della chiusura di C_1 rispetto alla π-riduzione,

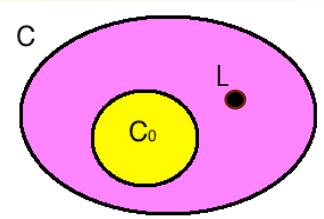
  per ogni altro $\mathbf{L_0 \in C_2}$**, vale che** $\mathbf{L_0 \in C_1}$
- **e, dunque,** $\mathbf{C_1 = C_2}$

Il linguaggio ‘‘più difficile’’ di una classe

- Riassumendo: abbiamo due classi di complessità C_1 e C_2 tali che $C_1 \subseteq C_2$,
- e sappiamo che **C_1 è chiusa** rispetto ad una qualche π-riduzione
- Se per caso trovassimo un linguaggio **L C_2–completo** rispetto a $\preccurlyeq_\pi$ allora
  - **da un ipotetico algoritmo che decide L utilizzando una quantità di risorse pari a quella che definisce la classe C_1 – cioè, se dimostrassimo che L $\in$ C_1**
  - **potremmo dedurre un algoritmo che decide qualunque problema in C_2 utilizzando una quantità di risorse pari a quella che definisce la classe C_1**
- Allora, **se riuscissimo a dimostrare che L $\in$ C_1 sapremmo automaticamente che tutti i linguaggi in C_2 sono anche in C_1 - ossia sapremmo che $C_1 = C_2$**
- Ma possiamo vederla anche in un altro modo: se $C_1 \subseteq C_2$ e **L è C_2–completo** e **se qualcuno riuscisse a dimostrare che $C_1 \neq C_2$** allora **sapremmo automaticamente che L $\notin$ C_1**
- L sarebbe un linguaggio ‘‘più difficile’’ fra tutti i linguaggi che stanno in C_2

Il linguaggio ‘‘più difficile’’ di una classe

- Abbiamo due classi di complessità C_1 e C_2 tali che $\mathbf{C_1 \subseteq C_2}$, e sappiamo che **$C_1$ è chiusa rispetto ad una qualche π-riduzione**
- Se per caso trovassimo un linguaggio L C_2–completo rispetto a $\preccurlyeq_\pi$ e
- **se qualcuno riuscisse a dimostrare che $C_1 \neq C_2$** allora **sapremmo automaticamente che $L \notin C_2$**
- Infatti:
- **Teorema 6.20**: Siano C e C_0 due classi di complessità tali che $C_0 \subseteq C$. Se C_0 è chiusa rispetto ad una π-riduzione allora, per ogni linguaggio L che sia C-completo rispetto a $\preccurlyeq_\pi$, $L \in C_0$ se e solo se $C = C_0$.
  - **Se $C = C_0$** , poiché L è C-completo e, dunque $L \in C$, allora $L \in C_0$.
  - **Viceversa, supponiamo che $L \in C_0$**. Poiché L è C completo rispetto a $\preccurlyeq_\pi$, allora, per ogni $L' \in C$, $L' \preccurlyeq_\pi L$.
    Poiché C_0 è chiusa rispetto a $\preccurlyeq_\pi$, allora, per ogni $L' \in C$, $L' \in C_0$: quindi, $C = C_0$.



Una particolare π-riduzione

- Dati due linguaggi, $L_1 \subseteq \Sigma_1^*$ e $L_2 \subseteq \Sigma_2^*$, diciamo che
  L_1 è *polinomialmente riducibile* a L_2 e scriviamo **$L_1 \preccurlyeq_p L_2$** se
- Esiste una funzione $f : \Sigma_1^* \to \Sigma_2^*$ tale che
- 1) f è totale e calcolabile in tempo polinomiale (in breve, $f \in FP$) – ossia,
  - è definita per ogni parola $x \in \Sigma_1^*$ e, inoltre,
  - esiste una macchina di Turing di tipo trasduttore T_f tale che, per ogni parola $x \in \Sigma_1^*$, la computazione $T_f(x)$ termina con la parola $f(x) \in \Sigma_2^*$ scritta sul nastro di output
  - esiste una costante c tale che: per ogni $x \in \Sigma_1^*$, **$dtime(T_f,x) \in O(|x|^c)$**
- 2) per ogni $x \in \Sigma_1^*$ vale che: $x \in L_1$ se e solo se $f(x) \in L_2$
  - **$\forall x \in \Sigma_1^* [x \in L_1 \leftrightarrow f(x) \in L_2]$**
- E siamo al paragrafo 6.8:
  - come sulla dispensa, d'ora in poi scriveremo sempre $\preccurlyeq$ invece di $\preccurlyeq_p$

Il nuovo strumento*

- Abbiamo due linguaggi, $L_1 \subseteq \Sigma_1^*$ e $L_2 \subseteq \Sigma_2^*$,
- e riusciamo a dimostrare che $\mathbf{L_1 \preccurlyeq_p L_2}$ (anzi, come abbiamo detto, leviamo la p: $\mathbf{L_1 \preccurlyeq L_2}$)
  - cioè, dimostriamo che esistono un trasduttore T_r e una costante c tali che, per ogni $x \in \Sigma_1^*$, e, inoltre, per ogni $x \in \Sigma_1^*$, $dtime(T_r, x) \in O(|x|^c)$
- Supponiamo di sapere che $L_2 \in DTIME[\ f(n)\ ]$
  - cioè, esiste un riconoscitore T_2 tale che, per ogni $y \in \Sigma_2^*$, $T_2(y)$ accetta se e soltanto se $y \in L_2$ e, inoltre, per ogni $y \in \Sigma_2^*$, $dtime(T_2, y) \in O(f(|y|))$
- Allora, possiamo costruire la seguente macchina T_1 : con input $x \in \Sigma_1^*$, T_1 opera in due fasi (ed utilizza due nastri)
  - FASE 1: T_1 simula $T_r(x)$ scrivendo l'output y sul secondo nastro
  - FASE 2: T_1 simula $T_2(y)$ sul secondo nastro: se $T_2(y)$ accetta allora anche T_1 accetta, se $T_2(y)$ rigetta allora anche T_1 rigetta.
- **T_1 decide L_1**:
  - perché $T_2(y)$ accetta se e solo se $y \in L_2$, e $y \in L_2$ se e solo se $x \in L_1$
- Ma quanto impiega T_1 a decidere L_1?

Un nuovo strumento

- Abbiamo due linguaggi, $L_1 \subseteq \Sigma_1^*$ e $L_1 \subseteq \Sigma_2^*$, e riusciamo a dimostrare che $\mathbf{L_1 \preccurlyeq L_2}$, e sappiamo che $\mathbf{L_2 \in DTIME[\ f(n)\ ]}$
- Allora, abbiamo costruito una macchina T_1 che decide L_1: ma quanto impiega T_1 a decidere L_1?
  - Con input x:
  - La FASE 1 termina in $O(|x|^c)$ passi
  - la FASE 2 termina in $O(f(|y|))$ passi
- Ma quanto è grande $|y|$ in funzione di $|x|$?
  - beh, poiché $T_r(x)$ impiega $O(|x|^c)$ passi per calcolare y
  - e in questo numero di passi sono conteggiati anche i passi che occorrono a scrivere y sul nastro di output
  - allora, $|y| \in O(|x|^c)$
- E, quindi, per ogni $x \in \Sigma_1^*$, $T_1(x)$ termina in $O(|x|^c + f(|x|^c))$ passi
- Ossia, $\mathbf{L_1 \in DTIME[\ n^c + f(\ n^c\ )\ ]}$

Il nuovo strumento: se DTIME[f(n)] ⊆ P

- In particolare: abbiamo due linguaggi, $L_1 \subseteq \Sigma_1^*$ e $L_2 \subseteq \Sigma_2^*$, e sappiamo che **$L_1 \preccurlyeq L_2$** ,
- abbiamo appena dimostrato che se **se $L_2 \in$ P allora $L_1 \in$ P**
  - infatti, in questo caso, esiste una costante k tale che $L_2 \in$ DTIME[n^k]
  - allora, da quanto visto alla pagina precedente, $L_1 \in$ DTIME[$n^c + (n^c)^k$] ⊆ P
- Il **Teorema 6.21** della dispensa 6 dimostra il solo caso "Se **$L_1 \preccurlyeq L_2$** e $L_2 \in$ P allora $L_1 \in$ P"
  - che è quel che abbiamo appena dimostrato (qui e nella pagina precedente)!
- Ossia, il **Teorema 6.21** della dispensa 6 dimostra che

  La classe P è chiusa rispetto alla riducibilità polinomiale

Il nuovo strumento: se DTIME[f(n)] ⊆ P

- **Teorema 6.21** della dispensa 6 dimostra che

  La classe P è chiusa rispetto alla riducibilità polinomiale

- Allo stesso modo si dimostra che, quando $L_1 \preccurlyeq L_2$, se $L_2 \in$ EXPTIME allora $L_1 \in$ EXPTIME
  - ossia, **la classe EXPTIME è chiusa rispetto alla riducibilità polinomiale**
- Ma si può dimostrare la stessa cosa con le classi non deterministiche:
  - Se $L_2 \in$ NP allora $L_1 \in$ NP,
  - Se $L_2 \in$ NEXPTIME allora $L_1 \in$ NEXPTIME,
  - Se avete voglia, provate a dimostrarlo per esercizio
- E anche per le classi spaziali
  - Se $L_2 \in$ PSPACE allora $L_1 \in$ PSPACE,
  - Se avete voglia, provate a dimostrarlo per esercizio

I linguaggi NP-completi

- A questo punto, abbandoniamo le generiche π-riduzioni e torniamo definitivamente alle nostre riduzioni polinomiali
  - da questo momento in poi, quando parleremo di riduzioni ci riferiremo sempre alle riduzioni polinomiali e utilizzeremo il simbolo per riferirci ad esse $\preccurlyeq$
- Un linguaggio $L \subseteq \Sigma^*$ è **NP-completo (rispetto alla riducibilità polinomiale)** se
  a) $L \in NP$
  b) per ogni altro $L_0 \in NP$, vale che $L_0 \preccurlyeq L$.
- I linguaggi NP-completi sono particolarmente importanti per il loro ruolo di possibili *linguaggi separatori* fra le classi P e NP:
- **Corollario 6.4: Se $P \neq NP$ allora, per ogni linguaggio NP-completo L, $L \notin P$.**
  - Supponiamo che L sia un linguaggio NP-completo e che $L \in P$.
  - Poiché L è NP-completo allora, per ogni linguaggio $L_0 \in NP$, $L_0 \preccurlyeq L$;
  - ma, se $L \in P$, poiché P è chiusa rispetto a $\preccurlyeq$, questo implica che, per ogni $L_0 \in NP$, $L_0 \in P$.
  - Ossia, P = NP, contraddicendo l'ipotesi.

I problemi NP-completi

- Ma quale è il senso del Corollario 6.4?
- Intanto che **è molto improbabile che un linguaggio NP-completo appartenga a P**
  - Perché ci interessa, dite? Ah, già, voi ancora non sapete nulla della congettura…
  - Ebbene, si sospetta che sia **P ≠ NP** – ma nessuno è mai riuscito a dimostrarlo, per questo è una congettura.. la ***congettura fondamentale della complessità computazionale***
  - e siccome è una questione importante, sulla dimostrazione della congettura (o della sua negazione) hanno messo una taglia da 1000000 di dollari!
  - Ma, dell'importanza della congettura, parleremo in seguito…
- Quindi: se vogliamo dimostrare che, *probabilmente*, non esiste un algoritmo deterministico che decide in tempo polinomiale un linguaggio che è in NP …
  … quel che dobbiamo fare è dimostrare che quel linguaggio è NP-completo
- E se, invece, abbiamo un linguaggio NP-completo e progettiamo un algoritmo deterministico che decide quel linguaggio in tempo polinomiale?
  - In tal caso, abbiamo vinto un milione di dollari
  - oppure, ehm, argh, abbiamo sbagliato qualcosa…

Uso delle riduzioni

- Ricordiamo che le riduzioni nel campo della calcolabilità si rivelano utili tanto per dimostrare che un linguaggio è decidibile/accettabile quanto per dimostrare che un linguaggio non è decidibile/accettabile: dato un linguaggio L_1
  - se dimostro che $L_1 \preccurlyeq L_2$, per un qualche altro linguaggio **L_2 decidibile,** allora, posso concludere che anche **L_1 è decidibile**
  - se dimostro che $L_0 \preccurlyeq L_1$, per un qualche altro linguaggio **L_0 non decidibile,** allora, posso concludere che anche **L_1 è non decidibile**
- Allo stesso modo, le riduzioni polinomiali sono uno strumento utile tanto per dimostrare che un linguaggio è in P quanto per dimostrare che un linguaggio *probabilmente* non è in P
- dato un linguaggio L_1
  - se dimostro che $L_1 \preccurlyeq L_2$, per un qualche altro linguaggio **$L_2 \in$ P,** allora, posso concludere che anche **$L_1 \in$ P**
  - se dimostro che $L_0 \preccurlyeq L_1$, per un qualche altro linguaggio **L_0 ,** allora, posso concludere che…
    L_1 non può essere "più facile" di L_0, ossia se L_0 *probabilmente* non appartiene a P allora anche L_1 *probabilmente* non appartiene a P
  - **ma di questo parleremo (e abbondantemente!) nella dispensa 9**