Lezione15 – classi complemento

Lezione del 24/04/2024

Classi di complessità complemento

- Torniamo un attimo al paragrafo 6.6: accanto alle classi introdotte all'inizio di questo paragrafo, possiamo considerare i corrispondenti complementi:
- coP = $\{L \subseteq \{0,1\}^* : L^C \in P\}$,
- coNP = $\{L \subseteq \{0,1\}^* : L^C \in NP\}$,
- E, allo stesso modo, le classi
  - coEXPTIME, coNEXPTIME,
  - coPSPACE
- E, in generale: $coDTIME[f(n)] = \{L \subseteq \{0,1\}^* : L^C \in DTIME[f(n)\}$,
  $coDSPACE[f(n)] = \{L \subseteq \{0,1\}^* : L^C \in DSPACE[f(n)\}$,
  $coNTIME[f(n)] = \{L \subseteq \{0,1\}^* : L^C \in NTIME[f(n)\}$,
  $coNSPACE[f(n)] = \{L \subseteq \{0,1\}^* : L^C \in NSPACE[f(n)\}$,
- Osserviamo che nella definizione delle classi di complessità complemento non viene specificato come vengono decisi (o accettati) i linguaggi che vi appartengono ma, invece, viene specificato come vengono decisi (o accettati) i **complementi** dei linguaggi che vi appartengono
- Tuttavia, questa differenza è irrilevante quando si parla di classi deterministiche

Classi di complessità complemento

- Osserviamo che nella definizione delle classi di complessità complemento non viene specificato come vengono decisi (o accettati) i linguaggi che vi appartengono ma, invece, viene specificato come vengono decisi (o accettati) i complementi dei linguaggi che vi appartengono
- Tuttavia, questa differenza è irrilevante quando si parla di classi deterministiche: infatti, sappiamo che
- **Teorema 6.11**: *Per ogni funzione totale calcolabile* $f : \mathbb{N} \to \mathbb{N}$,

  $DTIME[f(n)] = coDTIME[f(n)]$ e $DSPACE[f(n)] = coDSPACE[f(n)]$.

- E come viene dimostrato, in breve, questo teorema?
  - Si prende una macchina T che decide L tale che, per ogni x, $dtime(T,x) \in O(f(|x|))$ [o $dspace(T,x) \in O(f(|x|))$]
  - si costruisce una nuova macchina T' complementando gli stati di accettazione e di rigetto di T – ossia, si aggiungono le quintuple $\langle q_A, s, s, q'_R, F\rangle$ e $\langle q_R, s, s, q'_A, F\rangle$ per ogni $s \in \{0,1,\square\}$, dove q'_A e q'_R sono gli stati di accettazione e di rigetto di T'
  - **T' decide L^c** e $dtime(T',x) \in O(f(|x|))$ [o $dspace(T',x) \in O(f(|x|))$]

Classi di complessità complemento

- Osserviamo che nella definizione delle classi di complessità complemento non viene specificato come vengono decisi (o accettati) i linguaggi che vi appartengono ma, invece, viene specificato come vengono decisi (o accettati) i complementi dei linguaggi che vi appartengono
- Tuttavia, questa differenza è irrilevante quando si parla di classi deterministiche: infatti, dal Teorema 6.11 possiamo derivare
- **Corollario 6.3**: P = coP
- Ma anche che coPSPACE = PSPACE
- Possiamo arrivare alla stessa conclusione per le classi non deterministiche?
  - Cioè: possiamo utilizzare la stessa tecnica utilizzata nella dimostrazione del Teorema 6.11 nel caso non deterministico?
  - Possiamo complementare gli stati di accettazione e di rigetto di una macchina NT che accetta un linguaggio L al fine di accettare il complemento di L?

Classi di complessità complemento

- Possiamo complementare gli stati di accettazione e di rigetto di una macchina NT che *accetta* un linguaggio L al fine di *accettare* il complemento di L?
  - Perché la questione è proprio questa: le classi non deterministiche sono definite come classi di linguaggi *accettati* da macchine non deterministiche entro quantità limitate di istruzioni o celle di nastro
  - "Ma, come?!" state sicuramente pensando, dopo le scatole che ci ha fatto per dimostrarci che, sì, vabbé, sono definite sulla base dell'accettazione ma, in effetti, siccome le funzioni limite sono time- e space-constructible, allora quei linguaggi sono anche **decisi** entro le stesse quantità di risorse?! …
- Allora: è vero, anche se NP è definita come la classe dei linguaggi *accettati* in tempo non deterministico polinomiale, i linguaggi in NP sono, in effetti, linguaggi *decisi* da macchine non deterministiche in tempo polinomiale
- Tuttavia, ricordiamo che una macchina di Turing non deterministica NT
  - accetta un input x se **esiste** una computazione deterministica in NT(x) che termina in q_A
  - rigetta un input x se **ogni** computazione deterministica in NT(x) termina in q_R
- Ecco: il problema è proprio in questa (dannata) asimmetria nelle definizioni di accettazione e di rigetto

Facciamo un gioco

- Proviamo ad applicare la stessa tecnica usata nel teorema 6.11 ad un macchina non deterministica NT
  - costruiamo una nuova macchina NT' invertendo gli stati di accettazione e di rigetto di NT
  - e vediamo se NT' accetta (oppure no) il complemento del linguaggio accettato da NT

- Cominciamo scegliendo un linguaggio $L \subseteq \{0,1\}^*$ *accettato* da una macchina di Turing non deterministica NT

- E ricordiamo che il linguaggio complemento di L è $L^C = \{0,1\}^* - L$
  - ossia, per ogni $x \in \{0,1\}^*$
    - se $x \in L$ allora $x \notin L^C$
    - se $x \notin L$ allora $x \in L^C$

Facciamo un gioco

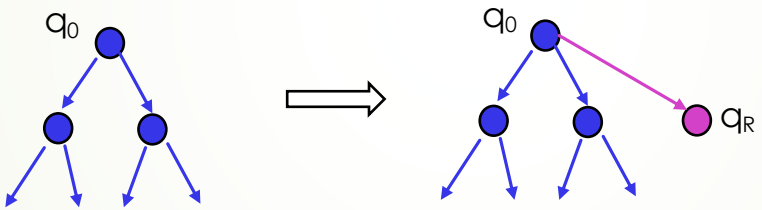
- Cominciamo scegliendo un linguaggio $L \subseteq \{0,1\}^*$ accettato da una macchina di Turing non deterministica NT
- E ricordiamo che il linguaggio complemento di L è $L^C = \{0,1\}^* - L$
  - ossia, per ogni $x \in \{0,1\}^*$
    - se $x \in L$ allora $x \notin L^C$
    - se $x \notin L$ allora $x \in L^C$
- Allora, una macchina non deterministica NT^C accetta L^C se, per ogni $x \in \{0,1\}^*$,
  - se $x \in L$ allora $\mathrm{NT}^C(x)$ *non accetta*
  - se $x \notin L$ allora $\mathrm{NT}^C(x)$ accetta
- e, quindi,
  - se $x \in L$ allora **ogni** computazione deterministica in $\mathrm{NT}^C(x)$ ***non*** termina in q_A
  - se $x \notin L$ allora **esiste** una computazione deterministica in $\mathrm{NT}^C(x)$ che termina in q_A

Facciamo un gioco

- Cominciamo scegliendo un linguaggio $L \subseteq \{0,1\}^*$ accettato da una macchina di Turing non deterministica NT
  - e proviamo ad applicare la stessa tecnica usata nel teorema 6.11 ad un macchina non deterministica NT, costruendo una nuova macchina NT' invertendo gli stati di accettazione e di rigetto di NT
- Un attimo, però: prima di invertire gli stati di accettazione e di rigetto di NT, costruiamo una nuova macchina NT_1 che, ancora, accetta L
- Prendiamo NT ed aggiungiamo all'insieme delle sue quintuple le quintuple
  $\langle q_0, s, s, q_R, F \rangle$ per ogni $s \in \{0,1, \square\}$
  - E questa è NT_1
  - ATTENZIONE: **per ogni $x \in \{0,1\}^*$ esiste *sempre* una computazione deterministica di $NT_1(x)$ che termina in q_R**

Facciamo un gioco

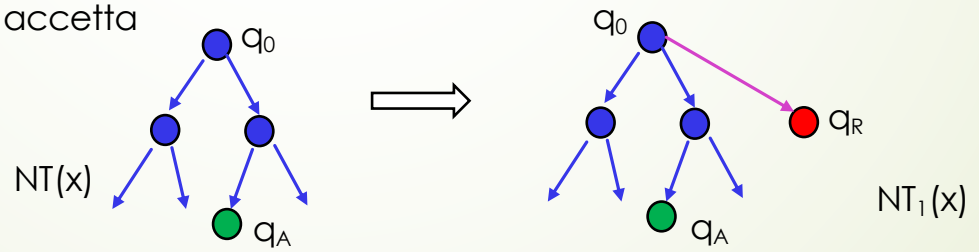
- Prendiamo NT , che accetta $L \subseteq \{0,1\}^*$, ed aggiungiamo all'insieme delle sue quintuple le quintuple $\langle q_0, s, s, q_R , F\rangle$ per ogni $x \in \{0,1, \square\}$
  - E questa è NT_1
  - **per ogni $x \in \{0,1\}^*$ esiste una computazione deterministica di $NT_1(x)$ che termina in q_R**



NT(x) $NT_1(x)$

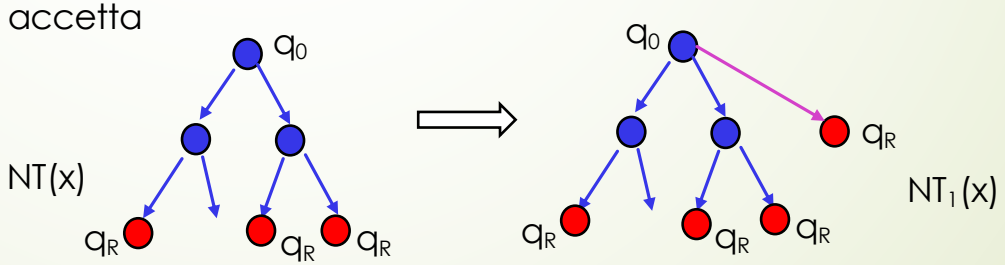
Facciamo un gioco

- Prendiamo NT , che accetta L ⊆ {0,1}* , ed aggiungiamo all'insieme delle sue quintuple le quintuple $\langle q_0, s, s, q_R , F\rangle$ per ogni x ∈ {0,1, □}
  - E questa è NT_1
  - **per ogni x ∈ {0,1}* esiste una computazione deterministica di $NT_1(x)$ che termina in q_R**
- **NT_1 accetta L**
- infatti: per ogni x ∈ L
  - poiché NT accetta L, allora NT(x) accetta
  - allora, esiste una computazione deterministica di NT(x) che termina in q_A
  - ma quella stessa computazione deterministica compare anche in $NT_1(x)$
  - e, quindi, $NT_1(x)$ accetta



Facciamo un gioco

- Prendiamo NT , che accetta L ⊆ {0,1}* , ed aggiungiamo all'insieme delle sue quintuple le quintuple $\langle q_0, s, s, q_R, F \rangle$ per ogni x ∈ {0,1, □}
  - E questa è NT_1
  - **per ogni x ∈ {0,1}* esiste una computazione deterministica di $NT_1(x)$ che termina in q_R**
- **NT_1 accetta L**
  - infatti: per ogni x ∈ L $NT_1(x)$ accetta
- e d'altra parte: per ogni x ∉ L
  - poiché NT accetta L, allora NT(x) non accetta (ossia, rigetta oppure non termina)
  - allora, non esiste alcuna computazione deterministica di NT(x) che termina in q_A
  - e allo stesso modo non esiste in $NT_1(x)$ una computazione deterministica che accetta
  - e, quindi, $NT_1(x)$ non accetta



Facciamo un gioco

- Dunque, abbiamo un linguaggio L ⊆ {0,1}* accettato dalla macchina non deterministica NT_1
  - e adesso applichiamo a NT_1 la stessa tecnica usata nel teorema : costruiamo una nuova macchina $\mathbf{NT_1^C}$ invertendo gli stati di accettazione e di rigetto di NT_1
- Ci aspetteremmo che $\mathbf{NT_1^C}$ accetti L^C… Sarà davvero così?
- Vediamo: scegliamo x ∈ {0,1}* e poniamo $x = x_1x_2 \dots x_n$
  - ossia, $x_1 \in \{0,1\}$ è il primo carattere di x, $x_2 \in \{0,1\}$ il secondo e così via
- **se $x \in L^C$** :
  - in $NT_1(x)$ esiste la computazione deterministica $\langle q_0, x_1, x_1, q_R, F\rangle$ che termina in q_R
  - e quella stessa computazione deterministica compare anche in $\mathbf{NT_1^C}$ (x) che, però, in $\mathbf{NT_1^C}$ termina in q_A
  - allora **$\mathbf{NT_1^C}$(x) accetta** – Bene!

Facciamo un gioco

- Dunque, abbiamo un linguaggio $L \subseteq \{0,1\}^*$ accettato dalla macchina non deterministica NT_1
  - e adesso applichiamo a NT_1 la stessa tecnica usata nel teorema : costruiamo una nuova macchina $\mathbf{NT_1^C}$ invertendo gli stati di accettazione e di rigetto di NT_1
- Ci aspetteremmo che $\mathbf{NT_1^C}$ accetti L^C… Sarà davvero così?
- Vediamo: scegliamo $x \in \{0,1\}^*$ e poniamo $x = x_1x_2 \ldots x_n$
- **Se $x \in L^C$, $\mathbf{NT_1^C(x)}$ accetta** – Bene!
- **Se $x \notin L^C$** :
  - se fosse vero che $\mathbf{NT_1^C}$ decide L^C allora $\mathbf{NT_1^C}(x)$ ***non*** dovrebbe accettare
  - ma in $NT_1(x)$ esiste la computazione deterministica $\langle q_0, x_1, x_1, q_R, F\rangle$ che termina in q_R
  - e quella stessa computazione deterministica compare anche in $\mathbf{NT_1^C}(x)$ che, però, in $\mathbf{NT_1^C}$ termina in q_A
  - allora **$\mathbf{NT_1^C(x)}$ accetta** – Bene! OPS! Cioè, no: MALE!

    $\mathbf{NT_1^C}(x)$ ***non*** dovrebbe accettare se $x \notin L^C$!
- Invece, **$\mathbf{NT_1^C(x)}$ accetta qualunque sia x**! Col cavolo che $\mathbf{NT_1^C}$ accetta L^C!

Facciamo un gioco

- Allora: anche se i linguaggi in NP sono, in effetti, linguaggi *decisi* da macchine di Turing non deterministiche in tempo polinomiale
- il fatto che una macchina di Turing non deterministica NT
  - accetta un input x se **esiste** una computazione deterministica in NT(x) che termina in q_A
  - rigetta un input x se **ogni** computazione deterministica in NT(x) termina in q_R
- proprio questa (dannata) asimmetria nelle definizioni di accettazione e di rigetto non permette di derivare una macchina che decide L^C invertendo gli stati di accettazione e di rigetto di una macchina non deterministica che decide L
- E questo significa che **non possiamo affermare che coNP = NP**
- Ma, tutto questo ragionamento, ci permette forse di affermare che coNP ≠ NP?
- Col cavolo!
  - la dimostrazione che coNP = NP potrebbe seguire una strada completamente diversa da quella dell'inversione degli stati finali di una macchina non deterministica…
- E allora?

Questioni di congetture

- Abbiamo detto più volte che la maggior parte delle inclusioni fra classi di complessità sono inclusioni deboli
  - nelle quali non si riesce a dimostrare che le due classi sono diverse
  - ma non si riesce nemmeno a dimostrare che le due classi sono uguali!
- Il caso più famoso è quello che riguarda le classi P e NP
  - sappiamo che $P \subseteq NP$ – e, quindi, che ogni problema in P è contenuto anche in NP
  - ma non sappiamo se $P = NP$ – ossia, se ogni problema in NP è contenuto, in effetti, in P
  - né sappiamo se $P \neq NP$ – ossia, se esiste un problema in NP che non è contenuto in P
- La **congettura fondamentale della teoria della complessità computazionale** ipotizza che **$P \neq NP$**
  - e sulla dimostrazione (o confutazione) di questa congettura pende una taglia da un milione di dollari!
- Ed ora abbiamo appena scoperto una nuova congettura:
- La **seconda congettura della teoria della complessità computazionale** ipotizza che **$coNP \neq NP$**

Relazione fra le due congetture

- In effetti, comunque, le due congetture non sono del tutto indipendenti, come descritto nel prossimo teorema
- **Teorema 6.23**: Se P = NP allora NP = coNP.
- Dimostrazione:
  - per il Corollario 6.3, P = coP
  - per ipotesi: P = NP e quindi coP = coNP
  - allora: NP = P = coP = coNP
- Il teorema afferma che: *se è falsa la Congettura Fondamentale della Teoria della Complessità Computazionale allora è falsa anche la Seconda Congettura della Teoria della Complessità Computazionale*
- Questo teorema può anche essere letto come: **se NP ≠ coNP allora P ≠ NP**
  - ossia: *se è vera la Seconda Congettura della Teoria della Complessità Computazionale allora è vera anche la Congettura Fondamentale della Teoria della Complessità Computazionale*
- L'affermazione inversa "se NP = coNP allora P = NP" non è invece stata dimostrata
- Per questo le due congetture sono, fino ad ora, due congetture distinte

Struttura della classe coNP

- **Teorema 6.24**: La classe coNP è chiusa rispetto alla riducibilità polinomiale.
  - Come detto sulla dispensa, "La dimostrazione è analoga a quella del Teorema 6.21 ed è lasciata per esercizio. "
  - Aggiungo che mi piacerebbe se qualcuno di voi lo facesse questo **UTILE** esercizio!
  - (E me lo inviasse)

- Come per tutte le classi di complessità, anche per la classe coNP possiamo definire linguaggi completi rispetto alla riducibilità polinomiale

- **DEFINIZIONE**: un linguaggio L è coNP-completo se
  - 1) $L \in coNP$
  - 2) per ogni linguaggio $L' \in coNP$, si ha che $L' \preccurlyeq L$

Struttura della classe coNP

- Come abbiamo visto la scorsa lezione, i linguaggi NP-completi sono i possibili linguaggi separatori fra P e NP
  - ossia, **nell'ipotesi P ≠ NP**
  - **un linguaggio NP-completo non può essere contenuto in P**
  - sono i linguaggi "più difficili" all'interno di NP
- La stessa cosa ci proponiamo di fare nella classe coNP
- Vogliamo mostrare che i linguaggi coNP-completi sono i candidati ad essere i linguaggi separatori fra NP e coNP
  - ossia che, **nell'ipotesi coNP ≠ NP** ,
  - **un linguaggio coNP-completo non può essere contenuto in NP**
  - che i linguaggi coNP-completi sono i linguaggi "più difficili" all'interno di coNP
- Questo è l'obiettivo dei prossimi due teoremi.

Struttura della classe coNP

- **Teorema 6.25**: Un linguaggio L è NP-completo se e soltanto se il suo complemento L^C è coNP-completo
- ⇒ Sia L un linguaggio NP-completo – mostriamo che L^C è coNP-completo
- 1) $L \in NP$ e, quindi, $L^C \in coNP$.
- 2) Dobbiamo mostrare che, per ogni $L_1 \in coNP$, vale che $\mathbf{L_1 \preccurlyeq L^C}$
  - sia allora L_1 un **qualsiasi** linguaggio in coNP (ossia, $\forall$ **$L_1 \in coNP$**): allora, **$L_1^C \in NP$**
  - poiché L è completo per la classe NP, allora per ogni $L_0 \in NP$, $L_0 \preccurlyeq L$: allora, in particolare, poiché $L_1^C \in NP$, vale che $\mathbf{L_1^C \preccurlyeq L}$
  - Questo significa che esiste una funzione $f_1 : \{0,1\}^* \rightarrow \{0,1\}^*$
    (ricordiamo che consideriamo linguaggi nell'alfabeto {0,1})

    tale che $f_1 \in FP$ e, **per ogni $x \in \{0,1\}^*$, $x \in L_1^C$ se e soltanto se $f_1(x) \in L$.**
  - Ma questo è equivalente a dire che, **per ogni $x \in \{0,1\}^*$, $x \notin L_1^C$ se e soltanto se $f_1(x) \notin L$**,
  - ossia, **per ogni $x \in \{0,1\}^*$, $x \in L_1$ se e soltanto se $f_1(x) \in L^C$**
  - ossia, $\mathbf{L_1 \preccurlyeq L^C}$
  - **Poiché L_1 è un qualsiasi linguaggio in coNP**, questo dimostra che L^C è completo per coNP.

Struttura della classe coNP

- **Teorema 6.25**: Un linguaggio L è NP-completo se e soltanto se il suo complemento L^C è coNP-completo
- ⇐ Sia L^C un linguaggio coNP-completo – mostriamo che L è NP-completo
- 1) $L^C \in$ coNP e, quindi, $L \in$ NP.
- 2) Dobbiamo mostrare che, per ogni $L_1 \in$ NP, vale che **$L_1 \preccurlyeq L$**
  - sia allora L_1 un **qualsiasi** linguaggio in NP (ossia, $\forall L_1 \in$ NP): allora, $L_1^C \in$ coNP
  - poiché L^C è completo per la classe coNP, allora per ogni $L_0 \in$ coNP, $L_0 \preccurlyeq L^C$: allora, in particolare, poiché $L_1^C \in$ coNP, vale che $L_1^C \preccurlyeq L^C$
  - Questo significa che esiste una funzione $f_1 : \{0,1\}^* \rightarrow \{0,1\}^*$
    (ricordiamo che consideriamo linguaggi nell'alfabeto {0,1})

    tale che $f_1 \in$ FP e, per ogni $x \in \{0,1\}^*$, **$x \in L_1^C$ se e soltanto se $f_1(x) \in L^C$**.
  - Ma questo è equivalente a dire che, **per ogni $x \in \{0,1\}^*$, $x \notin L_1^C$ se e soltanto se $f_1(x) \notin L^C$**, ossia, **per ogni $x \in \{0,1\}$, $x \in L_1$ se e soltanto se $f_1(x) \in L$** .
  - **Poiché L_1 è un qualsiasi linguaggio in NP**, questo dimostra che L è completo per NP.

Struttura della classe coNP

- **Teorema 6.26**: Se esiste un linguaggio L NP-completo tale che L ∈ coNP, allora NP = coNP.

- Dimostriamo il teorema mostrando prima che (1) coNP ⊆ NP e poi che (2) NP ⊆ coNP

- Sia L un qualunque linguaggio NP-completo tale che che L ∈ coNP
- (1) Poiché L ∈ coNP allora, **L^C ∈ NP**.
- Poiché L è NP-completo allora, per il Teorema 6.25, L^C è coNP-completo,
  - quindi, **per ogni L' ∈ coNP, si ha che L' ≤ L^C**.
- Ma NP è chiusa rispetto alla riducibilità polinomiale (Teorema 6.22)
  che significa che se accade che $L_1 \leq L_2$ e $L_2 \in NP$, allora $L_1 \in NP$
  e $L' \leq L^C$ e **L^C ∈ NP**
- allora, **per ogni linguaggio L' ∈ coNP, si ha che L' ∈ NP**.
- E questo dimostra che **coNP ⊆ NP**.

Struttura della classe coNP

- **Teorema 6.26**: Se esiste un linguaggio L NP-completo tale che L ∈ coNP, allora NP = coNP.

- Sia L un qualunque linguaggio NP-completo tale che che L ∈ coNP
- (2) Mostriamo ora l'inclusione opposta.
- Poiché L è NP-completo allora, per ogni L'' ∈ NP si ha che L'' $\preccurlyeq$ L
- ma **L ∈ coNP**.
- e inoltre coNP è chiusa rispetto alla riducibilità polinomiale (Teorema 6.24)
  che significa che se accade che $L_1 \preccurlyeq L_2$ e $L_2 \in$ coNP, allora $L_1 \in$ coNP
- Riassumendo: coNP è chiusa rispetto alla riducibilità polinomiale e
  per ogni L'' ∈ NP si ha che L'' $\preccurlyeq$ L e
  L ∈ coNP
- allora **per ogni L'' ∈ NP si ha che L'' ∈ coNP**
- E questo dimostra che **NP ⊆ coNP**.

- Infine, le due inclusioni **coNP ⊆ NP** e **NP ⊆ coNP** dimostrano il teorema.