Aniello Murano

PSPACE completezza

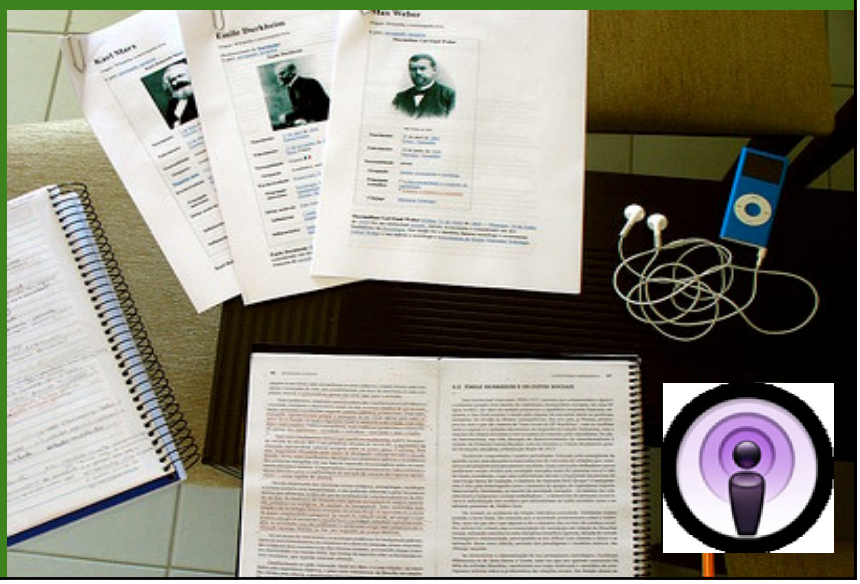
Lezione n.19
Parole chiave:
Pspace

Corso di Laurea:
Informatica

Codice:

Email Docente:
murano@ na.infn.it

A.A. 2008-2009



Definizione di Pspace Completezza

- **Definizione:** Un linguaggio **B** è ***PSPACE-complete*** se soddisfa le seguenti due condizioni:

  1. **B** è in PSPACE, e
  2. Ogni **A** in PSPACE è riducibile in tempo polinomiale a **B**.

  Se **B** soddisfa solo la condizione 2, diremo che è ***PSPACE-hard***.

- E' importante notare che utilizziamo ancora la riduzione in tempo piuttosto che quella in spazio

- Non usiamo quella di spazio perchè non vogliamo che la complessità del problema sia assorbita nella trasformazione.

Problema TQBF

- Un problema PSPACE-complete è quello della soddifacibilità delle formule booleane quantificate, ovvero le ***True Quantified Boolean Formulas (TQBF)***

- ***Universal quantifier*** ∀: $\forall x P(x)$ cioè "per ogni $x \in \{0,1\}$, $P(x)$ è true"
- ***Existential quantifier*** ∃: $\exists x P(x)$ cioè "per qualche $x \in \{0,1\}$, $P(x)$ è true"

- Noi consideriamo TQBF in forma prenessa. Ovvero, formule booleane sono prefisse da $\forall x$ o $\exists x$ per ciascuna variabile x.

- Esempi (true o false?):

$\forall x(x \vee \underline{x})$
$\exists x(x \vee \underline{x})$
$\exists x(x \wedge \underline{x})$
$\exists x \exists y(x \wedge y)$

$\forall x \forall y\ (x \vee y)$
$\forall x \exists y\ ((x \wedge y) \vee (\underline{x} \wedge \underline{y}))$
$\exists x \forall y\ ((x \wedge y) \vee (\underline{x} \wedge \underline{y}))$
$\exists z \forall x \exists y\ ((x \wedge y \wedge z) \vee (\underline{x} \wedge \underline{y} \wedge z))$

TQBF = {<ϕ> | ϕ è una vera (true) quantified Boolean formula}
(**T**rue **Q**uantified **B**oolean **F**ormulas)

PSPACE-completezza di TQBF

- **Teorema:** TQBF è PSPACE-completo

- Per mostrare che **TQBF∈PSPACE**, usiamo un algoritmo in spazio polinomiale che assegna valori di verità alle variabili e ricorsivamente valuta la formula.

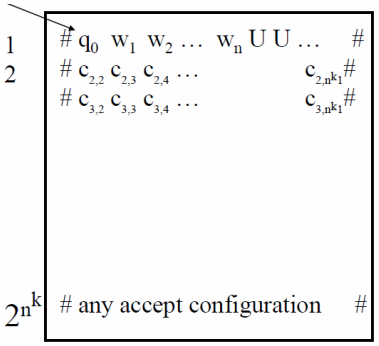
- Per l'hardness, procediamo in maniera similare a quanto fatto per NP.
  - Mostriamo che $A \leq_p$ **TQBF** per ogni **A∈PSPACE**, iniziando con una macchina M che lavora in spazio polinomiale per A.
  - Dunque, usiamo una riduzione in tempo polinomiale che mappa una stringa w ad una formula ϕ che codifica una simulazione di M su input w.
  - ϕ è vera se e solo se M accetta w.

Algoritmo in spazio polinomiale per TQBF

- Il seguente algoritmo decide TQBF:
  T = "Con input **<** ϕ**>** , una fully quantified Boolean formula:
  1. Se ϕ non contiene quantificatori, allora è un espressione con solo costanti così valutiamo ϕ e *accept* se è true, altrimenti, *reject*.
  2. Se ϕ è $\exists$**x** ψ, ricorsivamente chiamiamo **T** su ψ, prima con **0** sostituito a **x** e poi con **1** sostituito a **x**. Se il loro risultato è accept, Allora *accept*; altrimenti, *reject*.
  3. Se ϕ è $\forall$**x** ψ, ricorsivamente chiamiamo **T** su ψ, prima con **0** sostituito a **x** e poi con **1** sostituito a **x**. Se entrambi risultano accept, allora *accept*; altrimenti, *reject*."
- In pratica, valutiamo la formula con una procedura val nel modo seguente:
  - val(VERO)=vero e val(FALSO)=falso
  - val($\neg\varphi$)=$\neg$val(φ), val(φ1\/φ2)=val(φ1)\/val(φ2) e val(φ1/\$\varphi$2)=val($\varphi$1)/\val($\varphi$2)
  - val($\exists$ xφ)=val(φ | x=VERO) \/ val(φ | x=FALSO)
  - val($\forall$ xφ)=val(φ | x=VERO) /\ val(φ | x=FALSO)
- Ad ogni chiamata ricorsiva abbiamo una formula ridotta dove alcune variabili vengono instanziate. Tutta la ricorsione richiede dunque memoria vari a $|\varphi|$.
- La profondità massima della ricorsione è pari al più alla lunghezza della formula. Dunque, una quantità di memoria totale $(|\varphi|)^2$ è sufficiente.

PSPACE-Hardness per TQBF(1)

- Dimostriamo ora che il problema è completo. Dato un linguaggio L ϵ PSPACE, consideriamo la MT M che accetta L in spazio polinomiale.
- Sia x una parola di L. Costruiamo una formula φ di TQBF tale che φ è vera se e solo se M accetta x in spazio n^k.
- Usiamo una tecnica simile a quella utilizzate nel teorema di Cook-Levin. In particolare, assumiamo che
  - la macchina usa nastro seminfinito
  - ogni configurazione usa spazio n^k
  - la macchina cicla nella configurazione finale.
- Si noti che in questo caso la tabella da considerare ha n^k colonne e 2^{n^k} righe; quindi è di taglia esponenziale. Si veda la figura a lato.
- Si ricordi che nel caso di SAT, il tableau rappresentava un ramo della computazione ed era di n^k righe (e colonne)
- La formula che definiremo usa una variabile per ogni cella. Per semplicità nella formula ci riferiamo alle configurazioni come entità atomica.



PSPACE-Hardness per TQBF(2)

- Sia $m=n^k$. La formula quantificata che costruiremo è la seguente

  $\varphi(x) = \exists X_1 \exists X_2 (\varphi^M(X_1) \wedge \varphi^M(X_2) \wedge \varphi^I(X_1,x) \wedge \varphi^A(X_2) \wedge \varphi^T(X_1,X_2,2^m))$

- $\varphi(x)$ può essere interpretata nel seguente modo.
- X_1 ed X_2 sono due configurazioni (soddisfano il predicato φ^M).
- X_1 è una configurazione iniziale, contenente l'input x (soddisfa il predicato φ^I).
- X_2 è una configurazione finale (soddisfa il predicato φ^A).
- Esiste una computazione legale della macchina M che porta da X_1 a X_2 in 2^m passi (soddisfano il predicato $\varphi^T(X_1, X_2,2^m)$).

Osservazioni alla formula

- Ogni variabili X_i corrisponde ad una riga della tabella e quindi corrisponde ad m variabili del tipo $c_{i,j}$.
- La formula φ^M impone che vengano soddisfatte le proprietà di una configurazione relativa alla macchina M (espresse con le variabili $c_{i,j}$),
- la formula φ^I impone che sia soddisfatta la struttura della configurazione iniziale della macchina M con input x,
- la formula φ^A impone che sia soddisfatta la proprietà di una
- configurazione finale,
- la formula φ^T impone che siano soddisfatte le proprietà di una computazione eseguita dalla macchina M.

Ultimi passi per PSPACE-Hardness di TQBF

- Il problema principale è come definire la formula φ^T. Infatti se la definiamo come nel teorema di Savitch abbiamo:

  $$\varphi^T(X_1,X_2,t) = \exists X_3(\varphi^M(X_3) \wedge \varphi^T(X_1,X_3,t/2) \wedge \varphi^T(X_3,X_2,t/2)$$

- ma questo comporta una formula di lunghezza esponenziale.
- Usiamo invece la definizione seguente:

  $\varphi^T(X_1,X_2,t) = \forall X\ \forall Y\ \exists X_3(\varphi^M(X_3) \wedge [((X=X_1 \wedge Y=X_3) \vee (X=X_3 \wedge Y=X_2)) \rightarrow \varphi^T(X,Y,t/2)])$

  la cui lunghezza è $O(m \log 2^m)$.
- Pertanto la formula $\varphi(x)$ è anch'essa di lunghezza complessiva $O(m^2)$.

Formule come giochi

- Ciascuna formula TQBF (in forma prenessa) ϕ può essere vista come un gioco tra due giocatori **A** ed **E**.
- **A** ed **E**. muovono a turno instanziando le variabili, secondo le seguenti regole:
- Se $\phi = \exists x \psi(x)$, **E muove** instanziando **x=0** oppure **x=1**.
- Se $\phi = \forall x \psi(x)$, **A muove** instanziando **x=0** o **x=1**.
- In entrambi i casi, il gioco continua con $\psi(0)$ o $\psi(1)$, secondo della scelta fatta
- Il gioco continua finchè non sono eliminati tutti i quantificatori.
- Quando non ci sono più quantificatori, diremo che il giocatore **E** è il vincitore se la rimanente formula è valutata true. Altrimenti vince il giocatore **A**.
- Si condideri il seguente esempio. Chi vince?

$\exists x \forall y \exists z[(x \vee y) \wedge (y \vee z) \wedge (y \vee \underline{z})]$	**E** moves, selects **x=1**
$\forall y \exists z[(1 \vee y) \wedge (y \vee z) \wedge (y \vee \underline{z})]$	**A** moves, selects **y=0**
$\exists z[(1 \vee 0) \wedge (0 \vee z) \wedge (0 \vee \underline{z})]$	**E** moves, selects **z=1**
$(1 \vee 0) \wedge (0 \vee 1) \wedge (0 \vee \underline{1})$	**A** wins

Il problema formula-game

- Quando **E** ha un modo per assegnare dei valori alle variabili tali che per ogni scelta di **A** la formula risulta true, allora si dice che **E** ha una strategia vincente.
- Lo stesso dicasi per **A** su **E** se la formula risulta falsa.
- Chi ha una strategia vincente in $\exists x \forall y \exists z[(x \vee y) \wedge (y \vee z) \wedge (\underline{y} \vee \underline{z})]$?
  E: Seleziona **x=1**, e seleziona **z** come la negazione di quello che sceglie **A** per **y**.
- Chi ha una strategia vincente in $\exists x \forall y \exists z[(x \vee y) \wedge (y \vee z) \wedge (y \vee \underline{z})]$?
- Questa volta **A** ha una strategia vincente, infatti basta scegliere y=0.

Si consideri adesso il seguente insieme.

- **FORMULA-GAME = {<ϕ> | Il giocatore E ha una strategia vincente in ϕ}**
- **Teorema** FORMULA-GAME è PSPACE-completo.
- **Dimostrazione:** Questa segue dalla semplice osservazione che

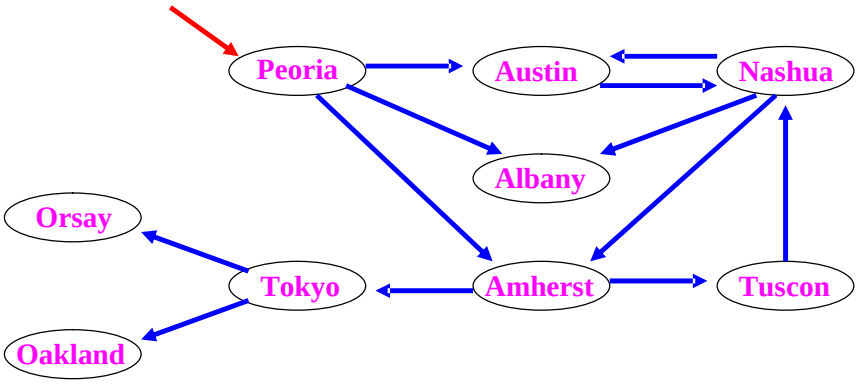
  FORMULA-GAME = TQBF.

  - Di fatti, ϕ è true se e solo se il giocatore **E** ha una strategia vincente nel gioco corrispondente.
  - Una dimostrazione più formale di questo teorema passa attraverso il numero dei quantificatore prefisso di ϕ

- Nelle prossime diapositive utilizzeremo **FORMULA-GAME** per provare la PSPACE-hardness di alcuni problemi.

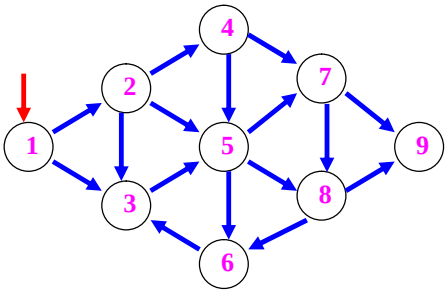
Il gioco Geografy dei bambini

- I giocatori, chiamati I e II, alterneranno i nomi delle città di ogni parte del mondo (inizia il giocatore I). Ogni città scelta deve iniziare con la stessa lettera che ha concluso la città precedente. Le ripetizioni non sono consentite. Il giocatore che non è in grado di continuare perde.
- Siamo in grado di modellare questo gioco con un grafo orientato i cui nodi sono le città del mondo. C'è arco da una città ad un altra se la prima può portare alla seconda utilizzando le regole del gioco.
- Un nodo è designato come il nodo di start. La condizione che le città non possono essere ripetute, significa che il percorso non deve ammettere cicli.



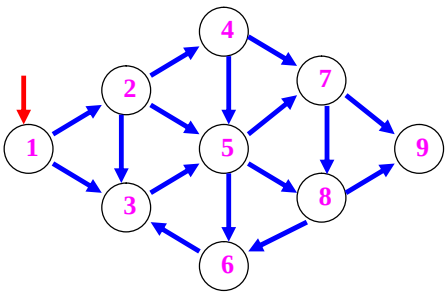
Geography generalized

- In ***Generalized Geography***, prendiamo un arbitrario grafo diretto con un nodo assegnato a start (invece del grafo associato con le attuali città). Chi ha la strategia vincente?



Geography generalized

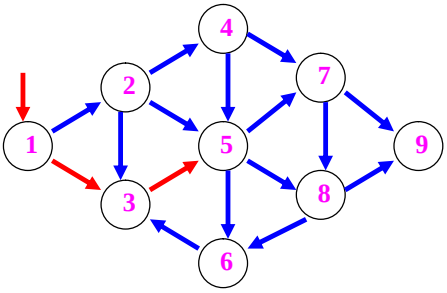
- In ***Generalized Geography***, prendiamo un arbitrario grafo diretto con un nodo assegnato a start (invece del grafo associato con le attuali città). Chi ha la strategia vincente?



Player I: Sceglie 3.

Geography generalized

- In ***Generalized Geography***, prendiamo un arbitrario grafo diretto con un nodo assegnato a start (invece del grafo associato con le attuali città). Chi ha la strategia vincente?

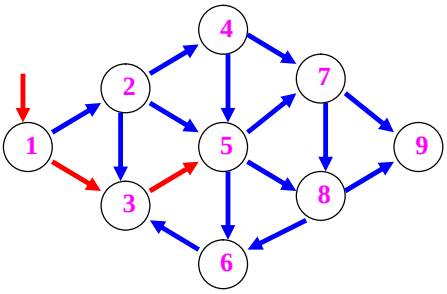


Player I: Sceglie 3.

II: Sceglie 5

Geography generalized

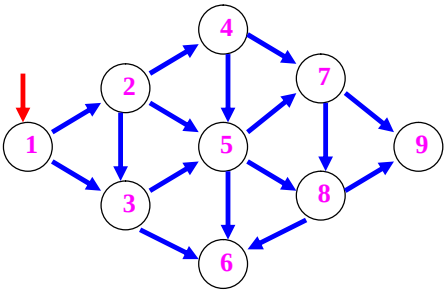
- In ***Generalized Geography***, prendiamo un arbitrario grafo diretto con un nodo assegnato a start.



Player I: Sceglie 3.
II: Sceglie 5

I: Seleziona 6 e

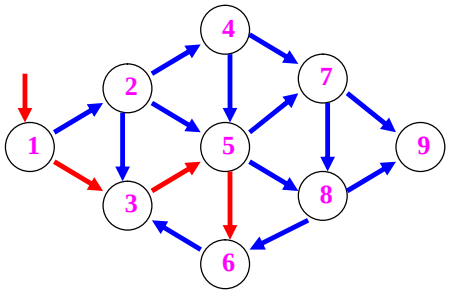
II: è bloccato.



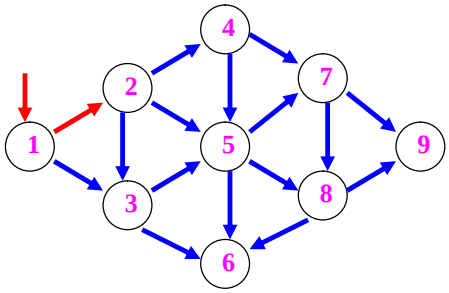
Chi ha una strategia vincente?
Se il giocatore I sceglie 3, il giocatore II sceglie 6 e vince.

Geography generalized

- In ***Generalized Geography***, prendiamo un arbitrario grafo diretto con un nodo assegnato a start.



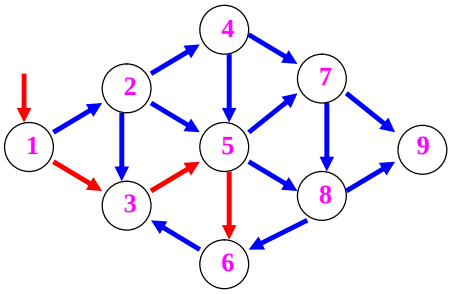
Chi ha la strategia vincente?
Player I: Sceglie 3. II se sceglie 5,
I seleziona 6, e II perde.



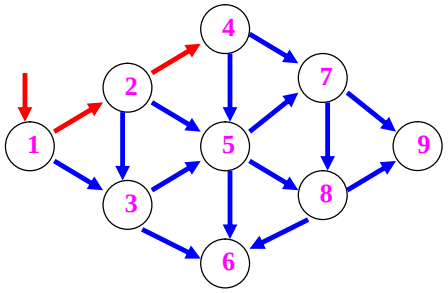
Chi ha la strategia vincente?
Player II. sicuramente Player I se sceglie 3. II sceglie 6 e vince,
Assumiamo che I sceglie 2.

Geography generalized

- In ***Generalized Geography***, prendiamo un arbitrario grafo diretto con un nodo assegnato a start.



Chi ha la strategia vincente?
Player I: Sceglie 3. II sceglie 5.
Ora I seleziona 6, e II perde.



Chi ha la strategia vincente?
Player II. Se I inizia come prima scegliendo 3. II sceglie 6 e vince.
Supponiamo invece che I sceglie 2 allora II sceglie 4. Se I sceglie 5, II sceglie 6 e vince, altrimenti se I sceglie 7, II sceglie 9 e vince.

GG e Pspace-completezza

- **GG = { < G,b> | il giocatore 1 ha una strategia vincente per il gioco di Generalized Geography giocato sul grafo G partendo dal nodo b }**
- **Apparteneza a PSPACE.** Un algoritmo ricorsivo simile a quello usato per TQBF determina quale giocatore ha la stategia vincente. Questo algoritmo lavora in spazio polinomiale per cui **GG∈PSPACE**.
- Per provare la **PSPACE-hardeness**, usiamo una riduzione in tempo polinomiale da FORMULA-GAME a GG.
  - Questa riduzione converte una formula game ϕ in un grafo generalized geography G in modo che giocare su G imita il giocare su ϕ.
  - In effeti i giocatori nel gioco generalized geography giocano una codifica del formula game.

Perchè GG∈PSPACE

- Il seguente algoritmo decide GG:

  M = "su input **< G,b>** , dove **G** è un grafo e **b** è un nodo di **G**:

  1. Se **b** ha grado uscente 0, *reject*, perchè il giocatore 1 perde immediatamente.
  2. Rimuovi il nodo **b** è tutti gli archi che sono connessi è crea un nuovo grafo H.
  3. Per ogni nodo $b_1,\ldots,b_k$ a cui **b** puntava originariamente, ricorsivamente chiama **M** su $\langle H,b_i \rangle$.
  4. Se tutte le chiamate su $\langle H,b_i \rangle$ non rifiutano, il giocatore 2 ha una strategia vincente nel gioco originale e quindi *reject*. Altrimenti, il giocatore 2 non ha una strategia vincente e quindi il giocatore 1 deve averne una, quindi accetta.

Perché GG è Nspace-hard

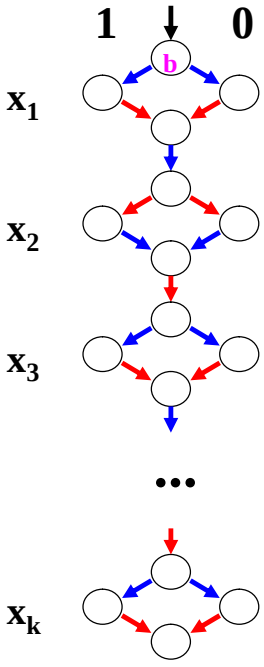
- Consideriamo una formula-game ϕ.
- Possiamo assumere che sia il primo che l'utimo quantificatore è esistenziale ($\exists$) e che il quantificatore e strettamente alternato. Se non è così, facilmente portare ϕ a questa forma aggiungendo variabili e quantificatori inutili.
- Inoltre, possiamo assumere che la parte quantifier-free di ϕ è una formula 3CNF, altrimenti può essere convertita in 3CNF.
- Si noti che le conversioni eventualmente richieste solra producono una formula equivalente e prendono un tempo polinomiale.
- Si consideri duenque la formula

$$\phi = \exists x_1 \forall x_2 \exists x_3 \forall x_4 \ldots \exists x_k [c_1 \wedge c_2 \wedge \ldots \wedge c_m],$$

dove ciascun c_i è una disgiunzione di tre litterali sulle variabili x_i.

- Descriviamo adesso un modo per convertire (in tempo polinomiale) ϕ in **<G,b>** tale che $\phi \in$ **FORMULA-GAME** se e solo se **<G,b>** $\in$ **GG**, cioè, **E** ha una strategia vincente in ϕ se e solo se Player I ha una strategia vincente in **<G,b>**.

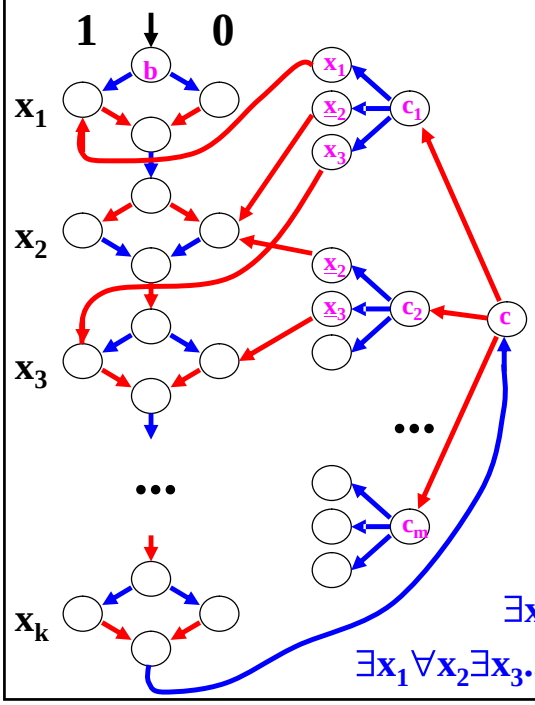
Perché GG è Nspace-hard



- La parte sinistra di **G** sarà rappresentata nel seguente modo:
  - Per ogni variabile creiamo un diamante.
  - Le frecce blu indicano le scelte del Giocatore I (i turni)
  - Le frecce rosse indicano le scelte del Giocatore II.
  - Le scelte che si trovano nella parte sinistra corrisponderanno alla scelta 1 nella formula di gioco, e le scelte che si trovano nella parte destra corrispon-deranno alla scelta 0.
  - Il grafo in figura è parziale e necessita di una estensione che mostriamo nella prossima diapositiva per un particolare esempio di ϕ.

$$\exists x_1 \forall x_2 \exists x_3 \forall x_4 \ldots \exists x_k [c_1 \wedge c_2 \wedge \ldots c_m]$$

Perché GG è Nspace-hard



- Nella parte destra c'è un nodo c, con collegamento dal diamante di x_k e collegato a m nodi c_i.
- Ogni c_i è collegato a tre nodi che rappresentano i letterali nella clausola c_i. Ogni letterale l_j di c_i elativo alla variabile x_j è collegato al diamante di x_j sul lato sinistro o destro a seconda che il letterale è non negato o in forma negata.
- Supponiamo ora che **A** ha una strategia vincente in ϕ. Allora, seguiamo la stessa strategia nella parte sinistra del grafo. Quindi c'è una clausola falsa $\mathbf{c_i}$, e quindi il player II sceglie quella clausola.
- Ora qualsiasi litterale $\mathbf{c_i}$ viene scelto dal player I, quest'ultimo è un letterale falso, quindi il cammino non può passare per esso.
- Quindi, il player II non è bloccato, e si muove verso il nodo a sinistra o a destra del corrispondente diamante.
- Ora il player I è bloccato. Cosi il player II ha una strategia vincente in **G**.

$$\exists x_1 \forall x_2 \exists x_3 \forall x_4 \ldots \exists x_k [c_1 \wedge c_2 \wedge \ldots c_m]$$

$$\exists x_1 \forall x_2 \exists x_3 \ldots \exists x_k [(x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_2 \vee \bar{x}_3 \vee \ldots) \wedge \ldots \wedge (\ldots)]$$