Parity games and mean-payoff games

Dario Balboni

November 20, 2020

Introduzione e Storia

Introduzione ai Graph Games

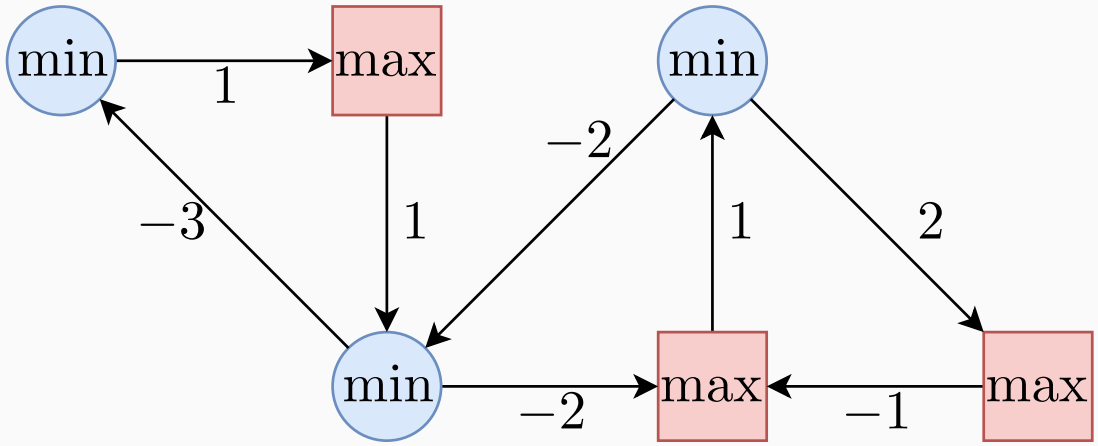


Figura 1: Esempio di mean-payoff game

Introduzione ai Graph Games

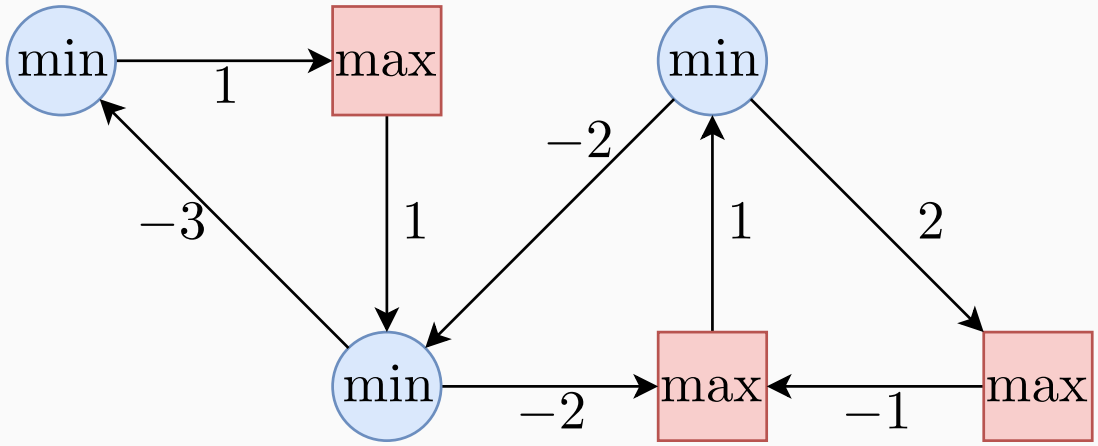


Figura 1: Esempio di mean-payoff game

Spostando la pedina dai vertici da loro posseduti, i giocatori costruiscono assieme una sequenza di pesi $\{w_i\}_i$, dalla quale si decide chi vince.

Introduzione ai Graph Games

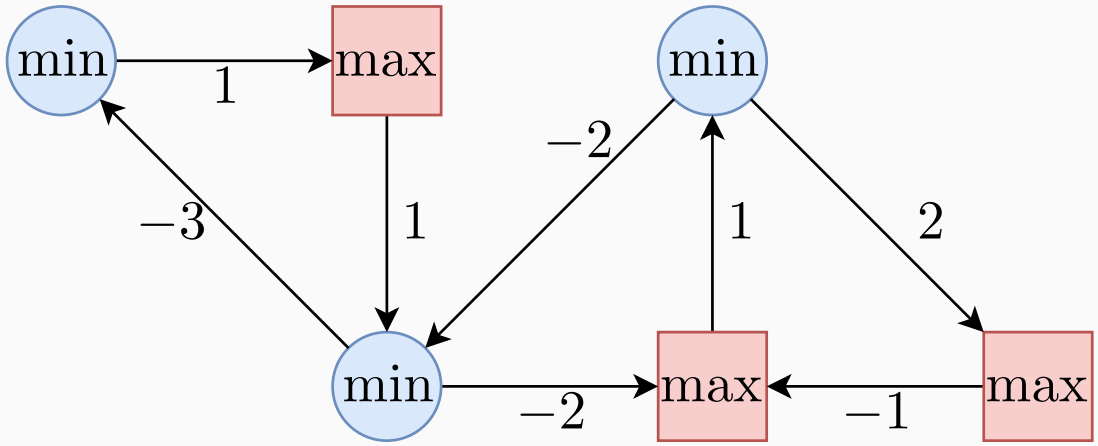


Figura 1: Esempio di mean-payoff game

Spostando la pedina dai vertici da loro posseduti, i giocatori costruiscono assieme una sequenza di pesi $\{w_i\}_i$, dalla quale si decide chi vince.

- Mean-payoff: $\nu(\{w_i\}_i) = \limsup_{k\to\infty} \frac{1}{k} \sum_{i\leq k} w_i$

Introduzione ai Graph Games

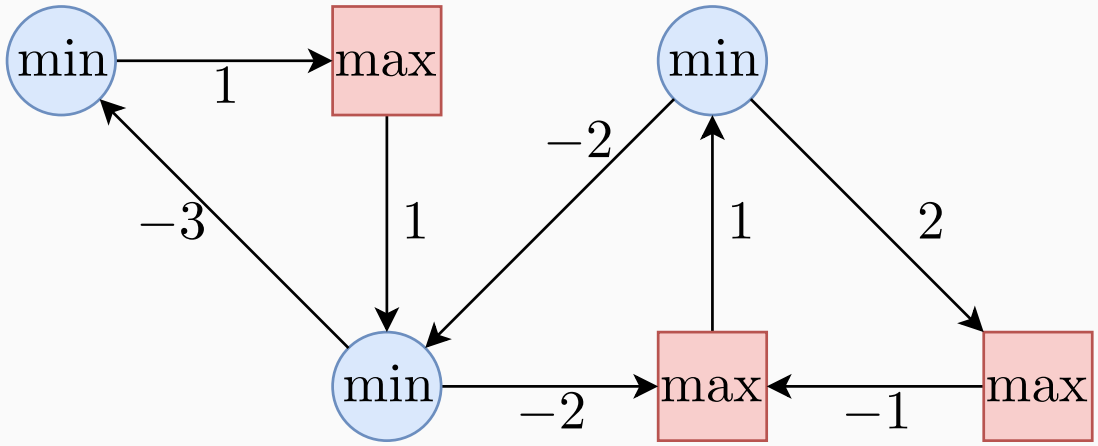


Figura 1: Esempio di mean-payoff game

Spostando la pedina dai vertici da loro posseduti, i giocatori costruiscono assieme una sequenza di pesi $\{w_i\}_i$, dalla quale si decide chi vince.

- Mean-payoff: $\nu(\{w_i\}_i) = \limsup_{k\to\infty} \frac{1}{k} \sum_{i\leq k} w_i$
- Parity: Max vince se il più alto numero che esce infinite volte è pari.

Introduzione ai Graph Games

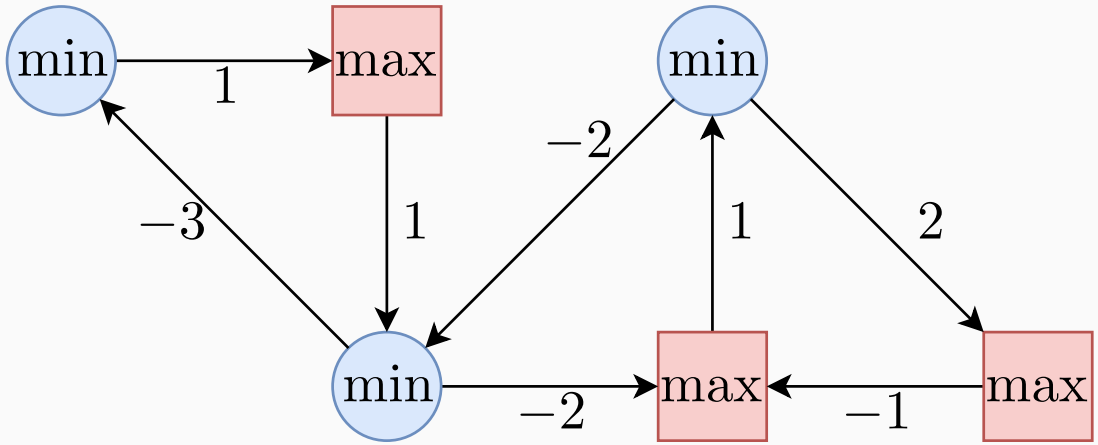


Figura 1: Esempio di mean-payoff game

Spostando la pedina dai vertici da loro posseduti, i giocatori costruiscono assieme una sequenza di pesi $\{w_i\}_i$, dalla quale si decide chi vince.

- Mean-payoff: $\nu(\{w_i\}_i) = \limsup_{k\to\infty} \frac{1}{k} \sum_{i\leq k} w_i$
- Parity: Max vince se il più alto numero che esce infinite volte è pari.
- Safety: alcuni archi sono "proibiti", se vengono percorsi Max perde.

Importanza dei Graph Games

50's

Lloyd Shapley introduce i graph games

- Applicazioni in economia (il mercato visto come avversario)
- Stocastici e multi-agente

Importanza dei Graph Games

50's

Lloyd Shapley introduce i graph games

- Applicazioni in economia (il mercato visto come avversario)
- Stocastici e multi-agente

70's

Ehrenfeucht e Mycielski introducono i mean-payoff games

Importanza dei Graph Games

50's

Lloyd Shapley introduce i graph games

- Applicazioni in economia (il mercato visto come avversario)
- Stocastici e multi-agente

70's

Ehrenfeucht e Mycielski introducono i mean-payoff games

- Equivalenti allo scheduling con constraint AND/OR (2004)

Importanza dei Graph Games

50's

Lloyd Shapley introduce i graph games

- Applicazioni in economia (il mercato visto come avversario)
- Stocastici e multi-agente

70's

Ehrenfeucht e Mycielski introducono i mean-payoff games

- Equivalenti allo scheduling con constraint AND/OR (2004)
- Pivoting rules di algoritmi del simplesso combinatorio (2014)

Importanza dei Graph Games

50's

Lloyd Shapley introduce i graph games

- Applicazioni in economia (il mercato visto come avversario)
- Stocastici e multi-agente

70's

Ehrenfeucht e Mycielski introducono i mean-payoff games

- Equivalenti allo scheduling con constraint AND/OR (2004)
- Pivoting rules di algoritmi del simplesso combinatorio (2014)

90's

Dexter Kozen introduce il μ-calcolo

- Logica per descrivere proprietà di sistemi di transizione finiti

Importanza dei Graph Games

50's

Lloyd Shapley introduce i graph games

- Applicazioni in economia (il mercato visto come avversario)
- Stocastici e multi-agente

70's

Ehrenfeucht e Mycielski introducono i mean-payoff games

- Equivalenti allo scheduling con constraint AND/OR (2004)
- Pivoting rules di algoritmi del simplesso combinatorio (2014)

90's

Dexter Kozen introduce il μ-calcolo

- Logica per descrivere proprietà di sistemi di transizione finiti
- Il model-checking è equivalente a risolvere i parity games

Richiami di teoria della complessità

Obiettivo della teoria della complessità: capire quante risorse computazionali servono per risolvere un problema utilizzando l'algoritmo migliore.

Richiami di teoria della complessità

Obiettivo della teoria della complessità: capire quante risorse computazionali servono per risolvere un problema utilizzando l'algoritmo migliore.

Non si riesce, ma si possono dividere i problemi in classi di complessità.

Richiami di teoria della complessità

Obiettivo della teoria della complessità: capire quante risorse computazionali servono per risolvere un problema utilizzando l'algoritmo migliore.

Non si riesce, ma si possono dividere i problemi in classi di complessità.

- P: Risolubili in tempo polinomiale (MCD, Linear programming, Prime).

Richiami di teoria della complessità

Obiettivo della teoria della complessità: capire quante risorse computazionali servono per risolvere un problema utilizzando l'algoritmo migliore.

Non si riesce, ma si possono dividere i problemi in classi di complessità.

- P: Risolubili in tempo polinomiale (MCD, Linear programming, Prime).
- NP: Soluzione verificabile in tempo polinomiale (Factoring, SAT).

Richiami di teoria della complessità

Obiettivo della teoria della complessità: capire quante risorse computazionali servono per risolvere un problema utilizzando l'algoritmo migliore.

Non si riesce, ma si possono dividere i problemi in classi di complessità.

- P: Risolubili in tempo polinomiale (MCD, Linear programming, Prime).
- NP: Soluzione verificabile in tempo polinomiale (Factoring, SAT).
  SAT $\in$ NP: c'è un certificato di "SI".

Richiami di teoria della complessità

Obiettivo della teoria della complessità: capire quante risorse computazionali servono per risolvere un problema utilizzando l'algoritmo migliore.

Non si riesce, ma si possono dividere i problemi in classi di complessità.

- P: Risolubili in tempo polinomiale (MCD, Linear programming, Prime).
- NP: Soluzione verificabile in tempo polinomiale (Factoring, SAT).
  SAT $\in$ NP: c'è un certificato di "SI". Simmetricamente definiamo
- coNP: Certificato di refutabilità controllabile in tempo polinomiale.

Richiami di teoria della complessità

Obiettivo della teoria della complessità: capire quante risorse computazionali servono per risolvere un problema utilizzando l'algoritmo migliore.

Non si riesce, ma si possono dividere i problemi in classi di complessità.

- P: Risolubili in tempo polinomiale (MCD, Linear programming, Prime).
- NP: Soluzione verificabile in tempo polinomiale (Factoring, SAT).
  SAT $\in$ NP: c'è un certificato di "SI". Simmetricamente definiamo
- coNP: Certificato di refutabilità controllabile in tempo polinomiale.
- NP $\cap$ coNP: Certificati sia per la solvibilità che per la non-solvibilità.

Richiami di teoria della complessità

Obiettivo della teoria della complessità: capire quante risorse computazionali servono per risolvere un problema utilizzando l'algoritmo migliore.

Non si riesce, ma si possono dividere i problemi in classi di complessità.

- P: Risolubili in tempo polinomiale (MCD, Linear programming, Prime).
- NP: Soluzione verificabile in tempo polinomiale (Factoring, SAT).
  SAT $\in$ NP: c'è un certificato di "SI". Simmetricamente definiamo
- coNP: Certificato di refutabilità controllabile in tempo polinomiale.
- NP $\cap$ coNP: Certificati sia per la solvibilità che per la non-solvibilità.
  - Pochi problemi noti in NP $\cap$ coNP (Factoring, SV approssimato).

Richiami di teoria della complessità

Obiettivo della teoria della complessità: capire quante risorse computazionali servono per risolvere un problema utilizzando l'algoritmo migliore.

Non si riesce, ma si possono dividere i problemi in classi di complessità.

- P: Risolubili in tempo polinomiale (MCD, Linear programming, Prime).
- NP: Soluzione verificabile in tempo polinomiale (Factoring, SAT).
  SAT $\in$ NP: c'è un certificato di "SI". Simmetricamente definiamo
- coNP: Certificato di refutabilità controllabile in tempo polinomiale.
- NP $\cap$ coNP: Certificati sia per la solvibilità che per la non-solvibilità.
  - Pochi problemi noti in NP $\cap$ coNP (Factoring, SV approssimato).
  - Ritenuti facili (se NP-completi collasso gerarchia polinomiale).

Richiami di teoria della complessità

Obiettivo della teoria della complessità: capire quante risorse computazionali servono per risolvere un problema utilizzando l'algoritmo migliore.

Non si riesce, ma si possono dividere i problemi in classi di complessità.

- P: Risolubili in tempo polinomiale (MCD, Linear programming, Prime).
- NP: Soluzione verificabile in tempo polinomiale (Factoring, SAT).
  SAT $\in$ NP: c'è un certificato di "SI". Simmetricamente definiamo
- coNP: Certificato di refutabilità controllabile in tempo polinomiale.
- NP $\cap$ coNP: Certificati sia per la solvibilità che per la non-solvibilità.
  - Pochi problemi noti in NP $\cap$ coNP (Factoring, SV approssimato).
  - Ritenuti facili (se NP-completi collasso gerarchia polinomiale).
  - Risolvere i graph games è in NP $\cap$ coNP.

Richiami di teoria della complessità

Obiettivo della teoria della complessità: capire quante risorse computazionali servono per risolvere un problema utilizzando l'algoritmo migliore.

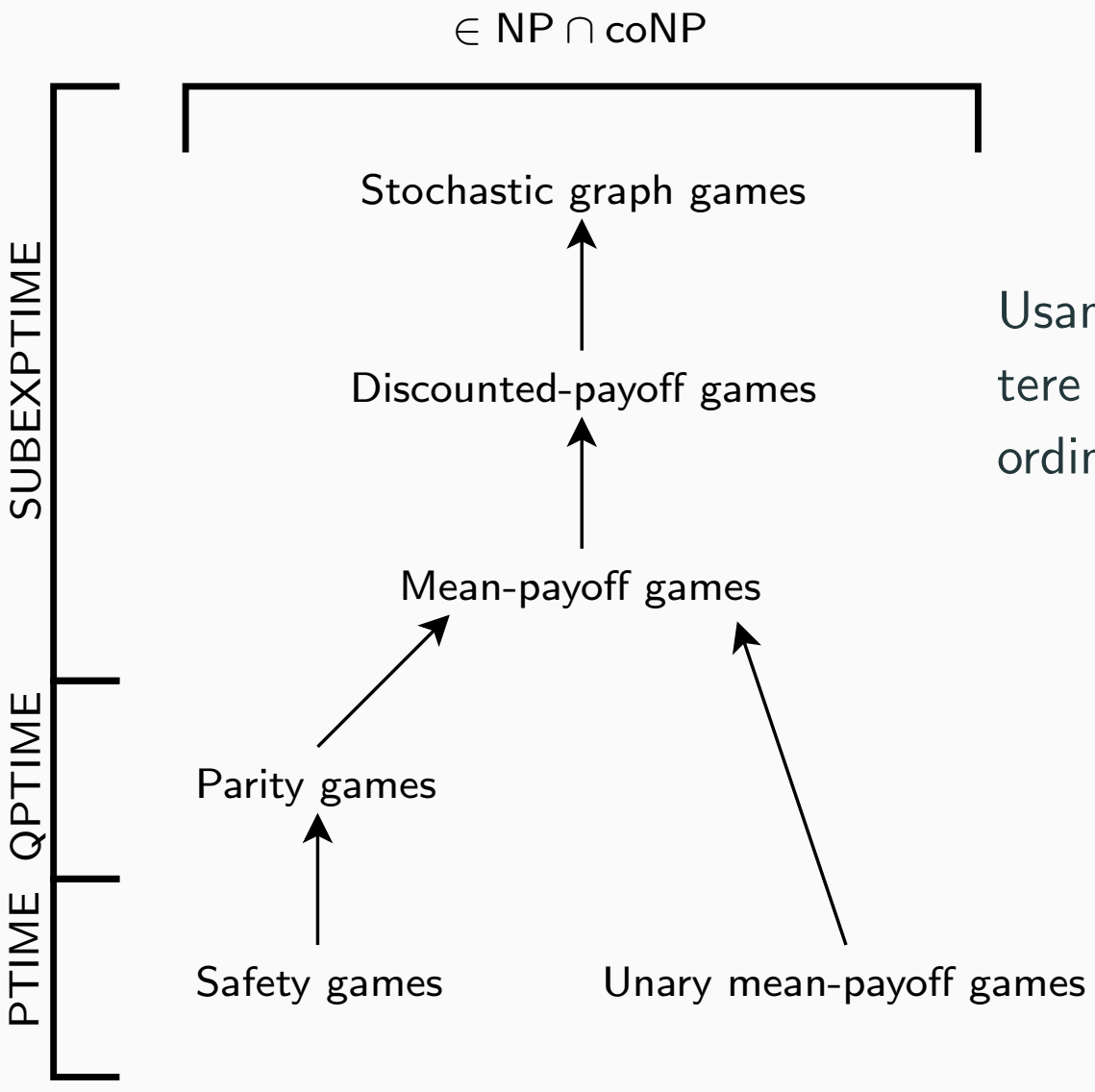
Non si riesce, ma si possono dividere i problemi in classi di complessità.

- P: Risolubili in tempo polinomiale (MCD, Linear programming, Prime).
- NP: Soluzione verificabile in tempo polinomiale (Factoring, SAT).
  SAT $\in$ NP: c'è un certificato di "SI". Simmetricamente definiamo
- coNP: Certificato di refutabilità controllabile in tempo polinomiale.
- NP $\cap$ coNP: Certificati sia per la solvibilità che per la non-solvibilità.
  - Pochi problemi noti in NP $\cap$ coNP (Factoring, SV approssimato).
  - Ritenuti facili (se NP-completi collasso gerarchia polinomiale).
  - Risolvere i graph games è in NP $\cap$ coNP.

Riduzione di Turing

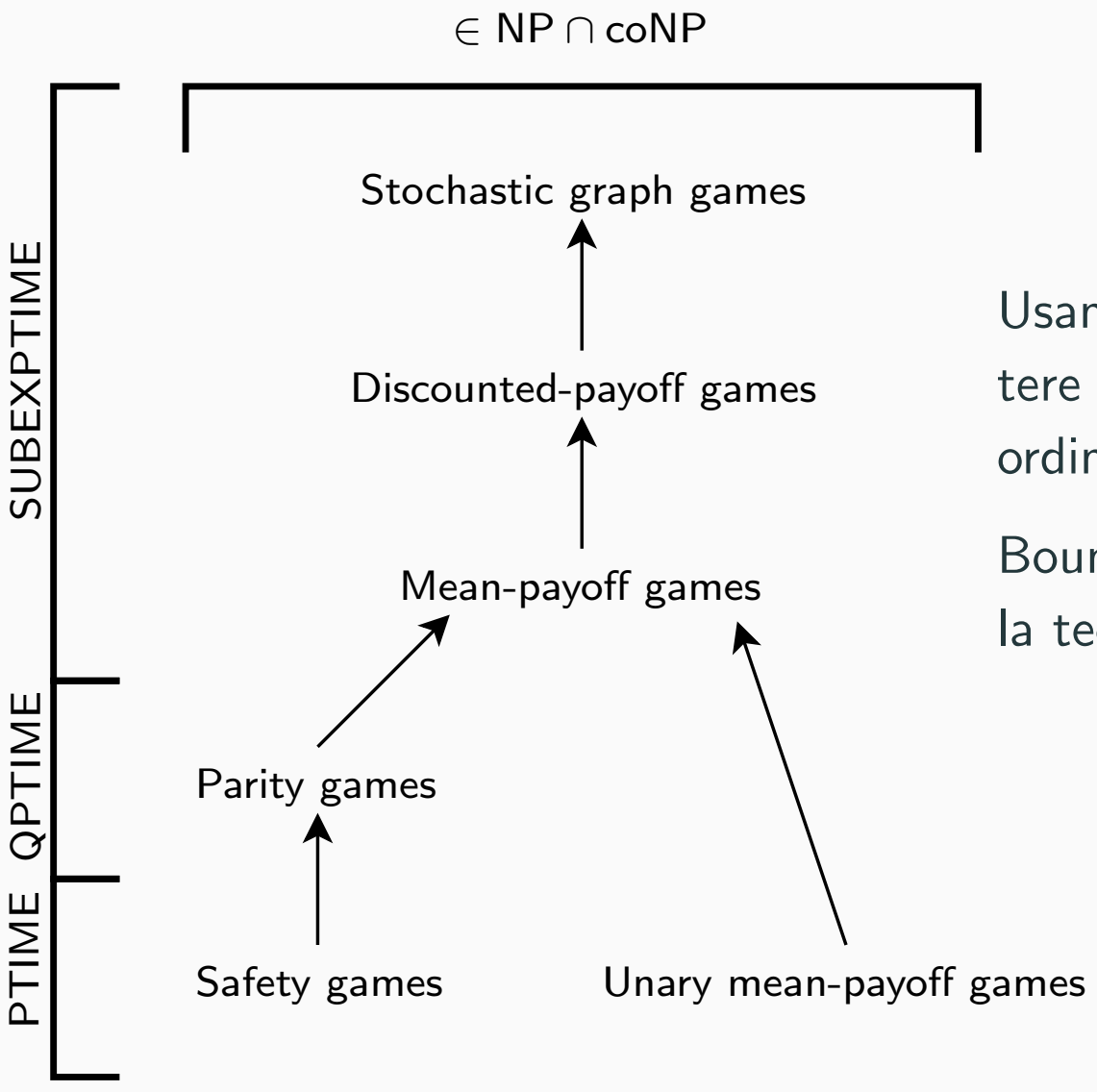
Un problema A è Turing-riducibile ad un problema B se, data una "scatola nera" che risolve B, posso costruire un algoritmo che risolve A.

Gerarchia dei Graph Games



Usando la riducibilità possiamo mettere i graph games in una gerarchia, ordinandoli "per difficoltà".

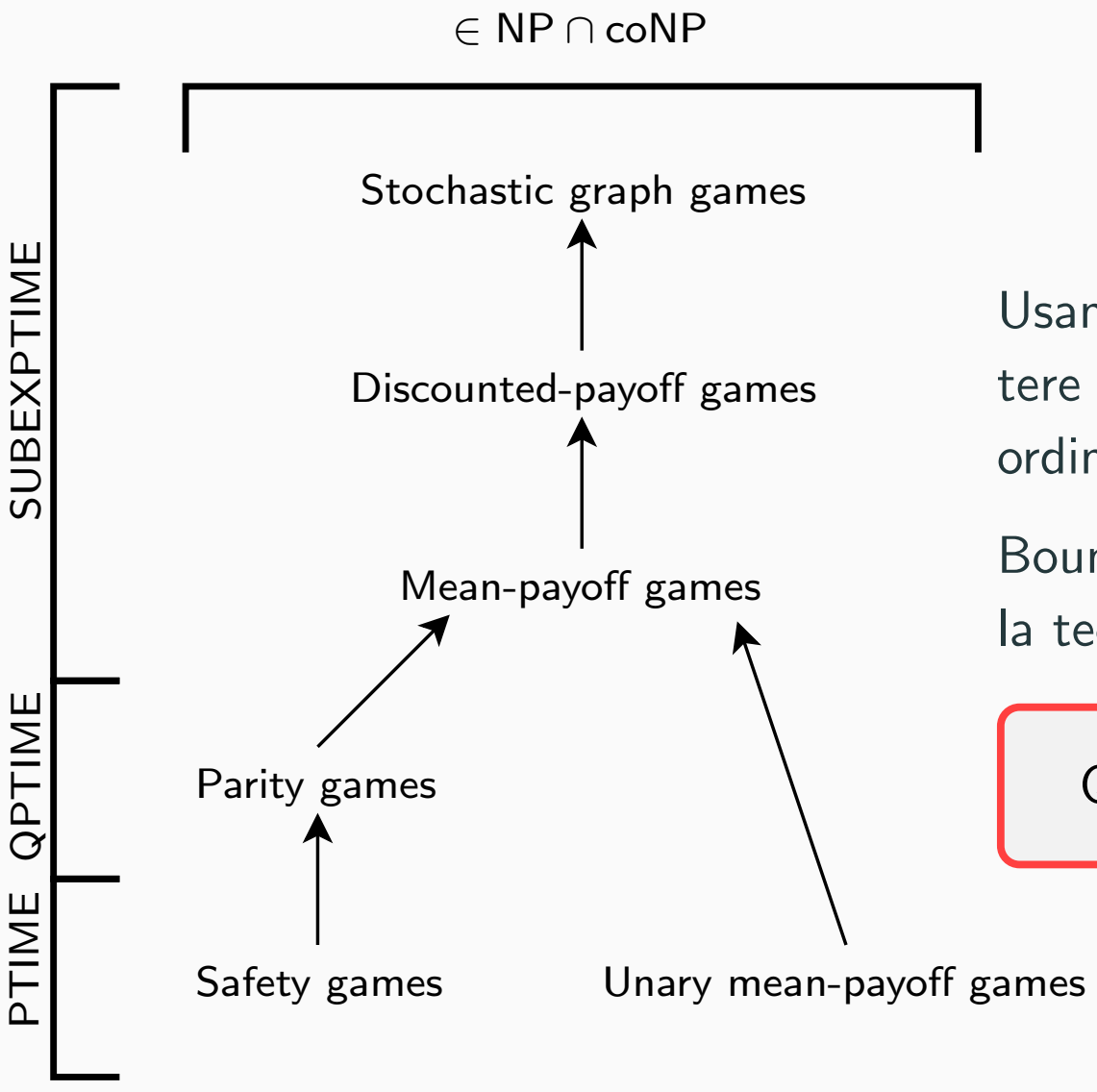
Gerarchia dei Graph Games



Usando la riducibilità possiamo mettere i graph games in una gerarchia, ordinandoli "per difficoltà".

Bound subesponenziale ottenuto con la teoria degli LP-type problems.

Gerarchia dei Graph Games



Usando la riducibilità possiamo mettere i graph games in una gerarchia, ordinandoli "per difficoltà".

Bound subesponenziale ottenuto con la teoria degli LP-type problems.

$$\text{QP} = O(e^{\log^{\alpha} n}) \text{ con } \alpha \in \mathbb{R}$$

Stato dell'arte per la complessità dei graph games

Anni '90

Ci si interroga sulla complessità dei graph games (Condon), primi algoritmi risolutivi (Zielonka), riduzioni (Zwick e Paterson) ed appartenenza ad $\mathrm{UP} \cap \mathrm{coUP}$ (Jurdziński).

Stato dell'arte per la complessità dei graph games

Anni '90 Ci si interroga sulla complessità dei graph games (Condon), primi algoritmi risolutivi (Zielonka), riduzioni (Zwick e Paterson) ed appartenenza ad $\mathrm{UP} \cap \mathrm{coUP}$ (Jurdziński).

Anni '00 Sviluppo di altri algoritmi (Small Progress Measures di Jurdziński) ed equivalenza dei giochi stocastici (Andersson e Miltersen).

Stato dell'arte per la complessità dei graph games

Anni '90 Ci si interroga sulla complessità dei graph games (Condon), primi algoritmi risolutivi (Zielonka), riduzioni (Zwick e Paterson) ed appartenenza ad $\mathrm{UP} \cap \mathrm{coUP}$ (Jurdziński).

Anni '00 Sviluppo di altri algoritmi (Small Progress Measures di Jurdziński) ed equivalenza dei giochi stocastici (Andersson e Miltersen).

2017 Calude, Jain, Khoussainov, Li e Stephan risolvono i parity games in tempo quasi-polinomiale.

Stato dell'arte per la complessità dei graph games

Anni '90 Ci si interroga sulla complessità dei graph games (Condon), primi algoritmi risolutivi (Zielonka), riduzioni (Zwick e Paterson) ed appartenenza ad $\mathrm{UP} \cap \mathrm{coUP}$ (Jurdziński).

Anni '00 Sviluppo di altri algoritmi (Small Progress Measures di Jurdziński) ed equivalenza dei giochi stocastici (Andersson e Miltersen).

2017 Calude, Jain, Khoussainov, Li e Stephan risolvono i parity games in tempo quasi-polinomiale.

2017-19 Vengono adatti algoritmi esistenti per renderli quasi-polinomiali (Succinct Progress Measures di Jurdziński e Lazić, Register-index di Lehtinen, Adattamento algoritmo Zielonka di Parys).

Stato dell'arte per la complessità dei graph games

Anni '90 Ci si interroga sulla complessità dei graph games (Condon), primi algoritmi risolutivi (Zielonka), riduzioni (Zwick e Paterson) ed appartenenza ad $\mathrm{UP} \cap \mathrm{coUP}$ (Jurdziński).

Anni '00 Sviluppo di altri algoritmi (Small Progress Measures di Jurdziński) ed equivalenza dei giochi stocastici (Andersson e Miltersen).

2017 Calude, Jain, Khoussainov, Li e Stephan risolvono i parity games in tempo quasi-polinomiale.

2017-19 Vengono adatti algoritmi esistenti per renderli quasi-polinomiali (Succinct Progress Measures di Jurdziński e Lazić, Register-index di Lehtinen, Adattamento algoritmo Zielonka di Parys).

2019 Czerwiński, Daviaud, Fijalkow, Jurdziński, Lazić e Parys dimostrano bound inferiori per gli algoritmi esistenti per parity.

Risoluzione algoritmica

Determinatezza posizionale

Determinatezza posizionale

Una strategia ρ si dice *posizionale* quando la mossa scelta dipende solo dal vertice corrente e non dalla storia passata della partita.

Determinatezza posizionale

Una strategia ρ si dice *posizionale* quando la mossa scelta dipende solo dal vertice corrente e non dalla storia passata della partita.

Un graph game si dice *posizionalmente determinato* se ciascun giocatore ha una strategia ottimale posizionale.

Determinatezza posizionale

Una strategia ρ si dice *posizionale* quando la mossa scelta dipende solo dal vertice corrente e non dalla storia passata della partita.

Un graph game si dice *posizionalmente determinato* se ciascun giocatore ha una strategia ottimale posizionale.

Parity: Emerson (1995), Mean-payoff: Ehrenfeuch e Mycielski (1979).

Determinatezza posizionale

Una strategia ρ si dice *posizionale* quando la mossa scelta dipende solo dal vertice corrente e non dalla storia passata della partita.

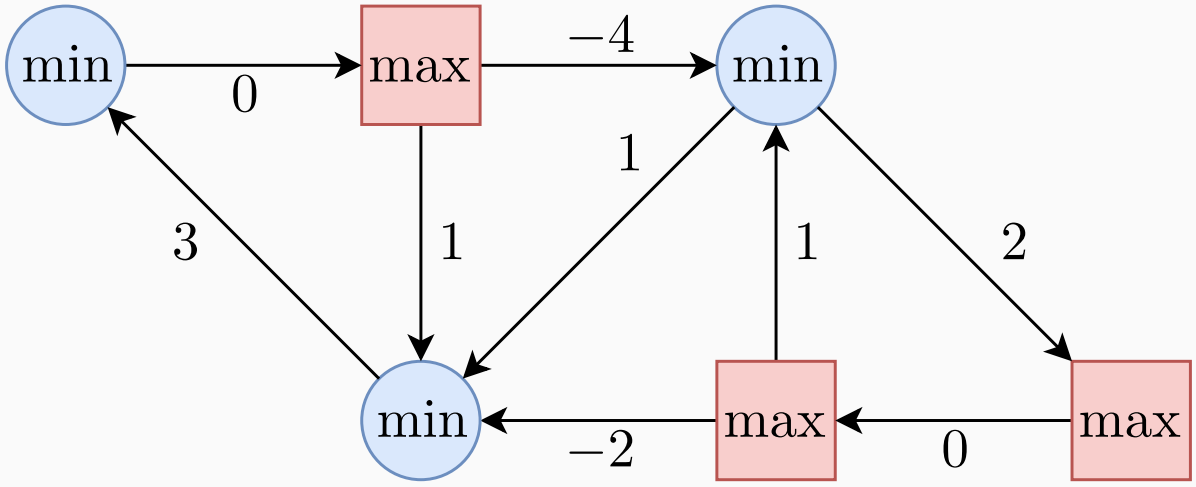
Un graph game si dice *posizionalmente determinato* se ciascun giocatore ha una strategia ottimale posizionale.

Parity: Emerson (1995), Mean-payoff: Ehrenfeuch e Mycielski (1979).

La determinatezza posizionale rende immediatamente decidibile il problema: posso scontrare tutte le strategie posizionali l'una contro l'altra.

In generale questo richiede tempo $O(2^m)$, dove m sono il numero di archi.

Risolvere i mean-payoff games con i potenziali

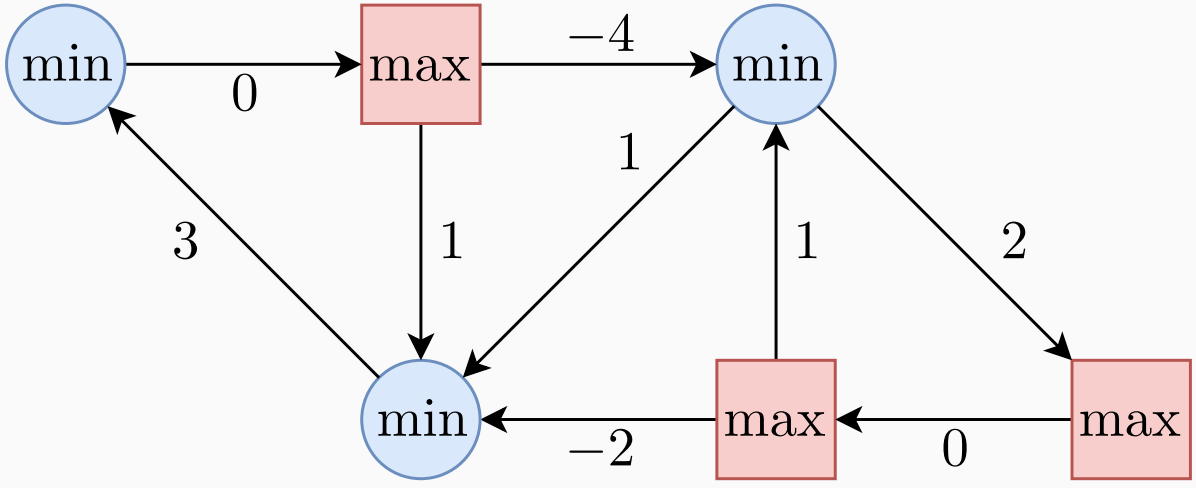


Se avessimo un mean-payoff game tale che per ogni vertice

- Max ha *almeno un* arco di uscita con peso non-negativo
- Min ha *tutti* gli archi di uscita con pesi non-negativo

allora sarebbe facile stabilire una strategia vincente per Max.

Risolvere i mean-payoff games con i potenziali



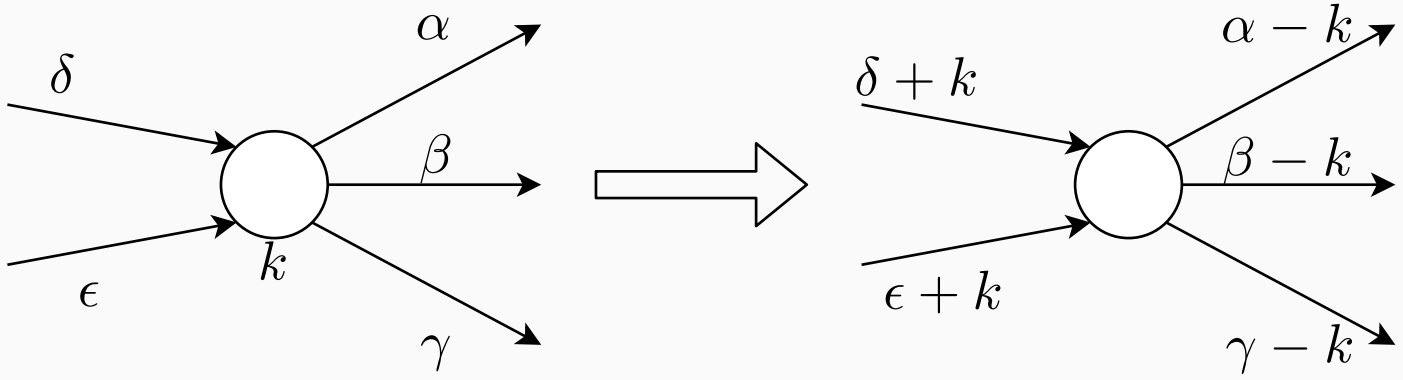
Se avessimo un mean-payoff game tale che per ogni vertice

- Max ha *almeno un* arco di uscita con peso non-negativo
- Min ha *tutti* gli archi di uscita con pesi non-negativo

allora sarebbe facile stabilire una strategia vincente per Max.

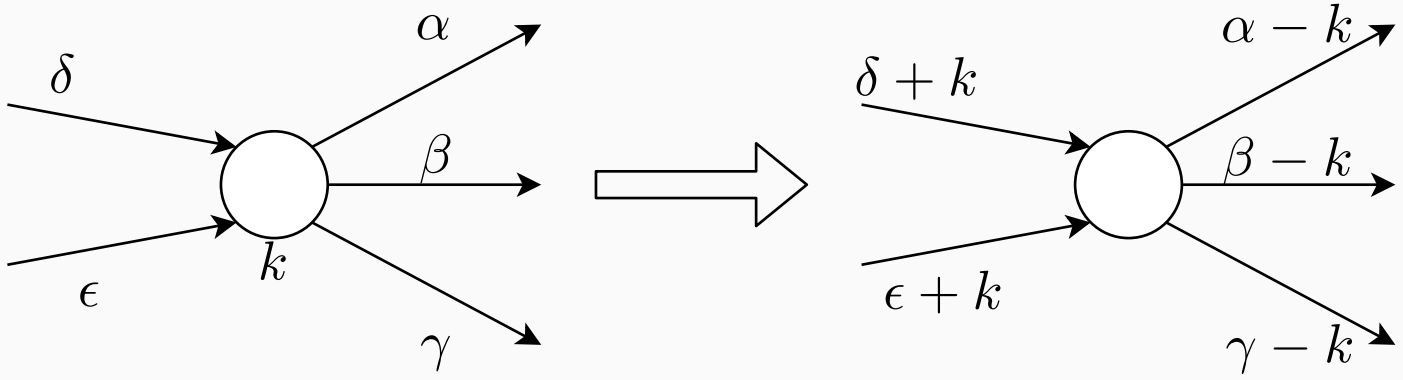
Un mean-payoff in cui Max vince può essere trasformato in questa forma.

Un mean-payoff in cui Max vince può essere trasformato in questa forma.



Strategie preservate dalla trasformazione indotta dal potenziale $k : V \to \mathbb{Z}$.

Un mean-payoff in cui Max vince può essere trasformato in questa forma.



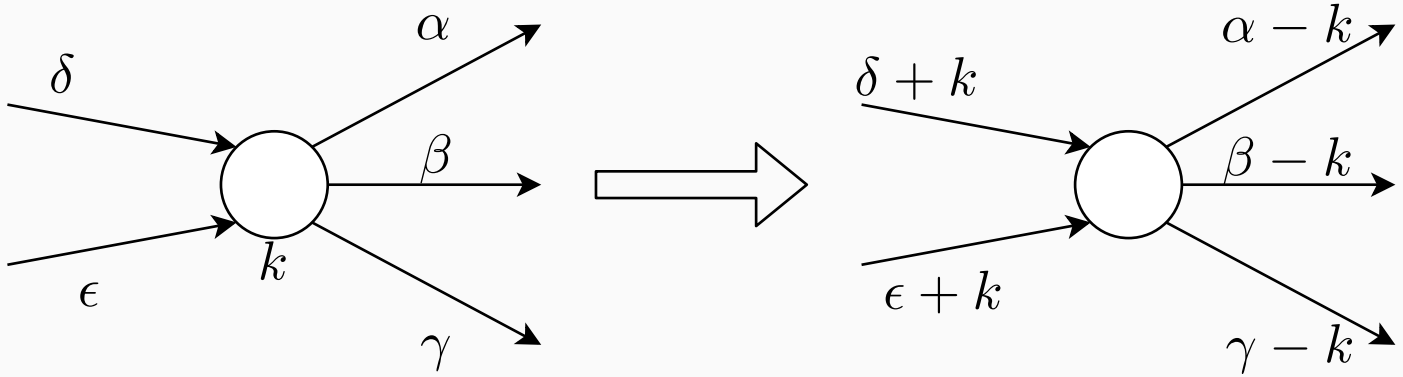
Strategie preservate dalla trasformazione indotta dal potenziale $k : V \to \mathbb{Z}$.

Imponiamo la condizione sul potenziale k in modo che da ogni nodo di Max esca almeno un arco non-negativo e da ogni nodo di Min escano solo archi non-negativi:

$$k(v) \leq \min_u \; k(u) + w(v,u) \qquad \text{se } v \in V_{\mathsf{Min}}$$

$$k(v) \leq \max_u \; k(u) + w(v,u) \qquad \text{se } v \in V_{\mathsf{Max}}$$

Un mean-payoff in cui Max vince può essere trasformato in questa forma.



Strategie preservate dalla trasformazione indotta dal potenziale $k : V \to \mathbb{Z}$.

Imponiamo la condizione sul potenziale k in modo che da ogni nodo di Max esca almeno un arco non-negativo e da ogni nodo di Min escano solo archi non-negativi:

$$k(v) \leq \min_u \; k(u) + w(v,u) \qquad T(k)(v) = \max\left\{k(v), \min_u \; k(u) + w(v,u)\right\}$$

$$k(v) \leq \max_u \; k(u) + w(v,u) \qquad T(k)(v) = \max\left\{k(v), \max_u \; k(u) + w(v,u)\right\}$$

Ne possiamo ricavare un algoritmo di update in tempo pseudo-polinomiale, ovvero lineare nel massimo peso W e polinomiale nel numero di nodi n.

Risolvere i parity games con le progress measures

- Idea come nei potenziali: teniamo un punteggio per ogni vertice.
  Ad ogni vertice associamo d numeri interi (uno per ogni priorità), che rappresentano quante volte ho visto una certa priorità da quando ho visto una priorità maggiore precedente.

Risolvere i parity games con le progress measures

- Idea come nei potenziali: teniamo un punteggio per ogni vertice.
  Ad ogni vertice associamo d numeri interi (uno per ogni priorità), che rappresentano quante volte ho visto una certa priorità da quando ho visto una priorità maggiore precedente.
- Se un giocatore può vincere allora può forzare un contatore di una sua priorità a raggiungere n, numero dei nodi. L'altro giocatore può fare la stessa cosa. Otteniamo quindi un gioco equivalente e *finito*.

Risolvere i parity games con le progress measures

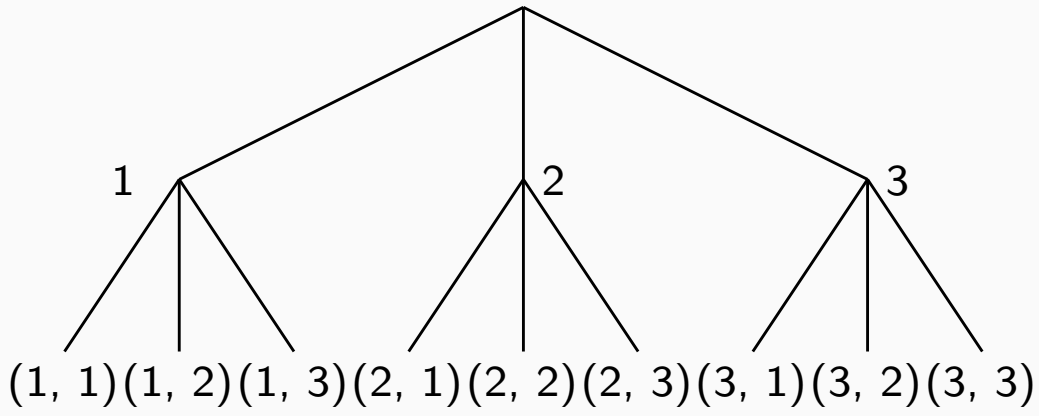
- Idea come nei potenziali: teniamo un punteggio per ogni vertice.
  Ad ogni vertice associamo d numeri interi (uno per ogni priorità), che rappresentano quante volte ho visto una certa priorità da quando ho visto una priorità maggiore precedente.
- Se un giocatore può vincere allora può forzare un contatore di una sua priorità a raggiungere n, numero dei nodi. L'altro giocatore può fare la stessa cosa. Otteniamo quindi un gioco equivalente e *finito*.

Possiamo però migliorare definendo un algoritmo di update simile al precedente per aggiornare i punteggi di ogni vertice.

L'algoritmo può avere runtime esponenziale. Infatti, è limitato solo dal numero di possibili scelte che può esplorare per ogni nodo, che sono n^d.

Progress Measures come alberi ordinati

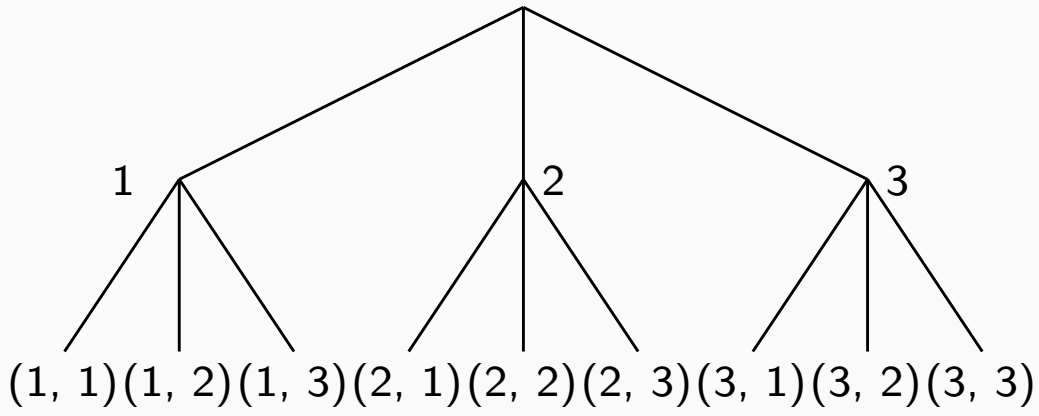
Posso vedere la lista di contatori come foglie di un albero ordinato.



Se il grafo ha n vertici l'albero ha n^d foglie.

Progress Measures come alberi ordinati

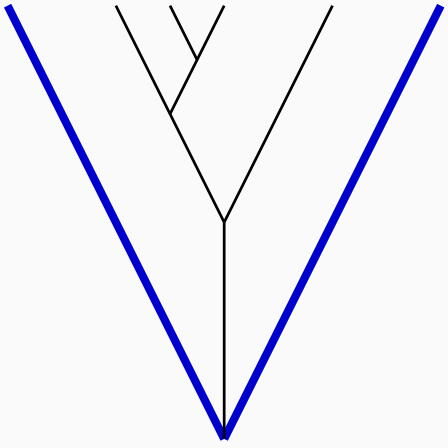
Posso vedere la lista di contatori come foglie di un albero ordinato.

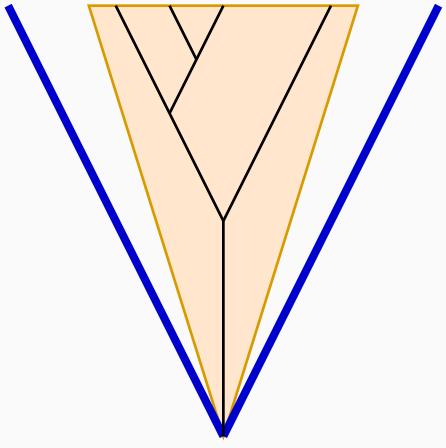


Se il grafo ha n vertici l'albero ha n^d foglie.

L'update è un operatore monotono sulle d-tuple di rami di questo albero ordinato, e ne esplora un sottoalbero.

Progress Measures Succinte

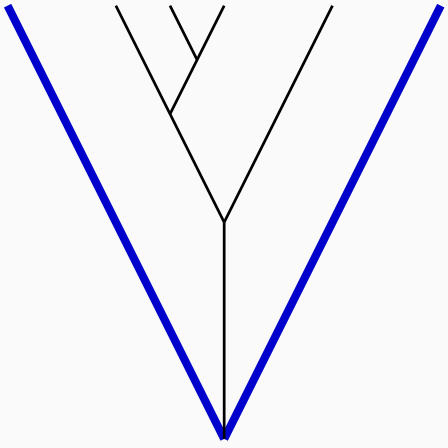


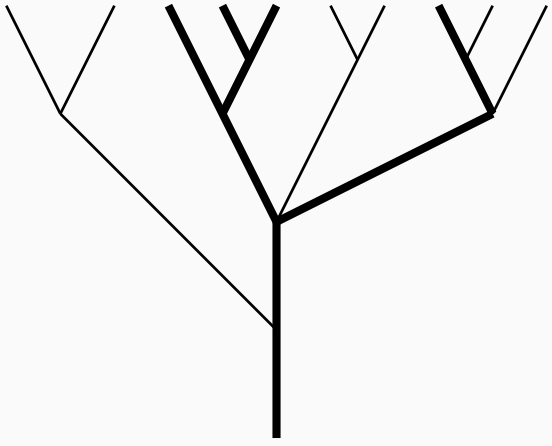


L'algoritmo di update è stabile

- per passaggio ad un sottoalbero che contiene la soluzione.

Progress Measures Succinte





L'algoritmo di update è stabile

- per passaggio ad un sottoalbero che contiene la soluzione.
- per immersione della soluzione in un altro albero, preservando l'ordine.

La variante costruisce un albero quasi-polinomiale in cui si immergono tutti gli alberi di taglia n e di altezza d. Un tale albero si dice (n, d)-universale.

È effettivamente possibile costruire un albero universale più piccolo?

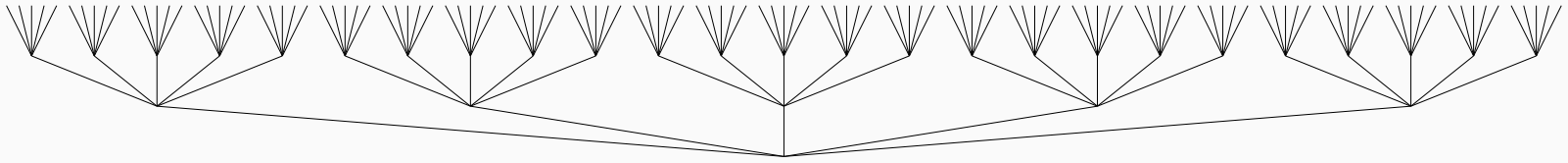


Figura 2: Albero $(5, 3)$-universale completo, con 125 foglie.

È effettivamente possibile costruire un albero universale più piccolo?

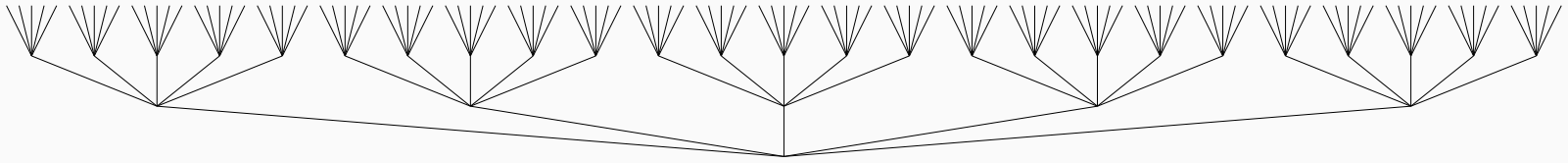


Figura 2: Albero $(5, 3)$-universale completo, con 125 foglie.

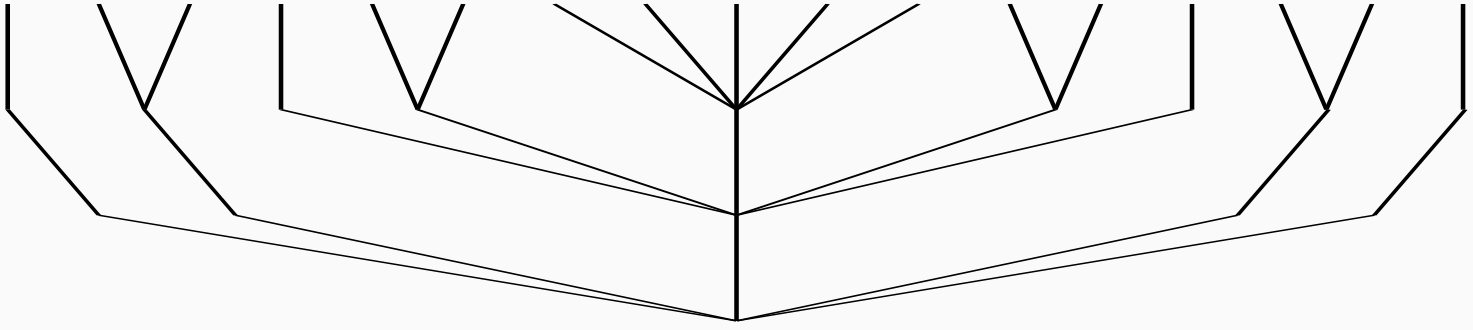
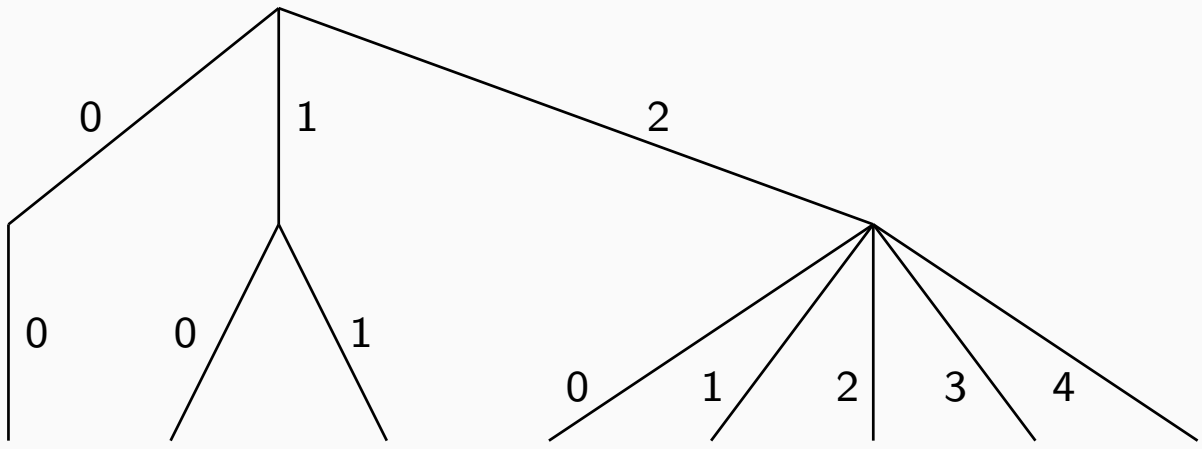
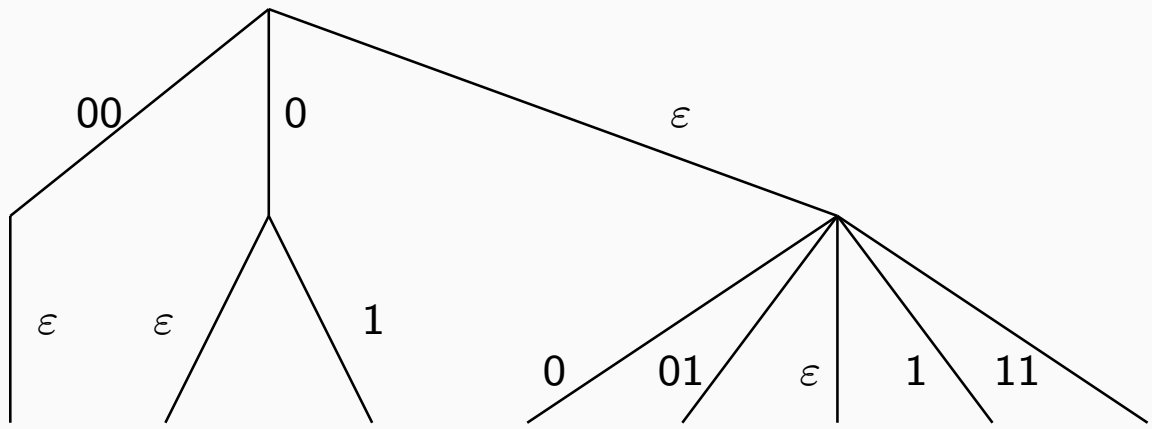


Figura 3: Minimo albero $(5, 3)$-universale, con 17 foglie.

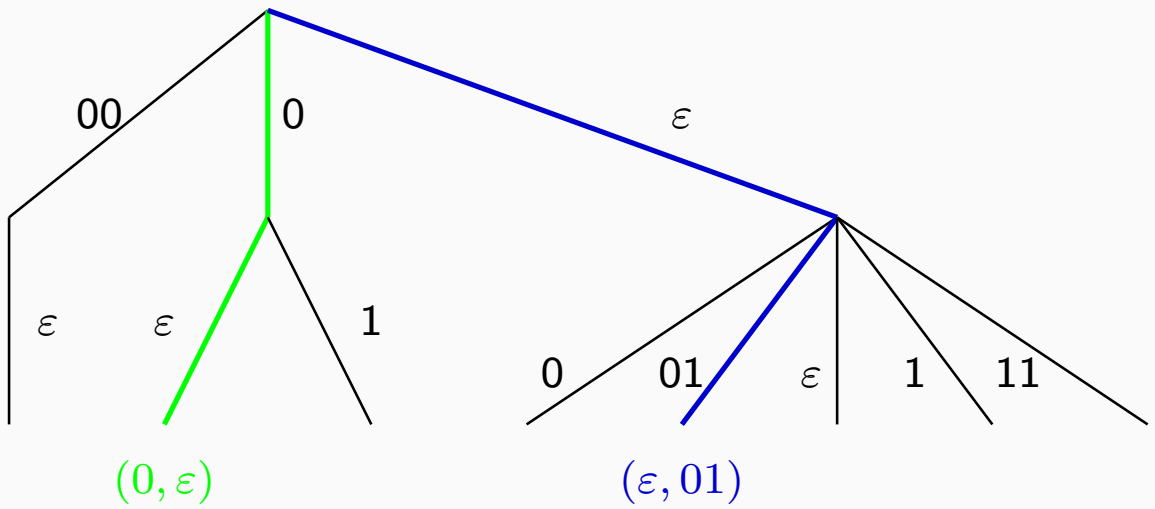
- Vogliamo trovare una codifica breve ad ogni foglia di una $\mu : V \to \mathbb{N}^d$.



- Vogliamo trovare una codifica breve ad ogni foglia di una $\mu : V \to \mathbb{N}^d$.
- Ricorsivamente dividiamo l'albero in tre parti: $M, L_<, L_>$ dove M è la direzione che divide l'albero in maniera bilanciata, $L_<$ sono le foglie minori di M, ed $L_>$ sono le foglie maggiori di M.
  Assegniamo poi una codifica che mantiene l'altezza alle tre parti dell'albero.



- Vogliamo trovare una codifica breve ad ogni foglia di una $\mu : V \to \mathbb{N}^d$.
- Ricorsivamente dividiamo l'albero in tre parti: $M, L_<, L_>$ dove M è la direzione che divide l'albero in maniera bilanciata, $L_<$ sono le foglie minori di M, ed $L_>$ sono le foglie maggiori di M.
  Assegniamo poi una codifica che mantiene l'altezza alle tre parti dell'albero.
- Ogni foglia dell'albero è codificata da una d-tupla di lunghezza $\leq \lceil \log n \rceil$
- Notiamo che preserva l'ordine dei valori $\mu(v)$, necessario per l'algoritmo.



- Vogliamo trovare una codifica breve ad ogni foglia di una $\mu : V \to \mathbb{N}^d$.
- Ricorsivamente dividiamo l'albero in tre parti: $M, L_<, L_>$ dove M è la direzione che divide l'albero in maniera bilanciata, $L_<$ sono le foglie minori di M, ed $L_>$ sono le foglie maggiori di M.
  Assegniamo poi una codifica che mantiene l'altezza alle tre parti dell'albero.
- Ogni foglia dell'albero è codificata da una d-tupla di lunghezza $\leq \lceil \log n \rceil$
- Notiamo che preserva l'ordine dei valori $\mu(v)$, necessario per l'algoritmo.
- Aggiungiamo un simbolo separatore: $(0, \varepsilon) \to 0\triangle, (10, 1) \to 10\triangle 1$

- Vogliamo trovare una codifica breve ad ogni foglia di una $\mu : V \to \mathbb{N}^d$.
- Ricorsivamente dividiamo l'albero in tre parti: $M, L_<, L_>$ dove M è la direzione che divide l'albero in maniera bilanciata, $L_<$ sono le foglie minori di M, ed $L_>$ sono le foglie maggiori di M.
  Assegniamo poi una codifica che mantiene l'altezza alle tre parti dell'albero.
- Ogni foglia dell'albero è codificata da una d-tupla di lunghezza $\leq \lceil \log n \rceil$
- Notiamo che preserva l'ordine dei valori $\mu(v)$, necessario per l'algoritmo.
- Aggiungiamo un simbolo separatore: $(0, \varepsilon) \to 0\triangle, (10, 1) \to 10\triangle 1$
- Il numero totale di codifiche è quindi

$$\binom{d + \lceil \log n \rceil + 1}{\lceil \log n \rceil} \leq (d + \lceil \log n \rceil + 1)^{\lceil \log n \rceil} = O(n^{\log n})$$

  e si ottiene così l'algoritmo quasi-polinomiale.

- Vogliamo trovare una codifica breve ad ogni foglia di una $\mu : V \to \mathbb{N}^d$.
- Ricorsivamente dividiamo l'albero in tre parti: $M, L_<, L_>$ dove M è la direzione che divide l'albero in maniera bilanciata, $L_<$ sono le foglie minori di M, ed $L_>$ sono le foglie maggiori di M.
  Assegniamo poi una codifica che mantiene l'altezza alle tre parti dell'albero.
- Ogni foglia dell'albero è codificata da una d-tupla di lunghezza $\leq \lceil \log n \rceil$
- Notiamo che preserva l'ordine dei valori $\mu(v)$, necessario per l'algoritmo.
- Aggiungiamo un simbolo separatore: $(0, \varepsilon) \to 0\triangle, (10, 1) \to 10\triangle 1$
- Il numero totale di codifiche è quindi

$$\binom{d + \lceil \log n \rceil + 1}{\lceil \log n \rceil} \leq (d + \lceil \log n \rceil + 1)^{\lceil \log n \rceil} = O(n^{\log n})$$

  e si ottiene così l'algoritmo quasi-polinomiale.
- Possiamo fare di meglio? Esiste un albero universale con meno foglie?

Alberi Universali e bound inferiori

Un albero (n, h)-universale è un albero ordinato dove si immergono in modo order-preserving tutti gli alberi ordinati di altezza h e con al più n foglie.

Minima dimensione di un albero universale

Un albero (n, h)-universale ha almeno $\binom{\lfloor \log n \rfloor + h - 1}{\lfloor \log n \rfloor - 1} \geq n^{\log h - 2}$ foglie.

Alberi Universali e bound inferiori

Un albero (n, h)-universale è un albero ordinato dove si immergono in modo order-preserving tutti gli alberi ordinati di altezza h e con al più n foglie.

Minima dimensione di un albero universale

Un albero (n, h)-universale ha almeno $\binom{\lfloor \log n \rfloor + h - 1}{\lfloor \log n \rfloor - 1} \geq n^{\log h - 2}$ foglie.

È un lower bound anche per gli approcci utilizzati dagli altri algoritmi, tutti basati sull'uso di "punteggi di gioco" da aggiornare.

Pertanto gli algoritmi attualmente noti non possono avere varianti che scendano sotto il bound quasi-polinomiale.

Un gioco intermedio

Importanza della codifica dei problemi

- Problema decisionale: output binario (vince / perde).
- Importante la codifica dell'input: determina la complessità di un algoritmo.

Importanza della codifica dei problemi

- Problema decisionale: output binario (vince / perde).
- Importante la codifica dell'input: determina la complessità di un algoritmo.
- Per i numeri interi si usa solitamente la scrittura in base, size$(n) = \lceil \log n \rceil$.
- Per rendere un problema artificialmente più facile si può usare la codifica unaria, ovvero porre size$(n) = n$.

Importanza della codifica dei problemi

- Problema decisionale: output binario (vince / perde).
- Importante la codifica dell'input: determina la complessità di un algoritmo.
- Per i numeri interi si usa solitamente la scrittura in base, size$(n) = \lceil \log n \rceil$.
- Per rendere un problema artificialmente più facile si può usare la codifica unaria, ovvero porre size$(n) = n$.

Il precedente algoritmo per i mean-payoff impiegherebbe tempo polinomiale se i pesi fossero scritti in unario.

Presentiamo un nuovo graph game, basato sul complicare l'insieme dei pesi ma al contempo semplificarlo scrivendo i pesi in unario.

Un gioco intermedio: Stacked Unary mean-payoff game

- Pesi polinomiali: $w_i = p(x) = \sum_{j \leq d} a_j x^j$
- Polinomi ordinati lessicograficamente: $p(x) \geq q(x) \Leftrightarrow \text{coeff}(p) \geq_{\text{Lex}} \text{coeff}(q)$
- Numeri codificati in unario: $\text{size}(2x^2 + 3x - 1) = 2 + 3 + 1$

Condizione di vittoria come i mean-payoff: $\limsup_k \frac{1}{n} \sum_{i \leq k} w_i(x) \gtreqless_{\text{Lex}} 0$.

Un gioco intermedio: Stacked Unary mean-payoff game

- Pesi polinomiali: $w_i = p(x) = \sum_{j \leq d} a_j x^j$
- Polinomi ordinati lessicograficamente: $p(x) \geq q(x) \Leftrightarrow \text{coeff}(p) \geq_{\text{Lex}} \text{coeff}(q)$
- Numeri codificati in unario: $\text{size}(2x^2 + 3x - 1) = 2 + 3 + 1$

Condizione di vittoria come i mean-payoff: $\limsup_k \frac{1}{n} \sum_{i \leq k} w_i(x) \gtreqless_{\text{Lex}} 0$.

Determinatezza posizionale

Gli stacked unary mean-payoff games sono posizionalmente determinati.

Un gioco intermedio: Stacked Unary mean-payoff game

- Pesi polinomiali: $w_i = p(x) = \sum_{j \leq d} a_j x^j$
- Polinomi ordinati lessicograficamente: $p(x) \geq q(x) \Leftrightarrow \text{coeff}(p) \geq_{\text{Lex}} \text{coeff}(q)$
- Numeri codificati in unario: $\text{size}(2x^2 + 3x - 1) = 2 + 3 + 1$

Condizione di vittoria come i mean-payoff: $\limsup_k \frac{1}{n} \sum_{i \leq k} w_i(x) \gtreqless_{\text{Lex}} 0$.

Determinatezza posizionale

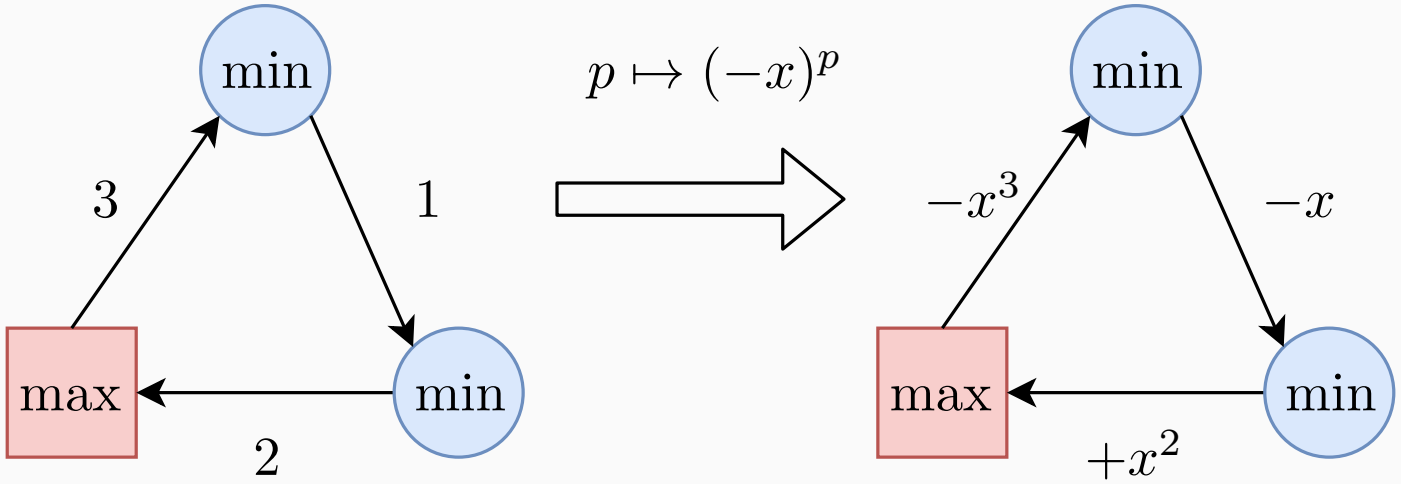
Gli stacked unary mean-payoff games sono posizionalmente determinati.

Complessità intermedia

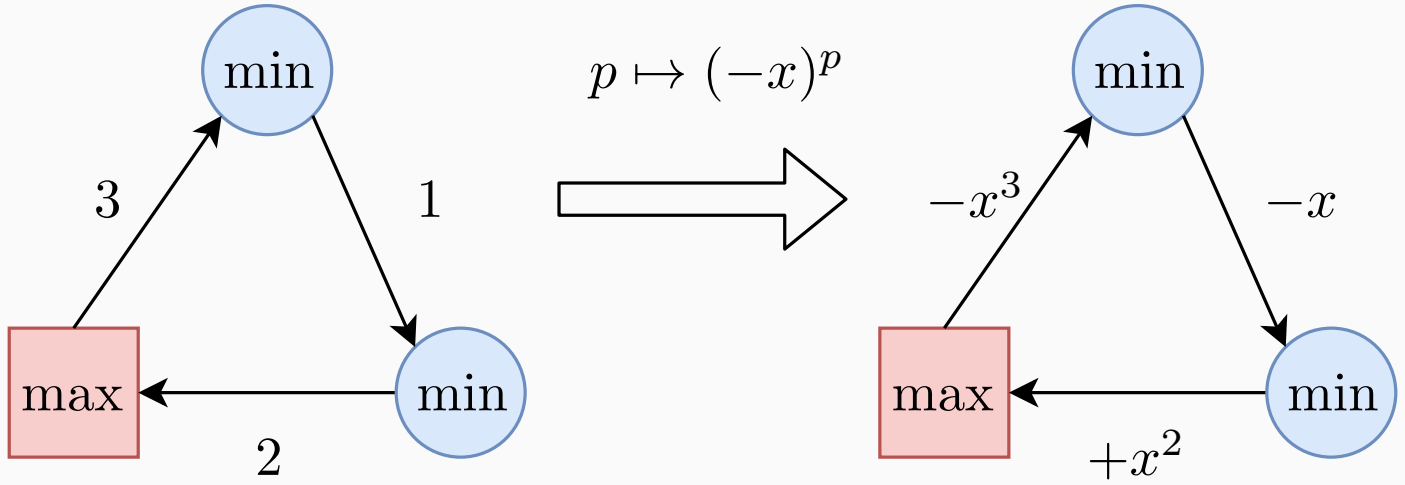
Gli stacked unary mean-payoff games sono di complessità intermedia tra i parity games ed i mean-payoff games.

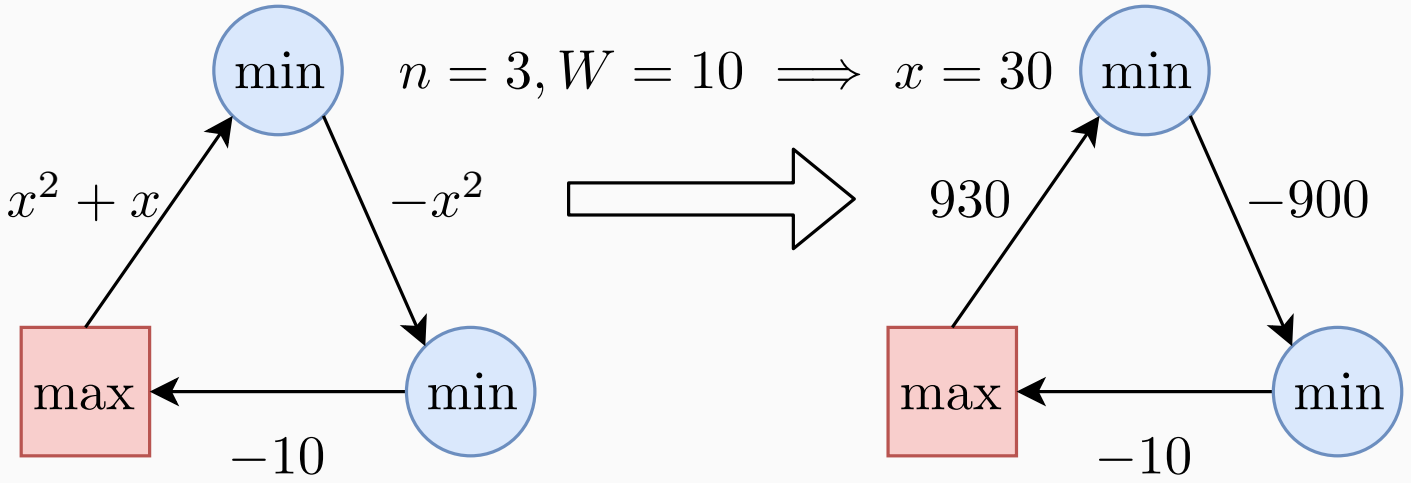
Determinatezza posizionale $\Longrightarrow$ basta la riduzione sul valore dei cicli.

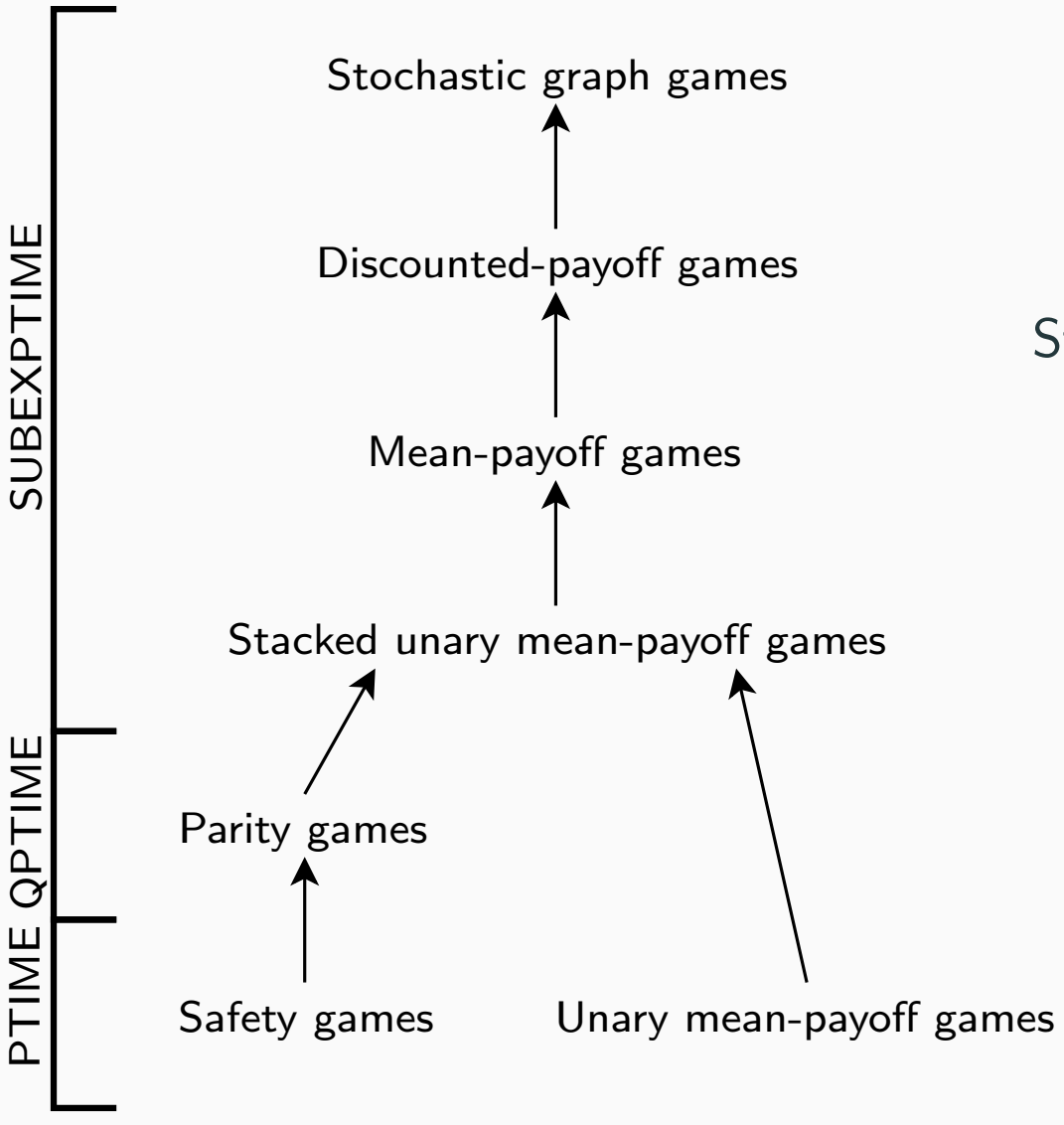
Determinatezza posizionale $\Longrightarrow$ basta la riduzione sul valore dei cicli.



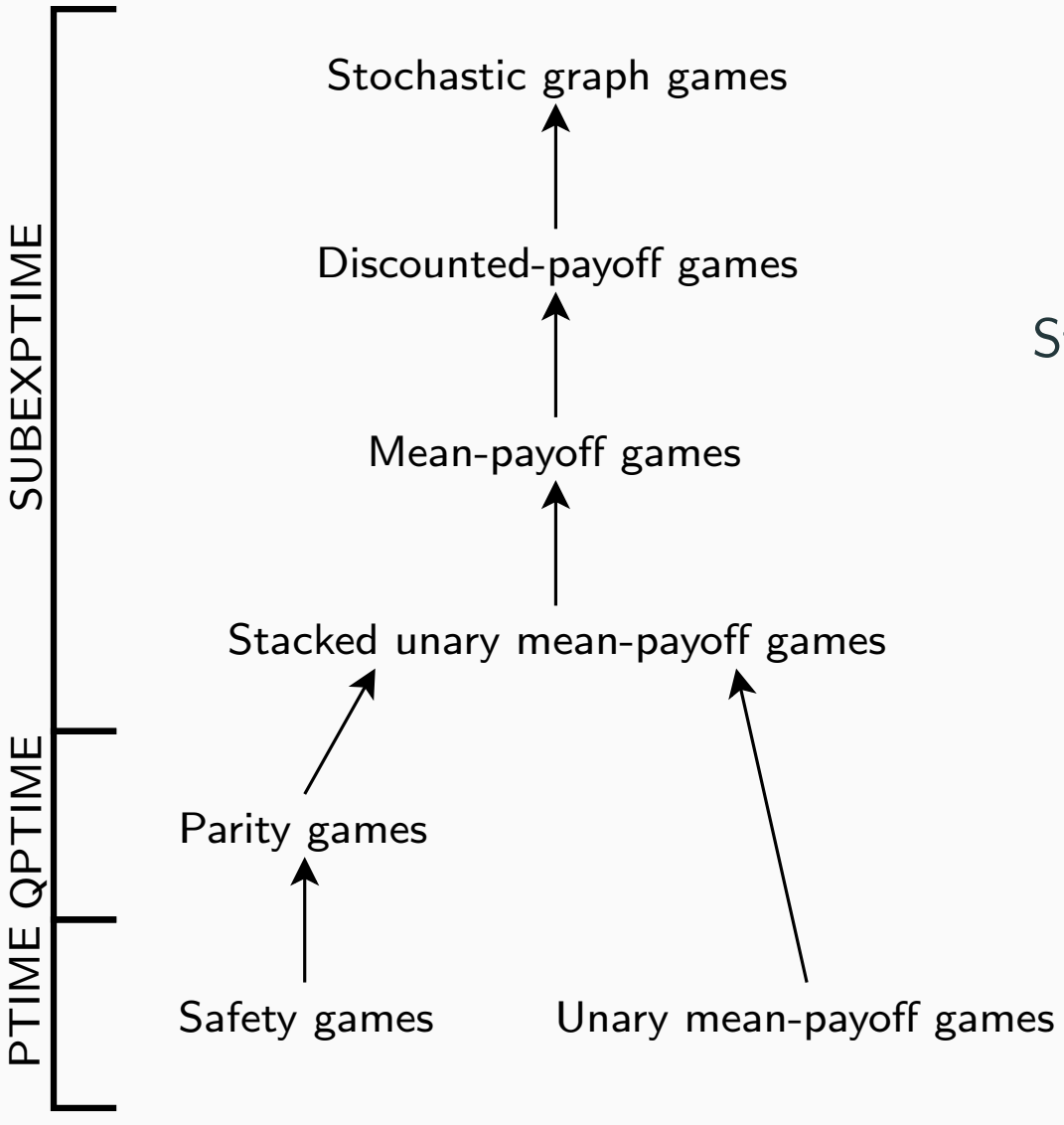
Determinatezza posizionale $\implies$ basta la riduzione sul valore dei cicli.





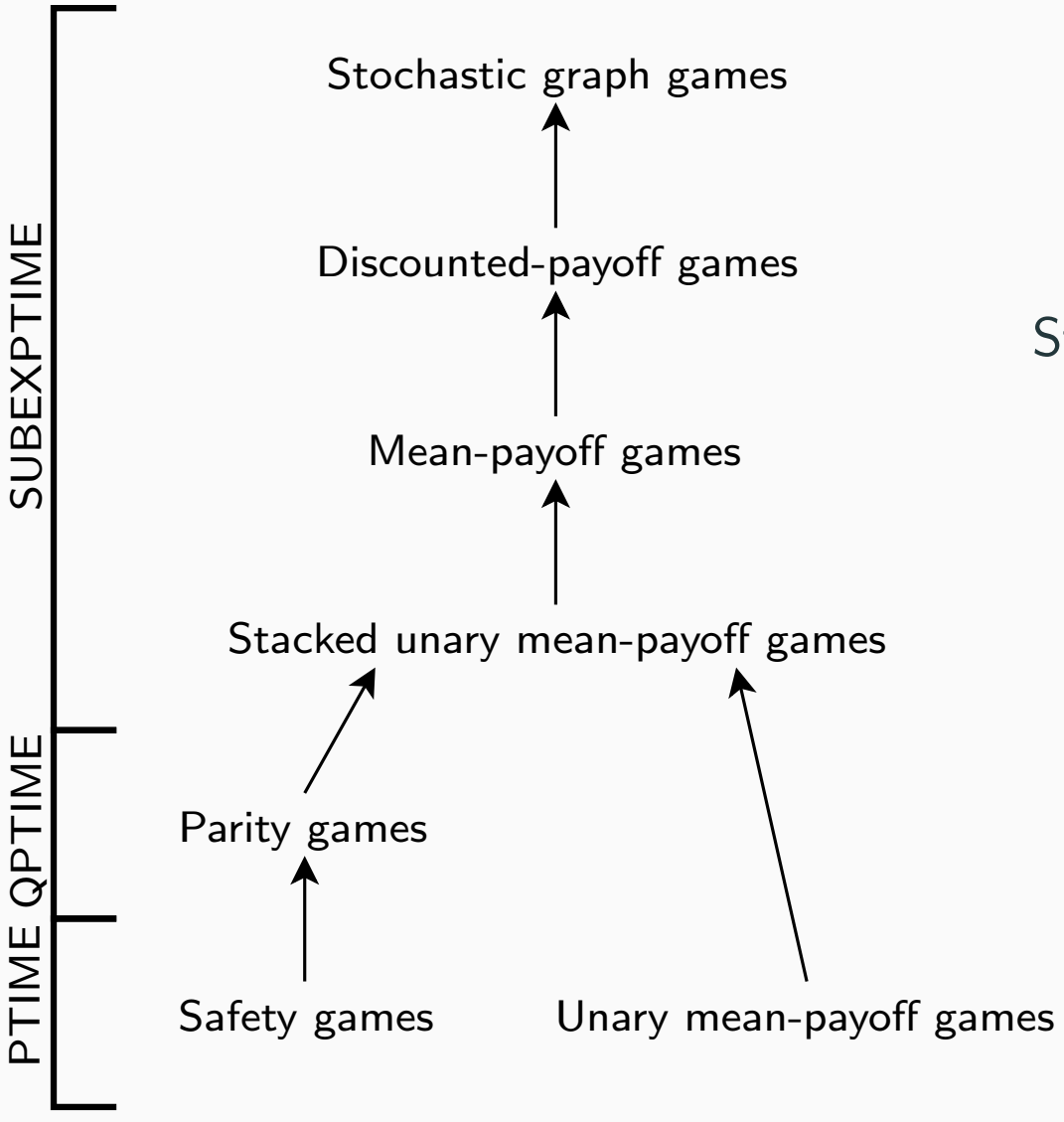


Stacked unary mean-payoff:



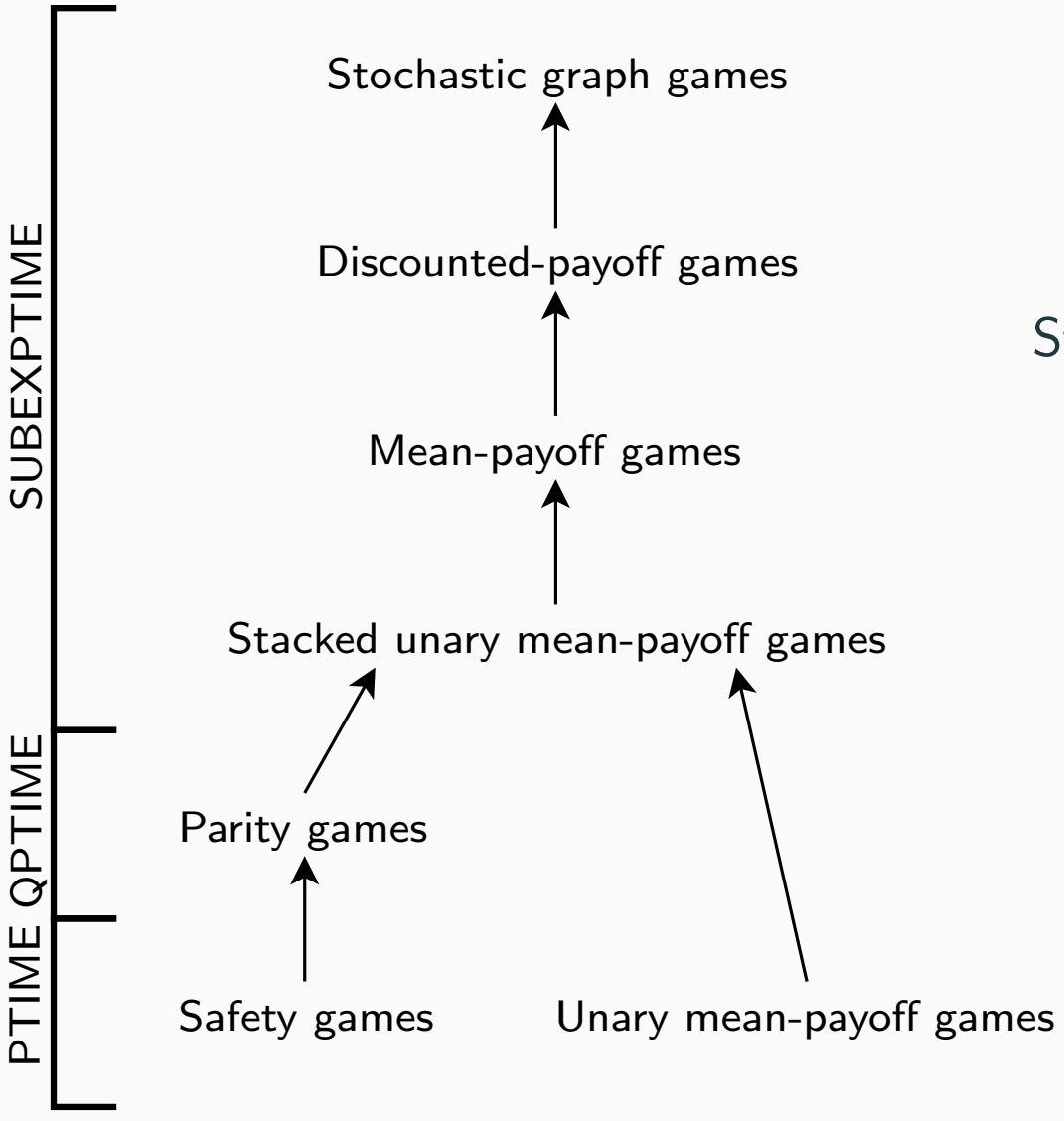
Stacked unary mean-payoff:

- Step necessario per arrivare a risolvere in modo efficiente i mean-payoff.



Stacked unary mean-payoff:

- Step necessario per arrivare a risolvere in modo efficiente i mean-payoff.
- Risolto in tempo esponenziale con variante progress measures.



Stacked unary mean-payoff:

- Step necessario per arrivare a risolvere in modo efficiente i mean-payoff.
- Risolto in tempo esponenziale con variante progress measures.
- Entrambi gli step non banali.

Grazie per l'attenzione