UNIVERSITÀ
DI TORINO

INTELLIGENZA ARTIFICIALE & LABORATORIO

Progetto CLINGO e PROLOG - Parte n.1 - Prof. Pozzato

Data: 15 Settembre 2023

Delmastro Andrea (912954) - Ferrero Fabio (926392) - Frumento Giulia (834773)

Università degli Studi di Torino

INDICE

PROLOG

INTRODUZIONE PROLOG: LA **DAMA**

In questo progetto viene utilizzato il linguaggio PROLOG al fine di modellare il comportamento di un sistema intelligente che sia in grado di giocare a **dama**. In particolare, il sistema deve essere in grado di restituire la **mossa migliore** per il **giocatore bianco** data la conformazione attuale della damiera.

Come vedremo in seguito, per simulare il gioco è stato implementato l'**algoritmo minimax con potatura alpha-beta** a profondità limitata.

MODELLAZIONE DEL DOMINIO: **DOMINO** E **STATO INIZIALE**

Il dominio è composto da una damiera, due giocatori (uno **bianco** e uno **nero**) e due tipi di pedine per ciascun giocatore (**pedina semplice** e **dama**).

Per rappresentare queste informazioni e lo stato iniziale si usano i seguenti predicati:

- `num_righe/1` e `num_colonne/1`: i due predicati rappresentano il numero di righe e di colonne della damiera;
- `occupata/2`: gli argomenti rappresentano la posizione e il tipo di pedina;
- `giocatore_pedina/3`: definisce per ogni giocatore il tipo di pedine disponibili e il loro peso;
- `giocatori_ordine/2`: permette di rappresentare l'ordine dei giocatori;

MODELLAZIONE DEL DOMINIO: **DOMINO** E **STATO INIZIALE**

- `max_muovere/1` e `min_muovere/1`: predicati che indicano i ruoli dei giocatori nell'algoritmo min-max;
- `peso_casella/3`: ogni casella nera della damiera ha un peso numerico che varia in base al tipo di pedina.
- `valore_damiera/1`: predicato che calcola il valore della damiera in base alle pedine presenti e la loro posizione;
- `lista_mosse/2`: restituisce la lista di mosse possibili per il giocatore del turno;
- `fine_partita/1`: predicato che indica la fine della partita quando un giocatore non ha più pedine o mosse da fare.

MODELLAZIONE DELLE AZIONI: I **MOVIMENTI** DELLE PEDINE

Le pedine si muovono in diagonale di una casella scura alla volta, quelle semplici possono andare *solo avanti* mentre le dame possono muoversi in *ogni direzione*.

Ogni pedina può mangiare quelle avversarie che si trovano in avanti, sulla casella diagonale accanto e che abbiano la casella successiva libera.

Una volta che una pedina semplice raggiunge il lato opposto della damiera viene *promossa* a dama.

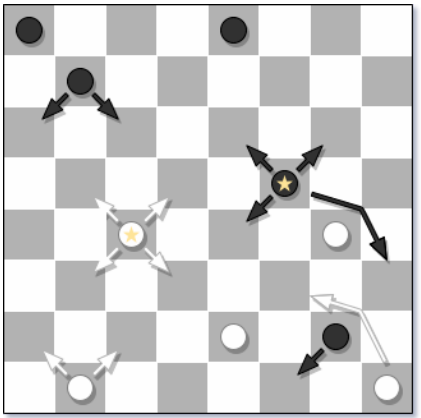

Figure: Mosse concesse

MODELLAZIONE DELLE AZIONI: I **MOVIMENTI** DELLE PEDINE

I predicati che modellano le **azioni possibili** per le pedine sono:

- `applicabile/2`: verifica l'applicabilità di una mossa per una data pedina, le mosse possibili sono spostarsi in diagonale o essere promossa a dama;
- `applicabile_mangia/2`: verifica l'applicabilità di una mossa con mangiata;
- `trasforma/3`: modifica i fatti per applicare la mossa sulla pedina scelta;
- `fai_azione/5` e `annulla_azione/5`: il primo predicato fa da wrapper al predicato `trasforma/3` e permette di ricordare i dati che vengono retratti per aggiornare lo stato della damiera. Il secondo predicato permette di ritornare allo stato originale della damiera annullando la mossa;
- `nemico/2`: verifica se la pedina accanto a quella da muovere è dell'avversario.

ALGORITMO MINIMAX CON POTATURA ALPHA-BETA

L'algoritmo minimax è un **algoritmo ricorsivo** che ricerca la mossa migliore in una data situazione. Per misurare la bontà di una mossa viene usata una **funzione di valutazione** che, nel nostro caso, si basa sulla somma dei pesi delle pedine moltiplicati per i pesi delle posizioni che occupano.

La complessità temporale per minimax è $O(b^m)$ mentre la complessità temporale per alpha-beta è $O(b^{m/2})$

ALGORITMO MINIMAX CON **POTATURA ALPHA-BETA**

La **potatura** permette ridurre il numero di nodi da visitare nell'albero. L'esplorazione si porta dietro gli estremi di un intervallo:

- `alpha`: *massimo lower bound*, ovvero il valore della scelta migliore per MAX;
- `beta`: *minimo upper bound*, ovvero il valore della scelta migliore per MIN.

Un nodo viene visitato solo se il suo valore stimato è compreso fra `alpha` e `beta`.

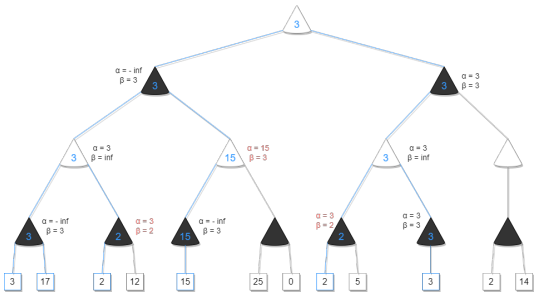

Figure: Algoritmo minimax con potatura

EFFICIENTAMENTO DELL'ALGORITMO TRAMITE **ORDINAMENTO DELLE AZIONI**

Considerare le azioni ordinate per efficienza all'atto della costruzione dell'albero (procedendo prima a espandere le azioni per *mangiare* e poi quelle per *muoversi*), sembrerebbe incrementare la quantità di potatura profonda dell'albero. Nonostante questo, i risultati mostrano come non esista una strategia generalmente migliore:

Profondità	**Ordine 1** (s)	**Ordine 2** (s)
8	4.2	2.5
9	10.8	5.9
10	6.7	8.15
11	66.3	30.1

Dove **Ordine 1** identifica l'ordine in cui vengono considerate prima le azioni di *movimento* e poi quelle per *mangiare*. **Ordine 2** è l'ordine opposto.

TEST MAPPA 1

Depth = 3 -> Mossa = pos(4, 3), mangiasusx
Tempo esecuzione = 21 ms

Depth = 4 -> Mossa = pos(4, 1), mangiasudx
Tempo esecusione = 41 ms

Depth = 6 -> Mossa = pos(4, 3), mangiasusx
Tempo esecuzione = 108 ms

Depth = 8 -> Mossa = pos(4, 3), sudx
Tempo esecuzione = 575 ms

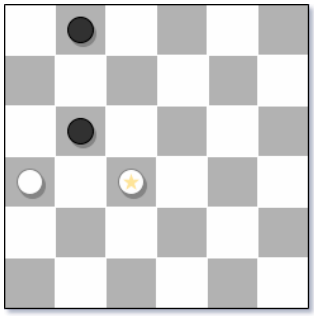

Figure: Conformazione damiera

TEST MAPPA 2

Depth = 3 -> Mossa = pos(1, 6), mangiagiusx
Tempo esecuzione = 16 ms

Depth = 4 -> Mossa = pos(1, 6), mangiagiusx
Tempo esecusione = 50 ms

Depth = 6 -> Mossa = pos(1, 6), mangiagiusx
Tempo esecuzione = 373 ms

Depth = 8 -> Mossa = pos(1, 6), mangiagiusx
Tempo esecuzione = 5251 ms

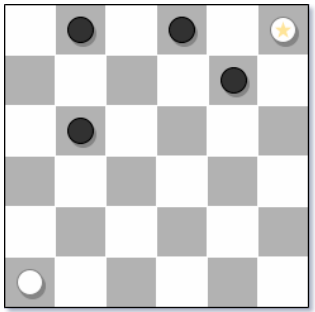

Figure: Conformazione damiera

TEST MAPPA 3

Depth = 3 -> Mossa = pos(1, 6), mangiagiusx
Tempo esecuzione = 40 ms

Depth = 6 -> Mossa = pos(1, 6), mangiagiusx
Tempo esecuzione = 746 ms

Depth = 7 -> Mossa = pos(1, 6), mangiagiusx
Tempo esecuzione = 3116 ms

Depth = 8 -> Mossa = pos(1, 6), mangiagiusx
Tempo esecuzione = 6740 ms

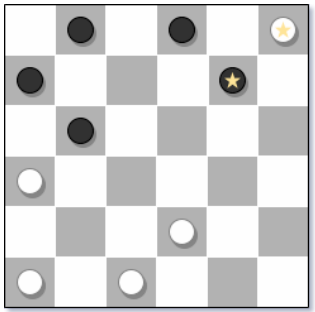

Figure: Conformazione damiera

TEST MAPPA 4

Depth = 3 -> Mossa = pos(6, 1), sudx
Tempo esecuzione = 56 ms

Depth = 6 -> Mossa = pos(6, 3), susx
Tempo esecuzione = 309 ms

Depth = 7 -> Mossa = pos(6, 3), susx
Tempo esecuzione = 651 ms

Depth = 8 -> Mossa = pos(5, 4), susx
Tempo esecuzione = 1564 ms

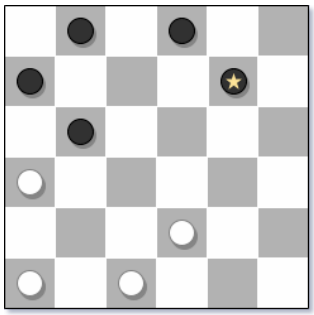

Figure: Conformazione damiera

CLINGO

INTODUZIONE CLINGO

In questo progetto viene utilizzato l'answer set solver CLINGO al fine di modellare un semplice problema di soddisfacimento di vincoli relativo alla generazione di un torneo.

MODELLAZIONE DEL DOMINIO: **DOMINIO** E **COSTANTI**

Le proprietà *statiche* del dominio sono state modellate attraverso opportune costanti:

```
1 #const teamsNumber    = 20.
2 #const daysNumber     = (teamsNumber * 2) - 2.
3 #const legDaysNumber  = daysNumber / 2.
4 #const rounds         = 2.
5 #const roundsInterval = 10.
```

Dove `legDaysNumber` si riferisce al numero di giornate di un girone, `roundsInterval` si riferisce al numero di giornate minimo che deve passare tra l'andata e il ritorno di una partita.

MODELLAZIONE DEL DOMINIO: **DOMINIO** E **COSTANTI**

Le squadre sono modellate attraverso un opportuno predicato `team`:

```
team(juventusFc; lincolnRedImpsFc; asRoma; ssLazio; ...)
```

Le città di residenza delle squadre sono modellate attraverso un opportuno predicato `city`:

```
city(turin; gibraltar; rome; suhareka; ...)
```

Ad ogni squadra è associata una città e uno stadio di casa:

```
homeGround(juventusFc, "Allianz Stadium", turin; ...)
```

MODELLAZIONE DEL DOMINIO: **DOMINIO** E **COSTANTI**

Il predicato `combination` rappresenta tutte le coppie ordinate di squadre diverse:

```
1 combination(HomeTeam, AwayTeam)
2   :- team(HomeTeam), team(AwayTeam), HomeTeam <> AwayTeam.
```

Lo stesso risultato si può ottenere in modo alternativo tramite l'utilizzo di aggregati:

```
1 (teamsNumber - 1)
2   { combination(HomeTeam, AwayTeam) : team(AwayTeam),
      AwayTeam <> HomeTeam }
3 (teamsNumber - 1)
4   :- team(HomeTeam).
```

A rappresentare che ogni squadra gioca in casa esattamente `teamsNumber - 1` partite contro altre squadre.

MODELLAZIONE DEI VINCOLI: UTILIZZO DEGLI **AGGREGATI**

Si è fatto uso estensivo degli aggregati per modellare i vincoli riguardanti l'assegnamento di una giornata ad una partita, il numero di partite per giornata e il numero di partite giocate ogni giornata da una squadra. Il più complesso è il seguente:

```
1 1
2   { match(Number, combination(Team, AwayTeam)) :
       combination(Team, AwayTeam);
3     match(Number, combination(HomeTeam, Team)) :
         combination(HomeTeam, Team) }
4 1
5   :- day(Number), team(Team).
```

A rappresentare che per ogni giornata e per ogni squadra viene giocata esattamente una partita in casa o una partita in trasferta.

MODELLAZIONE DEI VINCOLI: UTILIZZO DEGLI **INTEGRITY CONSTRAINT**

I restanti vincoli sono stati introdotti sotto forma di integrity constraint, ad esempio:

```
1 :- homeGround(Team1, Ground, City),
2    homeGround(Team2, Ground, City),
3    Team1 <> Team2,
4    match(Number, combination(Team1, _)),
5    match(Number, combination(Team2, _)).
```

A rappresentare che due squadre diverse che giocano nello stesso stadio nella stessa città non possono mai giocare in casa nella stessa giornata.

MODELLAZIONE DEI VINCOLI: VINCOLI **OPZIONALI**

Sono stati introdotti tutti i vincoli opzionali sotto forma di integrity constraint, ad esempio:

```
1 :- match(Number1, combination(Team1, Team2)),
2    match(Number2, combination(Team2, Team1)),
3    |Number1 - Number2| < roundsInterval.
```

A rappresentare che tra l'andata e il ritorno di una partita devono passare almeno `roundsInterval` giornate.

ESEMPIO RISULTATO: **GIRONE DI ANDATA**

Per questa esecuzione sono state considerate 4 squadre di calcio e 2 gironi, ciascuno composto da 3 giornate. Il tempo di esecuzione per trovare la risposta è di **0.016s**.

Table: Girone di andata

Giornata	Squadra Casa	Squadra ospite	Stadio
1	Juventus F.C.	A.C.F Fiorentina	Allianz Stadium, TO
1	S.S. Lazio	A.S. Roma	Stadio Olimpico, Roma
2	A.S. Roma	Juventus F.C.	Stadio Olimpico, Roma
2	A.C.F Fiorentina	S.S. Lazio	Artemio Franchi, FI
3	Juventus F.C	S.S. Lazio	Allianz Stadium, TO
3	A.S. Roma	A.C.F Fiorentina	Stadio Olimpico, Roma

ESEMPIO RISULTATO: **GIRONE DI RITORNO**

Per questa esecuzione sono state considerate 4 squadre di calcio e 2 gironi, ciascuno composto da 3 giornate. Il tempo di esecuzione per trovare la risposta è di **0.016s**.

Table: Girone di ritorno

Giornata	Squadra Casa	Squadra ospite	Stadio
4	A.C.F Fiorentina	A.S. Roma	Artemio Franchi, FI
4	S.S. Lazio	Juventus F.C.	Stadio Olimpico, Roma
5	S.S. Lazio	A.C.F Fiorentina	Stadio Olimpico, Roma
5	Juventus F.C.	A.S. Roma	Allianz Stadium, TO
6	A.C.F Fiorentina	Juventus F.C.	Artemio Franchi, FI
6	A.S. Roma	S.S. Lazio	Stadio Olimpico, Roma