Università degli studi di Torino

Dipartimento di Informatica

Intelligenza artificiale e laboratorio

Answer Set Programming

Giacomo Bonazzi - Marco Caldera - Giacomo Costarelli

1 dicembre 2019

Indice

1 Introduzione

Questa relazione ha lo scopo di presentare i punti principali affrontati durante lo sviluppo del progetto e di spiegare lo studio preliminare necessario per il corretto svolgimento dell'esercizio proposto durante l'insegnamento di Intelligenza Artificiale e Laboratorio A.A. 2018/2019.

Durante le lezioni ci sono stati presentati due esercizi. Tra di essi la nostra scelta è ricaduta sul primo dei due esercizi: la modellazione e la creazione del calendario delle partite della Champions League 2018/2019. Di seguito riportiamo i requisiti che ci sono stati proposti:

- Vi sono 32 squadre, provenienti da città di diverse nazioni europee.
- Le squadre vengono organizzate in 8 gironi (gruppo A, gruppo B, ..., gruppo H) da 4 squadre ciascuno.
- Due squadre della stessa nazione non possono essere collocate nello stesso gruppo.
- In ogni gruppo, ciascuna squadra deve affrontare due volte tutte le altre tre squadre del gruppo, una volta in casa e una volta in trasferta.
- Gli incontri vengono raggruppati in "giornate", durante le quali tutte le 32 squadre affrontano un'avversaria. Pertanto, il calendario si comporrà di 6 giornate, tre per le gare di andata e tre per le gare di ritorno.
- Due squadre della stessa città non possono giocare entrambe in casa durante la medesima giornata.
- Una squadra non può giocare più di due partite consecutive in casa o più di due partite consecutive in trasferta.

2 Progetto

Il progetto è sviluppato in ASP ed è stato completato attraversando quattro principali fasi: siamo partiti dallo studio della teoria del linguaggio logico stesso e dell'utilizzo di clingo, per passare poi alla fase di stesura dei dati del problema, allo sviluppo ed infine all'analisi dei risultati ottenuti.

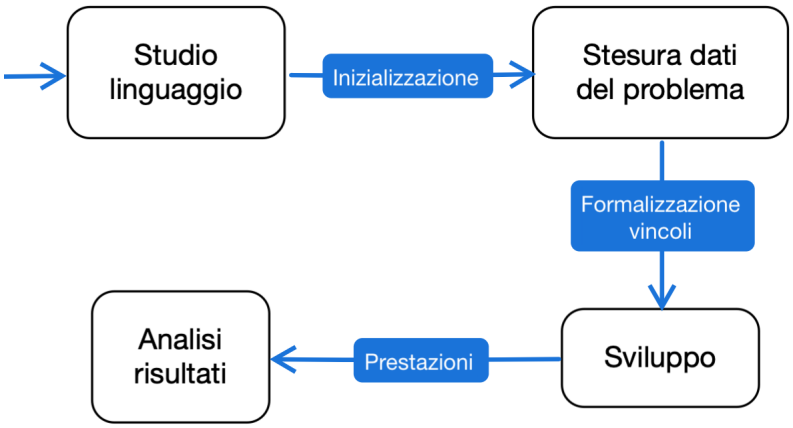

Figura 1: Outline del progetto

La Figura 1 illustra le quattro fasi appena citate. Ognuna di esse viene dettagliata nella relativa sezione ponendo particolare attenzione alle caratteristiche principali che ci hanno permesso di completare correttamente il progetto.

2.1 Answer Set Programming

Nella prima fase del progetto abbiamo studiato ASP [1] ed il suo funzionamento, in modo da chiarirci le idee e strutturare in maniera corretta lo sviluppo del nostro calendario per la Champions League.

L'answer set programming (ASP) è una metodologia di programmazione nata nel momento storico in cui si cercava di trovare una semantica per esprimere la negazione per fallimento (usata dagli interpreti Prolog). In seguito è diventata qualcosa di più, ovvero il fondamento di una nuova tecnica di programmazione. L'ASP, dal punto di vista sintattico è simile al Prolog, ma la natura delle soluzioni è diversa, infatti, in Prolog diamo un goal e cerchiamo una derivazione del goal a partire dalla base di conoscenza; invece, con ASP la soluzione non è la prova del goal, ma sono dei modelli detti answer set.

2.2 Modellazione del dominio

Una volta approfonditi gli aspetti tecnici legati al linguaggio abbiamo iniziato a considerare gli elementi necessari all'effettivo sviluppo del progetto.

Come prima cosa abbiamo l'esigenza di rappresentare il dominio. Abbiamo così preso le informazioni sulle squadre partecipanti alla Champions League 18/19 focalizzandoci, per ogni squadra, sulle seguenti informazioni:

- Nome
- Nazione
- Città

Analizzate le informazioni abbiamo così deciso, come mostrato nel Codice 1, di rappresentare esplicitamente, al fine di agevolare lo sviluppo, le correlazioni tra le tre tipologie di informazioni su ogni società. Queste correlazioni sono mostrate all'interno del Codice 1 e richiedono l'utilizzo di una tipologia di fatto chiamata *snc* (i.e., Squadra Nazione Città).

```
snc (
        club_brugge, belgio, bruges;
        monaco, francia, monaco;
        paris_saint_germain, francia, parigi;
        olympic_lione, francia, lione;
        borussia_dortmund, germania, dortmund;
        schalke_04, germania, gelsenkirchen ;
        bayern_monaco, germania, monaco_di_baviera;
        ...
).
```

Codice 1: Squadra-Nazione-Città

Per concludere correttamente la modellazione del dominio abbiamo dovuto ancora definire gli 8 possibili giorni in cui andare a distribuire le squadre partecipanti al torneo. Gli 8 giorni sono identificati dalle lettere maiscuole che vanno dalla A alla H e sono state rappresentate come mostrato nel Codice 2.

```
girone(
        gruppo_A; gruppo_B; gruppo_C; gruppo_D;
        gruppo_E; gruppo_F; gruppo_G; gruppo_H;
).
```

Codice 2: Gruppi

2.3 Sviluppo

Terminata la fase di modellazione del dominio abbiamo iniziato a considerare gli strumenti necessari ad implementare i vincoli da soddisfare per creare correttamente i gironi di champions.

Abbiamo eseguito una fase di *scratching* in cui il gruppo ha proposto possibili soluzioni ed idee iniziali così da poter iniziare a generare il calendario soddisfacendo i diversi vincoli e portanto così a termine correttamente l'esercizio.

Al fine di migliorare la chiarezza dell'elaborato abbiamo suddiviso concettualmente il progetto in 3 parti:

- Creazione dei gironi di champions
- Generazione partite
- Raffinamento vincoli

2.3.1 Creazione gironi di Champions

Inizialmente ci siamo occupati di gestire i vincoli relativi alla creazione degli 8 differenti gironi.

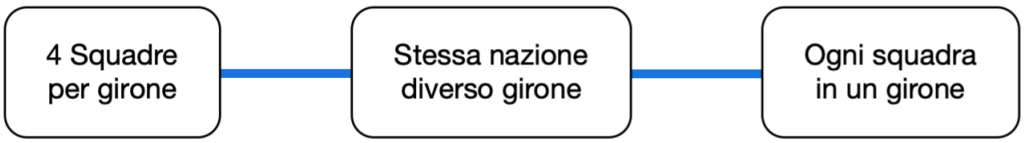

Figura 2: Vincoli gestiti in questa sezione

Analizzando in modo più approfondito i requisiti di progetto abbiamo individuato che il punto chiave per la risoluzione dei vincoli presenti in Figura 2 fosse la combinazione dei diversi gironi con la tripla *snc*. Per fare ciò abbiamo utilizzato i costrutti presenti all'interno del Codice 3.

```
%Ad ogni girone si assegnano esattamente 4 squadre.
4 {girone_contiene_squadra(G, S, N, C) : snc(S, N, C)} 4 :- girone(G).

%Ad ogni squadra si assegna uno ed uno solo girone.
1 {girone_contiene_squadra(G, S, N, C) : girone(G)} 1 :- snc(S, N, C).

%Due squadre della stessa nazione non possono essere nello stesso girone.
:-  girone_contiene_squadra(G, S1, N1, _),
    girone_contiene_squadra(G, S2, N2, _),
    S1 != S2,
    N1 == N2.
```

Codice 3: Generazione gironi

Il Codice 3 è costituito da due aggregati ed un integrity constraint[1]:

- **Riga 2**. L'aggregato presente a riga 2 serve per assegnare ad ogni girone (*girone(G)*) esattamente quattro squadre (i.e., esattamente quattro occorrenze di *snc*).
- **Riga 5**. Questo aggregato si occupa invece di garantire che ogni squadra (*snc(S,N,C)*) venga assegnata ad un unico girone.
- **Riga 8**. Il costrutto di riga 8 è invece un integrity constraint ed è utilizzato per escludere dal risultato tutti gli answer set conteneti gironi con squadre della stessa nazione (*N1==N2*).

Gli assegnamenti creati da queste regole sono rappresentati attraverso la tipologia di fatto chiamata *girone_contiene_squadra* che, al termine di questa fase, sarà formata da 32 fatti, uno per ogni squadra.

2.3.2 Generazione partite

Una volta determinati i gironi ci siamo occupati della creazione delle 6 giornate della prima fase della Champions League. Per generare queste giornate abbiamo prima modellato i vincoli generali (Figura 3) che ci hanno permesso di suddividere correttamente le partite e solo successivamente, nella Sezione 2.3.3, parleremo di come abbiamo concluso il progetto raffinando ulteriormente la soluzione al fine di rispettare anche gli ultimi vincoli del problema.

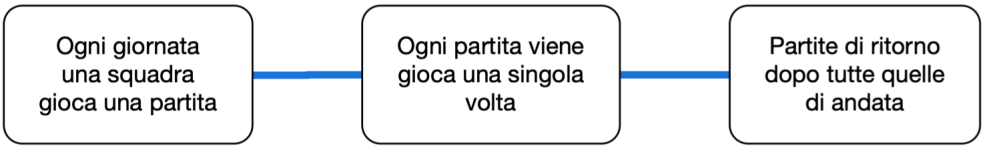

Figura 3: Vincoli per la generazione delle partite

Dopo aver analizzato questa parte del problema abbiamo valutato che, al fine di rispettare tutti i vincoli, ci sarebbe bastato modellare correttamente le tre partite di andata per poi, come vedremo, creare le partite di ritorno in maniera molto intuitiva.

[1]Una regola in cui non è presente la testa si chiama *integrity constraint*, ossia una regola che permette di specificare situazioni che non si possono verificare quindi al posto di $A :- AB$, é possibile scrivere : $-AB$ nel caso in cui AB non sia vero.

```
1  giornata(1..3).
2  2 {partita_andata(Girone, Squadra1,Squadra2, Gio): girone_contiene_squadra
       (Girone, Squadra1, _, _), girone_contiene_squadra(Girone, Squadra2, _,
        _ ), Squadra1 != Squadra2} 2 :- girone(Girone), giornata(Gio).
3
4  %due partite in casa nella stessa giornata
5  :-  partita_andata(Girone, Squadra1,Squadra2, N),  partita_andata(Girone,
       Squadra1,Squadra4, N), Squadra2 != Squadra4.
6  %due partite nella stessa giornata, una in casa una in trasferta
7  :-  partita_andata(Girone, Squadra1,Squadra2, N),  partita_andata(Girone,
       Squadra4,Squadra1, N), Squadra2 != Squadra4.
8  %due partite nella stessa giornata, una in trasferta ed una in casa
9  :-  partita_andata(Girone, Squadra1,Squadra2, N),  partita_andata(Girone,
       Squadra2,Squadra4, N), Squadra1 != Squadra4.
10 %due partite in trasferta nella stessa giornata
11 :-  partita_andata(Girone, Squadra1,Squadra2, N),  partita_andata(Girone,
       Squadra4,Squadra2, N), Squadra1 != Squadra4.
12
13 % Scartare tutti gli answer set in cui la stessa partita viene disputata
       in piu giornate.
14 :- partita_andata(Girone, Squadra1,Squadra2, N), partita_andata(Girone,
       Squadra1, Squadra2, N1), N != N1.
15
16 % Scartare tutti gli answer set in cui una partita di andata viene
       disputata come una partita di ritorno in una differente giornata d'
       andata.
17 :- partita_andata(Girone, Squadra1, Squadra2, _), partita_andata(Girone,
       Squadra2, Squadra1, _).%andata e ritorno in giornate diverse ma tra le
        partite di andata
```

Codice 4: Generazione gironi

Il Codice 4 mostra la prima parte del ragionamento che abbiamo fatto. In questo codice abbiamo cercato di formalizzare il seguente concetto: ogni girone è composto da quattro squadre, in ogni giornata di andata una delle quattro squadre di un girone affronta un'altra squadra dello stesso girone, formando così due partite. Al termine delle tre partite di andata ogni squadra deve aver affrontato le altre tre del suo girone.

Vediamo ora in dettaglio come abbiamo modellato il concetto all'interno delle righe del Codice 4:

- **Righe 1-2**. In queste righe modelliamo il fatto che per ogni giornata di andata e per ogni girone devono essere disputate due partite.
- **Riga 5-11**: I vincoli presenti in queste righe ci permettono di eliminare tutte quelle coppie di partite in cui una squadra, nella stessa giornata, gioca contro più squadre. Vengono dunque scartati gli answer set in

cui, data una giornata d'andata, una squadra gioca due partite in casa o due in trasferta o una in casa ed una in trasferta.

- **Riga 14**: Scartiamo tutte le soluzioni contenenti la stessa partita disputata più volte, essendo tali soluzioni certamente errate.

- **Riga 17**: Quest'ultimo vincolo del Codice 4 è molto importante. Senza questo vincolo una partita può venire considerata come partita d'andata semplicemente invertendo, rispetto ad un'altra partita d'andata, la posizione delle squadre. Questa situazione risulta però errata poiché staremmo creando una partita di ritorno in mezzo a quelle di andata.

Avendo ora modellato i principali vincoli, che permettono di definire correttamente la creazione delle partite di andata, possiamo andare a definire le partite di ritorno in maniera davvero molto semplice.

```
1 partita_ritorno(Girone, Squadra2,Squadra1, 6-Giornata+1) :-
2     partita_andata(Girone, Squadra1,Squadra2, Giornata).
```

Codice 5: Partite di ritorno

Come si può notare nel Codice 5, abbiamo scelto di definire una *partita_ritorno*[2] utilizzando semplicemente la definizione di *partita_andata* ed invertendo la posizione delle squadre.

La definizione della giornata in cui si deve disputare la gara di ritorno è data dal numero *6-Giornata+1* in modo tale che una data squadra di un girone incontri le altre tre squadre nell'ordine A B C C B A.

Esempio Ad esempio se abbiamo un fatto (squadra1=Juve, squadra2=PSG, giornata=1) la partita di ritorno creata è (squadra1=PSG, squadra2=Juve, giornata=6-1+1) infatti due squadre che si sono incontrate nella prima giornata devono incontrarsi nella sesta.

[2]La notazione posizionale utilizzata, considerata la coppia [Squadra1, Squadra2], vede Squadra1 come squadra di casa e Squadra2 come squadra in trasferta.

2.3.3 Raffinamento

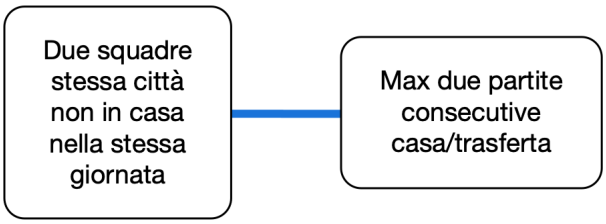

Figura 4: Vincoli di gestione delle partite in casa e numero di partite consecutive in casa/trasferta.

A questo punto sono rimasti solamente alcuni requisiti (Figura 4) ancora da modellare. Essi riguardano:

1. L'impossibilità di due squadre della stessa città di giocare in casa nella stessa giornata.

2. La necessità di non permettere ad una squadra di disputare più di due partite consecutivamente in casa o in trasferta.

```
1 uguale_citta(Squadra1, Squadra2) :-
2     snc(Squadra1, _, C1),
3     snc(Squadra2, _, C2),
4     C1 == C2,
5     Squadra1 != Squadra2.
6
7 % Scartare tutti gli answer set in cui due squadre della stessa citta
      giocano in casa nella stessa giornata
8 :- partita_andata( _, Squadra1, _, N), partita_andata( _, Squadra2, _, N),
      uguale_citta(Squadra1, Squadra2).
```

Codice 6: Codice sul vincolo per evitare che due squadre della stessa città giochino in casa in una stessa giornata

Il Codice 6 risolve il primo dei due vincoli rimasti. Per eliminare gli answer set contenenti giornate in cui due squadre della stessa città giocano entrambe in casa abbiamo definito una regola ed un integrity constraint:

- **Righe 2-6**. Abbiamo una regola che confronta due squadre per stabilire se sono della stessa città, se lo sono viene creato un fatto *uguale_ citta* tra di esse.

- **Riga 9**. Abbiamo definito un integrity constraint che si assicura che due squadre della stessa città non giochino in casa nella stessa giornata.

```
1  % Una Squadra gioca in casa tre volte consecutivamente se appare come
       prima squadra all'interno della partita.
2  tre_consecutive_casa(Squadra) :-
3      partita_andata(_, Squadra, _, 1),
4      partita_andata(_, Squadra, _, 2),
5      partita_andata(_, Squadra, _, 3).
6
7  % Una Squadra gioca in trasferta tre volte consecutivamente se appare tre
       volte come seconda squadra all'interno delle tre partite di partita.
8  tre_consecutive_trasferta(Squadra) :-
9      partita_andata(_, _, Squadra, 1),
10     partita_andata(_, _, Squadra, 2),
11     partita_andata(_, _, Squadra, 3).
12
13 :- tre_consecutive_casa(Squadra).
14 :- tre_consecutive_trasferta(Squadra).
```

Codice 7: Codice sul vincolo per evitare che vengano giocate consecutivamente 3 partite consecutive in casa o trasferta

Per quanto riguarda il secondo è ultimo vincolo abbiamo deciso di definire due regole e due integrity constraint (Codice 7). La formalizzazione del requisito poteva essere sintatticamente meno verbosa ed utilizzare solamente due integrity constraint ma, nell'ottica di sperimentare e semplificare la lettura, abbiamo deciso di separare il vincolo su più regole. In particolare la modellazione avviene in questo modo:

- **Righe 2-5** : Permette di creare un fatto *tre_consecutive_casa* relativo ad una squadra che gioca le tre partite di andata tutte in casa.
- **Righe 8-11** : Permette di creare un fatto *tre_consecutive_trasferta* relativo ad una squadra che gioca le tre partite di andata tutte in trasferta.
- **Righe 13-14** : Questi due vincoli ci permettono infine di assicurarci che nel nostro answer set restituito non siano presenti squadre che giocano tre partite consecutive in casa (riga 13) o in trasferta (riga 14)

È infine importante notare come, se tali vincoli sono rispettati per le partite di andata, allora vista la definizione di partita di ritorno lo sono anche per esse.

Grazie a quest'ultima porzione di codice l'esercizio si può considerare concluso in quanto tutti i requisiti iniziali risultano essere soddisfatti.

3 Valutazione risultati

Durante lo sviluppo ci siamo posti il problema di trovare un modo per testare i risultati restituiti da clingo in maniera semplice e veloce. L'output fornito da clingo non è facilmente leggibile e pertanto abbiamo abbiamo deciso di procedere in questo modo:

1. Trasformare l'output di clingo in un file .json strutturato.
2. Importare il file .json in una web application e mostrare graficamente i risultati della computazione.

```
[
  {
    "name": "girone_contiene_squadra",
    "args": [
      {
        "name": "gruppo_B"
      },
      {
        "name": "club_brugge"
      },
      {
        "name": "belgio"
      },
      {
        "name": "bruges"
      }
    ]
  },
  {...}
]
```

Codice 8: Porzione del file json generato

Per generale un file json strutturato (Codice 8) abbiamo utilizzando uno script python in grado di trasformare i fatti restituiti da clingo in un unico file json formato da due parametri:

- **name**. Questo parametro contiene il nome del fatto fornito in output da clingo
- **args**. Questo parametro contiene tutti gli argomenti del fatto che stiamo trattando in un dato momento.

Grazie a questo file .json abbiamo creato un'applicazione Angular[3] con cui abbiamo interpretato il file json e creato una pagina html contenente i risultati restituiti dal programma clingo.

Figura 5: Visualizzazione dei gironi di Champions League.

La Figura 5 mostra graficamente i gironi ottenuti dal programma clingo e permette in maniera davvero intuitiva di verificare che i relativi vincoli siano rispettati (e.g., non devono essere presenti più squadre italiane nello stesso girone).

[3]L'applicazione è visibile, grazie all'uso di Firebase all'indirizzo https://champions-1819.web.app (n.b., l'app è ottimizzata solamente per dispositivi desktop.

Figura 6: Visualizzazione della prima giornata Champions League.

La Figura 6 mostra invece una parte della prima giornata[4] e anche qui, grazie a questa visualizzazione, risulta molto più semplice capire se tutti i vincoli sono stati implementati correttamente (e.g., nella stessa giornata tutte le squadre giocano una partita).

4 Analisi risultati

Dopo esserci accertati della correttezza delle soluzioni trovate da clingo siamo passati alla fase di analisi dei tempi di esecuzione. Per l'analisi dei tempi abbiamo considerato insiemi di dieci esecuzioni, in cui i tempi sono risultati più che soddisfacenti.

[4]Per una visualizzazione chiara del risultato è consigliato aprire da computer desktop l'url dell'applicazione.

Esecuzione	Models	Time
1	1+	**6.234s**
2	1+	**6.421s**
3	1+	**6.278s**
4	1+	**6.484s**
5	1+	**6.421s**
6	1+	**6.250s**
7	1+	**6.328s**
8	1+	**6.359s**
9	1+	**6.390s**
10	1+	**6.234s**

Tabella 1: Risultati di 10 esecuzioni

Nella Tabella 1 riportiamo i risultati ottenuti su 10 esecuzioni e come si può notare il tempo medio per il soddisfacimento dei vincoli di una esecuzione risulta essere poco superiore ai 6 secondi.

I fattori che hanno portato a questi risultati sono sicuramente legati al fatto che abbiamo scelto di trattare esplicitamente la tripla *snc* (Squadra - Nazione - Città) senza necessariamente dover modellare questa relazione durante lo sviluppo e dunque usando delle ulteriori regole. Inoltre, la modellazione della *partita_ritorno* come una semplice conseguenza delle *partita_andata* ha sicuramente permesso di guadagnare altro tempo in termini di esecuzione.

5 Considerazioni finali

Questa relazione ha voluto racchiudere le informazioni principali che ci hanno permesso di completare correttamente il progetto. Il codice sviluppato ci ha dato modo di indagare su una metodologia di programmazione diversa da quelle a cui ci siamo abituati durante il percorso di studi universitario, consentendoci di consolidare le competenze teoriche sviluppate a lezione. In conclusione, possiamo ritenerci soddisfatti del programma sviluppato e dei risultati ottenuti, che sono conformi ai requisiti progettuali.

Listings

Riferimenti bibliografici

[1] http://wp.doc.ic.ac.uk/arusso/wp-content/uploads/sites/47/2015/01/clingo_guide.pdf