

Esercizio 1. Svolgere tutti i punti.

a-1) Si consideri il seguente programma logico e se ne calcolino gli answer set, illustrando adeguatamente il procedimento seguito.

```
a(Z) :- c(Z, Y), not b(Y), Y = Z + 1. 1
c(Y, Z) :- a(Y), b(Z), not a(Z). 2
c(Y, Y) v b(Y) :- a(Y), not b(Y). 3

c(3, 4).
a(2).
b(3).
```

a-2) Si aggiunga il seguente strong constraint al programma del punto precedente.

```
:- #count{X : b(X), not a(X)} < 2.
```

a-1)

~~a(3) :- c(3, 4), not b(4), 4 = 3 + 1~~

~~c(2, 3) :- a(2), b(3), not a(3).~~ % a(3) è sempre vero

~~c(3, 3) :- a(3), b(3), not a(3).~~

~~c(2, 2) | b(2) :- a(2), not b(2).~~ % b(2) non sarà mai vero

~~c(3, 3) | b(3) :- a(3), not b(3).~~ % b(3) è EDB

~~a(2) :- c(2, 3), not b(3) 3 = 2 + 1.~~ % è già EDB

The grounding termine

AS: {

A1: {[.. EDB ..], a(3), c(2, 2)} X

a-2)

```
:- #count{X : b(X), not a(X)} < 2.
```

A1: :- 0 < 2 SI

Inserendo questo strong constraint, verrebbe scelto A1, rendendo il programma inconsistente.

b) Si consideri ora un programma P (non è necessario sapere come è fatto) i cui answer set sono già stati calcolati e sono riportati di seguito.

```
A1: {girasole(1,4), girasole(2,4), girasole(1,3), girasole(2,3), margherita(5,1)}
A2: {girasole(1,4), girasole(2,4), margherita(4,3), girasole(1,1), girasole(2,1)}
A3: {margherita(1,4), girasole(1,3), girasole(2,3), girasole(1,1), girasole(2,1)}
A4: {girasole(1,4), girasole(2,4), girasole(1,3), girasole(2,3), girasole(1,1), girasole(2,1)}
```

Si supponga di aggiungere i seguenti weak constraint al programma P. Si calcoli quale sarebbe il costo di ognuno degli answer set riportati sopra, e si indichi quello ottimo, commentando il procedimento seguito.

```
% DLV syntax
:- girasole(X,Y), X < Y. [X:Y]
:- girasole(X,W), girasole(Y,Z), not margherita(X,Y). [X:Y]

% ASP-Core-2 syntax
:- girasole(X,Y), X < Y. [X@Y, X, Y]
:- girasole(X,W), girasole(Y,Z), not margherita(X,Y). [X@Y, X, Y, W, Z]
```

b)

A1: 12@1 12@2 3@3 3@4
 $\vdash \neg g(1,4), 1 < 4. [1@4, 1, 4]$
 $\vdash \neg g(2,4), 2 < 4. [2@4, 2, 4]$
 $\vdash \neg g(1,3), 1 < 3. [1@3, 1, 3]$
 $\vdash \neg g(2,3), 2 < 3. [2@3, 2, 3]$

% abbrevio

$\vdash \neg g(1,4), g(1,4), \neg m(1,1).$	$[1@1, 1, 1, 4, 4]$
$\vdash \neg g(2,4), g(1,4), \neg m(2,1).$	$[2@1, 2, 1, 4, 4]$
$\vdash \neg g(1,3), g(1,4), \neg m(1,1).$	$[1@1, 1, 1, 3, 4]$

M

1 2,3	1,4	2,1	2@1, 2, 1, 3, 4
-------	-----	-----	-----------------

A

1 1,4 , 2,4 ,	2,2	2@2, 1, 2, 4, 4
---------------	-----	-----------------

1 2,4 2,4	2,2	2@2, 2, 2, 4, 4
-----------	-----	-----------------

1 1,3 2,4	1,2	1@2, 1, 2, 3, 4
-----------	-----	-----------------

1 2,3 2,4	2,1	2@2, 2, 2, 3, 4
-----------	-----	-----------------

P

1 1,4 1,3	1,1	1@1, 1, 1, 4, 3
-----------	-----	-----------------

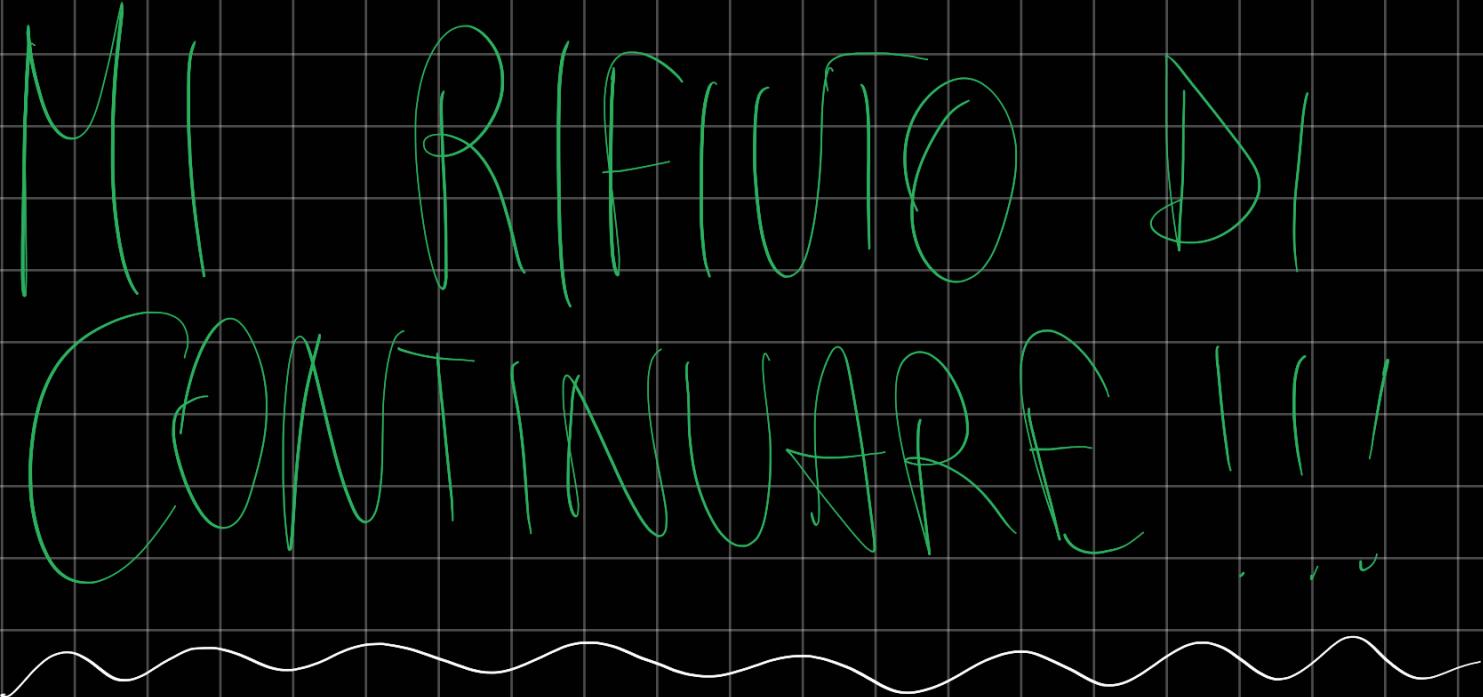
F

1 2,4 1,3	2,1	2@1, 2, 1, 4, 3
-----------	-----	-----------------

O

1,3 1,3	1,1	1@1, 1, 1, 3, 3
---------	-----	-----------------

2,3	1,3	2,1	2@1,2,1,3,3	R
1,4	2,3	1,2	1@2,1,2,4,3	C
2,4	2,3	2,2	2@2,2,2,4,3	H
1,3	2,3	1,2	1@2,1,2,3,3	
2,3	2,3	2,2	2@2,2,2,3,3	E



Esercizio 2. Renata Limbranata, la mogliettina del nostro vecchio amico Ciccio Pasticcio, ha di recente iniziato a frequentare un esclusivo circolo tennistico alle porte di Pasticciopoli. È tutta presa dalla sua nuova passione, e questo lascia a Ciccio un po' di tregua... o almeno così credeva lui! Infatti, la nuova direttrice tecnica del circolo, Fiammetta Laracchetta, ha in mente un nuovo sistema di allenamento piuttosto complesso, che richiede la formazione di squadre e di accoppiamenti tra i soci. Poteva Renata non chiamare in causa il suo adorato marito? Ovviamente no, e pertanto ora ci tocca aiutare Ciccio a scrivere un programma ASP che si occupi di ottenere quanto indicato di seguito.

- 1 • Ogni giocatore deve essere assegnato ad una squadra. Il numero di squadre massimo è noto all'inizio. Si tenga presente che il numero di squadre create può essere minore di quello massimo (cioè non tutte le squadre devono avere dei giocatori assegnati, mentre ogni giocatore deve finire esattamente in una squadra).
- 2 • Se un team non è vuoto, allora deve contenere giocatori di entrambi i sessi.
- 3 • Un primo desiderio, il meno importante di tutti, è che la forza complessiva (cioè la somma delle forze di ciascun membro) delle squadre sia bilanciata. In altri termini, date le forze complessive di due qualunque tra le squadre formate, se ne vuole minimizzare la differenza.
- 4 • La formula dell'allenamento prevede che ciascun team giochi esattamente contro un altro, e che i tutti membri di un team giochino contro tutti quelli del team accoppiato. Si devono pertanto stabilire gli accoppiamenti tra i team.
- Sono note delle statistiche calcolate sullo storico degli allenamenti nel circolo; in particolare, si ha un *grafo orientato pesato* i cui nodi sono i giocatori, e in cui esiste un arco $\langle a,b,wa \rangle$ se il giocatore "a" ha vinto "wa" incontri giocando contro il giocatore "b"; in questo caso, esisterà sempre anche un arco $\langle b,a,wb \rangle$ che indica quante volte il giocatore "b" ha battuto il giocatore "a". "wb+wa" è il numero totale di incontri tra "a" e "b". Pertanto, se due giocatori non hanno mai giocato, non ci sono archi tra loro; altrimenti c'è sempre una coppia di archi come quelli descritti (si noti che qualche arco potrebbe avere peso pari a zero, se uno dei giocatori non ha mai battuto l'altro).
- Nello stabilire gli accoppiamenti:
 - 5 ○ la cosa più importante è massimizzare il numero di incontri che si verificheranno tra i giocatori che non si sono mai incontrati prima;
 - 6 ○ se due giocatori che dovranno giocare a causa dell'accoppiamento tra i team si sono già incontrati in precedenza, si vuole minimizzare la differenza tra le mutue vittorie.
- 7 • Al fine di arricchire le statistiche, si desidera inoltre calcolare il numero totale di incontri già disputati tra gli accoppiamenti che si sono venuti a creare con le nostre scelte.

Modello dei dati in input

player(ID,Gender,Strength)
team(ID)
played(P1,P2,NWin)

← I giocatori del circolo, con il loro sesso e la loro forza stimata
← I possibili team che si possono comporre
← Le informazioni su incontri precedenti tra i giocatori del circolo

Modello dei dati in output

inTeam(Player,Team)
play(Team1,Team2)
history(N)

← La formazione dei team
← Gli accoppiamenti tra i team
← Il numero di incontri già disputati tra giocatori che si incontreranno da ora in poi

- $\text{inTeam}(P, T) \mid \text{inTeam}(P, T) :- \text{player}(P, -, -), \text{team}(T).$

% 1

$:- \text{inTeam}(P, T_1), \text{inTeam}(P, T_2), T_1 \neq T_2.$

% 2

$\text{nonVuoto}(T) :- \text{inTeam}(-, T).$

$:- \text{nonVuoto}(T), \#\{\text{P} : \text{inTeam}(P, T), \text{player}(P, m, -)\} = 0.$

$:- \text{nonVuoto}(T), \#\{\text{P} : (\text{inTeam}(P, T), \text{player}(P, f, -))\} = 0.$

% 3

$\text{forzaComplessiva}(T, F) :- \text{nonVuoto}(T), \#\{\text{S}, \text{P} : \text{inTeam}(P, T), \text{player}(P, -, S)\} = F.$

$\sim \text{forzaComplessiva}(T_1, F_1), \text{forzaComplessiva}(T_2, F_2), T_1 < T_2, D = F_1 - F_2,$

$\exists \text{abs}(D; \text{Cost}) [\text{Cost} @ 1, T_1, T_2]$

% 4

- $\text{play}(T_1, T_2) \mid \text{play}(T_1, T_2) :- \text{nonVuoto}(T_1), \text{nonVuoto}(T_2), T_1 \neq T_2.$

$\text{play}(T_2, T_1) :- \text{play}(T_1, T_2).$

$:- \text{play}(T_1, T_2), \text{play}(T_1, T_3), T_1 \neq T_2, T_2 \neq T_3, T_1 \neq T_3.$

$\text{inGame}(T) :- \text{play}(T, -).$

$:- \text{nonVuoto}(T), \text{not } \text{inGame}(T).$

% 5

$\sim \text{play}(T_1, T_2), \text{inTeam}(P_1, T_1), \text{inTeam}(P_2, T_2), \text{played}(P_1, P_2, W). [W @ 3, P_1, P_2]$

% 6

$\sim \text{play}(T_1, T_2), \text{inTeam}(P_1, T_1), \text{inTeam}(P_2, T_2), \text{played}(P_1, P_2, W_1), \text{played}(P_2, P_1, W_2),$

$D = W_1 - W_2, \exists \text{abs}(D, \text{Cost}), T_1 < T_2. [\text{Cost} @ 2, P_1, P_2]$

% 7

history(N) :- N = #sum{W,P1,P2 : play(T1,T2), inTeam(P1,T1), inTeam(P2,T2),
playeo(P1,P2,W)}.

