

### 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.

```
rubli(X,Y) :- cremlino(Y) :- mosca(X,Y), not russia(Y,Y). 2
russia(X,Y) :- vodka(X,Y,Z), vodka(_,_,Y), not mosca(X,Z).
rubli(X,Y) :- russia(X,Z), Y=Z+2. 3
vodka(1,1,1).    vodka(1,2,3).    vodka(1,3,4).
mosca(1,1).      cremlino(3).
```

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

```
:- #sum{ X : rubli(X,Y) } > 1.
```

a-1)

~~russia(1,3) :- vodka(1,3,4), vodka(1,2,3), not mosca(1,4).~~

~~russia(1,1) :- vodka(1,1,1), vodka(1,1,1), not mosca(1,1).~~ ← ~~mosca(1,1)~~ è EDB

~~rubli(1,1) | cremlino(1) :- mosca(1,1), not russia(1,1).~~

~~rubli(1,5) :- russia(1,3), 5 = 3 + 2~~

AS: {

A1: {[...EDB...], russia(1,3), rubli(1,1), rubli(1,5)}  
A2: {[...EDB...], russia(1,3), cremlino(1), rubli(1,5)}

}

a-2)

```
:- #sum{ X: rubli(X,Y) } > 1
```

A1: 1>1 NO

A2: 1>1 NO

Inserendo questo strong constraint, non venne scelto alcun AS perché, nonostante A1 contiene due istanze di "rubli", le somme non sono a distanza 1, ovvero "1" non viene controllata.

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: {aglio(3), aglio(4), lupomannaro(3), aglio(2), zombie(4), zombie(2)}  
A2: {aglio(3), aglio(4), lupomannaro(3), aglio(2), lupomannaro(4), zombie(2)}  
A3: {aglio(3), aglio(4), lupomannaro(3), aglio(2), vampiro(4), zombie(2)}  
A4: {aglio(3), aglio(4), lupomannaro(3), aglio(2), lupomannaro(4), vampiro(2)}  
A5: {aglio(3), aglio(4), lupomannaro(3), aglio(2), vampiro(4), vampiro(2)}
```

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

```
% DLV syntax  
:- zombie(X), zombie(Y), Y<X. [ X : Y ]  
:- aglio(X), vampiro(X). [ X : 2 ]  
:- argento(X,Y), lupomannaro(X). [ 1 : Y ]  
% ASP-Core-2 syntax  
:- zombie(X), zombie(Y), Y<X. [ X@Y, X,Y ]  
:- aglio(X), vampiro(X). [ X@2, X ]  
:- argento(X,Y), lupomannaro(X). [ 1@Y, X,Y ]
```

A1: 4@2  
: $\sim$  zombie(4), zombie(2), 2<4. [4@2, 4, 2]

A2: 0@1      OPTIMUM

A3: 4@2  
: $\sim$  aglio(4), vampiro(4) [4@2, 4]

A4: 2@2  
: $\sim$  aglio(2), vampiro(2). [2@2, 2]

A5: 6@2  
: $\sim$  aglio(2), vampiro(2). [2@2, 2]  
: $\sim$  aglio(4), vampiro(4). [4@2, 4]

**Esercizio 2.** Siamo a Pasticciopoli e la nostra amica Renata Limbranata, che non sa mai star ferma, ha deciso di iniziare una nuova attività. C'è un vecchio complesso multisala appena fuori città con una vista fantastica; ha deciso di comprarlo, ristrutturarlo e creare un centro eventi che sia sempre attivo! Ovviamente, ha una socia, la sua vecchia amica Ernesta Machefera; il problema

Corso di Laurea in Informatica  
**Corso di Intelligenza Artificiale**

### Prova d'esame del 17/07/2019

è che le due amiche non sono molto esperte del settore, e dopo un po' di tempo perdono la testa nel tentativo di gestire nel modo migliore l'uso delle strutture a seconda delle richieste di prenotazioni (che, per fortuna!, arrivano numerose). Come al solito, tocca al nostro vecchio Ciccio Pasticcio intervenire per salvare la situazione... e quindi tocca anche a voi. Si scriva un programma logico ASP che stabilisca come allocare le richieste di una giornata, tenendo conto delle specifiche indicate di seguito.

- Nel complesso ci sono un certo numero di sale, ciascuna con la propria capienza, tutte disponibili per l'arco della giornata (dall'inizio alla fine delle attività).
- A Renata ed Ernesta arrivano richieste di eventi, ciascuno di una certa tipologia, con un orario di inizio e fine e con un certo numero di persone coinvolte. Ciascun evento potrà essere accettato in una delle sale disponibili, oppure rifiutato.
- [c1] Un evento non può stare in due sale diverse!
- [c2] Naturalmente, un evento non può essere assegnato ad una sala che non sia abbastanza capiente per ospitarlo.
- [c3] Una sala non può ospitare contemporaneamente più di un evento, nemmeno se è abbastanza capiente; pertanto si devono evitare sovrapposizioni (anche parziali) di eventi nella stessa sala.
- Ci sono poi dei criteri fondamentali per l'ottimizzazione dell'uso delle risorse; sono elencati di seguito, IN ORDINE DI IMPORTANZA CRESCENTE:
  - [w1] Si desidera minimizzare il numero di sale usate (cioè, se possibile, è preferibile lasciare una sala completamente inutilizzata).
  - [w2] Se una sala è utilizzata è meglio che lo sia al massimo; pertanto, si vogliono ridurre, per ogni sala usata, i tempi in cui non è ospitato alcun evento.
  - [w3] Per ovvi motivi, è più facile chiudere un evento e prepararsi ad ospitare il successivo se questi sono dello stesso tipo. Pertanto, si vogliono minimizzare gli eventi di tipo diverso ospitati in una stessa sala.
  - [w4] Il profitto: vogliamo massimizzare il numero totale di persone ospitate.

#### MODELLO DEI DATI IN INPUT

inizioMin(I)	← l'orario a partire dal quale le sale sono disponibili
fineMax(F)	← l'orario fino al quale sono disponibili le sale
oreMax(O)	← il numero totale di ore in cui le sale sono disponibili (i.e., F-I)
sala(ID,C)	← le sale con la propria capienza
evento(ID,I,F,P,T)	← gli eventi, ciascuno con orario di inizio, orario di fine, persone coinvolte e tipologia

- *associato(ID,S) | associato(ID,S) :- evento(ID,-,-,-,-), sala(S,-).*  
*:- #count{ID: associato(ID,-)} = 0.*

% [c1]

*:- associato(ID,S1), associato(ID,S2), S1 != S2.*

% [c2]

*:- associato(ID,S), evento(ID,-,P,-), sala(S,C), P > C.*

% [c3]

/o [v3]  
:- associato(ID1, S), associato(ID2, S), ID1 != ID2, evento(ID1, 11, F1, -, -),  
evento(ID2, 12, F2, -, -), 12 < F1, 12 >= 11.

% [w1] OPPURE ...

salaUsata(S) :- associato(-, S). | associato(ID, S). [1@1, S]  
:- salaUsata(S). [1@1, S]

% [w2]

tempoliberoSala(S, T) :- O = #sum{TF, ID : associato(ID, S), evento(ID, 1, F, -, -),  
TF = F - 1}, salaUsata(S), T = 24 - O.  
:- tempoliberoSala(S, T). [T@2, S, T]

% [w3]

:- ~associato(ID1, S), associato(ID2, S), ID1 != ID2, evento(ID1, -, -, -, T1),  
evento(ID2, -, -, -, T2), T1 != T2. [1@3, ID1, ID2]

% [w4]

eventoAssociato(ID) :- evento(ID, -, -, -, -), associato(ID, -).  
eventoNonAssociato(ID, P) :- evento(ID, -, -, P, -), not eventoAssociato(ID).  
:- eventoNonAssociato(ID, P). [P@4, ID, P]

