La programmazione logica come metodologia per la codifica e la risoluzione di rompicapi

Agostino Dovier

Dipartimento di Matematica e Informatica
Università di Udine

04 Dicembre 2015

Programmazione in ASP

Aritmetica — vedi

http://sourceforge.net/projects/potassco/files/guide/

Come estensione sintattica, possiamo usare i seguenti “built-in”

```
plus      L + R
minus     L - R
uminus    - R
times     L * R
divide    L / R
modulo    L \ R
absolute  |  R|
power     L ** R
bitand    L & R
bitor     L ? R
bitxor    L ^ R
bitneg    ~ R
```

Programmazione in ASP

Aggregati

- ASP supporta l'utilizzo di *aggregati*
- Permettono di definire in modo intensionale funzioni associate ai valori degli argomenti degli atomi nell'answer set.
- Ci sono due principali aggregati: `count` e `sum`.
- Gli aggregati sono uno strumento di programmazione molto espressivo; la loro sintassi non è ancora "in evoluzione" (si consiglia di leggere il manuale più recente).

Programmazione in ASP

Aggregati ad esempi

```
dom(1..3).
p(1,1).  p(2,2).   p(3,3).
somma(S) :- S = #sum { Y : dom(X), p(X,Y) }.
```

L'output è

Programmazione in ASP

Aggregati ad esempi

```
dom(1..3).
p(1,1).  p(2,2).   p(3,3).
somma(S) :- S = #sum { Y : dom(X), p(X,Y) }.
```

L'output è `somma(6).`

Programmazione in ASP

Aggregati ad esempi

```
dom(1..3).
p(1,1).  p(2,2).   p(3,3).
somma(S) :- S = #sum { Y : dom(X), p(X,Y) }.
```

L'output è `somma(6).`

```
dom(1..3).
p(1,1).  p(2,2).   p(3,2).
somma(S) :- S = #sum { Y : dom(X), p(X,Y) }.
```

L'output è

Programmazione in ASP

Aggregati ad esempi

```
dom(1..3).
p(1,1).  p(2,2).   p(3,3).
somma(S) :- S = #sum { Y : dom(X), p(X,Y) }.
```

L'output è `somma(6).`

```
dom(1..3).
p(1,1).  p(2,2).   p(3,2).
somma(S) :- S = #sum { Y : dom(X), p(X,Y) }.
```

L'output è `somma(3).`

Qualcosa non torna?

Programmazione in ASP

Aggregati ad esempi

```
dom(1..3).
p(1,1).  p(2,2).   p(3,3).
somma(S) :- S = #sum { Y : dom(X), p(X,Y) }.
```

L'output è `somma(6).`

```
dom(1..3).
p(1,1).  p(2,2).   p(3,2).
somma(S) :- S = #sum { Y : dom(X), p(X,Y) }.
```

L'output è `somma(3).`

Qualcosa non torna? Considera insiemi, non multi-insiemi.

Programmazione in ASP

Aggregati ad esempi

```
dom(1..3).
p(1,1).  p(2,2).   p(3,2).
somma(S) :- S = #sum { Y,X : dom(X), p(X,Y) }.
```

L'output è `somma(5).`

Se rischiamo di "perdere" valori ripetuti, il trucco è quello di lavorare sulle coppie. Nelle versioni precedenti di gringo/clingo la sintassi era completamente diversa
(il passaggio non è stato indolore nella comunità...)

Viene sommata solo la prima variabille `Y` ma si collezionano le diverse coppie `(Y,X)` dunque si gestiscono anche le duplicazioni di doppioni.

Programmazione in ASP

Aggregati ad esempi

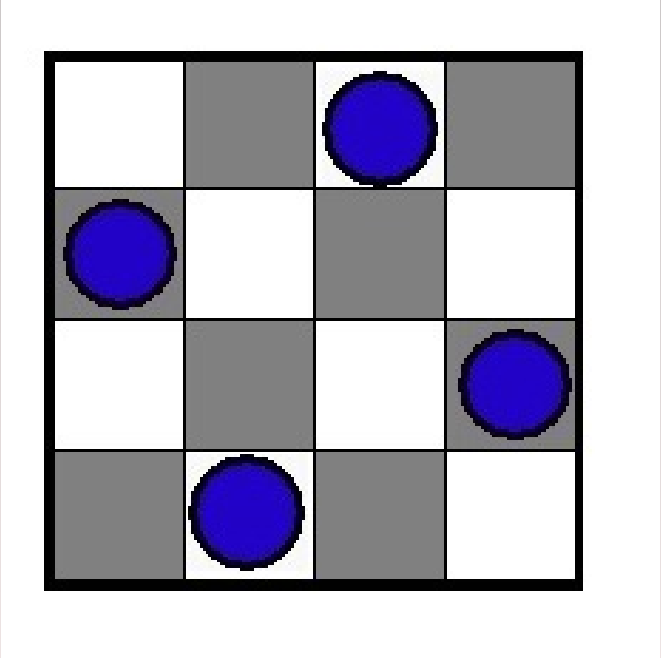
`count` ha sintassi (e problemi annessi) simili, ma conta il numero di atomi che soddisfano una condizione. Ad esempio:

```
dom(1..3).
p(1,1).  p(2,2).   p(3,3).
conta(S) :- S = #count{ p(X,Y):dom(X),p(X,Y),Y > 2 }.
```

L'output è `conta(1).`

N-Queens

4-Regine: sistemare 4 regine su una scacchiera 4×4 in modo tale che le regine non si attacchino.



Avevamo già visto un modeling “a vincoli” nella prima lezione. Vediamo ora il modello ASP.

Definiamo il predicato `queens/2`

```
numero(1..n).

%%% queen( i, j) sse nella riga i la regina e' in colonna j
%%% Per ogni colonna j c'e' esattamente una regina
1 {queen(I,J) : numero(I)} 1 :- numero(J).

%%% Attacco in orizzontale.
:- numero(I), numero(J1), numero(J2),
   J1 != J2,
   queen(I,J1), queen(I,J2).

%%% Attacco in diagonale
:- numero(I1),numero(I2),numero(J1),numero(J2),
   I1 != I2,
   queen(I1,J1), queen(I2,J2),
   | I1-I2 |  = | J1-J2 |.

#show queen/2.
```

Zoom sul vincolo 1

```
:- numero(I), numero(J1), numero(J2),
   J1 != J2,
   queen(I,J1), queen(I,J2).
```

Lo leggiamo come *Non è possibile che ci sia un numero I, un numero J1, un (altro) numero J2 con J1 $\neq$ J2, tali che ci sia la regina nella riga I sia nella colonna J1 che nella colonna J2*
Se volete

$$(\exists I)(\exists J_1)(\exists J_2)\,(J_1 \neq J_2 \wedge \texttt{queen}(I, J_1) \wedge \texttt{queen}(I, J_2)) \longrightarrow \texttt{false}$$

Zoom sul vincolo 1

```
:- numero(I), numero(J1), numero(J2),
   J1 != J2,
   queen(I,J1), queen(I,J2).
```

Lo leggiamo come *Non è possibile che ci sia un numero I, un numero J1, un (altro) numero J2 con J1 $\neq$ J2, tali che ci sia la regina nella riga I sia nella colonna J1 che nella colonna J2*
Se volete

$$(\exists I)(\exists J_1)(\exists J_2)\,(J_1 \neq J_2 \wedge \texttt{queen}(I, J_1) \wedge \texttt{queen}(I, J_2)) \longrightarrow \texttt{false}$$

Ovvero $A \rightarrow B$ che equivale a $\neg B \rightarrow \neg A$.
Ma $\neg B = \texttt{true}$ dunque semplicemente equivale a $\neg A$:

$$(\forall I)(\forall J_1)(\forall J_2)\,((\texttt{queen}(I, J_1) \wedge \texttt{queen}(I, J_2)) \longrightarrow J_1 = J_2)$$

Zoom sul vincolo 2

```
:- numero(I1),numero(I2),numero(J1),numero(J2),
   J1 != J2,
   queen(I1,J1), queen(I2,J2),
   | I1-I2 |  = | J1-J2 |.
```

Lo leggiamo come *Non è possibile che ci siano numeri I1,I2,J1,J2 con J1 $\neq$ J2, tali che ci sia tra la regina in colonna J1 e quella in colonna J2 valga il vincolo numerico (che garantisce l'attacco in diagonale)*

Zoom sul vincolo 2

```
:- numero(I1),numero(I2),numero(J1),numero(J2),
   J1 != J2,
   queen(I1,J1), queen(I2,J2),
   | I1-I2 |  = | J1-J2 |.
```

Lo leggiamo come *Non è possibile che ci siano numeri I1,I2,J1,J2 con J1 $\neq$ J2, tali che ci sia tra la regina in colonna J1 e quella in colonna J2 valga il vincolo numerico (che garantisce l'attacco in diagonale)*

Cosa succede se cambiamo I1 $\neq$ I2 con I1 $<$ I2?

Magic Square

Problem: Fill an $n \times n$ matrix (square) with the numbers $1, \ldots, n^2$ in such a way that each number is used once and the sum of each row, of each column, and of each of the two diagonals is the same.
Since the global sum is

$$\sum_{i=1}^{n^2} i = \frac{n^2(n^2+1)}{2}$$

each one of these sums amounts at $S = \frac{n^2(n^2+1)}{2n}$.

Magic Square

Problem: Fill an $n \times n$ matrix (square) with the numbers $1, \ldots, n^2$ in such a way that each number is used once and the sum of each row, of each column, and of each of the two diagonals is the same.
Since the global sum is

$$\sum_{i=1}^{n^2} i = \frac{n^2(n^2+1)}{2}$$

each one of these sums amounts at $S = \frac{n^2(n^2+1)}{2n}$.
E.g., with $n = 3$, $S = 15$.

2	7	6
9	5	1
4	3	8

An imperative solution

Scan permutations

```
int main(){

  int i=0, j=0, found;
  long int max;
  int quad [n*n];

  for(i=0;i<n;i++)
     for(j=0;j<n;j++)
         quad[i*n+j] = n*i+j+1;

  found = 0;
  max   = fact(n*n);
  printf("Tentativi massimi: %ld\n",max);

  while( !found &&  max>0) {
     if (check_sum(quad))
        found = 1;
     else {
        increment(quad);
        max--;
     }
   }

  if (found)
     stampamat(quad);
  else printf("No solution \n");
}
```

An imperative solution

Check

```
const int n=3;
int MAGIC=((n*n)*((n*n)+1))/(2*n);


int check_sum(int M [n*n]){
  int i,j, temp;
  int correct=1;

  // ROWS
  for(i=0;i<n;i++){
    temp=0;
    for(j=0;j<n;j++)
          temp=temp+M[n*i+j];
    if(temp != MAGIC)
       correct=0;
  }

  // COLS
  if (correct){
    for(j=0;j<n;j++){
      temp=0;
      for(i=0;i<n;i++)
          temp=temp+ M[n*i+j];
      if(temp != MAGIC)
         correct=0;
    }
  }
  //DIAG 1
  if (correct){
      temp=0;
      for(i=0;i<n;i++)
          temp=temp+ M[n*i+i];
      if(temp != MAGIC)
          correct=0;
  }

  // DIAG2
  if (correct){
      temp=0;
      for(i=0;i<n;i++)
          temp=temp+ M[n*i+n-i-1];
      if(temp != MAGIC)
          correct=0;
  }

return correct;
}
```

An imperative solution

Dijkstra, A Discipline of Programming, Prentice-Hall, 1997, pp. 71

```
void increment(int M [n*n]){
    int i,j,t;

    i = n*n - 1;
    while (M[i-1] > M[i]) i--;

    j = n*n;
    while (M[j-1] <= M[i-1]) j--;

    // swap values at positions (i-1) and (j-1)
    t = M[i-1];
    M[i-1] = M[j-1];
    M[j-1] = t;

    i++;
    j = n*n;
    while (i < j) {
       // swap values at positions (i-1) and (j-1)
       t = M[i-1];
       M[i-1] = M[j-1];
       M[j-1] = t;
       i++;
       j--;
    }
}
```

An imperative solution

Auxiliary procedures

```
long int fact(int m){
  int i;
  long int t=1;
  for(i=1;i <=m; i++) t=t*i ;
  return t;
}

void stampamat(int quad [n*n]){
   int i, j;

   for(i=0;i<n;i++){
     printf("| ");
     for(j=0;j<n;j++)
          printf(" %d |",quad[i*n+j]);
     printf("\n ---- \n");
   }
}
```

ASP modeling

Just a quick view

```
lato(1..n).
valore(1..n*n).
diag(1..2).
magicval(((n*n)*(n*n+1))/(2*n)).

1 { magic(X,Y,V) : valore(V) } 1 :- lato(X),lato(Y).
1 { magic(X,Y,V) : lato(X),lato(Y) } 1 :- valore(V).


sum_cols(X, S ) :- S = #sum{ V : magic(X,L,V), lato(L)}, lato(X).
sum_rows(Y, S ) :- S = #sum{ V : magic(L,Y,V), lato(L)}, lato(Y).
sum_diag(1, S ) :- S = #sum{ V : magic(L,L,V), lato(L)}.
sum_diag(2, S ) :- S = #sum{ V : magic(L,n-L+1,V), lato(L)}.

:- lato(X), sum_cols(X,V), magicval(T), V != T.
:- lato(X), sum_rows(X,V), magicval(T), V != T.
:- diag(D), sum_diag(D,V), magicval(T), V != T.

#show magic/3.
```

Efficiency

```
Agostino@ACUFENE /cygdrive/c/Documents and Settings/Agostino/Dropbox/VARIE
$ time ./a.exe
Tentativi massimi: 362880
HAI VINTO
|  2 | 7 | 6 |
 ----
|  9 | 5 | 1 |
 ----
|  4 | 3 | 8 |
 ----

real    0m0.094s
user    0m0.015s
sys     0m0.046s
Agostino@ACUFENE /cygdrive/c/Documents and Settings/Agostino/Dropbox/VARIE
$ time ./a.exe
Tentativi massimi: 20922789888000
HAI VINTO
|  1 | 2 | 15 | 16 |
 ----
|  12 | 14 | 3 | 5 |
 ----
|  13 | 7 | 10 | 4 |
 ----
|  8 | 11 | 6 | 9 |
 ----

real    74m16.214s
user    74m13.376s
sys     0m0.077s
```

Efficiency

```
$ clingo -c n=3 quadrato.lp
Clingo version 4.4.0
Reading from quadrato.lp
Solving...
Answer: 1
magic(1,1,2) magic(1,3,6) magic(1,2,7) magic(2,3,1) magic(2,2,5) magic(2,1,9) m
agic(3,2,3) magic(3,1,4) magic(3,3,8)
SATISFIABLE

Models       : 1+
Calls        : 1
Time         : 0.077s (Solving: 0.06s 1st Model: 0.06s Unsat: 0.00s)
CPU Time     : 0.078s

Agostino@ACUFENE /cygdrive/c/Users/Agostino/Dropbox/DIDATTICA/AI-LP-GIOCHI/CODI
CI
$ clingo -c n=4 quadrato.lp
Clingo version 4.4.0
Reading from quadrato.lp
Solving...
Answer: 1
magic(1,2,4) magic(1,4,5) magic(1,3,10) magic(1,1,15) magic(2,2,1) magic(2,4,8)
 magic(2,3,11) magic(2,1,14) magic(3,1,3) magic(3,3,6) magic(3,4,9) magic(3,2,1
6) magic(4,1,2) magic(4,3,7) magic(4,4,12) magic(4,2,13)
SATISFIABLE

Models       : 1+
Calls        : 1
Time         : 1.146s (Solving: 1.04s 1st Model: 1.04s Unsat: 0.00s)
CPU Time     : 1.139s

Agostino@ACUFENE /cygdrive/c/Users/Agostino/Dropbox/DIDATTICA/AI-LP-GIOCHI/CODI
```

Efficiency

```
Agostino@ACUFENE /cygdrive/c/Users/Agostino/Dropbox/DIDATTICA/AI-LP-GIOCHI/CODI
CI
$ clingo -c n=5 quadrato.lp
clingo version 4.4.0
Reading from quadrato.lp
Solving...
Answer: 1
magic(1,1,2) magic(1,4,7) magic(1,2,16) magic(1,3,17) magic(1,5,23) magic(2,5,3
) magic(2,1,5) magic(2,3,13) magic(2,4,20) magic(2,2,24) magic(3,3,6) magic(3,2
,10) magic(3,4,12) magic(3,1,18) magic(3,5,19) magic(4,2,1) magic(4,3,8) magic(
4,5,9) magic(4,4,22) magic(4,1,25) magic(5,4,4) magic(5,5,11) magic(5,2,14) mag
ic(5,1,15) magic(5,3,21)
SATISFIABLE

Models       : 1+
Calls        : 1
Time         : 95.666s (Solving: 95.11s 1st Model: 95.10s Unsat: 0.00s)
CPU Time     : 95.457s

Agostino@ACUFENE /cygdrive/c/Users/Agostino/Dropbox/DIDATTICA/AI-LP-GIOCHI/CODI
CI
$ clingo -c n=6 quadrato.lp
clingo version 4.4.0
Reading from quadrato.lp
Solving...
Answer: 1
magic(1,3,6) magic(1,5,15) magic(1,1,19) magic(1,6,20) magic(1,4,21) magic(1,2,3
0) magic(2,2,4) magic(2,6,7) magic(2,5,18) magic(2,1,22) magic(2,4,26) magic(2,3
,34) magic(3,2,1) magic(3,1,5) magic(3,4,12) magic(3,3,28) magic(3,6,32) magic(3
,5,33) magic(4,6,8) magic(4,4,11) magic(4,1,16) magic(4,3,23) magic(4,2,24) magi
c(4,5,29) magic(5,6,9) magic(5,4,10) magic(5,5,14) magic(5,3,17) magic(5,2,25) m
agic(5,1,36) magic(6,5,2) magic(6,3,3) magic(6,1,13) magic(6,2,27) magic(6,4,31)
 magic(6,6,35)
SATISFIABLE

Models       : 1+
Calls        : 1
Time         : 42111.245s (Solving: 42108.98s 1st Model: 42108.97s Unsat: 0.00s)
CPU Time     : 42084.000s

Agostino@ACUFENE /cygdrive/c/Users/Agostino/Dropbox/DIDATTICA/AI-LP-GIOCHI/CODIC
```

Lesson learnt

- If you don't have a great idea, direct encoding in C/JAVA etc is useless
- This applies to all NP complete problems
- ASP solvers (and SAT solvers and CP solvers) are nowadays much more clever than any heuristic we can develop, implement and test in a reasonable time

ASP modeling

The case $n = 3$

```
lato(1..3).
valore(1..9).
diag(1..2).
magicval(15).

% Unicita' di valore per cella e non ripetizione dei valori
1 { magic(X,Y,V) : valore(V) } 1 :- lato(X),lato(Y).
1 { magic(X,Y,V) : lato(X),lato(Y)  } 1 :- valore(V).

% Calcolo delle somme
sum_col(X,V1+V2+V3)  :- magic(X,1,V1), magic(X,2,V2), magic(X,3,V3).
sum_row(Y,V1+V2+V3)  :- magic(1,Y,V1), magic(2,Y,V2), magic(3,Y,V3).
sum_diag(1,V1+V2+V3) :- magic(1,1,V1), magic(2,2,V2), magic(3,3,V3).
sum_diag(2,V1+V2+V3) :- magic(1,3,V1), magic(2,2,V2), magic(3,1,V3).

% If a column has a wrong sum then diff_sum is true
diff_sum :- lato(X),  sum_col(X,V), magicval(S), X != S.
% If a row has a wrong sum then diff_sum is true
diff_sum :- lato(X),  sum_row(X), magicval(S), X != S.
% If a diagonal has a wrong sum then diff_sum is true
diff_sum :- sum_diag(1,V),magicval(S), X != S.
diff_sum :- sum_diag(2,V),magicval(S), X != S.
% diff_sum cannot happen
:- diff_sum.
```

ASP modeling

The case $n = 3$

```
lato(1..3).
valore(1..9).
diag(1..2).
magicval(15).

% Unicita' di valore per cella e non ripetizione dei valori
1 { magic(X,Y,V) : valore(V) } 1 :- lato(X),lato(Y).
1 { magic(X,Y,V) : lato(X),lato(Y)  } 1 :- valore(V).

% Calcolo delle somme
sum_col(X,V1+V2+V3)  :- magic(X,1,V1), magic(X,2,V2), magic(X,3,V3).
sum_row(Y,V1+V2+V3)  :- magic(1,Y,V1), magic(2,Y,V2), magic(3,Y,V3).
sum_diag(1,V1+V2+V3) :- magic(1,1,V1), magic(2,2,V2), magic(3,3,V3).
sum_diag(2,V1+V2+V3) :- magic(1,3,V1), magic(2,2,V2), magic(3,1,V3).

% It cannot happen that for one column the sum is wrong
 :- lato(X),  sum_col(X,V), magicval(S), X != S.
% It cannot happen that for one row the sum is wrong
  :- lato(X),  sum_row(X), magicval(S), X != S.
% It cannot happen that for one diagonal the sum is wrong
  :- sum_diag(1,V),magicval(S), X != S.
  :- sum_diag(2,V),magicval(S), X != S.
```

ASP modeling

The general case

```
lato(1..n).
valore(1..n*n).
diag(1..2).
magicval(((n*n)*(n*n+1))/(2*n)).

1 { magic(X,Y,V) : valore(V) } 1 :- lato(X),lato(Y).
1 { magic(X,Y,V) : lato(X),lato(Y) } 1 :- valore(V).

% Use aggregates for computing the sum in a compact way
sum_cols(X, S ) :- S = #sum{ V : magic(X,L,V), lato(L)}, lato(X).
sum_rows(Y, S ) :- S = #sum{ V : magic(L,Y,V), lato(L)}, lato(Y).
sum_diag(1, S ) :- S = #sum{ V : magic(L,L,V), lato(L)}.
sum_diag(2, S ) :- S = #sum{ V : magic(L,n-L+1,V), lato(L)}.

% It cannot happen that for one column the sum is wrong (for all lato(X))
:- lato(X), sum_cols(X,V), magicval(T), V != T.
% It cannot happen that for one row the sum is wrong (for all lato(X))
  :- lato(X), sum_rows(X,V), magicval(T), V != T.
% It cannot happen that for one diagonal the sum is wrong (for all diag(D))
:- diag(D), sum_diag(D,V), magicval(T), V != T.

#show magic/3.
```

Lesson learnt

- Use of cardinality constraints $1\{\cdots\}1$ for forcing relations to be functions
- Use of cardinality constraints $1\{\cdots\}1$ for forcing injectivity.
- Use of aggregates (sum) for compact encoding of properties
- Use of constraints for removing wrong solutions. They allow to express *universal quantification*.

La capra e il cavolo

Un pastore deve attraversare un fiume portando sull'altra riva un lupo e una capra affamati e un cavolo gigante. Ha a disposizione una barca a remi con la quale può traghettare un solo oggetto o animale alla volta. Ma, attenzione, non può lasciare da soli:

- il lupo e la capra perché il lupo si mangia la capra;
- la capra ed il cavolo perché la capra si mangia il cavolo.

Quanti viaggi deve fare per portare sull'altra riva il lupo, la capra ed il cavolo?

La capra e il cavolo

Un problema di planning

```
time(1..n).
place(left;right;boat).
object(man;goat;cabbage;wolf).

%%% In any time, any object is exactly in one place.
1 { on(T,O,P) : place(P) } 1 :- time(T), object(O).

%%% INITIAL STATE
on(1,goat,left).    on(1,cabbage,left).
on(1,wolf,left).     on(1,man,left).

%%%     IF (goat and cabbage) OR (goat and wolf) are in place P,
%%%     THEN the man is in P
on(T,man,P) :- on(T,goat,P), on(T,cabbage,P), time(T), place(P).
on(T,man,P) :- on(T,goat,P), on(T,wolf,P), time(T), place(P).

%%% Boat effect (it is important where the motion has started)
on(T+2,O,right):- on(T+1,O,boat), on(T,O,left),  time(T), object(O).
on(T+2,O,left) :- on(T+1,O,boat), on(T,O,right), time(T), object(O).
```

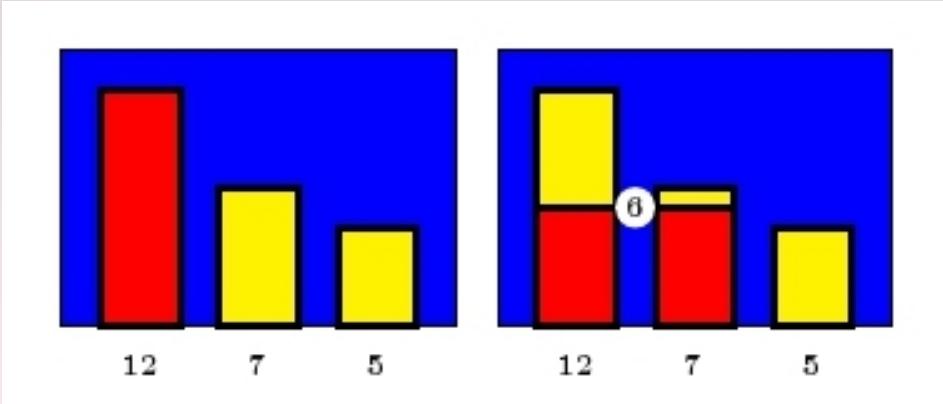
La capra e il cavolo

Un problema di planning (come “indovinare” il giusto `n` ?)

```
%%% IF someone is in boat, then the man must be on boat.
on(T,man,boat) :- on(T,O,boat), time(T), object(O).

%%% The boat contains from 0 to 2 objects
0 { on(T,O,boat) : object(O) } 2 :- time(T).

%%% INERTIA rules
:- on(T+1,O,left), on(T,O,right), time(T), object(O).
:- on(T+1,O,right), on(T,O,left), time(T), object(O).

%%% FINAL STATE (goal)
final :- on(n,goat,right), on(n,cabbage,right),
         on(n,wolf,right), on(n,man,right).
:- not final.

#show on/3.
```

The three barrels

"Ci sono tre botti di capacità N (pari), $N/2+1$, e $N/2-1$ (esempio: 12,7,5). All'inizio la botte più capace è piena di vino e le altre sono vuote. Si vuol raggiungere una conformazione in cui la più piccola è vuota e le altre due contengono esattamente la stessa quantità. Le sole azioni permesse sono di svuotare una botte in un'altra (non è permesso di misurare i flussi o l'altezza del vino o pesare le botti)."



The three barrels

Un problema di planning

```
litri(0..12).
botte(5;7;12).
tempo(0..t).

% C'e' uno ed uno solo solo versamento ald ogni tempo T
1{ versa(T,X,Y): botte(X), botte(Y), X != Y} 1 :- tempo(T), T < t.

% Stato iniziale
contains(0,12,12).    contains(0,7,0).   contains(0,5,0).

% Stato finale
goal :- contains(t,12,6), contains(t,7,6), contains(t,5,0).
:- not goal.
```

The three barrels

Effetto del versare:

```
contains(T+1,X,0) :-
    tempo(T),tempo(T+1),
    botte(X),  botte(Y), X != Y,
    litri(LX), litri(LY),
    %%%
    versa(T,X,Y),
    contains(T,X,LX),contains(T,Y,LY),
    Y-LY > LX.
contains(T+1,Y,LX+LY) :-
    tempo(T),  tempo(T+1),
    botte(X),  botte(Y), X != Y,
    litri(LX), litri(LY),
    %%%
    versa(T,X,Y),
    contains(T,X,LX), contains(T,Y,LY),
    Y-LY > LX.
```

The three barrels

Effetto del versare:

```
contains(T+1,X,LX-(Y-LY)) :-
tempo(T),  tempo(T+1),
     botte(X),  botte(Y), X != Y,
     litri(LX), litri(LY),
versa(T,X,Y),
contains(T,X,LX),
contains(T,Y,LY),
Y-LY <= LX.
contains(T+1,Y,Y) :-
tempo(T),  tempo(T+1),
     botte(X),  botte(Y), X != Y,
     litri(LX), litri(LY),
versa(T,X,Y),
contains(T,X,LX),
contains(T,Y,LY),
Y-LY <= LX.
```

The three barrels

Inertia:

```
contains(T+1,B,L) :-
      tempo(T),tempo(T+1),litri(L),
      botte(X),  botte(Y), botte(B),
      X != Y, X != B, Y != B
      versa(T,X,Y).

%%% Vincoli (non si versa da una vuota ne' in una piena)
:- botte(X), botte(Y), X!=Y, tempo(T), versa(T,X,Y), contains(T,X,0).
:- botte(X), botte(Y), X!=Y, tempo(T), versa(T,X,Y), contains(T,Y,Y).

#show versa/3.
```

Conclusioni

- Abbiamo visto l'uso ricorrente del vincolo di cardinalità 1 . . . 1 nel modeling
- Abbiamo visto come usare i vincoli per eliminare in modo compatto situazioni sbagliate.
- Abbiamo usato gli aggregati (`sum`)
- Abbiamo visto i problemi di planning (stato iniziale/stato finale, azioni)

Esercizi

- Data una scacchiera $n \times n$, e due numeri h e k, si risolva il problema di sistemare h torri e k alfieri in modo che non si attacchino vicendevolmente.
- Dato un rettangolo $m \times n$, si trovi un cammino per un cavallo degli scacchi che passi una ed una sola volta per ogni cella (x, y) (punto di partenza e di arrivo sono distinti).
- Si codifichi il problema dal film Die Hard: `http://puzzles.nigelcoldwell.co.uk/twentytwo.htm`
- Si consideri il gioco della vita di Conway: `https://it.wikipedia.org/wiki/Gioco_della_vita`. Si consideri una scacchiera di dimensione $n \times n$ e si scriva un programma ASP in grado di trovare una configurazione iniziale con al massimo k celle vive che al tempo t permette di avere almeno h celle vive.
- Si scriva un programma ASP che permette di identificare configurazioni stabili, con esattamente t elementi vivi.