AUTOMATED REASONING

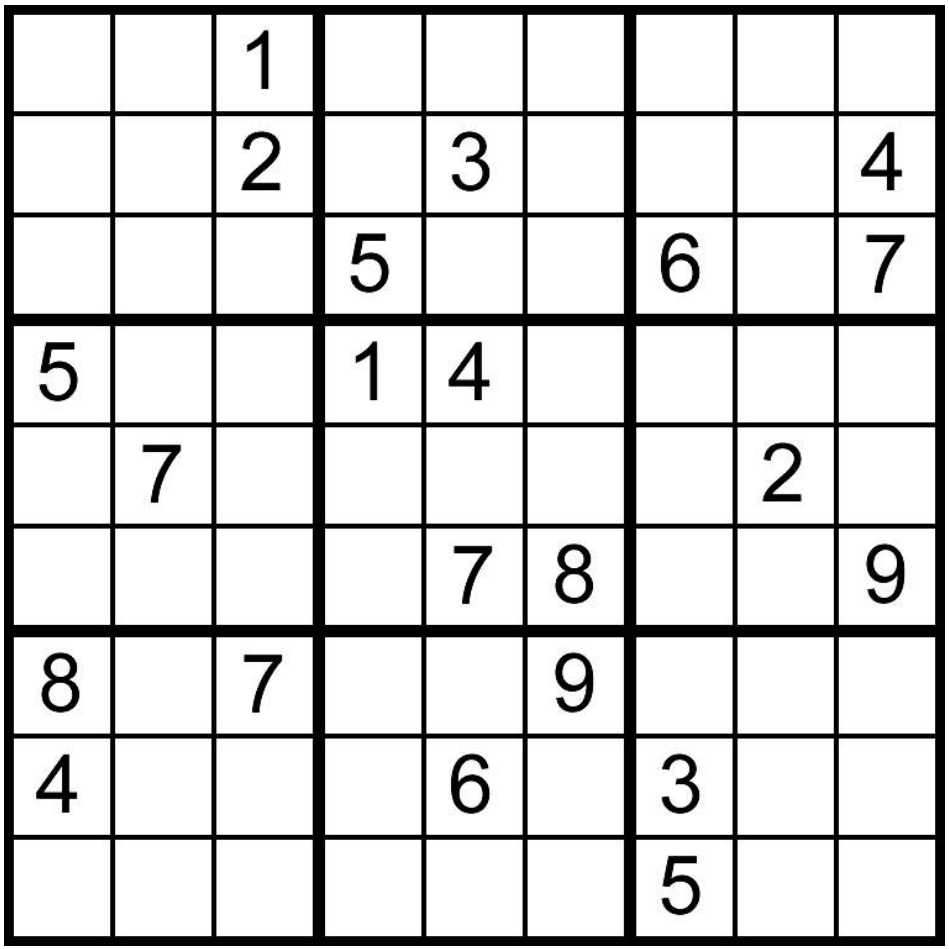
Agostino Dovier

Università di Udine
CLPLAB

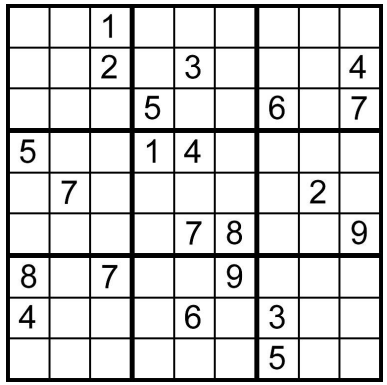
Udine, November 2016

Modeling CSP with ASP

SUDOKU



SUDOKU



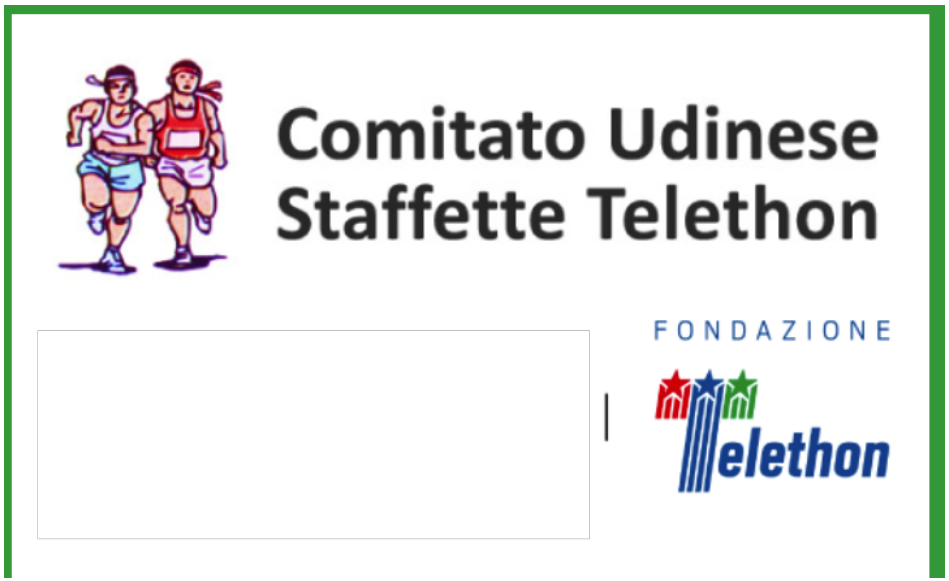
We want to define the predicate `x(riga,colonna,valore)`

```
x(1,3,1).
x(2,3,2). x(2,5,3). x(2,9,4).
x(3,4,5). x(3,7,6). x(3,9,7).
x(4,1,5). x(4,4,1). x(4,5,2).
x(5,2,7). x(5,8,2).
x(6,5,7). x(6,6,8). x(6,9,9).
x(7,1,8). x(7,3,7). x(7,6,9).
x(8,1,4). x(8,5,6). x(8,7,3).
x(9,7,5).
```

SUDOKU

```
coord(1..9).
val(1..9).

% Assign a pair (X,Y) to a subsquare
square(I,X,Y) :-
   coord(X), coord(Y), coord(I),
   I == (X-1) / 3 + 3*((Y-1) / 3) + 1.

% x is a function
1 { x(X,Y,N) : val(N) } 1   :- coord(X),coord(Y).

% x in injective on columns
1 { x(X,Y,N) : coord(X) } 1 :- coord(Y), val(N).

% x is injective on rows
1 { x(X,Y,N) : coord(Y) } 1 :- coord(X), val(N).

% x is injective in subsquares
1 { x(X,Y,N) : square(I,X,Y) } 1  :- val(N), coord(I).
```

TELETHON


Comitato Udinese
Staffette Telethon
FONDAZIONE
Telethon

TELETHON

```
slot(1..4). %%% cambia con 24
runner(X) :- can(X,_).
runner(X) :- ok(X,_).
available(X,T) :- can(X,T).
available(X,T) :- ok(X,T).

%% Biiezione runner/slot
1 { corrialle(X,T): slot(T) } 1 :- runner(X).
1 { corrialle(X,T): runner(X)} 1 :- slot(T).

%% Controllo disponibilita'
:- runner(X), slot(T), corrialle(X,T), not available(X,T).

%%% MINIMIZZIAMO GLI SLOT non OK
fastidio(F) :- F = #count{X : runner(X), slot(T), corrialle(X,T), not ok(X,T)}.
#minimize {F:fastidio(F)}.

#show corrialle/2.
#show fastidio/1.

%%% INPUT DATA, For each of the 24 player
%%% ok - good, can -maybe
can(ago,1). ok(ago,2). can(ago,3).
can(simone,1). can(simone,2). ok(simone,4).
ok(alberto,2). can(alberto,4).
can(fausto,1). can(fausto,4).
```

SOCCER TOURNAMENT

SOCCER TOURNAMENT

```
squadra(a;b;c;d; e;f;g;h).
giorno(1..7).   %% 1 lunedi', ..., 7 domenica
feriale(1..5).  %%
settimana(1..s).
orario(17;19;21).

slot(G,S,O)     :- giorno(G),  settimana(S), orario(O).
squadreord(X,Y) :- squadra(X), squadra(Y), X < Y.
```

SOCCER TOURNAMENT

```
testaserie(a;b;c;d).
testeserie(A,B) :- testaserie(A),testaserie(B), A < B.

% Every pair of teams plays each other exactly once
1 { gioca(X,Y,G,S,O): slot(G,S,O) } 1 :- squadreord(X,Y).
% For each time slot, there is at most one match
0 { gioca(X,Y,G,S,O): squadreord(X,Y)} 1 :- slot(G,S,O).
```

SOCCER TOURNAMENT

Auxiliary Predicates:

```
oggi(X,S,X,S)              :- giorno(X),settimana(S).
domani(X,S,X+1,S)          :- giorno(X),giorno(X+1),settimana(S).
domani(7,S,1,S+1)          :- settimana(S),settimana(S+1).
dopodomani(X,S,X+2,S)      :- giorno(X),giorno(X+2),settimana(S).
dopodomani(7,S,2,S+1)      :- settimana(S),settimana(S+1).
dopodomani(6,S,1,S+1)      :- settimana(S),settimana(S+1).
good(X1,S1,X2,S2) :-
      giorno(X1),giorno(X2),
      settimana(S1),settimana(S2),
      not oggi(X1,S1,X2,S2),
      not domani(X1,S1,X2,S2),
      not domani(X2,S2,X1,S1),
      not dopodomani(X1,S1,X2,S2),
      not dopodomani(X2,S2,X1,S1).
```

SOCCER TOURNAMENT

Constraints:

```
% At least three days between matches

:- squadreord(X,Y), squadreord(X,Z), Y < Z,
   slot(G1,S1,O1), slot(G2,S2,O2),
   gioca(X,Y,G1,S1,O1),gioca(X,Z,G2,S2,O2),
   not good(G1,S1,G2,S2).
:- squadreord(X,Y), squadreord(Z,X), Y != Z,
   slot(G1,S1,O1), slot(G2,S2,O2),
   gioca(X,Y,G1,S1,O1),gioca(Z,X,G2,S2,O2),
   not good(G1,S1,G2,S2).
:- squadreord(Y,X), squadreord(Z,X), Y < Z,
   slot(G1,S1,O1), slot(G2,S2,O2),
   gioca(Y,X,G1,S1,O1),gioca(Z,X,G2,S2,O2),
   not good(G1,S1,G2,S2).
```

SOCCER TOURNAMENT

Constraints:

```
% No matches Tue and Wed at 21.00 (there's champions league in TV)
:- squadreord(X,Y), settimana(S), gioca(X,Y,2,S,21).
:- squadreord(X,Y), settimana(S), gioca(X,Y,3,S,21).
% No matches Sat and Sun at 21.00 (there's the Serie A)
:- squadreord(X,Y), settimana(S), gioca(X,Y,6,S,21).
:- squadreord(X,Y), settimana(S), gioca(X,Y,7,S,21).

%%%% best teams don't fight each other in the first week
:- testeserie(X,Y), slot(G,S,O), gioca(X,Y,G,S,O), S < 2.
%%%% At most one match between best teams in the same day
:- testeserie(X1,X2), testeserie(X3,X4),
   slot(G,S,O1), slot(G,S,O2), O1 < O2,
   gioca(X1,X2,G,S,O1), gioca(X3,X4,G,S,O2).

%%% There are not working days without matches
slotoccupato(G,S,O) :-
     slot(G,S,O),squadreord(X,Y),gioca(X,Y,G,S,O).
giorno_occupato(G,S) :-
     slot(G,S,O), slotoccupato(G,S,O).
:- feriale(G), settimana(S), not giorno_occupato(G,S).
```

SOCCER TOURNAMENT

USING MINIMIZE

Constraints:

```
giorno_singolo(G,S) :-
   giorno(G),settimana(S), orario(O1), orario(O2), orario(O3),
   O1 != O2, O1 != O3, O2 < O3,
   slotoccupato(G,S,O1),
   not slotoccupato(G,S,O2),
   not slotoccupato(G,S,O3).

%%% If single match, it is at 19
:- giorno(G), settimana(S), giorno_singolo(G,S), slotoccupato(G,S,17).
:- giorno(G), settimana(S), giorno_singolo(G,S), slotoccupato(G,S,21).

%%% If two matches no hole inside
:- giorno(G), settimana(S), not giorno_singolo(G,S),
   slotoccupato(G,S,17), not slotoccupato(G,S,19), slotoccupato(G,S,21).

%%% MINIMZE THE DAYS WITH SINGLE MATCH
contagiornisingoli(N) :- N = #count{G,S : giorno_singolo(G,S)}.
#minimize {N:contagiornisingoli(N)}.
```

SOCCER TOURNAMENT

"Special" constraints:

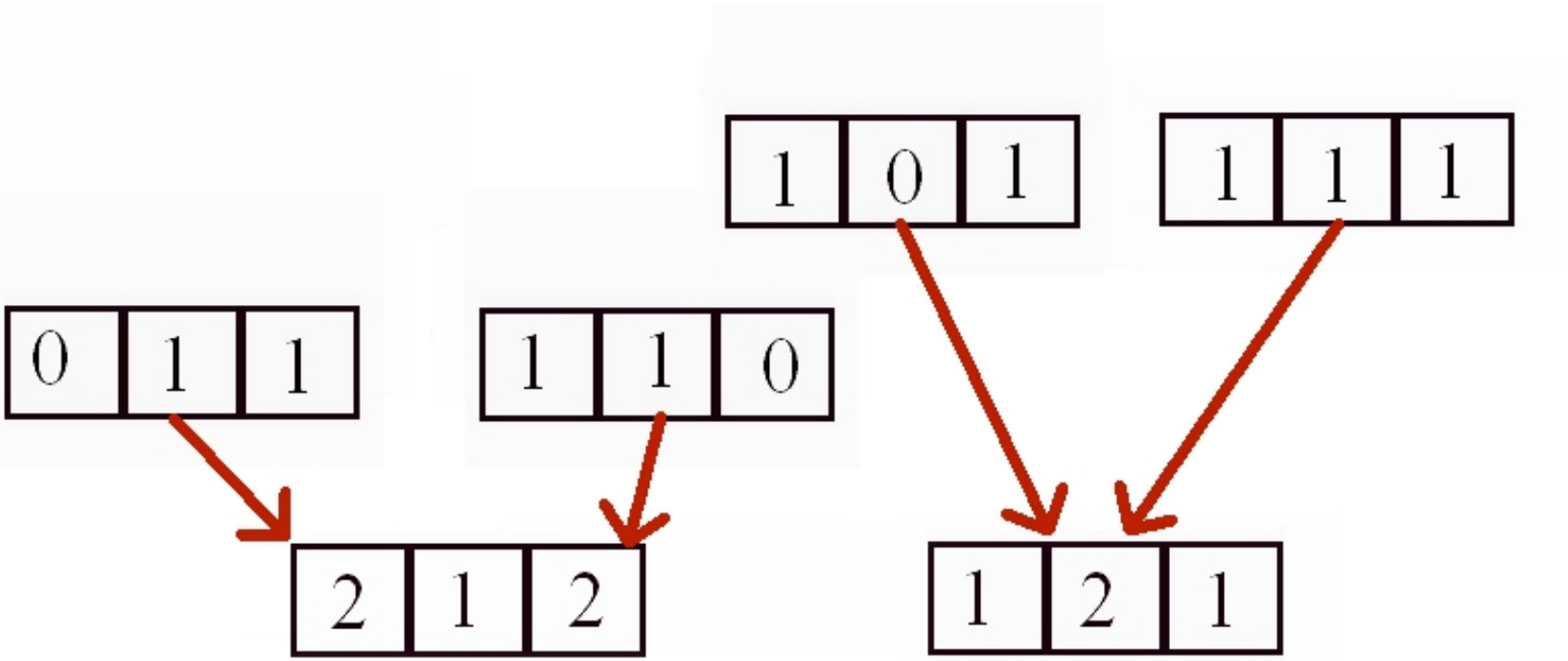
```
% The team "a" can't play on saturday
% (some players play basketball)
:- settimana(S), orario(O), squadra(Y), gioca(a,Y,6,S,O).
:- settimana(S), orario(O), squadra(Y), gioca(Y,a,6,S,O).
% The team "a" can't play on monday
% (some girlfriends close the shop)
:- settimana(S), orario(O), squadra(Y), gioca(b,Y,1,S,O).
:- settimana(S), orario(O), squadra(Y), gioca(Y,b,1,S,O).
% In the first and last day there must be at least 2 matches
:-  orario(O1), orario(O2), O1 < O2,
    not slotoccupato(1,1,O1), not slotoccupato(1,1,O2).
:-  orario(O1), orario(O2), O1 < O2,
    not slotoccupato(7,s,O1), not slotoccupato(7,s,O2).
```

Output predicates

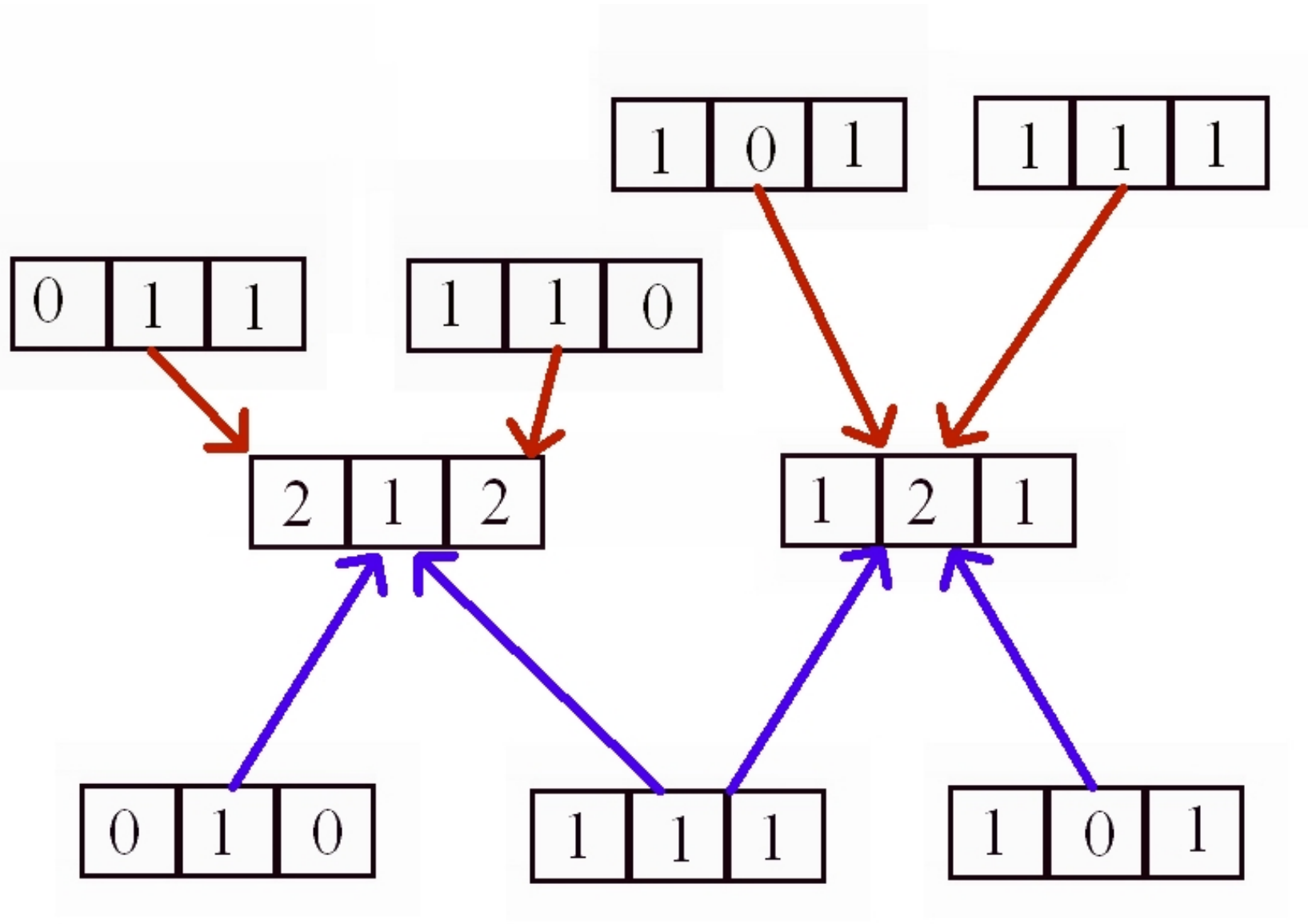
```
day(A,X1,X2,X3,X4,X5,X6) :-
    giorno(G),
    settimana(S),
    A = G + 7*(S-1),
    completamento(X1,X2,G,S,17),
    completamento(X3,X4,G,S,19),
    completamento(X5,X6,G,S,21).
completamento(X,Y, G,S,O) :-
    slot(G,S,O), squadreord(X,Y), gioca(X,Y,G,S,O).
completamento(nil,nil, G,S,O) :-
    slot(G,S,O), not slotoccupato(G,S,O).
```

HAPLOTYPE INFERENCE

HAPLOTYPE INFERENCE



HAPLOTYPE INFERENCE



HAPLOTYPE INFERENCE

- Let $H \subseteq \{0,1\}^n$ be a set of *haplotypes* and
- $G \subseteq \{0,1,2\}^n$ be a set of *genotypes*.
- Given $h_1, h_2 \in H$ and $g \in G$, $\{h_1, h_2\}$ explains g if and only if $|h_1| = |h_2| = |g|$ and $\forall i \in [1..n]$:

$$\begin{array}{lcl} g[i] \leq 1 & \longrightarrow & h_1[i] = h_2[i] = g[i] \\ g[i] = 2 & \longrightarrow & h_1[i] \neq h_2[i] \end{array}$$

- A set of haplotypes H explains a set of genotypes G if for all $g \in G$ there are $h_1, h_2 \in H$ such that $\{h_1, h_2\}$ explains g.
- Given a set of genotypes G and an integer k, the *haplotype inference problem (HIP)* by pure parsimony is the problem of finding a set H that explains G and such that $|H| = k$ (decision version: NP complete).

- m input genotypes ($1 \ldots m$)
- $2m$ potential haplotypes ($1 \ldots 2m$, not necessarily distinct)
- The i-th genotype g_i is explained by haplotypes h_{2i} and h_{2i-1}
- `geno(1..m).`, `site(1..n).` and `haplo(1..2m).`
- Facts: `g(i,j,k).` describes genotypes, where
  - i is the i-th genotype,
  - j is the position within the genotype and
  - k is the value in $\{0,1,2\}$
- `h(i,j).` describe haplotypes.
  It is in the model iff the value of the haplotype i at position j is 1

HAPLOTYPE INFERENCE

THE ASP MODELING

```
  % If g(i,j)=0 or 1 assign deterministically
  % h(2i) and h(2i-1).
(1)     h(2*I-1,J) :- g(I,J,1).
(2)     h(2*I,J)   :- g(I,J,1).
(3)     :- h(2*I-1,J), g(I,J,0).
(4)     :- h(2*I,J),   g(I,J,0).
  % If g(i,j)=2 then exactly one between
  % h(2i) and h(2i-1) is 1
(5)     h(2*I-1,J) :- g(I,J,2), not h(2*I,J).
(6)     h(2*I,J)   :- g(I,J,2), not h(2*I-1,J).
```

HAPLOTYPE INFERENCE

THE ASP MODELING

```
(7)    representative_haplo(A) :-
                haplo(A), not cover_someone(A).
(8)    cover_someone(B) :-
                haplo(A;B), A < B, samehaplo(A,B).
```

- Define the representatives.
- The lowest index haplotype of a set of equal ones is selected as representative.

HAPLOTYPE INFERENCE

THE ASP MODELING

```
(7)    representative_haplo(A) :-
                haplo(A), not cover_someone(A).
(8)    cover_someone(B) :-
                haplo(A;B), A < B, samehaplo(A,B).

(9)    { samehaplo(A,B) } :- haplo(A), haplo(B), A < B.
(10)   :- samehaplo(A,B), haplo(A;B), A < B,
              site(S), h(A,S), not h(B,S).
(11)   :- samehaplo(A,B), haplo(A;B), A < B,
              site(S), not h(A,S), h(B,S).
```

- We have a (possibly empty) set of `samehaplo`.
- Constraints: can't be `samehaplo` and a site S with $\neq$ values.
- `not(_,S)` for the case haplotype at site S is 0.
- Symmetry breaking.

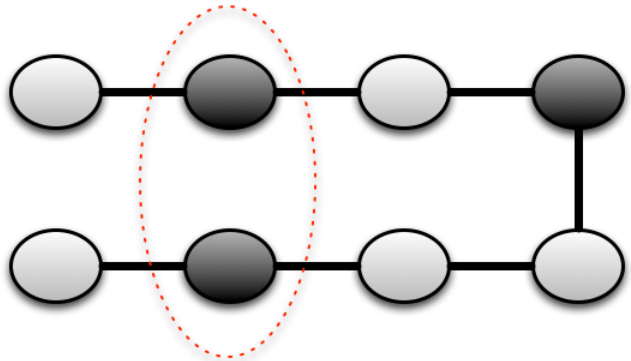
```
% Count the number of representative and minimize it
(12)   diff_haplo(N) :-
```

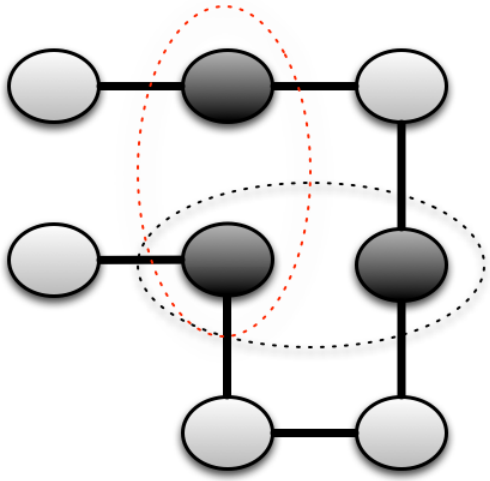
HAPLOTYPE INFERENCE

THE ASP MODELING

```
(7)     representative_haplo(A) :-
                haplo(A), not cover_someone(A).
(8)     cover_someone(B) :-
                haplo(A;B), A < B, samehaplo(A,B).

(9)     { samehaplo(A,B) } :- haplo(A), haplo(B), A < B.
(10)    :- samehaplo(A,B), haplo(A;B), A < B,
              site(S), h(A,S), not h(B,S).
(11)    :- samehaplo(A,B), haplo(A;B), A < B,
              site(S), not h(A,S), h(B,S).

  % Count the number of representative and minimize it
(12)    diff_haplo(N) :-
                N = #count{ A:representative_haplo(A) }.
(13)    #minimize{ N:diff_haplo(N)}.
```

- The input is a list S of amino acids $S = s_1, \ldots, s_n$,
- where $s_i \in \{h, p\}$
- Each s_i is placed on a 2D grid with integer coordinates
- Any pair of two amino acids can't occupy the same position
- If two amino acids are at distance 1, they are in contact





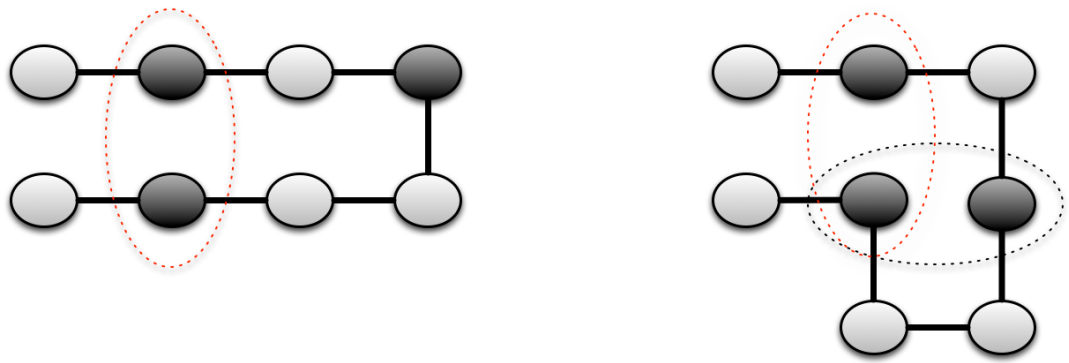
The protein structure prediction problem

Model

- A folding is a function $\omega : \{1, \ldots, n\} \longrightarrow \mathbb{N}^2$ where
- $\forall i \, \text{next}(\omega(i), \omega(i+1))$ and
- $\forall i, j \, (i \neq j \rightarrow \omega(i) \neq \omega(j))$
- $\text{next}(\langle X_1, Y_1 \rangle, \langle X_2, Y_2 \rangle) \Longleftrightarrow |X_1 - X_2| + |Y_1 - Y_2| = 1$.
- Find a folding that minimizes the (simplified) energy function:

$$E(S, \omega) = \sum_{\substack{1 \leq i \leq n-2 \\ i+2 \leq j \leq n}} \text{Pot}(s_i, s_j) \cdot \text{next}(\omega(i), \omega(j))$$

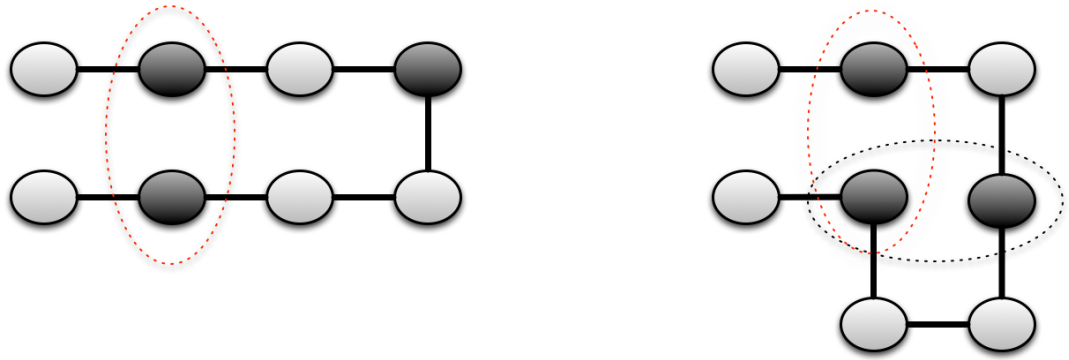
where $\text{Pot}(p, p) = \text{Pot}(h, p) = \text{Pot}(p, h) = 0$ and $\text{Pot}(h, h) = -1$.



- A folding is a function $\omega : \{1, \ldots, n\} \longrightarrow \mathbb{N}^2$ where
- $\forall i \, \text{next}(\omega(i), \omega(i+1))$ and
- $\forall i, j \, (i \neq j \rightarrow \omega(i) \neq \omega(j))$
- $\text{next}(\langle X_1, Y_1 \rangle, \langle X_2, Y_2 \rangle) \Longleftrightarrow |X_1 - X_2| + |Y_1 - Y_2| = 1$.
- Find a folding that minimizes the (simplified) energy function:

$$E(S, \omega) = \sum_{\substack{1 \leq i \leq n-2 \\ i+2 \leq j \leq n}} \text{Pot}(s_i, s_j) \cdot \text{next}(\omega(i), \omega(j))$$

where $\text{Pot}(p, p) = \text{Pot}(h, p) = \text{Pot}(p, h) = 0$ and $\text{Pot}(h, h) = -1$.



- The positions can be restricted to $[1 \ldots 2n] \times [1 \ldots 2n]$
- The first two amino acids are set to $\omega(1) = \langle n, n \rangle$ and $\omega(2) = \langle n, n+1 \rangle$
- Sequence is represented by n facts:

```
prot(1,p).  prot(2,h).  prot(3,p).  prot(4,h).
prot(5,p).  prot(6,p).  prot(7,h).  prot(8,p).
```

- Each placement is stored as `sol(i,X,Y).`

ASP Encoding

```
(1)   size(N)          :-  N = #count{ I:prot(I,J) }.
(2)   board(1..2*N)    :- size(N).
(3)   range(X,Y)       :- size(N), board(X;Y),
                        |X-N| < N , |Y-N| < N.
```

- Count the protein length and defines the board size
- `range` is loaded with all combinations from 1 to 2N

```
(4)   sol(1,N,N)       :- size(N).
(5)   sol(2,N,N+1)     :- size(N).

(6)   1 { sol(I,X,Y) : range(X,Y) } 1 :- prot(I,Amino).

(7)   :- prot(I1,A1), prot(I2,A2), I1<I2,
         sol(I1,X,Y), sol(I2,X,Y),  range(X,Y).
```

ASP Encoding

```
(1)   size(N)          :-  N = #count{ I:prot(I,J) }.
(2)   board(1..2*N)    :- size(N).
(3)   range(X,Y)       :- size(N), board(X;Y),
                       |X-N| < N , |Y-N| < N.

(4)   sol(1,N,N)       :- size(N).
(5)   sol(2,N,N+1)     :- size(N).
```

- Set the first two amino acids positions

```
(6)   1 { sol(I,X,Y) : range(X,Y) } 1 :- prot(I,Amino).

(7)   :- prot(I1,A1), prot(I2,A2), I1<I2,
         sol(I1,X,Y), sol(I2,X,Y),  range(X,Y).
```

ASP Encoding

```
(1)  size(N)          :-  N = #count{ I:prot(I,J) }.
(2)  board(1..2*N)    :- size(N).
(3)  range(X,Y)       :- size(N), board(X;Y),
                         |X-N| < N , |Y-N| < N.

(4)  sol(1,N,N)       :- size(N).
(5)  sol(2,N,N+1)     :- size(N).

(6)  1 { sol(I,X,Y) : range(X,Y) } 1 :- prot(I,Amino).
```

- Each amino acid *I* is associated to a unique position

```
(7)  :- prot(I1,A1), prot(I2,A2), I1<I2,
        sol(I1,X,Y), sol(I2,X,Y),  range(X,Y).
```

ASP Encoding

```
(1)  size(N)          :-  N = #count{ I:prot(I,J) }.
(2)  board(1..2*N)    :- size(N).
(3)  range(X,Y)       :- size(N), board(X;Y),
                      |X-N| < N , |Y-N| < N.

(4)  sol(1,N,N)       :- size(N).
(5)  sol(2,N,N+1)     :- size(N).

(6)  1 { sol(I,X,Y) : range(X,Y) } 1 :- prot(I,Amino).

(7)  :- prot(I1,A1), prot(I2,A2), I1<I2,
            sol(I1,X,Y), sol(I2,X,Y),  range(X,Y).
```

- Constraint: two distinct amino acids can't overlap

ASP Encoding

```
(8)   :- prot(I1,A1), prot(I2,A2),  I2>1,
                I1==I2-1, not next(I1,I2).
(9)    next(I1,I2) :- prot(I1,A1), prot(I2,A2), I1<I2,
                sol(I1,X1,Y1), sol(I2,X2,Y2),
                range(X1,Y1), range(X2,Y2),
                1 == |X2-X1| + |Y2-Y1|.
```

- Constraint: two consecutive amino acids in sequence are `next` in space

```
(10)  energy_pair(I1,I2) :- prot(I1,h), prot(I2,h),
                next(I1,I2), I1+2<I2,
                1==(I2-I1)  \ 2. %%% "\" is "mod"
(11)  energy(N) :- N = #count{(I1,I2):energy_pair(I1,I2)}.
(12)  #maximize { N:energy(N)}.
```

ASP Encoding

```
(8)   :- prot(I1,A1), prot(I2,A2),  I2>1,
                I1==I2-1, not next(I1,I2).
(9)    next(I1,I2) :- prot(I1,A1), prot(I2,A2), I1<I2,
                sol(I1,X1,Y1), sol(I2,X2,Y2),
                range(X1,Y1), range(X2,Y2),
                1 == |X2-X1| + |Y2-Y1|.

(10)  energy_pair(I1,I2) :- prot(I1,h), prot(I2,h),
                 next(I1,I2), I1+2<I2,
                 1==(I2-I1)  \ 2. %%% "\" is "mod"
(11)  energy(N) :- N = #count{(I1,I2):energy_pair(I1,I2)}.
(12)  #maximize { N:energy(N)}.
```

- If two amino acids of type *h* are `next`, count one contact (energy_pair)
- Optimization: at least one amino acid in sequence between the pair
- Optimization: only odd distances in space can contribute to a contact