Modelling with CP

Nadjib LAZAAR

LIRMM
(COCONUT team)

IMAGINA Masters course
24 02. 2015

Overview

- Formulation in CP
  - Vocabulary (variables, domains)
  - Constraints

Overview

- Formulation in CP
  - Vocabulary (variables, domains)
  - Constraints
- ReFormulation-s in CP
  - Auxiliary variables / Channeling constraints
  - Redundant constraints
  - Global constraints
  - Symmetry breaking

Motivations

Numerous CP modeling languages and platforms have been developed (OPL, ZINC, GECODE, CHOCO…) for solving combinatorial problems that arise in optimization, planning, scheduling…

Motivations

Numerous CP modeling languages and platforms have been developed (OPL, ZINC, GECODE, CHOCO…) for solving combinatorial problems that arise in optimization, planning, scheduling…

Assembly line management for car production
(Sanlaville, ROADEF05)

Motivations

Numerous CP modeling languages and platforms have been developed (OPL, ZINC, GECODE, CHOCO…) for solving combinatorial problems that arise in optimization, planning, scheduling…

Assembly line management for car production
(Sanlaville, ROADEF05)

Recommender System
(McSherry & Aha, IJCAI07; Felfernig & Burke, ICEC08)
Continuous Auctions
(Lozano et al., CP10)
Portfolio Designs
(Flener et al., Constraints07)

Motivations

Numerous CP modeling languages and platforms have been developed (OPL, ZINC, GECODE, CHOCO…) for solving combinatorial problems that arise in optimization, planning, scheduling…

Assembly line management for car production
(Sanlaville, ROADEF05)

Recommender System
(McSherry & Aha, IJCAI07; Felfernig & Burke, ICEC08)
Continuous Auctions
(Lozano et al., CP10)
Portfolio Designs
(Flener et al., Constraints07)

Air traffic management
(Flener et al., JATM07)
Railway management
(Chiu et al., Constraints02; Rodriguez & Kermad, Comprail98)

Constraint Programming

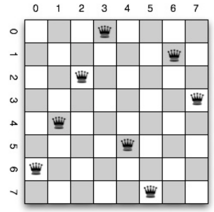
- Constraint Satisfaction Problem (CSP)
  - Vocabulary (X,D)=(variables, domains)
  - Set of Constraints C

Constraint Programming

- Constraint Satisfaction Problem (CSP)
  - Vocabulary (X,D)=(variables, domains)
  - Set of Constraints C

Specification (n-queens):

The n-queens problem asks how to place n queens on an n x n chess board so that none of them can hit any other in one move.

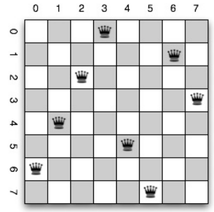


Constraint Programming

- Constraint Satisfaction Problem (CSP)
  - Vocabulary (X,D)=(variables, domains)
  - Set of Constraints C

Specification (n-queens):

The n-queens problem asks how to place n queens on an n x n chess board so that none of them can hit any other in one move.



CSP:

Variables: $X = \{X_i | i \in [1,n]\}$

Domains: $X_i \in X, \ D(X_i) = \{1, 2, \ldots, n\}$

Constraints:

- $C_{lines} = \{X_i \neq X_j \mid i,j \in [1,n], i \neq j\}$
- $C_{diag1} = \{X_i \neq X_j + j - i \mid i,j \in [1,n], i \neq j\}$
- $C_{diag2} = \{X_i \neq X_j + i - j \mid i,j \in [1,n], i \neq j\}$

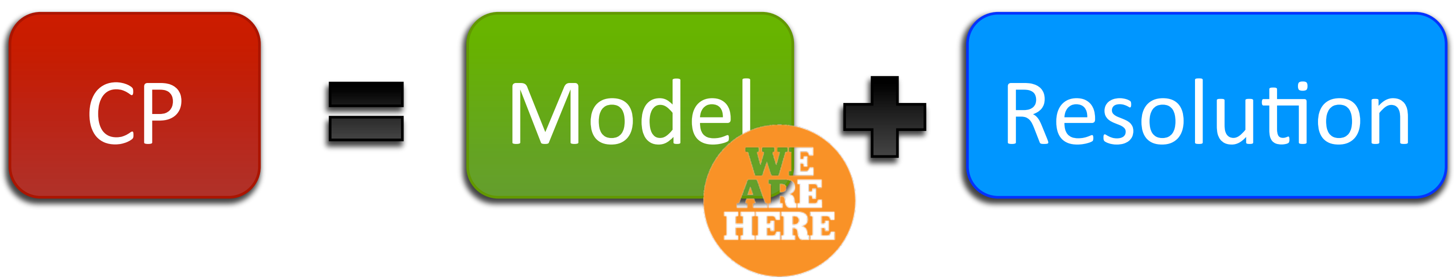
Constraint Programming

- Constraint Satisfaction Problem (CSP)
  - Vocabulary (X,D)=(variables, domains)
  - Set of Constraints C

CP = Model + Resolution

Constraint Programming

- Constraint Satisfaction Problem (CSP)
  - Vocabulary (X,D)=(variables, domains)
  - Set of Constraints C



Problem = conjunction of sub-Problems

- In CP a problem can be viewed as a conjunction of sub-problems that we are able to solve
- A sub-problem can be trivial ($x \neq y$), or complex (cumulative resources of limited capacities)

    → sub-problem = constraint

Constraints

- Predefined constraints: arithmetic ($x < y$, $x = y + z$, $|x-y| > k$, alldiff, cardinality, sequence …)

Constraints

- Predefined constraints: arithmetic ($x < y$, $x = y + z$, $|x-y| > k$, alldiff, cardinality, sequence …)
- Constraints given in extension by the list of allowed (or forbidden) combinations of values

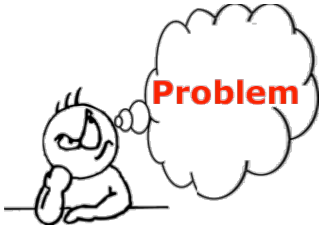
Constraints

- Predefined constraints: arithmetic ($x < y$, $x = y + z$, $|x-y| > k$, alldiff, cardinality, sequence …)
- Constraints given in extension by the list of allowed (or forbidden) combinations of values
- User-defined constraints: any algorithm can be encapsulated

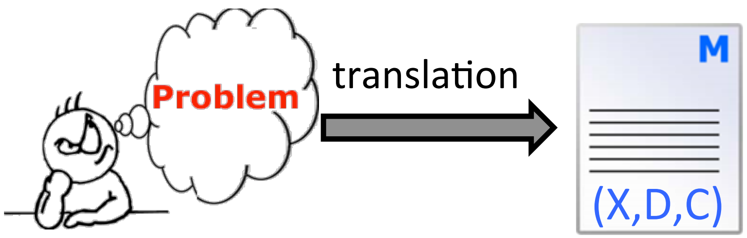
Constraints

- Predefined constraints: arithmetic ($x < y$, $x = y + z$, $|x-y| > k$, alldiff, cardinality, sequence …)
- Constraints given in extension by the list of allowed (or forbidden) combinations of values
- User-defined constraints: any algorithm can be encapsulated
- Logical combination of constraints using OR, AND, NOT, XOR operators.

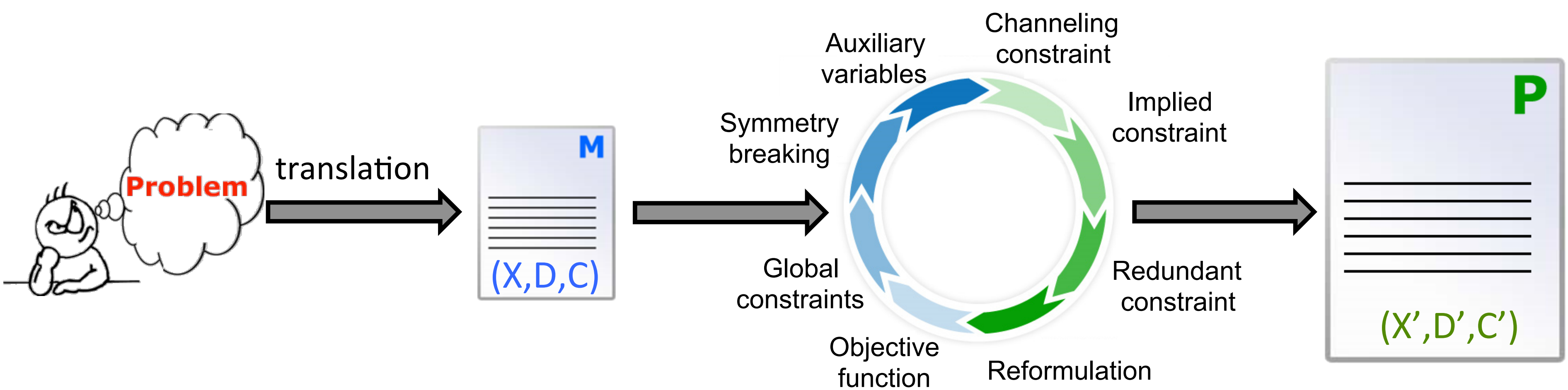
Modelling in CP



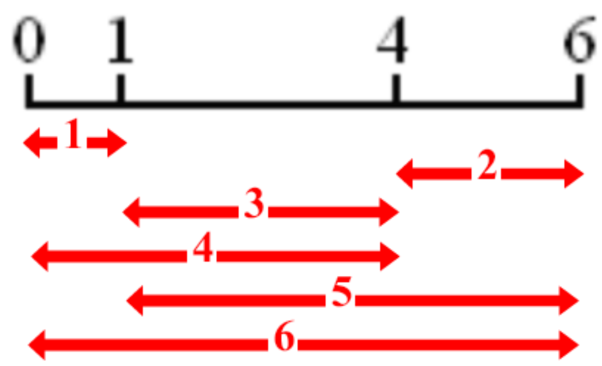
Modelling in CP



Modelling in CP



Golomb Rulers Example



Optimal Golomb Rulers (OGR-25) — *Completed 25 October 2008*[(after 3006 days)

[distributed.net, wiki]

Golomb Rulers Example

M

```
using CP;

int m=...;

dvar int x[1..m] in 0..m*m;

minimize x[m];

subject to
{
(1)   forall (i in 1..m-1)
         x[i] < x[i+1];
(2)   forall (i in 1..m, j in 1..m,
           k in 1..m, l in 1..m:
           (i < j, k < l))
         x[j] - x[i] != x[l] - x[k];
}
```

Golomb Rulers Example

M

```
using CP;

int m=...;

dvar int x[1..m] in 0..m*m;

minimize x[m];

subject to
{
(1)   forall (i in 1..m-1)
         x[i] < x[i+1];
(2)   forall (i in 1..m, j in 1..m,
            k in 1..m, l in 1..m:
            (i < j, k < l))
         x[j] - x[i] != x[l] - x[k];
}
```

P

```
using CP;

int m=...;
tuple indexerTuple {int i;
              int j;}
{indexerTuple} indexes1 = {<i, j> | ordered i,j in 1..m};
{indexerTuple} indexes2 = {<i, j> | ordered i,j in 1..m div 2};

dvar int x[1..m] in 0..m*m;
dvar int d[indexes1];

minimize d[1,m];

subject to {
(1) x[1]==0;
(2) forall (i in 1..m-1)   x[i] < x[i+1];
(3) forall(ind in indexes1)    d[ind] == x[ind.i]-x[ind.j];
(4) x[m] >= (m * (m - 1)) / 2;
(5) x[2] <= x[m]-x[m-1];
(6) allDifferent(all(ind in indexes2) d[ind]);
(7) forall(ind1,ind2,ind3:(ind1.i==ind2.i)&&(ind2.j==ind3.i)&&
      (ind1.j==ind3.j)&&(ind1.i<ind2.j<ind1.j))
      d[ind1] == d[ind2]+d[ind3];
(8) forall(ind1,ind2,ind3,ind4 in indexes2 :
      (ind1.i==ind2.i)&&(ind1.j==ind3.j)&&(ind2.j==ind4.j)
      &&(ind3.i==ind4.i)&&(ind1.i<m-1)&&(3<ind1.j<m+1)
            &&(2<ind2.j<m)&&(1<ind3.i<m-1)
            &&(ind1.i<ind3.i<ind2.j< ind1.j))
d[ind1] == (d[ind2]+d[ind3]-d[ind4]);
(9) forall(ind in indexes2, k in 1..m div 2)
      (x[ind.i+1]==x[ind.i]+k) => (x[ind.j+1]!=x[ind.j]+k);
}
```

Auxiliary variables / channeling constraints

- Auxiliary variables are introduced into a model to facilitate the expression of a sub-problem or to introduce better constraints (in terms of propagation)

Auxiliary variables / channeling constraints

- Auxiliary variables are introduced into a model to facilitate the expression of a sub-problem or to introduce better constraints (in terms of propagation)

M

```
dvar int x,y,z;

subject to {

x-y != x-z;
x-y != y-z;
x-z != y-z;
}
```

Auxiliary variables / channeling constraints

- Auxiliary variables are introduced into a model to facilitate the expression of a sub-problem or to introduce better constraints (in terms of propagation)

M

```
dvar int x,y,z;

subject to {

x-y != x-z;
x-y != y-z;
x-z != y-z;
}
```

Equivalent

P

```
dvar int x,y,z;

dvar int d[1..3];

subject to {

x-y == d[1];
x-z == d[2];
y-z == d[3];
AllDifferent( d );

}
```

Golomb Rulers Example

M

```
using CP;

int m=...;

dvar int x[1..m] in 0..m*m;

minimize x[m];

subject to
{
(1)   forall (i in 1..m-1)
         x[i] < x[i+1];
(2)   forall (i in 1..m, j in 1..m,
           k in 1..m, l in 1..m:
           (i < j, k < l))
         x[j] - x[i] != x[l] - x[k];
}
```

P

```
using CP;

int m=...;
tuple indexerTuple {int i;
              int j;}
{indexerTuple} indexes1 = {<i, j> | ordered i,j in 1..m};
{indexerTuple} indexes2 = {<i, j> | ordered i,j in 1..m div 2};

dvar int x[1..m] in 0..m*m;
dvar int d[indexes1];

minimize d[1,m];

subject to {
(1) x[1]==0;
(2) forall (i in 1..m-1)   x[i] < x[i+1];
(3) forall(ind in indexes1)    d[ind] == x[ind.i]-x[ind.j];
(4) x[m] >= (m * (m - 1)) / 2;
(5) x[2] <= x[m]-x[m-1];
(6) allDifferent(all(ind in indexes2) d[ind]);
(7) forall(ind1,ind2,ind3:(ind1.i==ind2.i)&&(ind2.j==ind3.i)&&
      (ind1.j==ind3.j)&&(ind1.i<ind2.j<ind1.j))
      d[ind1] == d[ind2]+d[ind3];
(8) forall(ind1,ind2,ind3,ind4 in indexes2 :
      (ind1.i==ind2.i)&&(ind1.j==ind3.j)&&(ind2.j==ind4.j)
      &&(ind3.i==ind4.i)&&(ind1.i<m-1)&&(3<ind1.j<m+1)
            &&(2<ind2.j<m)&&(1<ind3.i<m-1)
            &&(ind1.i<ind3.i<ind2.j< ind1.j))
d[ind1] == (d[ind2]+d[ind3]-d[ind4]);
(9) forall(ind in indexes2, k in 1..m div 2)
      (x[ind.i+1]==x[ind.i]+k) => (x[ind.j+1]!=x[ind.j]+k);
}
```

Redundant constraints

- Redundant (implied) constraints are constraints implied by the constraints defining the problem

Redundant constraints

- Redundant (implied) constraints are constraints implied by the constraints defining the problem

```
dvar int x,y,z;

subject to {

x < y;
y < z;

}
```

Redundant constraints

- Redundant (implied) constraints are constraints implied by the constraints defining the problem

M

```
dvar int x,y,z;

subject to {

x < y;
y < z;

}
```

Equivalent

P

```
dvar int x,y,z;

subject to {

x < y;
y < z;
x < z;

Lex(x,y,z);

}
```

Golomb Rulers Example

M

```
using CP;

int m=...;

dvar int x[1..m] in 0..m*m;

minimize x[m];

subject to
{
(1)   forall (i in 1..m-1)
         x[i] < x[i+1];
(2)   forall (i in 1..m, j in 1..m,
            k in 1..m, l in 1..m:
            (i < j, k < l))
         x[j] - x[i] != x[l] - x[k];
}
```

P

```
using CP;

int m=...;
tuple indexerTuple {int i;
                    int j;}
{indexerTuple} indexes1 = {<i, j> | ordered i,j in 1..m};
{indexerTuple} indexes2 = {<i, j> | ordered i,j in 1..m div 2};

dvar int x[1..m] in 0..m*m;
dvar int d[indexes1];

minimize d[1,m];

subject to {
(1) x[1]==0;
(2) forall (i in 1..m-1)   x[i] < x[i+1];
(3) forall(ind in indexes1)    d[ind] == x[ind.i]-x[ind.j];
(4) x[m] >= (m * (m - 1)) / 2;
(5) x[2] <= x[m]-x[m-1];
(6) allDifferent(all(ind in indexes2) d[ind]);
(7) forall(ind1,ind2,ind3:(ind1.i==ind2.i)&&(ind2.j==ind3.i)&&
      (ind1.j==ind3.j)&&(ind1.i<ind2.j<ind1.j))
      d[ind1] == d[ind2]+d[ind3];
(8) forall(ind1,ind2,ind3,ind4 in indexes2 :
      (ind1.i==ind2.i)&&(ind1.j==ind3.j)&&(ind2.j==ind4.j)
      &&(ind3.i==ind4.i)&&(ind1.i<m-1)&&(3<ind1.j<m+1)
            &&(2<ind2.j<m)&&(1<ind3.i<m-1)
            &&(ind1.i<ind3.i<ind2.j< ind1.j))
d[ind1] == (d[ind2]+d[ind3]-d[ind4]);
(9) forall(ind in indexes2, k in 1..m div 2)
      (x[ind.i+1]==x[ind.i]+k) => (x[ind.j+1]!=x[ind.j]+k);
}
```

Global Constraints

- Global constraints capture a relation between a non-fixed number of variables

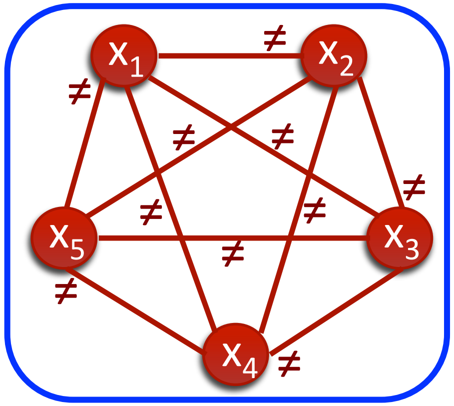
Global Constraints

- Global constraints capture a relation between a non-fixed number of variables
- Why global constraints?
  - Expressiveness (semantic globality)
  - Effectiveness of propagation (operational globality)
  - Efficiency of propagation (algorithmic globality)

Global Constraints

- Global constraints capture a relation between a non-fixed number of variables
- Why global constraints?
  - Expressiveness (semantic globality)
  - Effectiveness of propagation (operational globality)
  - Efficiency of propagation (algorithmic globality)

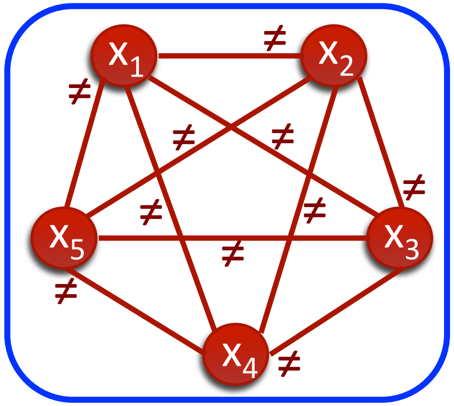
allDifferent



Global Constraints

- Global constraints capture a relation between a non-fixed number of variables
- Why global constraints?
  - Expressiveness (semantic globality)
  - Effectiveness of propagation (operational globality)
  - Efficiency of propagation (algorithmic globality)

allDifferent



- More than 423 global constraints in the catalog (http://sofdem.github.io/gccat/)

Golomb Rulers Example

M

```
using CP;

int m=...;

dvar int x[1..m] in 0..m*m;

minimize x[m];

subject to
{
(1)   forall (i in 1..m-1)
         x[i] < x[i+1];
(2)   forall (i in 1..m, j in 1..m,
            k in 1..m, l in 1..m:
            (i < j, k < l))
         x[j] - x[i] != x[l] - x[k];
}
```

P

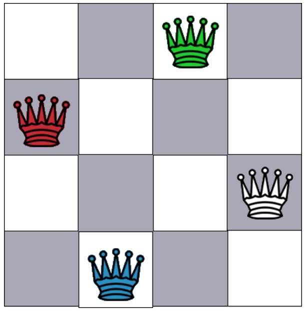
```
using CP;

int m=...;
tuple indexerTuple {int i;
              int j;}
{indexerTuple} indexes1 = {<i, j> | ordered i,j in 1..m};
{indexerTuple} indexes2 = {<i, j> | ordered i,j in 1..m div 2};

dvar int x[1..m] in 0..m*m;
dvar int d[indexes1];

minimize d[1,m];

subject to {
(1) x[1]==0;
(2) forall (i in 1..m-1)   x[i] < x[i+1];
(3) forall(ind in indexes1)    d[ind] == x[ind.i]-x[ind.j];
(4) x[m] >= (m * (m - 1)) / 2;
(5) x[2] <= x[m]-x[m-1];
(6) allDifferent(all(ind in indexes2) d[ind]);
(7) forall(ind1,ind2,ind3:(ind1.i==ind2.i)&&(ind2.j==ind3.i)&&
      (ind1.j==ind3.j)&&(ind1.i<ind2.j<ind1.j))
      d[ind1] == d[ind2]+d[ind3];
(8) forall(ind1,ind2,ind3,ind4 in indexes2 :
      (ind1.i==ind2.i)&&(ind1.j==ind3.j)&&(ind2.j==ind4.j)
      &&(ind3.i==ind4.i)&&(ind1.i<m-1)&&(3<ind1.j<m+1)
            &&(2<ind2.j<m)&&(1<ind3.i<m-1)
            &&(ind1.i<ind3.i<ind2.j< ind1.j))
d[ind1] == (d[ind2]+d[ind3]-d[ind4]);
(9) forall(ind in indexes2, k in 1..m div 2)
      (x[ind.i+1]==x[ind.i]+k) => (x[ind.j+1]!=x[ind.j]+k);
}
```

Symmetry breaking

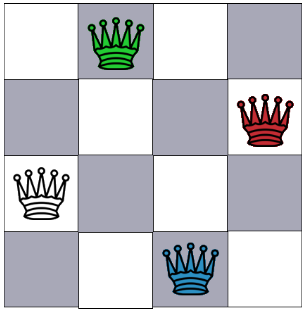
- A symmetry in CP is a permutation of values or variables (or both) that preserves (non-) solutions

Symmetry breaking

- A symmetry in CP is a permutation of values or variables (or both) that preserves (non-) solutions
- Symmetry on values:



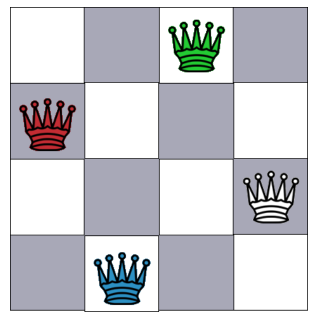
S=[2 , 4 , 1 , 3]



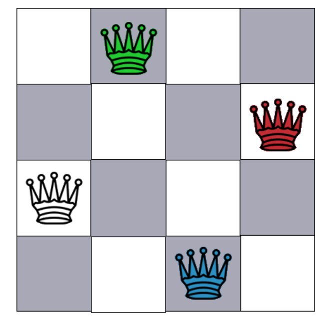
S’=[3 , 1 , 4 , 2]

Symmetry breaking

- A symmetry in CP is a permutation of values or variables (or both) that preserves (non-) solutions
- Symmetry on values:

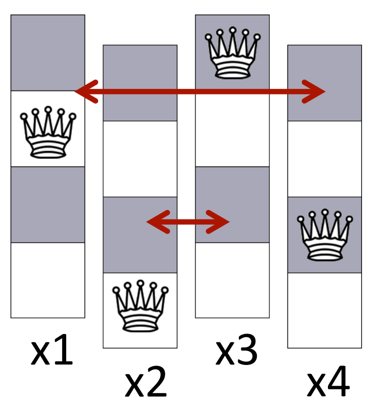


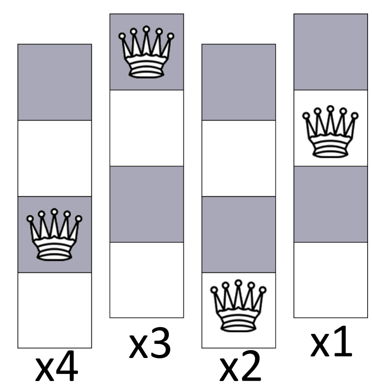
S=[2 , 4 , 1 , 3]



S’=[3 , 1 , 4 , 2]

- Symmetry on variables:





Golomb Rulers Example

M

```
using CP;

int m=...;

dvar int x[1..m] in 0..m*m;

minimize x[m];

subject to
{
(1)   forall (i in 1..m-1)
         x[i] < x[i+1];
(2)   forall (i in 1..m, j in 1..m,
           k in 1..m, l in 1..m:
           (i < j, k < l))
         x[j] - x[i] != x[l] - x[k];
}
```

P

```
using CP;

int m=...;
tuple indexerTuple {int i;
              int j;}
{indexerTuple} indexes1 = {<i, j> | ordered i,j in 1..m};
{indexerTuple} indexes2 = {<i, j> | ordered i,j in 1..m div 2};

dvar int x[1..m] in 0..m*m;
dvar int d[indexes1];

minimize d[1,m];

subject to {
(1) x[1]==0;
(2) forall (i in 1..m-1)   x[i] < x[i+1];
(3) forall(ind in indexes1)    d[ind] == x[ind.i]-x[ind.j];
(4) x[m] >= (m * (m - 1)) / 2;
(5) x[2] <= x[m]-x[m-1];
(6) allDifferent(all(ind in indexes2) d[ind]);
(7) forall(ind1,ind2,ind3:(ind1.i==ind2.i)&&(ind2.j==ind3.i)&&
      (ind1.j==ind3.j)&&(ind1.i<ind2.j<ind1.j))
      d[ind1] == d[ind2]+d[ind3];
(8) forall(ind1,ind2,ind3,ind4 in indexes2 :
      (ind1.i==ind2.i)&&(ind1.j==ind3.j)&&(ind2.j==ind4.j)
      &&(ind3.i==ind4.i)&&(ind1.i<m-1)&&(3<ind1.j<m+1)
            &&(2<ind2.j<m)&&(1<ind3.i<m-1)
            &&(ind1.i<ind3.i<ind2.j< ind1.j))
d[ind1] == (d[ind2]+d[ind3]-d[ind4]);
(9) forall(ind in indexes2, k in 1..m div 2)
      (x[ind.i+1]==x[ind.i]+k) => (x[ind.j+1]!=x[ind.j]+k);
}
```