Proving Symmetries by Model Transformation

C. Mears T. Niven

Faculty of IT
Monash University

Workshop on Constraint Modelling and Reformulation, 2010

Symmetries In CSP Instances

- Symmetries can be used to improve CSP solving.
- It is good to know when your CSP has symmetries.

Finding Symmetries in CSP Instances

- There are many ways to find symmetries in CSP instances:
  - Try swapping variables around in the constraints.
  - Turn the CSP into a graph (with varying detail) and find the graph's symmetries.
  - Find all the solutions!
- The more accurate methods tend to be too slow for real-sized instances.

Symmetries in CSP Models Instead

- Instead of looking at big instances, examine the *model* itself.
- Symmetries in the model apply to all instances.
- Find once, use often.

Symmetry Detection Framework

1. Find symmetries of small instances.
2. Generalise those symmetries to the model.
3. Gather the most promising symmetries.

4. **Prove that the symmetries hold on the model.**

Symmetry Detection Framework

1. Find symmetries of small instances.
2. Generalise those symmetries to the model.
3. Gather the most promising symmetries.

4. **Prove that the symmetries hold on the model.**

CSP Models

What is a model?

A CSP Model: MiniZinc

```
% Latin Square
int: size;
set of int: range = 1..size;

% Decision variables
array[range, range, range] of var 0..1: x;

% Constraints
constraint forall (i, j in range)
              (sum (k in range) (x[i,j,k]) = 1);
constraint forall (i, k in range)
              (sum (j in range) (x[i,j,k]) = 1);
constraint forall (j, k in range)
              (sum (i in range) (x[i,j,k]) = 1);
```

A CSP Model: MiniZinc

```
% N-queens
int : n;
set of int : rg = 1..n;

array[rg,rg] of var 0..1 : x;

constraint forall (i in rg)
           (sum (j in rg) (x[i,j]) = 1);
constraint forall (j in rg)
           (sum (i in rg) (x[i,j]) = 1);
constraint forall (k in 3..2*n-1)
           (sum (i,j in rg where i+j=k) (x[i,j]) <= 1);
constraint forall (k in 2-n..n-2)
           (sum (i,j in rg where i-j=k) (x[i,j]) <= 1);
```

Our Method

Given a potential symmetry σ:

1. Apply σ to each constraint $c \in C$,
2. Check if $\sigma(c)$ is in C.

If all $\sigma(c)$ are in C, then σ is a symmetry.

Applying a Symmetry

- Symmetries that manipulate indices.
- Examples:
  - Dimensions swap: $x[i, j] \Mapsto x[j, i]$.
  - Indices inverted: $x[i] \Mapsto x[N - i + 1]$.
  - Values inverted: $x[i] \Mapsto N - x[i] + 1$.
  - Arbitrary index permutation: $x[i] \Mapsto x[\varphi(i)]$.
- Find each $x[i_1, \cdots]$ occurrence and replace it.

Applying a Symmetry (examples)

```
aa(X, t([I,J,K]) <=> aa(X, t([J,I,K]))).
```

```
constraint forall (i, j in range)
        (sum (k in range) (x[i,j,k]) = 1);
```

```
constraint forall (i, j in range)
        (sum (k in range) (x[j,i,k]) = 1);
```

Applying a Symmetry (examples)

```
aa(X, t([I|R])) <=> aa(X, t([U-I+L|R])).
```

(where L and U are the lower and upper bounds of the index.)

```
constraint forall (i, j in range)
       (sum (k in range) (x[i,j,k]) = 1);
```

```
constraint forall (i, j in range)
       (sum (k in range) (x[n-i+1,j,k]) = 1);
```

Checking $\sigma(c) \in C$.

- For each $\sigma(c)\ldots$
- $\ldots$ is there a $c' \in C$ such that $c' \equiv \sigma(c)$?

- Probably not.

Normalisation

```
forall(i,j in range) (sum (k in range) (x[i,j,k]) = 1);
forall(i,k in range) (sum (j in range) (x[i,j,k]) = 1);
forall(j,k in range) (sum (i in range) (x[i,j,k]) = 1);
```

```
forall(i,j in range) (sum (k in range) (x[j,i,k]) = 1);
forall(i,k in range) (sum (j in range) (x[j,i,k]) = 1);
forall(j,k in range) (sum (i in range) (x[j,i,k]) = 1);
```

Normalisation (1)

```
forall (i, j in range)
       (sum (k in range) (x[i,j,k]) = 1);
```

```
forall (i, j in range)
       (sum (k in range) (x[j,i,k]) = 1);
```

Normalisation (1)

```
forall (i, j in range)
       (sum (k in range) (x[i,j,k]) = 1);
```

```
forall (j, i in range)
       (sum (k in range) (x[i,j,k]) = 1);
```

Normalisation (1)

Rule: put generators in alphabetical order.

```
gen('$cc'(decl(int,Var1,no,VarKind1,A1),
          gen('$cc'(decl(int,Var2,no,VarKind2,A2),Rest)))) <=>
        Var1 `$>` Var2 |
    gen('$cc'(decl(int,Var2,no,VarKind2,A2),
              gen('$cc'(decl(int,Var1,no,VarKind1,A1),Rest)))).
```

Normalisation (1)

```
forall (i, j in range)
       (sum (k in range) (x[i,j,k]) = 1);
```

```
forall (i, j in range)
       (sum (k in range) (x[i,j,k]) = 1);
```

Normalisation (2)

```
forall (i, j in range)
       (sum (k in range) (x[i,j,k]) = 1);
```

```
forall (i, j in range)
       (sum (k in range) (x[n-i+1,j,k]) = 1);
```

Normalisation (2)

Rule: make array indices single variables.

Normalisation (2)

```
forall (i, j in range)
       (sum (k in range) (x[i,j,k]) = 1);

            a = n-i+1   ;   i = n-a+1

forall (i, j in range)
       (sum (k in range) (x[n-i+1,j,k]) = 1);
```

Normalisation (2)

```
forall (i, j in range)
       (sum (k in range) (x[i,j,k]) = 1);
```

```
forall (n-a+1, j in range)
       (sum (k in range) (x[n-(n-a+1)+1,j,k]) = 1);
```

Normalisation (2)

```
forall (i, j in range)
       (sum (k in range) (x[i,j,k]) = 1);
```

```
forall (n-a+1, j in range)
       (sum (k in range) (x[a,j,k]) = 1);
```

Normalisation (2)

```
forall (i, j in range)
       (sum (k in range) (x[i,j,k]) = 1);
```

```
forall (n-a+1, j in range)
       (sum (k in range) (x[a,j,k]) = 1);
```

Normalisation (2)

Rule: $(U - x + L) \in L..U \quad \mapsto \quad x \in L..U.$

```
decl(int, U-X+L, gen_var(L..U), VK, Ann) <=>
    decl(int, X, gen_var(L..U), VK, Ann).
```

Normalisation (2)

```
forall (i, j in range)
       (sum (k in range) (x[i,j,k]) = 1);
```

```
forall (a, j in range)
       (sum (k in range) (x[a,j,k]) = 1);
```

Normalisation (3)

Other rules:

```
X-Y <=> X+(-Y).
-(X+Y) <=> -(X) + -(Y).
-(-(X)) <=> X.
X+(-(X)) <=> term(X) | i(0).
i(0)+X <=> X.
```

Normalisation (3)

Other rules:

```
permutation(P,permutation(inverse(P),X)) <=> X.
permutation(inverse(P),permutation(P,X)) <=> X.

alldifferent(permutation(P,X)) <=>
    alldifferent(X).

card(permutation(P,X)) <=>
    card(X).

permutation(P,X) != permutation(P,Y) <=> X != Y.
```

Results

- Problems:
  - Latin square
  - Steiner Triples
  - Balanced Incomplete Block Design
  - Social Golfers
  - N-queens
- Succeeds on most of the symmetries.

Where It Fails

```
forall (k in 3..2*n-1)
  (sum (i,j in rg where i+j=k) (x[i,j]) <= 1);
forall (k in 2-n..n-2)
  (sum (i,j in rg where i-j=k) (x[i,j]) <= 1);
```

Where It Fails

```
forall (k in 3..2*n-1)
  (sum (i,j in rg where i+j=k) (x[i,j]) <= 1);
forall (k in 2-n..n-2)
  (sum (i,j in rg where i-j=k) (x[i,j]) <= 1);
```

```
forall (k in 3..2*n-1)
  (sum (i,j in rg where i+j=k) (x[n-i+1,j]) <= 1);
forall (k in 2-n..n-2)
  (sum (i,j in rg where i-j=k) (x[n-i+1,j]) <= 1);
```

Where It Fails

```
forall (k in 3..2*n-1)
   (sum (i,j in rg where i+j=k) (x[i,j]) <= 1);
forall (k in 2-n..n-2)
   (sum (i,j in rg where i-j=k) (x[i,j]) <= 1);
```

```
forall (k in 3..2*n-1)
   (sum (n-a+1,j in rg where n-a+1+j=k) (x[a,j]) <= 1);
forall (k in 2-n..n-2)
   (sum (n-a+1,j in rg where n-a+1-j=k) (x[a,j]) <= 1);
```

Where It Fails

```
forall (k in 3..2*n-1)
   (sum (i,j in rg where i+j=k) (x[i,j]) <= 1);
forall (k in 2-n..n-2)
   (sum (i,j in rg where i-j=k) (x[i,j]) <= 1);
```

```
forall (k in 3..2*n-1)
   (sum (a,j in rg where n-a+1+j=k) (x[a,j]) <= 1);
forall (k in 2-n..n-2)
   (sum (a,j in rg where n-a+1-j=k) (x[a,j]) <= 1);
```

Future Work

- Normalise pairs of constraints *mutually*.
  - (mostly done!)
- More flexibility/robustness.
- Apply more symmetries.

Thanks!

Questions?