Proving Symmetries by Model Transformation

C. Mears[1], T. Niven[1], M. Jackson[2], and M. Wallace[1]

{Chris.Mears,Todd.Niven,Mark.Wallace}@monash.edu,
M.G.Jackson@latrobe.edu.au

[1] Faculty of IT, Monash University, Australia
[2] Department of Mathematics, La Trobe University, Australia

A MiniZinc Models

Latin Square (integer).

```
int : n;
array[1..n,1..n] of var 1..n: x;

constraint forall (i in 1..n) (alldifferent (j in 1..n) (x[i,j]));
constraint forall (j in 1..n) (alldifferent (i in 1..n) (x[i,j]));
```

The variable `x[i,j]` being equal to `k` represents the cell at row `i` and column `j` having the value `k`.

Steiner Triples.

```
int: nb = n * (n-1) div 6;
array[1..nb,1..n] of var 0..1: x;

constraint forall (i in 1..nb)
  (sum (j in 1..n) (x[i,j])= 3);
constraint forall (i,j in 1..nb where i!=j)
  (sum (k in 1..n) (x[i,k] * x[j,k]) <= 1);
```

The variable `x[i,j]` being equal to 1 represents the element `j` belonging to the triple `i`.

BIBD.

```
int: v;
int: k;
int: lambda;
int: b = (lambda * v * (v - 1)) div (k * (k - 1));
int: r = (lambda * (v - 1)) div (k - 1);

array [1..v, 1..b] of var bool: m;

constraint forall (i in 1..v) (sum (j in 1..b) (bool2int(m[i, j])) = r);

constraint forall (j in 1..b) (sum (i in 1..v) (bool2int(m[i, j])) = k);

constraint forall (i1, i2 in 1..v where i1 != i2)
            (sum (j in 1..b) (bool2int(m[i1,j] /\ m[i2,j])) = lambda);
```

The variable `m[i,j]` being equal to 1 represents the object `i` belonging to block `j`.

Social Golfers.

```
int: n_groups;
int: n_per_group;
int: n_rounds;
int: n_golfers = n_groups * n_per_group;

set of int: groups = 1..n_groups;
set of int: group = 1..n_per_group;
set of int: rounds = 1..n_rounds;
set of int: golfers = 1..n_golfers;

array [rounds, groups] of var set of golfers: x :: is_output;

constraint forall (r in rounds, g in groups)
    (card(x[r, g]) = n_per_group);

constraint forall (r in rounds)
    (all_disjoint (g in groups) (x[r, g]));

constraint forall (a, b in golfers where a < b)
    (sum (r in rounds, g in groups)
        (bool2int(a in x[r, g] /\ b in x[r,g])) <= 1);
```

The variable `x[i,j]` is the set of players who are in Group `j` in Week `i`.

N-Queens (Boolean).

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

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

The variable `x[i,j]` being equal to 1 represents a queen being placed in square `(i,j)`.

N-Queens (integer).

```
int: n;
array [1..n] of var 1..n: x;

constraint
    forall (i,j in 1..n where i != j) (
        x[i] != x[j] /\ x[i]+i != x[j]+j /\ x[i]-i != x[j]-j
    );
```

The variable `x[i]` taking the value `j` means that the queen in column `i` is in row `j`.

B Proofs for Section 3

Here we give details for showing undecidability of the solution symmetries. The argument given in Section 3.3 is essentially complete, though the more detailed discussion of tiling given here may give some extra assistance. The underlying tiling construction required for Section 3.2 is substantially more technical though.

Our tiles are formed from a Turing machine program T, which runs (deterministically) forever when started on the blank tape in the initial state 0. Recall that the problem of deciding if state 1 is eventually reached is undecidable; subject to minor modification of the program T we may assume without loss of

generality that there is a single possible transition into state 1 in the program definition. The program T encodes into a tiling problem by square tiles with restricted adjacency relations, so that each successive tiled row of the first quadrant of the plane corresponds to each successive tape configuration. We direct the reader to Robinson's original presentation [3] for details, but give a rough description of the basic tiles to aid the discussion (note we consider a version tiling only the positive quadrant, starting from a Turing machine on a one-way infinite tape). There are four general kinds of tiles as follows.

- Start tiles. These encode the initial configuration.
- Alphabet tiles. These tiles translate standard tape cell content (that is, a symbol written on some tape cell) from a row to the row above, with no change to the content.
- Action tiles. These tiles enact the program commands: there is a tile for each command of the Turing machine program. If the program asks for a transition to the right (for example), then the new state information is displayed on the right hand side of the tile and dually for left transitions. These are the only tiles that can be placed above a tile whose upper edge encodes a symbol *and* a state.
- Merge tiles. These are for placing next to the action tiles, to receive the new state information. There is a left and a right merge tile for each combination of state and symbol: for a right-moving merge tile corresponding to state q and symbol s, the left edge is designed to match the right edge of an action tile encoding a transition into q, and the bottom edge is designed to match the tape symbol s of the tile below it. The upper edge shows the combination (q, s) encoding the information that the machine head is in state q reading s in this square.

If we let tile 0 denote the start tile representing the machine in square 0 in initial state and let tile 1 denote the action tile corresponding to the unique transition into state 1, then for every such deterministic Turing machine program T, the corresponding set of tiles will tile the first quadrant, with tile 0 at position $(0, 0)$, and the problem of deciding (given arbitrary such T) if the tile 1 is placed is Σ_1^0-complete. For CSP models we are interested not in tiling the plane, but rather in tiling an $N \times N$ grid, and not every such bounded tiling corresponds to a restriction of some full tiling of the positive quadrant. The main issue that arises for Robinson's construction is that one can place a left-moving merge tile on the right hand boundary, thus, in essence, introducing a second tape head from the right: there is no control over the state of this second head. This presents a challenge to our basic strategy (see Section 3.1), as it may allow for the placement of the designated symmetry-breaking tile 1, even if the original program T did not reach the special state 1.

There are several ways to fix this by adjusting the labelling of the tiles (such as adding border tiles, or adding extra information to the tiles, recording whether they are left or right of the machine head). However as we are concerned with CSP models (not tilings per se), the easiest fix is to add a constraint asking that *no left moving merge tile be placed at the right hand border.* This can be done

by adding a unary relation M on the domain of tiles corresponding to the set of tiles that are not left moving merge tiles and constrain the variable `x[N-1,i]` to be in M.

For basic tiling of an $N \times N$ grid, the machine head, starting at position $(0, 0)$ can never reach a position (i, j) with $i > j$ as it can only move one step at time and each step involves placement of a new row: if it moves rightward at every step, the corresponding action tiles are placed in positions (i, i). In this instance, the extra constraint (no left-moving merge tiles on the boundary) forces every tiling of an $N \times N$ grid to be a restriction of some tiling of the plane (obviously, this is a property that is very particular to the construction being used). However one of our arguments involves a distortion of this tiling, so that each of the original tiles becomes a 2×1-rectangle, with successive rows offset by 1 placement. In this situation it is possible for (the distorted versions of) action tiles to "fall off" the right edge of the $N \times N$ grid. As it turns out, provided we constrain the outer perimeter not to involve the placement of (the distorted version of) a left-moving merge tile, it is still true that tilings of an $N \times N$ grid by these rectangular tiles correspond to restrictions of tilings of the full $\mathbb{N} \times \mathbb{N}$ positive quadrant. This is because, while the actual machine head may move leftwards again later, the tape configurations are encoded in rows of this tiling so that each configuration is shifted one step rightwards at every row. Thus the head is unable to move leftwards quickly enough to re-enter the grid. (A sketch of the basic idea of this distorted tiling can be seen in Figure 2.)

We now give the details of the argument for proving the undecidability of recognising the dimension swap symmetry for a single 2-dimensional matrix variable (Section 3.2). The argument is substantially more technical than for the value inversion symmetry.

We will refer to the set of points $\{(i, j) \in \mathbb{N} \times \mathbb{N} \mid i > j\}$ as the *first octant*: it is the lower half of the positive part of the first quadrant of an integer lattice. We first show how to translate the basic tiling problem (can the plane be tiled with placement of tile 0 at $(0, 0)$ with tile 1 placed somewhere) to a corresponding problem of tiling the first octant of the plane, this time requiring that tile 0 is placed at $(1, 0)$ (and asking if tile 1 is placed somewhere). To do this we explain how to translate any finite set of tiles into a new set such that the original set can tile the first quadrant if and only if the new set can tile the first octant. The idea is to replace each square tile by a rectangle of two new squares. Let $\mathcal{T} = \{0, \ldots, n-1\}$ be a set of "tiles" with adjacency relations $\sim_h$ and $\sim_v$. Now consider the $2n$-element set $\mathcal{T} \times \{L, R\}$: for convenience we write i_L in place of (i, L), and similarly for i_R. We now define the relations $\sim_h$ and $\sim_v$ as follows.

- For each i we have $i_L \sim_h i_R$.
- If $i \sim_h j$, then $i_R \sim_h r_L$.
- If $i \sim_v j$, then $i_R \sim_v j_L$.
- For each i and each j we define $i_L \sim_v j_R$.

The idea is demonstrated in Figure 1. In the left picture, four copies of some tiles have been placed, with a copy of tile 0 at position $(0, 0)$, tile 1 at position $(1, 0)$, tile 2 at position $(0, 1)$ and tile 3 at position $(1, 1)$ (in this particular

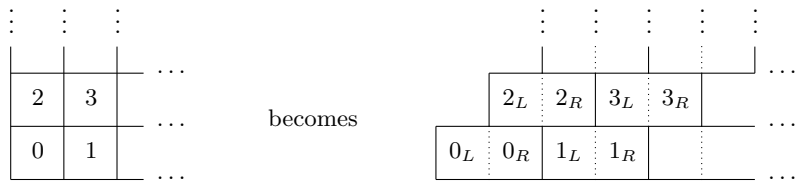

Fig. 1. The construction for tiling the first octant.

enumeration of the tiles, the tile 1 plays no distinguished role). The right picture shows the corresponding tiling of the first octant: boundaries between left and right copies of tiles are dotted: the relation $\sim_h$ only allows two halves of the same tile to be placed horizontally adjacent. It is routinely seen that the constructed set of tiles can tile the first octant with 0_L placed at $(1, 0)$ if and only if the original set can tile the positive quadrant with tile 0 placed at $(0, 0)$. Moreover, every tiling by the first set of tiles corresponds in an obvious way to a unique tiling by the second set and vice versa.

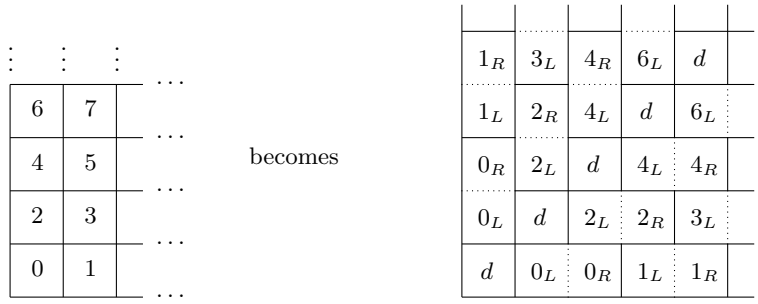

Fig. 2. A typical solution to the symmetric tiling CSP, starting from the basic tiling of the plane by square tiles.

We now construct a CSP model based around a tiling of the first octant. The idea is constraint the points of an $N \times N$ grid to be tiled in the first octant, with tile 0 placed at position $(1, 0)$, but also in the second octant in symmetry with the first octant: tile 0 will be placed at position $(0, 1)$, and the constraints for positions in the second octant will switch the roles of horizontal and vertical adjacency. These can be written as the following constraints:

```
constraint forall (i,j in 0..N-1 with i>j) (x[i-1,j] ~h x[i,j])
constraint forall (i,j in 0..N-1 with i>j) (x[i,j-1] ~v x[i,j])
constraint forall (i,j in 0..N-1 with i<j) (x[i-1,j] ~v x[i,j])
constraint forall (i,j in 0..N-1 with i<j) (x[i,j-1] ~h x[i,j])
constraint x[0,1]=0_L
constraint x[1,0]=0_L
```

We introduce a separate tile—tile d—for placement along the diagonal.

```
constraint forall (i in 0..N-1) (x[i,i]=d)
```

We constraint the outer perimeter to not include left-moving merge tiles; letting M denote the set of tiles other than (distorted) left-moving merge tiles, we add:

```
constraint forall (i in 0..N-1) (x[N-1,i] in M) and (x[i,N-1] in M)
```

An example of the situation is shown in Figure 2, starting from a conventional tiling on the left, and how the corresponding solution to the distorted CSP model might be visualised.

This corresponds to the adjusted form of Step 1 from the Basic Strategy of Section 3.1. After this distortion, the original tile 1 has a left and right version; we proceed along the lines of Step 2 using tile 1_L in place of tile 1. Let us duplicate this tile to produce tile $1'_L$. Every existing adjacency relating tile 1_L is extended to tile $1'_L$ (so tile $1'_L$ may be placed left of tile 1_R, for example). At this stage the CSP model admits the constraint symmetry $x[i,j] \mapsto x[j,i]$, and has solutions for every value of N. We now add one further constraint: that tile 1_L cannot be placed in the second octant and tile $1'_L$ cannot be placed in the first octant.

If the machine running the original Turing machine program T (deterministic and started on the blank tape in state 0) eventually reaches the distinguished state 1, then for large enough N, a solution of the standard tiling (with tile 0 at $(0,0)$) involves placement of tile 1. Then in our distorted CSP model, we must assign, for some $i > j$ the value 1_L to $x[i,j]$ and the value $1'_L$ to $x[j,i]$. The dimension swap symmetry fails for this CSP model. Otherwise, if T never enters state 1, then no solution to the original tiling involves placement of tile 1 (even over $N \times N$ grid, provided we constrain the right edge of the grid to avoid left-moving merge tiles), and similarly no $x[i,j]$ can be given the value 1_L or $1'_L$ in any solution to the distorted CSP model. Hence in this case, the CSP model admits the dimension swap symmetry. This completes the reduction of the Halting problem to the failure of dimension swap solution symmetries in CSP models (Section 3.2).

We mention that these techniques seem related to the undecidability results proved for infinite CSPs by Dantchev and Valencia [1]. Indeed, tiling arguments provide a reasonable simplification to the Turing machine argument given there for 2-dimensional infinite CSPs with just successor on the index set: only a single variable x is required using tilings (and only successor is required to complete the proof for undecidability of the value inversion symmetry). Moreover, it is easy to use tiling arguments to show that it is highly undecidable (Σ^1_1-hard) to decide if an infinite CSP has a solution, even when restricted to a single 2-dimensional variable and with index arithmetic restricted to successor and order. Indeed, the problem of tiling the first quadrant such that tile 0 is placed at position $(0,0)$ and tile 1 placed infinitely often in the first row is Σ^1_1-complete (Harel [2]) and can be expressed as an infinite CSP: to the basic tiling problem given in Section 3 (but with no bound on the values of i and j), one must add the constraint

```
constraint (x[0,0]=0)
constraint forall i exists j>i (x[j,0]=1)
```

Dantchev and Valencia [1] also show that it is undecidable as to whether or not an infinite CSP with a single 1-dimensional variable has a solution, provided

that the full power of Presburger arithmetic is allowed on the index set. while we do not give details, it is possible to adapt this argument in a manner similar to the Basic Strategy in Section 3.1 to obtain undecidability of some solution symmetries for 1-dimensional variables (such as value inversion: dimension swap has no meaning for 1-dimensional variables)

References

1. Dantchev, S., Valencia, F.: On the computational limit of infinite satisfaction. In: Proceedings of the 2005 ACM Symposium on Applied Computing. pp. 393–397 (2005)
2. Harel, D.: Effective transformations on infinite trees with applications to high undecidability, dominoes and fairness. J. ACM pp. 224–248 (1986)
3. Robinson, R.: Undecidability and nonperiodicity for tilings of the plane. Inventiones Math. 12, 177–209 (1971)