KNIGHT PROBLEM

1. Problem description

This problem (sometimes also called Euler tour) involves moving a knight over a chessboard according to the rules of chess such that every field is visited exactly once, after which the knight returns to the initial starting position. If the chessboard fields are represented as the 64 nodes of a graph, solving the problem involves finding a Hamiltonian path where each edge represents a valid move for the knight.

```
range ChessBoard 1..64;
{ChessBoard} Knightmove[i in ChessBoard] = { j | j in ChessBoard:
     i mod 8 =1 &
          (j=i-15 \/ j=i-6 \/ j=i+10 \/ j=i+17)
     \/ i mod 8 = 2 &
          (j=i-17 \/ j=i-15 \/ j=i-6 \/ j=i+10 \/ j=i+15 \/ j=i+17 )
     \/ i mod 8 >= 3 & i mod 8 <= 6 &
          (j=i-17 \/ j=i-15 \/ j=i-10 \/ j=i-6 \/ j=i+6 \/ j=i+10 \/ j = i+15 \/ j=i+17)
     \/ i mod 8 = 7 &
          (j=i-17 \/ j=i-15 \/ j=i-10 \/ j=i+6 \/ j=i+15 \/ j=i+17)
     \/ i mod 8 = 0 &
          (j=i-17 \/ j=i-10 \/ j=i+6 \/ j=i+15)
};
var ChessBoard jump[ChessBoard];
solve {
     forall(p in ChessBoard)
        jump[p] in Knightmove[p];
     circuit(jump);
     forall(p in ChessBoard)
        sum(c in n Knightmove[p]) (jump[c] = p) = 1;
};
```

Figure 1. The knight problem in OPL

2. OPL model

The problem model of figure 1 is taken from [1, sec. 10.3], it relies on the numbering system shown in figure 2 on the next page. The core of the model is the array of sets `Knightmove` which returns for each of the 64 positions the set of possible next fields allowed for the knight. The array `jump` records for each of the 64 positions the next position and is thus in the appropriate format for the `circuit` constraint [1, pp. 101-102],

Date: January 2004.

which specifies that `jump` represents a Hamiltonian circuit. The preceding quantified constraint in figure 1 on the preceding page merely constrains the domain of `jump[p]` to the set of permissible moves as returned by `Knightmove[p]`. The last constraint is semantically redundant, as it requires that each of the 64 positions appears exactly once in the range of the array `jump`.

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	49	50	51	52	54	55	56
57	58	59	60	61	62	63	64

FIGURE 2. Numbering system

3. Z MODEL

We can transform the model in figure 1 on the previous page quite straightforwardly. Some simplifications are possible, we point out the differences.

3.1. **Data modelling.** As before, we use an alias for the range *ChessBoard*.

$$ChessBoard == 1 \,..\, 64$$

The following represents the specification of the function *Knightmove* (which has been an array of sets in the OPL analogue), using explicit enumeration. Expressions of this kind are natural in Z and we could simplify some terms as shown above. No parentheses are needed, since $\land$ binds stronger than $\lor$ [3, p. 69].

$$Knightmove : ChessBoard \longrightarrow (\mathbb{F}\ ChessBoard)$$

$$\begin{aligned}
Knightmove = \{ & i : ChessBoard; j : \mathbb{F}\ ChessBoard \mid \\
& i \bmod 8 = 1 \ \land\ j = \{i-15, i-6, i+10, i+17\} \ \lor \\
& i \bmod 8 = 2 \ \land\ j = \{i-17, i-15, i-6, i+10, i+15, i+17\} \ \lor \\
& (i \bmod 8 \in 3\,..\,6) \ \land\ j = \{i-17, i-15, i-10, i-6, i+6, i+10, i+15, i+17\} \ \lor \\
& i \bmod 8 = 7 \ \land\ j = \{i-17, i-15, i-10, i+6, i+15, i+17\} \ \lor \\
& i \bmod 8 = 0 \ \land\ j = \{i-17, i-10, i+6, i+15\} \\
\}&
\end{aligned}$$

3.2. **Problem constraints.** The main problem constraints are (i) the field constraints on *jump* dependent on the current position and (ii) the circuit constraint. We provide an analogue of the latter in the Appendix, but found it more natural here to integrate this constraint into the signature of *jump*. This is possible since, as we have shown in the Appendix, the circuit global constraint is equivalent to the specification of a bijection whose domain and range sets are identical.

Euler_Tour

$$jump : ChessBoard \rightarrowtail\!\!\!\!\!\rightarrow ChessBoard$$

$$\forall\, p : ChessBoard \bullet jump(p) \in Knightmove(p)$$
$$\forall\, p : ChessBoard \bullet \#\{c : Knightmove(p) \mid jump(c) = p\} = 1$$

For comparison, we have also included the redundant constraint of the model in figure 1, being the last expression in the predicate block. The result is a notably succinct specification.

4. Literature references

Interesting in comparison with the OPL model of [1, sec. 10.3] is the NCL variant in [5, sec. 6.5], which was also published in [6, sec. 6.4]. This variant is based on the use of constraint references, a feature which is unique to NCL and allows a terse formulation without requiring a specialised global constraint. It is thus between the extremes of a low-level formulation and the more abstract format of using global constraints. A succinct ALICE variant of the problem using the circuit global constraint can be found in [2, p. 88]. Even earlier than that, Wirth presented the stepwise refinement of the problem into a recursive PASCAL program in [4, pp. 137–143], using two integer arrays to record the valid knight moves (comparable to figure 1).

References

[1] Pascal Van Hentenryck. *The OPL Optimization Programming Language*. MIT Press, January 1999.
[2] Jean-Louis Laurière. A Language and a Program for Stating and Solving Combinatorial Problems. *Artificial Intelligence*, 10(1):29–127, February 1978.
[3] J. M. Spivey. *The Z Notation: A Reference Manual*. J. M. Spivey, Oriel College, Oxford OX1 4EW, second edition, 1998. First published 1992 by Prentice Hall, current version on http://spivey.oriel.ox.ac.uk/mike/zrm/.
[4] Niklaus Wirth. *Algorithms + Data Structures = Programs*. Series in Automatic Computation. Prentice-Hall, 1976.
[5] Jianyang Zhou. *The Manual of NCL - Version 1.2*. Loria, Campus Scientifique - BP 239, 54506, Vandoeuvre-les-Nancy, France, 1998.
[6] Jianyang Zhou. Introduction to the constraint language NCL. *Journal of Logic Programming*, 45(1-3):71–103, September 2000.