

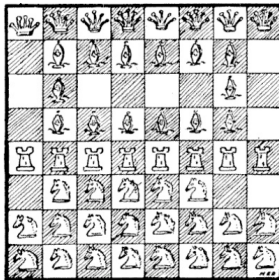
Encoding the Crowded Chessboard Puzzle in B

Michael Leuschel

November 2016

Crowded Chessboard puzzle

306.—THE CROWDED CHESSBOARD.



The puzzle is to rearrange the fifty-one pieces on the chessboard so that no queen shall attack another queen, no rook attack another rook, no bishop attack another bishop, and no knight attack another knight. No notice is to be taken of the intervention of pieces of another type from that under consideration—that is, two queens will be considered to attack one another although there may be, say, a rook, a bishop, and a knight between them. And so with the rooks and bishops. It is not difficult to dispose of each type of piece separately; the difficulty comes in when you have to find room for all the arrangements on the board simultaneously.

(EBook # 16713 from Project Gutenberg)

Crowded Chessboard puzzle

- puzzle 306 from Dudeney, Amusement in Mathematics, 1917
- “The puzzle is to rearrange the fifty-one pieces on the chessboard so that no queen shall attack another queen, no rook attack another rook, no bishop attack another bishop, and no knight attack another knight. No notice is to be taken of the intervention of pieces of another type from that under consideration—that is, two queens will be considered to attack one another although there may be, say, a rook, a bishop, and a knight between them. And so with the rooks and bishops. It is not difficult to dispose of each type of piece separately; the difficulty comes in when you have to find room for all the arrangements on the board simultaneously.”

Various Encodings

- Puzzle much more challenging than 8-Queens; closer to “real” industrial applications; but can still be presented relatively compactly; Encodings:
 - generic Prolog solution: using CLP(FD) + reification with sum constraint; 100 lines of Prolog, but **not** successful for solving $n=8$
 - Z3 SMT Solution: hard-coded for 8 queens; various encodings (board \mapsto figures vs figures \mapsto position), **not** successful for $n=8$;
 - generate SAT Encoding (7836 lines) from a Python program (235 lines): $n = 8$ solved in 8 seconds by Z3, 2 seconds by Sat4j
- Can we write a declarative specification **and** solve it ?

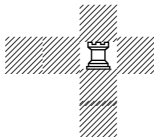
B Encoding

- the B model is very readable; we will return to performance later
- below we first solve this puzzle where we set n to 4
- we use ProB version 1.6.2 – *beta1*
- we have the following set of possible indexes:

$$\text{let } \text{Idx} = 1..n \rightsquigarrow \{1, 2, 3, 4\}$$

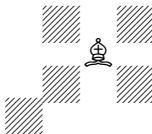
Rook Horizontal/Vertical Moves

- let $moveHV = \lambda(i, j). (i \in ldx \wedge j \in ldx | \{k, l | k \in ldx \wedge l \in ldx \wedge (i \mapsto j) \neq (k \mapsto l) \wedge (i = k \vee j = l)\})$
- E.g., $moveHV(2 \mapsto 3) = \{(1 \mapsto 3), (2 \mapsto 1), (2 \mapsto 2), (2 \mapsto 4), (3 \mapsto 3), (4 \mapsto 3)\}$, visualized below:



Bishop Diagonal Moves

- let $moveDiag = \lambda(i, j). (i \in Idx \wedge j \in Idx | \{k, l | k \in Idx \wedge l \in Idx \wedge (i \mapsto j) \neq (k \mapsto l) \wedge (k - i = l - j \vee i - k = l - j)\})$
- E.g., $moveDiag(2 \mapsto 3) = \{(1 \mapsto 2), (1 \mapsto 4), (3 \mapsto 2), (3 \mapsto 4), (4 \mapsto 1)\}$, visualized below:



Knight Moves

- let $dist = \lambda(i, j).(i \in \mathbb{Z} \wedge j \in \mathbb{Z} | (IF\ i \geq j\ THEN\ i - j\ ELSE\ j - i\ END))$
- let $moveK = \lambda(i, j).(i \in Idx \wedge j \in Idx | \{k, l | k \in Idx \wedge l \in Idx \wedge i \neq k \wedge j \neq l \wedge dist(i \mapsto k) + dist(j \mapsto l) = 3\})$
- E.g., we have
 $moveK(2 \mapsto 3) = \{(1 \mapsto 1), (3 \mapsto 1), (4 \mapsto 2), (4 \mapsto 4)\}$,
 visualized below:



All Moves

- single higher-order function,
 - which for each piece returns the attacking function
 - which in turn for each position on the board returns the set of attacked positions
- *let* $attack = \{Rook \mapsto moveHV, Bishop \mapsto moveDiag, Queen \mapsto \lambda p.(p \in Idx \times Idx | moveHV(p) \cup moveDiag(p)), Knight \mapsto moveK, Empty \mapsto Idx \times Idx \times \{\emptyset\}\}$

Quick Validation

- *let* $crd = \lambda x. (x \in \mathbb{P}(\mathbb{Z} \times \mathbb{Z}) | card(x))$
- $\lambda P. (P \in PIECES | crd[ran(attack(P))])$, shown in the table below:

Quick Validation

- $\text{let } \text{crd} = \lambda x. (x \in \mathbb{P}(\mathbb{Z} \times \mathbb{Z}) | \text{card}(x))$
- $\lambda P. (P \in \text{PIECES} | \text{crd}[\text{ran}(\text{attack}(P))])$, shown in the table below:
















<i>Pce</i>	<i>Attacks</i>
<i>Queen</i>	$\{9, 11\}$
<i>Rook</i>	$\{6\}$
<i>Bishop</i>	$\{3, 5\}$
<i>Knight</i>	$\{2, 3, 4\}$
<i>Empty</i>	$\{0\}$

Specifying number of pieces of each type

- $\exists nrPcs. (nrPcs \in PIECES \rightarrow 0..n^2 \wedge (Queen \mapsto n \in nrPcs \wedge Rook \mapsto n \in nrPcs \wedge Bishop \mapsto 5 \in nrPcs \wedge Knight \mapsto 2 \in nrPcs) \wedge \Sigma(p).(p \in PIECES | nrPcs(p)) = n^2) nrPcs = \{(Queen \mapsto 4), (Rook \mapsto 4), (Bishop \mapsto 5), (Knight \mapsto 2), (Empty \mapsto 1)\}$

Solving Crowded Chessboard Puzzle for $n = 4$

- $\exists \text{board}. (\text{board} \in \text{Idx} \times \text{Idx} \rightarrow \text{PIECES} \wedge \forall (\text{pos}, \text{piece}). (\text{pos} \mapsto \text{piece} \in \text{board} \Rightarrow \forall \text{pos2}. (\text{pos2} \in \text{attack}(\text{piece})(\text{pos}) \Rightarrow \text{board}(\text{pos2}) \neq \text{piece})) \wedge \forall P. (P \in \text{PIECES} \Rightarrow \text{card}(\{p \mid p \in \text{dom}(\text{board}) \wedge \text{board}(p) = P\}) = \text{nrPcs}(P))) \rightsquigarrow \text{TRUE}$
- first solution is visualized below:

Compare B with 100 line Prolog non-solution

```
solve(N,Knights,Sol) :- length(Sol,N),
    maplist(pieces(N),Sol), append(Sol,AllPieces),
    Bishops is 2*N-2, Empty is N*N - 2*N - Bishops - Knights,
    global_cardinality(AllPieces,[0-Empty, 1-N, 2-N, 3-Bishops,4-Knights]),
    maplist(exactly_one(1),Sol), % one queen on every row
    maplist(exactly_one(2),Sol), % one rook on every row
    transpose(Sol,TSol),
    maplist(exactly_one(1),TSol), % one queen on every col
    maplist(exactly_one(2),TSol), % one rook on every col
    findall(diag(D,Sol),diagonal(Sol,D),Diagonals),
    maplist(at_most_one(1,Sol),Diagonals), % check queens do not attack each other
    maplist(at_most_one(3,Sol),Diagonals), % ditto for bishops
    knights_ok(Sol),labeling([ffc],AllPieces),prboard(Sol).

...
exactly_one(Piece,List) :- count(Piece,List,'#=',1).
at_most_one(Piece,Sol,diag(D,Sol)) :- at_most_one(Piece,D).
at_most_one(Piece,List) :- count(Piece,List,'#<',2).

...
check_knight(Index,Line,K1) :-
    (nth1(Index,Line,K2) -> not_two_knights(K1,K2) ; true).
not_two_knights(K1,K2) :- (K1 #= 4) #=> (K2 #\= 4).

...
```

Compare B with 601 line Z3 non-solution

```
; (set-logic QF_FD)
; datatype pair for column / row tuples
(declare-datatypes (T1 T2) ((Pair (mk-pair (first T1) (second T2)))))
...
; reduce symmetry by sorting queens
(assert (= (first q1) 1))
(assert (= (first q2) 2))
...
; unwind quantifier
; as z3 does not detect saturation of the implications
; LHS in the quantifier
(assert (not_same_diagonal q1 q2))
...
(assert (not_reachable_knight k20 k21))
(assert (distinct r1 r2 r3 r4 r5 r6 r7 r8
                  q1 q2 q3 q4 q5 q6 q7 q8
                  b1 b2 b3 b4 b5 b6 b7 b8 b9
                  b10 b11 b12 b13 b14
                  k1 k2 k3 k4 k5 k6 k7 k8
                  k9 k10 k11 k12 k13 k14 k15
                  k16 k17 k18 k19 k20 k21))
...
```

Compare B with 296 line Python SAT solution

```
...
def totalizer_c2(var_tuple,left_vars,right_vars,outputs):
    (a,b,r) = var_tuple
    if a > len(left_vars) and b > len(right_vars):
        return "-" + get_var(outputs[r-1]) + " 0"
    if a > len(left_vars):
        return get_var(right_vars[b-1]) + " -" + get_var(outputs[r-1]) + " 0"
    if b > len(right_vars):
        return get_var(left_vars[a-1]) + " -" + get_var(outputs[r-1]) + " 0"
    return get_var(left_vars[a-1]) + " " + get_var(right_vars[b-1]) + " -" + ge

...
# encode knights movement
for numx in range(1,n):
    for numy in range(1,n):
        for (otherx,othery) in {(numx+1,numy+2),(numx-1,numy+2),(numx+1,numy-2)}:
            if otherx > 0 and otherx < 9 and othery > 0 and othery < 9:
                output += ["-" + get_var(var_name("knight",numx,numy)) + " -" +
s = card_constraints_adder("queen",num_queens,n)
output += s
s = card_constraints_adder("rook",num_rooks,n)
output += s
...
```


Readability and Performance

- this solution is very readable, much more readable than the SMT, Prolog or Python SAT generator
- it is a natural mathematical specification of the problem
- but, as such ProB cannot solve it for $n = 8$
- it is higher-order, so it cannot be translated using Kodkod
















Readability and Performance

- with our new KODKOD-SAT bridge, we simply annotate parts of this higher-order B model
- ProB can then, with the help of Kodkod, solve this puzzle on the annotated B encoding in a few seconds !
- we have obtained a readable and efficient model, much faster than more low-level SMT or Prolog encodings !
- DEMO

Solving Crowded Puzzle using ProB+KODKOD

- $\exists board1. (KODKOD(1, board1, n, bool(board1 \in 1..n * n \rightarrow PIECES)) \wedge KODKOD(1, board1, n, bool(card(\{p|p \in dom(board1) \wedge board1(p) = Queen\}) = n \wedge card(\{p|p \in dom(board1) \wedge board1(p) = Rook\}) = n \wedge card(\{p|p \in dom(board1) \wedge board1(p) = Bishop\}) = 5 \wedge card(\{p|p \in dom(board1) \wedge board1(p) = Knight\}) = 2 \wedge card(\{p|p \in dom(board1) \wedge board1(p) = Empty\}) = n * n - 2 * n - 5 - 2)) \wedge \forall (piece, i, j, i2, j2). (i \mapsto j \in (1..n) \times (1..n) \wedge i2 \mapsto j2 \in attack(piece)(i \mapsto j) \Rightarrow KODKOD(1, board1, attack \mapsto i \mapsto j \mapsto i2 \mapsto j2, bool(board1((i - 1) * n + j) = piece \Rightarrow board1((i2 - 1) * n + j2) \neq piece))) \wedge KODKOD_SOLVE(1, board1 \mapsto n, n \mapsto attack)) \rightsquigarrow TRUE$

First Solution




















































			
			
			
			

Solving the full Puzzle

- we set n to 8 and let $Idx = 1..n \rightsquigarrow \{1, 2, 3, 4, 5, 6, 7, 8\}$
- the number of attacked squares is:

<i>Pce</i>	<i>Attacks</i>
<i>Queen</i>	$\{21, 23, 25, 27\}$
<i>Rook</i>	$\{14\}$
<i>Bishop</i>	$\{7, 9, 11, 13\}$
<i>Knight</i>	$\{2, 3, 4, 6, 8\}$
<i>Empty</i>	$\{0\}$

Solving the full Puzzle

Crowded Chessboard Solution from 1917



Here is the solution. Only 8 queens or 8 rooks can be placed on the board without attack, while the greatest number of bishops is 14, and of knights 32. But as all these knights must be placed on squares of the same colour, while the queens occupy four of each colour and the bishops 7 of each colour, it follows that only 21 knights can be placed on the same colour in this puzzle. More than 21 knights can be placed alone on the board if we use both colours, but I have not succeeded in placing more than 21 on the "crowded chessboard." I believe the above solution contains the maximum number of pieces, but possibly some ingenious reader may succeed in getting in another knight.

(EBook # 16713 from Project Gutenberg)

Answer from 2016: There is no solution with 22 Knights: Kodkod

Statistics: 679 ms translation, 34112 ms solving, 16201 clauses, 5313 variables, 325 primary variables.

Impossible Hard Sudoku

- by Norvig <http://norvig.com/sudoku.html>
- took Norvig's solver 24 minutes and took ProB 21 minutes with original B model (most normal Sudokus take ProB < 0.1 second)
- with new Kodkod-ProB bridge total time about 4 seconds
- Kodkod stats: 415 ms translation, 270 ms solving, 309151 clauses, 155235 variables
- higher-order set gets compiled away by ProB; only primitive constraints sent to Kodkod and then to SAT
- with original FM'12 Kodkod backend it was very difficult to write a B version of the puzzle that could be translated.

Outlook

- Applications:
 - SlotTool to make it even faster
 - bounded model checking for railway/interlocking examples
- Automatic inference of annotations
- make more robust; allow nested calls