



Checklist for Designing or Reading a Model

- 1 Each array index occurs twice in the comment on the array's declaration;
`array[I, J] of ...: X; % X[i, j] = the ... i ... j`
- 2 Each array index range either starts from 1 or is an `enum`, for clarity
- 3 Beware of decision variables declared without tight domains
- 4 No decision variable has a non-inlined equality constraint
- 5 No decision variable of type `opt τ` is declared explicitly (in this course)
- 6 No `sum | forall(i in 1..x)` with a decision variable `x` is used
- 7 No equality constraints are pushed into an `if then else` expression
- 8 Beware of `where θ` and `if θ` with test `θ` on decision variables
- 9 Beware of explicit (`<->`) and implicit (`bool2int(...)`) reification
- 10 Beware of logical negation and disjunction: `not`, `\vee`, `exists`, `xor`,
`xorall`, `if θ then ϕ else ψ endif`, `<-, ->`, `<->`
- 11 Beware of nonlinear, `pow`, `div`, `mod` constraints on decision variables



Checklist for Designing or Reading a Model

- 12 Constraint predicates with the most specific meanings are used
- 13 Global constraints are used, instead of their definitions
- 14 Constraints over shared decision variables are ideally merged
- 15 The `element` predicate is not used explicitly, for readability
- 16 Functions on small sets are encoded by implicit `element`, if need be
- 17 Relations over small sets are encoded by `regular` or `table`, if faster than a formulation in the scope of checklist items 6 to 11 of Topic 2



Conventions of all Slides (recommended!)

- Scalar identifiers (`bool`, `enum` items, `int`) start with a lowercase letter.
- Mass identifiers (`array`, `enum`, `set`) start with an uppercase letter.
- Arrays have self-explanatory function identifiers: a given|unknown total function $f: X \rightarrow Y$ can be modelled as `array[X] of par|var Y: F`.
- Index identifiers are lowercase and mnemonic: memory aid.
- Comments about the *next* line end in “：“, like line 2 in the example below.

Example

```
1 int: nQueens; % the given number of queens
2 % Row[c] = the row number of the queen in column c:
3 array[1..nQueens] of var 1..nQueens: Row;
```

Variable `Row[c]` is like `Row(c)`, denoting the function `Row` applied to arg. `c`.
The array `Row` is *not* a variable, but an *array of variables*: it has row numbers,
but calling it `Rows` would make `Rows[c]` seem to denote a *set* of rows for `c`!



Ideas for Debugging and Accelerating a Model

- If there are no solutions (or missing solutions) to a known-to-be satisfiable instance, then:
 - Use `assert`, `trace`, `trace_exp`, ...:
see [Section 4.2.1.9 of the MiniZinc Handbook](#).
 - Comment away some constraints to increase the solution set: if the model now has solutions, then the commented-away constraints are unsatisfiable.
 - In the IDE or CLI, choose findMUS as the backend
in order to find a minimal unsatisfiable subset (MUS) of the constraints:
see [Section 3.8 of the MiniZinc Handbook](#).
- In the IDE, choose “Run > Profile compilation” in order to see per model line the numbers of constraints and decision variables generated by its flattening, and the flattening time: if some of these numbers are extreme, then you probably ran afoul of items of the checklist on the next slide.
- In the IDE, choose “Run > Compile” in order to inspect the flat code.