

# Constraint Programming Puzzles in B

Michael Leuschel

September 2016

## Introduction

- ProB can be used to process Latex files, i.e., ProB scans a given Latex file and replaces certain commands (such as `\probexpr`) by processed results.
- the following slides were generated (on 25/10/2016 – 7h465s) this way using ProB version 1.6.1 – rc (`WedOct1910 : 53 : 282016 + 0200`) and the command:  
`probcli -latex presentation_raw.tex presentation.tex`

## `\probexpr`

The `\probexpr` command takes a B expression as argument and evaluates it. By default it shows the B expression and the value of the expression.

- `\probexpr{{1}\/{2**100}}` in the raw Latex file will yield:  
 $\{1\} \cup \{2^{100}\} = \{1, 1267650600228229401496703205376\}$
- `\probexpr{{1}\/{2**10}}{ascii}` instructs ProB to use the B ASCII syntax:  
 $\{1\} \setminus \{2 ** 10\} = \{1, 1024\}$

## `\probrep1`

The `\probrep1` command takes a REPL command and executes it. By default it shows only the output of the execution, e.g., in case it is a predicate `TRUE` or `FALSE`.

- `\probrep1{2**10>1000}` in the raw Latex file will yield:  
*TRUE*
- `\probrep1{let DOM = 1..3}` outputs a value and will define the variable `DOM` for the remainder of the Latex run:  
`{1,2,3}`
- `\probrep1{f:DOM >-> DOM}{solution}{time}` shows the solution of a predicate and solving time:  
 $f = \{(1 \mapsto 3), (2 \mapsto 2), (3 \mapsto 1)\}$  (*0ms*)

## Other ProB Latex Commands

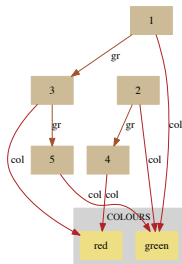
- `\prohtable` show an expression (usually a relation) as a Latex table
- `\probdot` show an expression (again, usually a relation) as a Dot graph
- `\probprint` just pretty-print a formula
- `\probif{Test}{Then}{Else}` a conditional, evaluating a predicate or boolean expression
- `\probfor{ID}{Set}{Body}` iteration

## Overview

- We now show that some constraint problems can be encoded very easily in B
- However, solving constraints in a language such as B is often considered “too difficult”
- These examples show that some examples at least can be solved by ProB
- Long term of goal of research on ProB: make B suitable as a high-level constraint modelling language

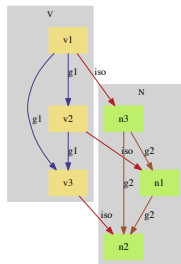
## Graph Coloring

- Let us first define a directed graph  $gr = \{(1 \mapsto 3), (2 \mapsto 4), (3 \mapsto 5)\}$
- We want to color this graph using  $cols = \{red, green\}$
- We simply set up a total function from nodes to  $cols$  and require that neighbours in  $gr$  have a different colour:
- Solution found by ProB for  $\exists col.(col \in 1..5 \rightarrow cols \wedge \forall(x, y).(x \mapsto y \in gr \Rightarrow col(x) \neq col(y)))$ :



## Graph Isomorphism

- Let us define two directed graphs  $g1 = \{(v1 \mapsto v2), (v1 \mapsto v3), (v2 \mapsto v3)\}$  and  $g2 = \{(n1 \mapsto n2), (n3 \mapsto n1), (n3 \mapsto n2)\}$
- We can check  $g1$  and  $g2$  for isomorphism by trying to find a solution for:  $\exists iso. (iso \in V \mapsto N \wedge \forall v. (v \in V \Rightarrow iso[g1[\{v\}]] = g2[iso[\{v\}]])$
- ProB has found a solution, which is shown below:





## Subset Sum Example (from Peter Stuckey)

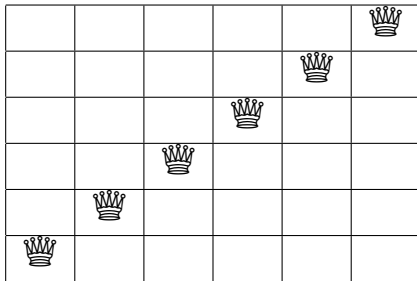
- Problem:  
 “Find 4 different integers between 1 and 5 that sum to 14”
- B Formulation:  
 $\exists S.(S \subseteq 1..5 \wedge \text{card}(S) = 4 \wedge \Sigma(z).(z \in S | z) = 14)$
- one solution found by ProB:  $S = \{2, 3, 4, 5\}$
- all solutions found by ProB:  
 $\{S | S \subseteq 1..5 \wedge \text{card}(S) = 4 \wedge \Sigma(z).(z \in S | z) = 14\} =$   
 $\{\{2, 3, 4, 5\}\}$  (0ms)
- Note: in another language: `[W,X,Y,Z] :: 1..5,`  
`all_different([W,X,Y,Z]), W+X+Y+Z #=14,`  
`labeling([X,Y,Z,W])`

## Coins Puzzle

- We have various bags each containing coins of different values  
 $coins = \{16, 17, 23, 24, 39, 40\}$ .
- Puzzle: In total 100 coins are stolen; how many bags are stolen for each coin value?
- one solution found by ProB:  $stolen = \{(16 \mapsto 2), (17 \mapsto 4), (23 \mapsto 0), (24 \mapsto 0), (39 \mapsto 0), (40 \mapsto 0)\}$
- all solutions found by ProB:  
 $\{s \mid s \in coins \rightarrow \mathbb{N} \wedge \Sigma(x).(x \in coins \mid x * s(x)) = 100\} =$   
 $\{\{(16 \mapsto 2), (17 \mapsto 4), (23 \mapsto 0), (24 \mapsto 0), (39 \mapsto 0), (40 \mapsto 0)\}\} (0ms)$
- Observe:  $coins$  is not bounded







## N-Queens (1)

- Place  $n$  queens on a  $n \times n$  chessboard so that no two queens attack each other
- We solve the puzzle for  $n=6$ .
- First, we place one queen on each row and column by using a total injection constraint:  $\exists \text{queens} . (\text{queens} \in 1..n \mapsto 1..n)$ .



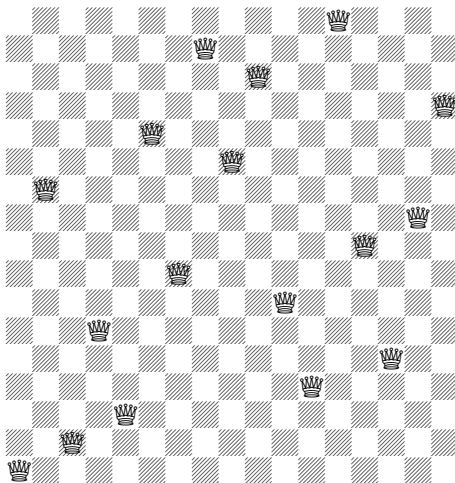
## N-Queens (2)

- We now ensure that queens cannot take each other on diagonals:  $queens \in 1..n \mapsto 1..n \wedge \forall (q1, q2). (q1 \in 1..n \wedge q2 \in 2..n \wedge q2 > q1 \Rightarrow queens(q1) + (q2 - q1) \neq queens(q2) \wedge queens(q1) + (q1 - q2) \neq queens(q2))$ .
- Shown below is a solution found by ProB  
 $queens = \{(1 \mapsto 5), (2 \mapsto 3), (3 \mapsto 1), (4 \mapsto 6), (5 \mapsto 4), (6 \mapsto 2)\}$  (0ms)

## N-Queens (3)

For  $n=17$  (20ms) we get:



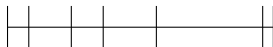
## Golomb Ruler

- A Golomb ruler with  $n = 7$  marks of length  $len = 25$  has the property that all distances between distinct marks are different

- The following expresses the problem in B:

$$\exists a.(a \in 1..n \rightarrow 0..len \wedge \forall i.(i \in 2..n \Rightarrow a(i-1) < a(i)) \wedge \forall (i1, j1, i2, j2).(i1 > 0 \wedge i2 > 0 \wedge j1 \leq n \wedge j2 \leq n \wedge i1 < j1 \wedge i2 < j2 \wedge (i1 \mapsto j1) \neq (i2 \mapsto j2) \Rightarrow a(j1) - a(i1) \neq a(j2) - a(i2)))$$

- A solution found by ProB is shown graphically below

$$a = \{(1 \mapsto 0), (2 \mapsto 2), (3 \mapsto 6), (4 \mapsto 9), (5 \mapsto 14), (6 \mapsto 24), (7 \mapsto 25)\} \text{ (80ms)}$$


## Validation of Golomb Ruler result

We can compute the  $(n * (n - 1)) / 2 = 21$  distances:

<i>Nr</i>	<i>i</i>	<i>j</i>	<i>delta</i>
1	1	2	2
2	1	3	6
3	1	4	9
4	1	5	14
5	1	6	24
6	1	7	25
7	2	3	4
8	2	4	7
9	2	5	12
10	2	6	22
11	2	7	23
12	3	4	3
13	3	5	8
14	3	6	18
15	3	7	19
16	4	5	5
17	4	6	15
18	4	7	16
19	5	6	10
20	5	7	11
21	6	7	1

## Latin Squares

- Let us construct a latin square of order 6 using indices in  $\{1,2,3,4,5,6\}$ .
- We want to construct a square  
 $\exists sol.(sol \in idx \times idx \rightarrow idx \wedge \forall(i, j1, j2).(i \in idx \wedge j1 \in idx \wedge j2 \in idx \wedge j1 > j2 \Rightarrow sol(i \mapsto j1) \neq sol(i \mapsto j2) \wedge sol(j1 \mapsto i) \neq sol(j2 \mapsto i)))$
- A solution is shown below (20 ms):

3	1	2	4	5	6
2	3	1	6	4	5
1	6	5	2	3	4
4	2	6	5	1	3
5	4	3	1	6	2
6	5	4	3	2	1



## Sudoku (1)

- We define the domain: *let*  $DOM = 1..9$
- We now compute the pairs of positions on columns that need to be different: *let*  $Diff1 = \{x1, x2, y1, y2 | y1 \in DOM \wedge y2 \in DOM \wedge x1 \in DOM \wedge x2 \in DOM \wedge x1 < x2 \wedge y1 = y2\}$ ; this gives rise to  $card(Diff1) = 324$  pairs of positions
- Now the same for rows:  
*let*  $Diff2 = \{x1, x2, y1, y2 | y1 \in DOM \wedge y2 \in DOM \wedge x1 \in DOM \wedge x2 \in DOM \wedge x1 = x2 \wedge y1 < y2\}$

## Sudoku (2)

- A solution to  $\exists Board.(Board \in DOM \rightarrow (DOM \rightarrow DOM) \wedge \forall(x1, x2, y1, y2).(x1 \mapsto x2 \mapsto y1 \mapsto y2 \in Diff1 \cup Diff2 \Rightarrow Board(x1)(y1) \neq Board(x2)(y2))) \rightsquigarrow TRUE$  is shown below.
- This does not yet take into account the all different constraint on the subsquares: *let*  $Sub = \{\{1, 2, 3\}, \{4, 5, 6\}, \{7, 8, 9\}\}$

2	5	4	1	3	6	7	8	9
1	4	2	3	6	5	9	7	8
4	9	3	2	1	8	5	6	7
3	1	8	7	9	2	4	5	6
6	3	9	8	7	1	2	4	5
5	2	1	9	8	7	6	3	4
7	6	5	4	2	9	8	1	3
8	7	6	5	4	3	1	9	2
9	8	7	6	5	4	3	2	1

## Sudoku (3)

- We now compute the pairs of positions that need to be different within each subsquare:

let  $Diff3 = \{x1, x2, y1, y2 \mid x1 \geq x2 \wedge (x1 = x2 \Rightarrow y1 > y2) \wedge (x1 \mapsto y1) \neq (x2 \mapsto y2) \wedge \exists(s1, s2).(s1 \in Sub \wedge s2 \in Sub \wedge x1 \in s1 \wedge x2 \in s1 \wedge y1 \in s2 \wedge y2 \in s2)\}$

- We combine all position pairs: let  $Diff = Diff1 \cup Diff2 \cup Diff3$
- A solution to  $\exists Board.(Board \in DOM \rightarrow (DOM \rightarrow DOM) \wedge \forall(x1, x2, y1, y2).(x1 \mapsto x2 \mapsto y1 \mapsto y2 \in Diff \Rightarrow Board(x1)(y1) \neq Board(x2)(y2))) \rightsquigarrow TRUE$  (50ms) is below:

2	7	5	1	4	3	8	6	9
1	3	6	7	9	8	2	4	5
8	4	9	5	6	2	7	1	3
7	1	2	8	3	5	4	9	6
4	6	3	2	1	9	5	7	8
5	9	8	4	7	6	1	3	2
6	5	4	3	2	1	9	8	7
3	2	1	9	8	7	6	5	4
9	8	7	6	5	4	3	2	1

## External Data Sources (1)

- ProB can read in XML and CSV files using external functions
- Let us read a CSV file containing data about chemical elements: `let data = READ_CSV_STRINGS("elementdata.csv")`
- The data is of type `seq(STRING → STRING)` and contains `size(data) = 118` entries.
- The first entry is `data(1) = {"Atomic_Number" ↦ "1"}, {"Atomic_Radius" ↦ "79"}, {"Atomic_Weight" ↦ "1.00794"}, {"Boiling_Point" ↦ "20.28"}, {"Covalent_Radius" ↦ "32"}, {"Density" ↦ "0.0708 (@ -253degC)"}, {"Electronic_Configuration" ↦ "1s1"}, {"First_Ionisation_Energy" ↦ "1311.3"}, {"Heat_Evaporation" ↦ "0.904 (H-H)"}, {"Heat_Fusion" ↦ "0.117 (H-H)"}, {"Lattice" ↦ "HEX"}, {"Lattice_Constant" ↦ "3.75"}, {"Melting_Point" ↦ "14.01"}, {"Name" ↦ "Hydrogen"}, {"Oxidation_States" ↦ "1 -1"}, {"Pauling_Electronegativity" ↦ "2.2"}, {"Specific_Heat" ↦ "14.267 (H-H)"}, {"Specific_Volume" ↦ "14.1"}, {"Symbol" ↦ "H"}, {"Thermal_Conductivity" ↦ "0.1815"}`
- Note that the read function is generic: it works for any CSV file; empty cells lead to undefined fields

## External Data Sources (2)

- We can determine the atomic elements with one letter symbols using the expression

$\{i, nm \mid data(i)("Symbol") = nm \wedge STRING\_LENGTH(nm) = 1\};$   
 the result is shown below (including the element's name):

<i>i</i>	<i>s</i>	<i>nm</i>
1	"H"	"Hydrogen"
5	"B"	"Boron"
6	"C"	"Carbon"
7	"N"	"Nitrogen"
8	"O"	"Oxygen"
9	"F"	"Fluorine"
15	"P"	"Phosphorus"
16	"S"	"Sulfur"
19	"K"	"Potassium"
23	"V"	"Vanadium"
39	"Y"	"Yttrium"
53	"I"	"Iodine"
74	"W"	"Tungsten"
92	"U"	"Uranium"

## Data Validation (1)

Data validation is one area where B's expressivity is very useful:

- We can check that the index corresponds to the atomic number:  $\forall i.(i \in \text{dom}(\text{data}) \Rightarrow i = \text{STRING\_TO\_INT}(\text{data}(i)(\text{"Atomic\_Number"}))) \rightsquigarrow \text{TRUE}$
- It is often useful to define auxiliary functions: *let*  $aw = \lambda i.(i \in \text{dom}(\text{data}) \mid \text{DEC\_STRING\_TO\_INT}(\text{data}(i)(\text{"Atomic\_Weight"}), 4));$  for example  $aw(1) = 10079$ .
- Here, B is used almost like a functional programming language

## Data Validation (2)

- we can see that  $aw$  is **not** defined for all entries:

$$dom(data) - dom(aw) =$$

$\{104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118\}$ , e.g.,  
for  $i = 104 \wedge name = \text{"Rutherfordium"}$

- We can check if the atomic weights are ordered:

$$\forall(i, j). (i \in dom(aw) \wedge j \in dom(aw) \wedge i < j \Rightarrow aw(i) \leq aw(j)) \rightsquigarrow FALSE$$

- One counter example is

$$i = 18 \wedge j = 19 \wedge aw_i = 399480 \wedge aw_j = 390983 \wedge name_i = \text{"Argon"} \wedge name_j = \text{"Potassium"}$$

## Data Validation (3)

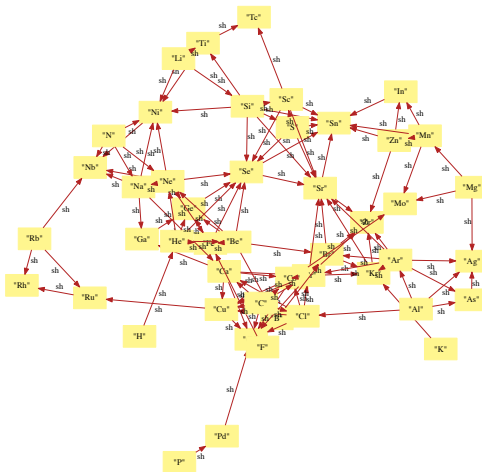
- All counter examples are shown in the table below
- From [chemistry.about.com](http://chemistry.about.com), "... it seems intuitively obvious that increasing the number of protons would increase the atomic mass. However, ...you will see that cobalt (atomic number 27) is more massive than nickel (atomic number 28). Uranium (number 92) is more massive than neptunium (number 93)."

<i>Element1</i>	<i>aw1</i>	<i>Element2</i>	<i>aw2</i>
"Argon"	"39.948"	"Potassium"	"39.0983"
"Cobalt"	"58.9332"	"Nickel"	"58.6934"
"Plutonium"	"244.0642"	"Americium"	"243.0614"
"Tellurium"	"127.6"	"Iodine"	"126.90447"
"Thorium"	"232.0381"	"Protactinium"	"231.03588"
"Uranium"	"238.0289"	"Neptunium"	"237.048"



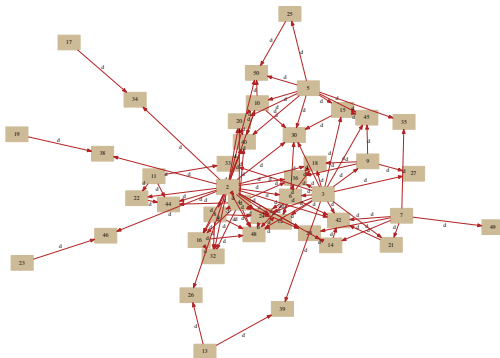
# Visualization (1)

Visualise the first 50 elements and show which symbol names share a common character:



## Visualization (2)

Visualise divisibility of first 50 numbers:



## Infinite Functions

- A finite function is  $\lambda x.(x \in 1..5 | x * x) = \{(1 \mapsto 1), (2 \mapsto 4), (3 \mapsto 9), (4 \mapsto 16), (5 \mapsto 25)\}$
- ProB auto-detects infinite functions:  
 $let\ cube = \lambda x.(x \in \mathbb{Z} | x * x * x) \rightsquigarrow \lambda x.(x \in \mathbb{Z} | x * x * x)$
- They can be used for constraint solving  
 $\{y | cube(cube(y)) = 512\} = \{2\}$  (note: no domain restriction required)
- The following function is not immediately detected as infinite, but ProB detects it cannot expand it:  
 $let\ isqrt = \{x, r | x \in \mathbb{N} \wedge r \in \mathbb{N} \wedge r * r \leq x \wedge (r+1) * (r+1) > x\}.$   
 We can still use the function:
  - $isqrt(103) = 10$
  - $isqrt[1..20] = \{1, 2, 3, 4\}$
  - $closure1(isqrt)[\{1024\}] = \{1, 2, 5, 32\}$

## Uses of the Latex Mode

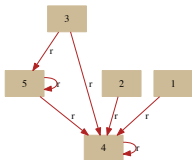
- model documentation: generate a documentation for a formal model, that is guaranteed to be up-to-date and shows the reader how to operate on the model.
- worksheets for particular tasks: can replace a formal model, the model is built-up by Latex commands and the results shown. This is probably most appropriate for smaller, isolated mathematical problems in teaching.
- validation reports for model checking or assertion checking results,
- coverage reports for test-case generation,
- as a help to debug a model, and extract information,
- documentation of ProB's features, ...

The End

End of the Latex and Constraint Solving Demo

## Using Kodkod

- ProB can make use of the Kodkod library to translate part of B to SAT
- in the ProB REPL you just need to type `:kodkod` before a predicate to activate this
- here is a predicate that works well with Kodkod and not so well with ProB's default solver:  $\exists r. (r \in 1..5 \leftrightarrow 1..5 \wedge (r; r) = r \wedge r \neq \emptyset \wedge \text{dom}(r) = 1..5 \wedge r[2..3] = 4..5)$



# Kodkod Experiments

- $\exists(z, x, y).(z \in 101..102 \wedge \{x, y\} = \{z\})$ 
  - ProB:  $z = 101 \wedge x = 101 \wedge y = 101$  (10ms)
  - kodkod:  $z = 102 \wedge x = 102 \wedge y = 102$  (240ms)
  - z3:  $x = 101 \wedge y = 101 \wedge z = 101$  (70ms)

## Send More Money

- we can compute all solutions to this classical puzzle using the B expression

$$\{S, E, N, D, M, O, R, Y \mid S * 1000 + E * 100 + N * 10 + D + M * 1000 + O * 100 + R * 10 + E = M * 10000 + O * 1000 + N * 100 + E * 10 + Y \wedge [S, E, N, D, M, O, R, Y] \in 1..8 \rightarrow 0..9 \wedge S > 0 \wedge M > 0 \wedge \text{card}(\{S, E, N, D, M, O, R, Y\}) = 8\} = \{(((((((9 \mapsto 5) \mapsto 6) \mapsto 7) \mapsto 1) \mapsto 0) \mapsto 8) \mapsto 2)\}$$