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 $25/11/2016 - 13h5624s$) this way using ProB version $1.6.2 - beta1$ ($TueNov2215:53:312016 + 0100$) 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**10}}` in the raw Latex file will yield: $\{1\} \cup \{2^{10}\} = \{1, 1024\}$
- `\probexpr{{1}\/{2**10}}{ascii}` instructs ProB to use the B ASCII syntax:
  `{1} \/ {2 ** 10} = {1,1024}`

\probrepl

The \probrepl 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.

- \probrepl{2**10>1000} in the raw Latex file will yield:
  TRUE
- \probrepl{let DOM = 1..3} outputs a value and will define the variable DOM for the remainder of the Latex run:
  $\{1,2,3\}$
- \probrepl{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

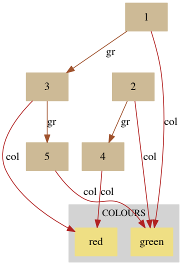
- `\probtable` show an expression (usually a relation) as a Latex table
- `\probdot` show an expression (again, usually a relation) as a Dot graph
- `\probprint` just pretty-print a formula
- `\probif`{Test}{Then}{Else} a conditional, evaluating a predicate or boolean expression
- `\probfor`{ID}{Set}{Body}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

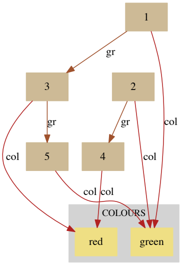
Graph Coloring

- Let us first define a directed graph $gr = \{(1 \mapsto 3), (2 \mapsto 4), (3 \mapsto 5)\}$



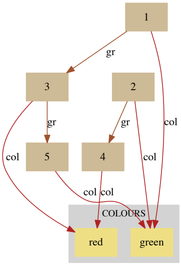
Graph Coloring

- Let us first define a directed graph $gr = \{(1 \mapsto 3), (2 \mapsto 4), (3 \mapsto 5)\}$
- We want to color this graph using $cols = \{red, green\}$



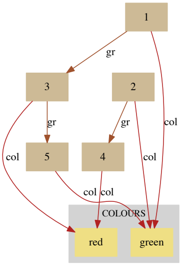
Graph Coloring

- 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:



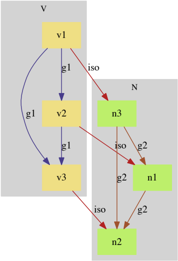
Graph Coloring

- 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 $\exists col.(col \in 1..5 \rightarrow cols \wedge \forall(x, y).(x \mapsto y \in gr \Rightarrow col(x) \neq col(y)))$:



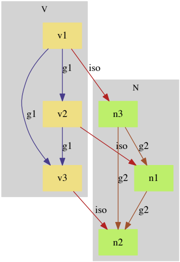
Graph Isomorphism

- We define two directed graphs $g1 = \{(v1 \mapsto v2), (v1 \mapsto v3), (v2 \mapsto v3)\}$ and $g2 = \{(n1 \mapsto n2), (n3 \mapsto n1), (n3 \mapsto n2)\}$



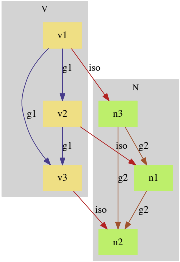
Graph Isomorphism

- We define two directed graphs $g1 = \{(v1 \mapsto v2), (v1 \mapsto v3), (v2 \mapsto v3)\}$ and $g2 = \{(n1 \mapsto n2), (n3 \mapsto n1), (n3 \mapsto n2)\}$
- The successors of $v1$ in $g1$ are $g1[\{v1\}] = \{v2, v3\}$



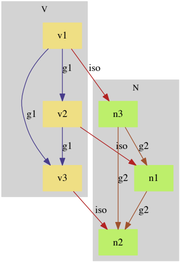
Graph Isomorphism

- We define two directed graphs $g1 = \{(v1 \mapsto v2), (v1 \mapsto v3), (v2 \mapsto v3)\}$ and $g2 = \{(n1 \mapsto n2), (n3 \mapsto n1), (n3 \mapsto n2)\}$
- The successors of $v1$ in $g1$ are $g1[\{v1\}] = \{v2, v3\}$
- We can check $g1$ and $g2$ for isomporhism by trying to find a solution for: $\exists iso.(iso \in V \rightarrowtail\!\!\!\!\twoheadrightarrow N \wedge \forall v.(v \in V \Rightarrow iso[g1[\{v\}]] = g2[iso[\{v\}]]))$.



Graph Isomorphism

- We define two directed graphs $g1 = \{(v1 \mapsto v2), (v1 \mapsto v3), (v2 \mapsto v3)\}$ and $g2 = \{(n1 \mapsto n2), (n3 \mapsto n1), (n3 \mapsto n2)\}$
- The successors of $v1$ in $g1$ are $g1[\{v1\}] = \{v2, v3\}$
- We can check $g1$ and $g2$ for isomporhism by trying to find a solution for: $\exists iso.(iso \in V \rightarrowtail\!\!\!\!\twoheadrightarrow N \land \forall v.(v \in V \Rightarrow iso[g1[\{v\}]] = g2[iso[\{v\}]]))$.
- 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"

Subset Sum Example (from Peter Stuckey)

- Problem:
  "Find 4 different integers between 1 and 5 that sum to 14"
- B Formulation:
  $\exists S.(S \subseteq 1..5 \wedge card(S) = 4 \wedge \Sigma(z).(z \in S|z) = 14)$
- one solution found by ProB: $S = \{2, 3, 4, 5\}$
- all solutions found by ProB:
  $\{S|S \subseteq 1..5 \wedge card(S) = 4 \wedge \Sigma(z).(z \in S|z) = 14\} = \{\{2, 3, 4, 5\}\}$ (*0ms*)
- Note: in a constraint programming 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?

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: $\{s | s \in coins \to \mathbb{N} \wedge \Sigma(x).(x \in coins | x * s(x)) = 100\} = \{\{(16 \mapsto 2), (17 \mapsto 4), (23 \mapsto 0), (24 \mapsto 0), (39 \mapsto 0), (40 \mapsto 0)\}\}$ (*0ms*)
- Observe: *coins* is not bounded

Latin Squares

- Let us construct a latin square of order 6 using indices in $\{1,2,3,4,5,6\}$.

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 $\exists sol.(sol \in idx \times idx \to idx \wedge \forall(i, j1, j2).(i \in idx \wedge j1 \in idx \wedge j2 \in idx \wedge j1 > j2 \Rightarrow sol(i \mapsto j1) \neq sol(i \mapsto j2) \wedge sol(j1 \mapsto i) \neq sol(j2 \mapsto i)))$
- 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

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, ...

Uses:FORMAT_TO_STRING manual entry

3.9 FORMAT_TO_STRING

This external function takes a format string (see Section 3.10) and a B sequence of values and generates an output string, where the values have been inserted into the format string in place of the `~w` placeholders.

- the length of sequence must correspond to the number of `~w` in the format string.
- the format string follows the conventions of SICStus Prolog. E.g., one can use `~n` for newlines.

Example uses:

- *FORMAT_TO_STRING*("~w |-> ~w", $\{(1 \mapsto 20), (2 \mapsto 30)\}$) = "20 |-> 30"
- *FORMAT_TO_STRING*("~w |-> ~w", [*TO_STRING*(20), "twenty"]) = "20 |-> twenty"
- *FORMAT_TO_STRING*("set = ~w", [{3..4, 1..2}]) = "set = {{1,2},{3,4}}"
- *FORMAT_TO_STRING*("~w ^ ~w is ~w~i", $[2, 10, 2^{10}, 0]$) = "2 ^ 10 is 1024"
- *FORMAT_TO_STRING*("{~w,~w}", [{2}, {3}]) = "{{2},{3}}"

Uses: B course notes

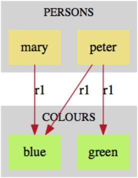
1 Relations

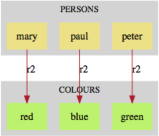
In mathematics a binary relation over the sets A and B is defined to be a subset of $A \times B$. The Cartesian product $A \times B$ in turn is defined to be the set of pairs $a \mapsto b$ such that $a \in A$ and $b \in B$.

Take for example

- $PERSONS = \{peter, paul, mary\}$ and
- $COLOURS = \{red, green, blue\}$, then
- $PERSONS \times COLOURS = \{(peter \mapsto red), (peter \mapsto green), (peter \mapsto blue), (paul \mapsto red), (paul \mapsto green), (paul \mapsto blue), (mary \mapsto red), (mary \mapsto green), (mary \mapsto blue)\}$.

A particular relation could be $let\ r1 = \{peter \mapsto green, peter \mapsto blue, mary \mapsto blue\}$. Another one is $let\ r2 = \{peter \mapsto green, paul \mapsto blue, mary \mapsto red\}$. Both are illustrated in Fig. 1.





The End

End of the Latex and Constraint Solving Demo