ProB Latex Comands

Michael Leuschel

Institut für Informatik, Heinrich-Heine-Universität Düsseldorf
Universitätsstr. 1, D-40225 Düsseldorf
{michael.leuschel}@hhu.de

1 Overview

ProB can be used to process Latex files, i.e., ProB scans a given Latex file and replaces certain commands by processed results.

Usage A typical usage would be as follows:

```
probcli FILE -init -latex Raw.tex Final.tex
```

Note: the FILE and -init commands are optional; they are required in case you want to process the commands in the context of a certain model. Currently the ProB Latex commands mainly support B and Event-B models, TLA+ and Z models can also be processed but all commands below expect B syntax. You can add more commands if you wish, e.g., set preferences using `-p PREF VAL` or run model checking `--model-check`. The Latex processing will take place after most other commands, such as model checking.

You will probably want to put the probcli call into a Makefile, in particular when you want to generate dot graphics using ProB(see below). This file was generated by putting the following into the Makefile:

```
prob_latex_doc.tex: prob_latex_doc_raw.tex
   probcli -latex prob_latex_doc_raw.tex prob_latex_doc.tex
```

Applications

- 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: for certain tasks the Latex document can replace a formal model, the model is built-up by Latex commands and the results shown in the final Latex output. This is probably most appropriate for smaller, isolated mathematical problems.
- validation reports for model checking or assertion checking results
- coverage reports for test-case generation,
- funding proposals with a B modelling of workpackages, tasks and partners,
- and hopefully many more.

2 ProB Latex Commands

2.1 General Aspects

Currently every ProB Latex command has to be put onto a **single** line. In future this may be relaxed. Every command has the following form:

```
\probCMD{ARG}{Opt1}...{Optn}
```

The arguments `Opt1` to `Optn` are optional.

If you add dummy Latex command definitions for the ProB commands you can use Latex to process the original, raw Latex file without running ProB, e.g., to check for errors and general layout. E.g., for the first command presented below you could write:

```
\newcommand{\probexpr}[1]{#1}
```

Also, ProB commands are not replaced within line comments starting with `%`. However, ProB commands are processed within block comments (started by `\begin{comment}`).

2.2 Evaluating Expressions

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.

Here are a few examples:

- `\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}`
- `\probexpr{{1}\/{2**10}}{value}` means that only the value will be displayed:
  $\{1, 1024\}$
- `\probexpr{"B-String"}{value}{string}` means that for string results the B quotes are removed:
  B-String

2.3 Executing REPL commands

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.

Here are a few examples:

- `\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]:
$\{1, 2, 3\}$
– there is a special form for the let command: `\problet{DOM}{1..3}`, it has the same effect as the command above, but also prints out the let predicate itself:
$let\ DOM = 1..3 \rightsquigarrow \{1, 2, 3\}$
– `\probrepl{a:(DOM * DOM)-->DOM}` uses the above variable `DOM`:
$TRUE$
– `\probrepl{f:DOM >-> DOM}{solution}{time}` shows the solution of a predicate and solving time:
$f = \{(1 \mapsto 3), (2 \mapsto 2), (3 \mapsto 1)\}$ $(20ms)$

2.4 Generating Latex Tables

The `\probtable` command takes a B expression as argument, evaluates it and shows it as a table. Valid options are `no-headings`, `no-tabular`, `no-hline`, `no-row-numbers` as well as `max-table-size=N` where N is a number.

Here are a few examples:

– `\probtable{{(1,2)|->3,(4,5)|->9}}` in the raw Latex file will yield:

Nr	$prj1$	$prj2$	$prj2$
1	1	2	3
2	4	5	9

– `\probtable{a:((1..2) * BOOL)-->>DOM}{no-row-numbers}` yields:

$prj1$	$prj2$	$prj2$
1	$FALSE$	1
1	$TRUE$	1
2	$FALSE$	2
2	$TRUE$	3

– `\probtable{a:((1..2) * BOOL)-->>DOM}{no-headings}{no-tabular}{no-row-numbers}` means we have to provide ourselves the tabular environment and can set it up and format it accordingly:

Idx	Truth	Range
1	$FALSE$	1
1	$TRUE$	1
2	$FALSE$	2
2	$TRUE$	3

2.5 Generating Dot Graphics

The `\probdot` command takes a B expression or predicate as argument, evaluates it and translates it into a dot graphic. It takes either one or two additional file

[1] Unless it is removed using `:unlet DOM`

arguments. The first additional argument is always the name/path of the generated dot file. The second optional argument is the name/path of the generated pdf file, it will be generated using the dot command. You can give ProB the path to the dot command at startup using the `DOT` preference, e.g., using:

```
probcli -latex Raw.tex Final.tex -p DOT /usr/local/bin/dot
```

When the second additional argument is missing, you need to generate the PDF yourself, e.g., using `sfdp` in the Makefile. Note, you can also provide as third optional argument `sfdp`, in which case ProB will call sfdp directly for you. You may have to provide the path to sfdp using the preference `SFDP`.

Here are an few examples:

- `\probdot{bij:DOM>->>DOM & !x.(x:DOM=>bij(x)/=x)}{figures/bij.dot}{figures/bij.pdf}` generates the file bij.pdf. It then can be included using the command `includegraphics`, see Figure 1.
- `\probdot{a={TRUE}*(1..10) & s=%x.(x:1..3|x*x)}{figures/ab.dot}` generates the file ab.dot. In the associated Makefile we use `sfdp` to generate the graphic included in Figure 2. (We could have used the command: `\probdot{a={TRUE}*(1..10) & s=%x.(x:1..3|x*x)}{figures/ab.dot}{figures/ab.pdf}{sfdp}` instead; the Makefile gives us more control, e.g., to set options of sfdp.) The example also shows how to display two relations/graphs in one figure.

The Makefile for Figure 2 contains:

```
figures/ab.pdf: figures/ab.dot
     $(SFDP) -Tpdf figures/ab.dot > figures/ab.pdf
```

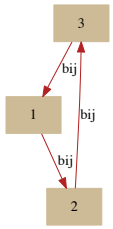


Abb. 1. Illustrating a bijection

2.6 Pretty Printing Formulas

The `\probprint` command takes an expression or predicate and pretty prints it.
Some symbols require the `bsymb.sty` package to be imported using `\usepackage{bsymb}`.
If the formula is a predicate the second argument should be `pred`.

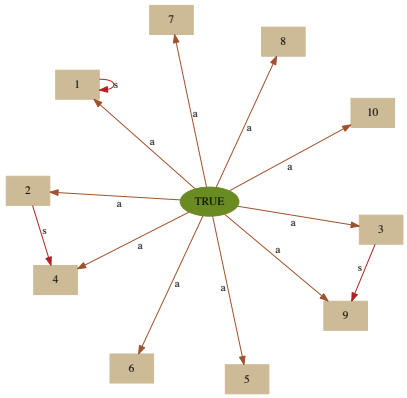


Abb. 2. Illustrating two graphs in a figure

– `\probprint{bool({1}<|{1|->2,2|->3}|>>{4}:NATURAL+->INTEGER)}` yields:
  $bool((\{1\} \lhd \{(1 \mapsto 2), (2 \mapsto 3)\}) \rhd \{4\} \in \mathbb{N} \nrightarrow \mathbb{Z})$
– `\probprint{2>1 & 3-2=1}{pred}` yields:
  $2 > 1 \land 3 - 2 = 1$

2.7 Conditional

The `\probif` command takes an expression or predicate and two Latex texts. If the expression evaluates to TRUE the first branch is processed, otherwise the other one is processed.

Here are a few examples:

– `\probif{2**10>1000}{$\top$}{$\bot$}` in the raw Latex file will yield:

  $\top$
– `\probif{bool(2**10<1000)}{$\top$}{$\bot$}` in the raw Latex file will yield:

  $\bot$

2.8 Repetition

The `\probfor` command takes an identifier, a set expression and a Latex text, and processes the Latex text for every element of the set expression, setting the identifier to a value of the set.

For example, below we embed the command:
`\probfor{ii}{1..5}{ \item the square of \probexpr{ii} is \probexpr{ii*ii} }`
within an itemize environment to generate a list of entries:

– the square of ii = 1 is ii * ii = 1
– the square of ii = 2 is ii * ii = 4
– the square of ii = 3 is ii * ii = 9
– the square of ii = 4 is ii * ii = 16
– the square of ii = 5 is ii * ii = 25

3 Examples

3.1 Visualising all bijections

In Figure 3 we show all 6 bijections for $DOM = \{1, 2, 3\}$ to itself. We have used the command `\probrepl{let bijs = SORT(DOM >->>DOM)}` to compute all bijections and sort them into a sequence (using the external function `SORT`). This command is in a comment block, so its output does not appear in the Latex file. We then use one `\probfor{ii}{dom(bijs)}{...}` to generate the PDFs and one to include the graphics in the figure. The Latex code does not know how many bijections there are.

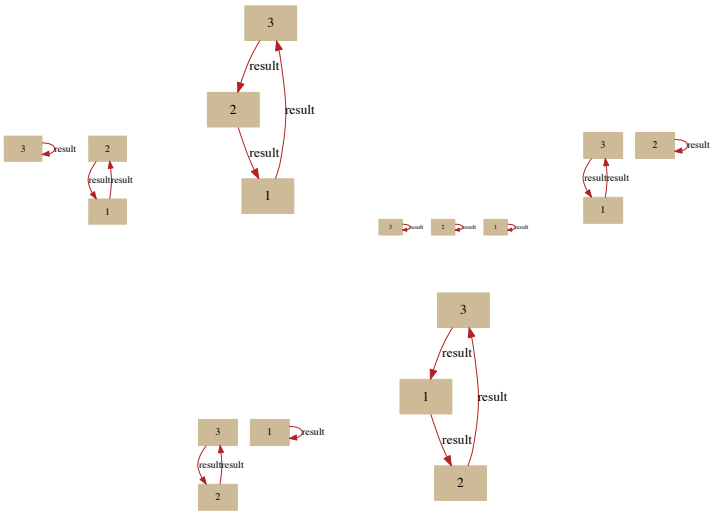


Abb. 3. Illustrating all bijections from $DOM = \{1, 2, 3\}$ to itself

Indeed, we can use very similar commands to generate in Figure 4 we show all 2 bijections between 1..2 and $BOOL$.

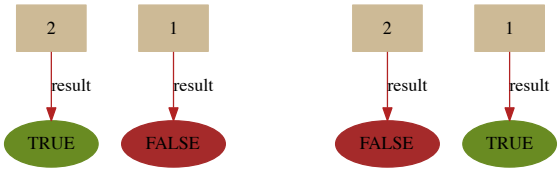
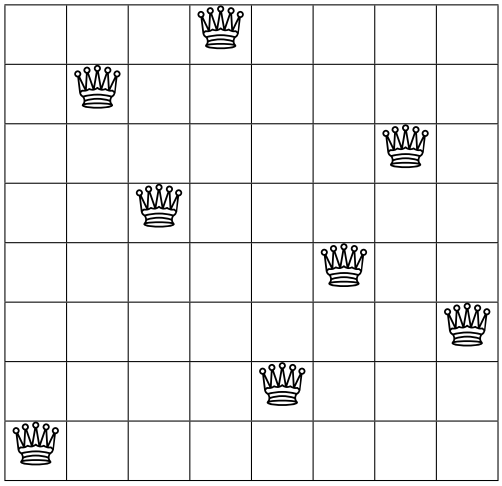


Abb. 4. Illustrating all bijections between 1..2 and $BOOL$

3.2 N-Queens

Let us solve the N-Queens problem for n=8. For this we use `\probrepl` to solve the predicate $\exists queens.(queens \in 1..n \rightarrowtail 1..n \wedge \forall(q1, q2).((q1 \in 1..n \wedge q2 \in 2..n) \wedge q2 > q1 \Rightarrow queens(q1)+(q2-q1) \neq queens(q2) \wedge queens(q1)+(q1-q2) \neq queens(q2)))$. We use the optional argument `{store}` to store the found value of `queens` (similar to a let). We then set up a tabular environment in which we next two `\probfor` commands (one for the rows and then one for the columns), with a `\probif` to generate the symbol `\WhiteQueenOnWhite` from the skak package whenever a queen is at the present position.



3.3 Primes

Using a similar scheme as used to display the N-Queens solution, we can visualise the prime numbers up to 99:

		2	3		5		7		
	11		13				17		19
			23						29
	31						37		
	41		43				47		
			53						59
	61						67		
	71		73						79
			83						89
							97		

4 Version and Configuration Information

4.1 Version Information

Using external functions you can obtain various information which can be useful to include in the generated Latex files. For example, this document was generated using ProB version 1.6.1 – *beta*3. Here are the external functions used for this Latex file, they can be found in the machine `ProBLatex.mch` and in the imported files in ProB's `stdlib` folder (e.g., `LibraryMeta.def`).

```
DEFINITIONS
 "LibraryMeta.def";
 EXTERNAL_FUNCTION_SORT(X) == POW(X) --> POW(INTEGER*X);
 SORT(x) == [];
```

The `SORT` function was used above in Section 3.1. Using the `PROB_INFO_STR` external function we can gather the following information about ProB (the `prolog-version` information is not shown):

Nr	*Flag*	*StringValue*
1	"current-time"	"7/9/2016 - 12h41 1s"
2	"java-command-path"	"/usr/bin/java"
3	"java-version"	"1.8.0_73-b02"
4	"parser-version"	"2016-06-17 13:04:11.769"
5	"prob-last-changed-date"	"Wed Sep 7 09:35:42 2016 +0200"
6	"prob-revision"	"d03d3d7b42c45ccd2dfb66e880979efa5f614330"
7	"prob-version"	"1.6.1-beta3"

Using the `PROB_INFO_INT` external function we can gather the following information about ProB and its current state:

Nr	*Flag*	*IntValue*
1	“current-state-id”	5
2	“now-timestamp”	1473244861
3	“processed-states”	6
4	“prolog-atoms-bytes-used”	1554981
5	“prolog-atoms-nb-used”	29466
6	“prolog-choice-bytes-used”	1296
7	“prolog-gc-count”	0
8	“prolog-gc-time”	0
9	“prolog-global-stack-bytes-used”	10490272
10	“prolog-local-stack-bytes-used”	1320
11	“prolog-memory-bytes-free”	8347472
12	“prolog-memory-bytes-used”	148938736
13	“prolog-runtime”	1640
14	“prolog-trail-bytes-used”	244512
15	“prolog-walltime”	4610
16	“states”	7
17	“transitions”	6

4.2 Model Information

Using the `PROJECT_INFO` external function we can gather information about the B project (the `absolute-files` information is not shown):

Nr	*Flag*	*Value*
1	“assertion_labels”	{“thm1”, “thm2”}
2	“constants”	∅
3	“files”	{“LibraryMeta.def”, “ProBLatex.mch”}
4	“invariant_labels”	{“inv0”, “inv1”, “inv2”}
5	“main-file”	{“ProBLatex.mch”}
6	“operations”	{“Inc”}
7	“sets”	∅
8	“variables”	{“x”}

4.3 Preferences

Using the `GET_PREF` and `GET_PREF_DEFAULT` external function we can gather information about preferences and, e.g., display the non-default preferences:

Nr	*Preference*	*Value*
1	“DOT”	“/usr/local/bin/dot”
2	“REPL_CACHE_PARSING”	“true”

4.4 History

Using the `STATE_AS_STRING` and `HISTORY` external function we can gather information about the current history (e.g., can be used to display a counter-example after model checking).

Step	*ID*	*State*
1	0	“(x=1)”
2	1	“(x=2)”
3	2	“(x=3)”
4	3	“(x=4)”
5	4	“(x=5)”

Using the `STATE_SUCC` external predicate we can also inspect the entire state space:

ID	*SuccID*
−1	0
0	1
1	2
2	3
3	4
4	5

4.5 Status of Assertions, Invariants, etc.

Using the `FORMULA_INFOS` and `FORMULA_VALUES` external function we can gather information about invariants, guards and assertions (actually everything that is displayed in the ProB evaluation view):

1 $/ * @thm1 * / 2^{10} = 1024$
TRUE
2 $/ * @thm2 * / \Sigma(a).(a \in 1..100 | a) = 5050$
TRUE

Here are the invariants and their status:

1 $/ * @inv0 * / x \in \mathbb{N}$
TRUE
2 $/ * @inv1 * / union(\{\lambda' | \exists a.(a \subseteq 1..3 \wedge \lambda' = a)\}) = 1..3$
TRUE
3 $/ * @inv2 * / inter(\{\lambda' | \exists a.(a \subseteq 1..3 \wedge \lambda' = a)\}) = \varnothing$
TRUE

5 Issues and to do’s

The Latex interface is still evolving, commands and command format may change. The same is true for the relevant external functions.

- Error handling could be improved (check all arguments valid; try and make raw Latex be valid),
- refactor eval_codes and eval_strings in general
- check performance of parsing via Java (seems to take 10-20 ms per parsing call), we now cache parse results,

- add `probinclude` command; can be used e.g. for graphical visualisation of history,
- more external functions (setStateID,...), maybe more or less complete reification of interface, repl and tk and prob2_interface commands ideally implemented via external function ?
- the rendering of strings is not yet optimal.