From Failure to Proof: The ProB Disprover for B and Event-B

Sebastian Krings(✉), Jens Bendisposto, and Michael Leuschel

Institut für Informatik, Universität Düsseldorf, Universitätsstr. 1,
40225 Düsseldorf, Germany
{krings,bendisposto,leuschel}@cs.uni-duesseldorf.de

Abstract. The ProB disprover uses constraint solving to find counter-examples for B proof obligations. As the ProB kernel is now capable of determining whether a search was exhaustive, one can also use the disprover as a prover. In this paper, we explain how ProB has been embedded as a prover into Rodin and Atelier B. Furthermore, we compare ProB with the standard automatic provers and SMT solvers used in Rodin. We demonstrate that constraint solving in general and ProB in particular are able to deal with classes of proof obligations that are not easily discharged by other provers and solvers. As benchmarks we use medium sized specifications such as landing gear systems, a CAN bus specification and a railway system. We also present a new method to check proof obligations for inconsistencies, which has helped uncover various issues in existing (sometimes fully proven) models.

1 Introduction and Motivation

Both the B-method [1] and its successor Event-B [2] are state-based formal methods rooted in set theory. They are used for the formal development of software and systems that are correct by construction. This usually involves formal proofs of different properties of the specification.

In former work [23] we described a disprover based on using ProB's constraint solver to automatically find counter-examples for given proof obligations and thus saving the user from spending time in a futile interactive proof attempt. Say that we have to prove that the goal G is a logical consequence of the hypotheses $H_1, \ldots, H_n$. The ProB disprover then tries to find a solution for the formula $H_1 \wedge \ldots \wedge H_n \wedge \neg G$. If it can find a solution, the proof cannot succeed and the solution is a counter-example.

In [23] we already made the observation that in some cases, namely if we neither encounter infinite sets nor deferred sets[1] whose cardinality is unbounded, the absence of a counter-example is actually a proof. We thus suggested as future work to implement an analysis that checks if the absence of a counter-example is

Parts of this research have been sponsored by the EU funded FP7 project 287563 (ADVANCE) and by an industrial project funded by Alstom.

[1] Deferred sets are sets which are not given upfront by enumerating their elements.

R. Calinescu and B. Rumpe (Eds.): SEFM 2015, LNCS 9276, pp. 199–214, 2015.
DOI: 10.1007/978-3-319-22969-0_15

a valid proof. This work has been finalized in the last year: ProB now keeps track of infinite enumeration, in particular the scope in which an infinite enumeration has occurred and whether a solution has been found or not. This enables our technique to detect if the search for a counter-example was exhaustive, i.e., we can now use ProB as a prover. Note that we go beyond the suggested future work of [23]: we allow variables with an infinite domain to occur, as long as they do not have to be enumerated exhaustively. We have also improved the core algorithm of [23] in various ways, by allowing to focus on selected hypotheses and by providing a way to detect inconsistencies in the hypotheses or potential bugs in the disprover. In this paper we have also conducted a thorough empirical evaluation, comparing our constraint-based proof with existing provers for B and Event-B. This study shows that the constraint-based proof fares surprisingly well for a variety of case studies.

2 Constraint-Based Proof Technique

In the following section we describe how ProB can be used as a prover inside Rodin [3] and Atelier B [12]. First, we provide a short introduction to the constraint solving capabilities of ProB in Sect. 2.1. Further technical details regarding ProB's kernel can be found in [21,22] or [20]. Following, Sect. 2.2 will outline how ProB was embedded into Rodin's proof architecture. Section 2.3 will explain the integration of ProB into Atelier B. Afterwards, in Sect. 2.4 we will show how ProB can be used to detect inconsistencies in the model.

2.1 ProB's Constraint Solving Kernel

The ProB constraint solver is based on CLP(FD)-style constraint-propagation [11], i.e., the variables of a B specification are annotated with possible values (e.g., in the form of intervals for integer variables). This information is propagated from one variable to another, e.g., if we know that x is in the range 0..8 and the predicate $x = y + 2$ holds, then y must be in the range $-2..6$. As a last resort, ProB enumerates undetermined variables when no further propagation is possible. While doing so, the solver tracks where and why enumeration occurs. It is able to distinguish between safe and unsafe enumerations, i.e., if all possible values of a variable have to be tried out or if a single solution is sufficient. This is done by observing the context[2] in which an enumeration occurs. Exhaustive enumeration can then be detected individually for each variable and later be transferred to the whole constraint if possible. Let us look at a few example constraints, where we suppose all free variables to be existentially quantified:

- $i \in \{1, 2, 1024, 2048\} \wedge i > 2 \quad \vdash_? \quad i \; mod \; 2 = 1$
  Here, we have the two hypotheses $i \in \{1, 2, 1024, 2048\}$ and $i > 2$ and we want to prove that $i \; mod \; 2 = 1$ is a logical consequence. Hence, we would construct the formula $i \in \{1, 2, 1024, 2048\} \wedge i > 2 \wedge \neg(i \; mod \; 2 = 1)$ and try

[2] This includes quantification, negation and arbitrarily nested combinations of them.

to find solutions for i. For this formula, ProB finds two solutions ($i = 1024$ and $i = 2048$) and no infinite enumeration has occurred (ProB has narrowed down the interval of i to 3..2048 before enumeration has started). As such, we can conclude that $G \equiv i \; mod \; 2 = 1$ is *not* a logical consequence of the hypotheses $H_1 \equiv i \in \{1, 2, 1024, 2048\}$ and $H_2 \equiv i > 2$.

- $i \in \{1, 2, 1024, 2048\} \wedge i > 2 \quad \vdash_? \quad i \; mod \; 2 = 0$
  For the opposite of the goal, i.e., $i \; mod \; 2 \neq 1$ or equivalently $i \; mod \; 2 = 0$, we construct the formula $i \in \{1, 2, 1024, 2048\} \wedge i > 2 \; \wedge \; \neg(i \; mod \; 2 = \; 0)$. In this case ProB finds no solution and no infinite enumeration has occurred. As such, we have proven that $i \; mod \; 2 = 0$ follows logically from $i \in \{1, 2, 1024, 2048\} \wedge i > 2$.
- $i > 20 \quad \vdash_? \quad i \; mod \; 2 = 1$
  If we want to prove that $(i \; mod \; 2 = 1)$ is a logical consequence of $i > 20$, we construct the formula $i > 20 \wedge \; \neg(i \; mod \; 2 = 1)$. ProB finds a solution $(i = 22)$, but infinite enumeration has occurred in the sense that the possible values of i lie in the interval 22..∞. However, in this context this is not an issue, as a solution has been found. As such, we can conclude that $i \; mod \; 2 = 1$ is not a logical consequence of $i > 20$.
- $i > 20 \quad \vdash_? \quad (i \; mod \; 2 = 0 \vee i \; mod \; 1001 \neq 800)$
  Finally, if we want to prove that $(i \; mod \; 2 = 0 \vee i \; mod \; 1001 \neq 800)$ is a logical consequence of $i > 20$, we get the formula $i > 20 \wedge \neg(i \; mod \; 2 = 0 \vee i \; mod \; 1001 \neq 800)$. Here ProB finds no solution, but an "enumeration warning" is produced. Indeed, the constraint solver has narrowed down the possible solutions for i to the interval 801..∞, but with the default search settings no solution has been found. Here, we cannot conclude that $i \; mod \; 2 = 0 \vee i \; mod \; 1001 \neq 800$ is a logical consequence of $i > 20$. Indeed, $i = 1801$ is a counter-example.[3]

2.2 Integration into Rodin for Event-B

When working on a proof obligation, Rodin keeps track of two sets of hypotheses: the set of *all* available hypothesis for the target goal and a *user-selected* subset. The idea is to be able to reduce the search space of the automatic provers by excluding irrelevant hypotheses. In the case of the ProB prover we could, for instance, get rid of hypotheses that are irrelevant for the proof but contain variables over infinite domains, deferred sets or complicated constraints.

This approach cannot lead to false positives, because limiting the number of available hypothesis cannot render a formerly unprovable sequent provable. However, disproving while omitting hypotheses can lead to false negatives if the hypotheses are too weak for a proof. For instance, say the goal G is $i \; mod \; 2 = 1$ and the hypotheses are $i \in \{1, 2, 3\}$ (H_1) and $i \neq 2$ (H_2). ProB will not find a counter-example for $H_1 \wedge H_2 \wedge \neg G$ but it will find a (false) counter-example for $H_1 \wedge \neg G$.

Figure 1 outlines how the disprover proceeds in more detail:

[3] Which ProB can find if we enlarge the default search space, e.g., by adding $i < 10000$ as additional constraint.

1. We first try to solve the predicate $H_1 \wedge ... \wedge H_m \wedge \neg G$, i.e., the negated goal together with *all* available hypotheses. If we find a solution, we report the proof obligation as *unprovable* and insert the counter-example inside the Rodin proof tree. If no counter-example is found and search was exhaustive, the initial sequent is *proven*, because no counter-example *exists*.
2. If the constraint solver is unable to prove or disprove the predicate in step 1, we reduce the number of hypotheses to the user-selected hypotheses and again look for a counter-example. The three possible outcomes are:
   - A contradiction is detected with the reduced set of hypotheses. This is still a valid *proof*, as removing hypotheses can only introduce further counter-examples but not remove them.
   - If we find a solution, we report a *possible* counter-example, but leave the proof obligation status as *unknown*. However, we do not interfere with the ongoing proof effort, as the proof obligation might still be provable using all hypotheses.
   - Otherwise we return without a result (status is *unknown*).

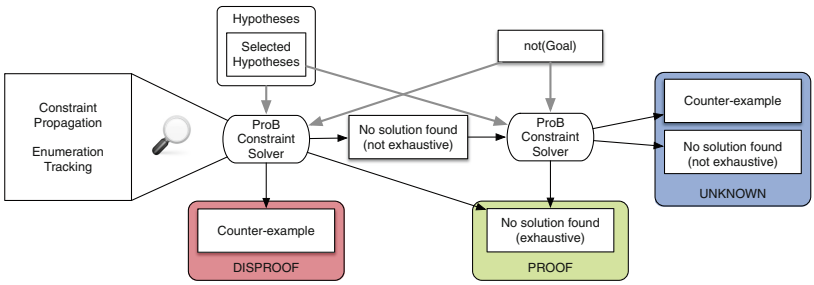


Fig. 1. Disproving algorithm

2.3 Integration into Atelier B for Classical B

The integration of ProB into Atelier B is closer to the original implementation of the disprover explained in [23].[4] Within Atelier B a proof obligation is translated into a B machine, where all hypotheses are put into the properties clause and the assertions clause contains an implication of the form $SelHyp \Rightarrow Goal$. Here, $Goal$ is the proof goal, and $SelHyp$ are the selected hypotheses. The latter are empty if `prob(0)` is called from Atelier B and contain all hypotheses H_1 which have a variable in common with $Goal$ if `prob(1)` is called. When `prob(2)` is called, Atelier B recursively adds all further hypotheses which have variables in common with H_1. The selection algorithm is the same that is used for the other Atelier B provers (e.g., `pp(0)`, `pp(1)`, `pp(2)`). It is also possible to specify a time-out `t` in milliseconds: `prob(n|t)`. Once the machine is constructed, Atelier B calls the command line version of ProB, which tries to find a counter-example

[4] This work was conducted in a joint project with ClearSy (Lilian Burdy, Etienne Prun) and funded by Alstom (Fernando Mejia).

to $SelHyp \Rightarrow Goal$ and writes the result to an intermediate file. The possible result values are very similar to above:

- no counter-example exists: the proof obligation is proven,
- no counter-example found (with reason being either time-out, deferred sets used or enumeration warning): the proof obligation status is unknown,
- counter-example found: the proof obligation status is still unknown, but not provable from the selected hypotheses.

2.4 Inconsistency Detection

After the algorithms outlined in Sects. 2.2 and 2.3 return a proof, a second phase can be triggered as outlined in Fig. 2: We try to find a proof for the negation of the goal. This time, we send $H_1 \wedge ... \wedge H_m \wedge G$ to the constraint solver. The result allows us to decide, whether the goal predicate G played a role in the original proof. If the negated goal can be proven as well, we detected a contradiction in the hypotheses. Contradicting hypotheses might occur due to an error in the model, in particular if they are detected at the root of the proof tree.[5] Hence, the user should be notified if they occur in a successful proof.

If contradicting hypotheses or disproven obligations have been found, ProB can afterwards compute the unsat core in order to provide smaller counter-examples and ease understanding of shortcomings in the underlying model. This helped us to identify the cause of several bugs in the Stuttgart 21 model and in one of the published landing gear case studies (see Sect. 3.2).

Furthermore, this two-phase analysis can be used to detect bugs in ProB: if the search for a counter-example fails to explore certain cases, it might be independent of the goal. Hence, we can detect if ProB correctly spots contradictions using crafted sequents. In fact, we did detect an error in a prototypical optimisation (common-subexpression elimination), which we did not use in this paper. We could even go further and apply other provers to the unsat core generated by ProB in order to validate a proof effort by a second toolchain.

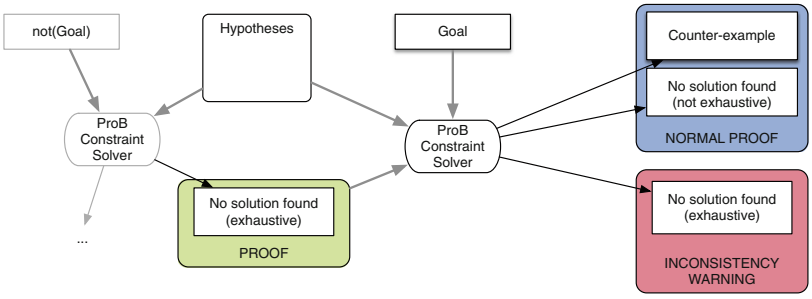


Fig. 2. Inconsistency detection

[5] Deeper within a proof, contradicting hypotheses can occur "naturally".

3 Empirical Evaluation and Comparison

In this section, we compare ProB to several other provers available for the Rodin platform [3], i.e., Rodin's automatic tactic and the SMT plug-in [14,15].

Our evaluation leads us to the following conclusions:

- In many cases ProB can discharge proof obligations that cannot be discharged by other provers. Each additional obligation that is discharged actually saves time and money.
- None of the provers can be replaced by the others.
- The performance of a prover is influenced by the surrounding tactic, including other provers. While the influence of a tactic on ProB is only marginal, it is quite strong for other provers.

3.1 Experimental Setup

For our experiments, we have used Rodin 3.1, version 2.1.0 of the Atelier B provers plugin and version 1.2.1 of the SMT plugin, with the bundled version 2.4.1 of CVC3 and the bundled development version of veriT. We have used a timeout of 5 s for each SMT solver, run in succession. ProB was used in version 1.5.1-beta1, connected through the disprover plugin version 3.0.8. Again, a timeout of 5 s was used for each constraint solving attempt with a maximum of two attempts per proof obligation (see Fig. 1). We used a global timeout of 25 s for a whole tactic.

All benchmarks were run on a MacBook Pro featuring a 2.6 GHz i7 CPU and 8 GB of RAM. We did not run proof attempts in parallel to avoid issues due to hyper-threading or scheduling. We developed an evaluation plugin[6] for the Rodin platform that applies the user- or pre-defined proof tactics to selected proof obligations.

We used the following combined tactics as they represent closely what can be utilized by end-users:

- The automatic tactic that comes with Rodin. It applies a number of rewriting rules and decision procedures to the proof tree. For instance, it checks if the goal is included in the set of hypotheses and thus discharged. The automatic tactic is applied until a fixpoint is reached or the process times out. This is the "Default Auto Tactic" of Rodin where the calls to PP and ML have been removed.
- In a second step, we used this tactic in its original state, i.e., with the PP and ML provers from Atelier B enabled.
- The SMT plugin [14,15] applies two different SMT solvers (veriT [10] and CVC3 [7]) to the original goal. We used the default SMT tactic that calls PP and ML as well.

[6] See https://github.com/wysiib/ProverEvaluationPlugin for sources and instructions.

– Finally, we add ProB to the tactic as well. It is applied to the goal before the other provers.

In addition we benchmarked the provers alone, i.e. without tactics. This gives us a better picture of the individual power of each prover.

– PP and ML from Atelier B together,
– SMT plugin on its own, using both veriT and CVC3, and
– ProB alone.

We used the following models for our benchmarks:

– Answers to the ABZ-2014 landing gear case study [9]. Beside our own version [18], we also used the three models by Su and Abrial [26], a model by André, Attiogbé and Lanoix [4], as well as a model by Mammar and Laleau [24].
– A model of the Stuttgart 21 Railway station interlocking by Wiegard, derived from Chap. 17 of [2] with added timing and performance modeling.
– A model of a controller area network (CAN) bus developed by Colley.
– A formal development of a graph coloring algorithm by Andriamiarina and Méry. The graphs to be colored are finite, but unbounded and not fixed in the model.
– A model of a pacemaker by Méry and Singh [25].

The models were selected so as to cover a variety of use cases. The landing gear model [18] contains mainly enumerated sets; hence we suspected ProB to perform well. We included several other versions of the case study to investigate how modelling style influenced prover performance. On the other end of the spectrum, the graph coloring model uses only deferred sets. Hence, we expected ProB not to perform well, as finite enumeration is not possible. The other models were expected to lie in between those extremes. We do not claim that our selection is representative. Indeed, we could have selected more models using (mostly) deferred sets; but this would have just confirmed that ProB's prover is disabled for proof obligations involving deferred sets.

For raw data and additional visualizations see http://www.stups.hhu.de/ProB/index.php5/Sefm2015. Rodin is available on http://www.event-b.org. The provers are available from update sites included in Rodin.[7]

3.2 Results

The benchmark results for the tactics can be found in Tables 1 and 2 and Figs. 3, 4 and 5, while the results for the provers alone are in Table 3 and part (b) of Fig. 3. Table 1 shows the total number of proof obligations discharged, as well as the percentage of proof obligations discharged using ML/PP together with SMT and in the last column the percentage discharged by using these two proof tactics together with the ProB disprover. Each Venn diagram shows how many

[7] For a standalone version of ProB see http://www.prob2.de.

Table 1. Benchmark results: Discharged Event-B proof obligations

Model	# POs	Tactic alone	+ML/PP	+ML/PP+ SMT	+ML/PP+ SMT+ProB
Landing gear system 1, Su et al.	2328	2022	2190	2303	2306
Landing gear system 2, Su et al.	1188	817	915	1169	1173
Landing gear system 3, Su et al.	341	134	152	205	262
CAN bus, Colley	534	289	398	403	388
Graph coloring, Andriamiarina et al.	254	122	166	170	169
Landing gear system, Hansen et al.	74	64	65	67	74
Landing gear system, Mammar et al.	433	218	297	381	397
Landing gear system, Andre et al.	619	180	214	319	450
Pacemaker, Neeraj Kumar Singh	370	258	354	364	369
Stuttgart 21 interlocking, Wiegard	202	37	33	97	147

Table 2. Benchmark results: Event-B Average Runtimes (in seconds/po)

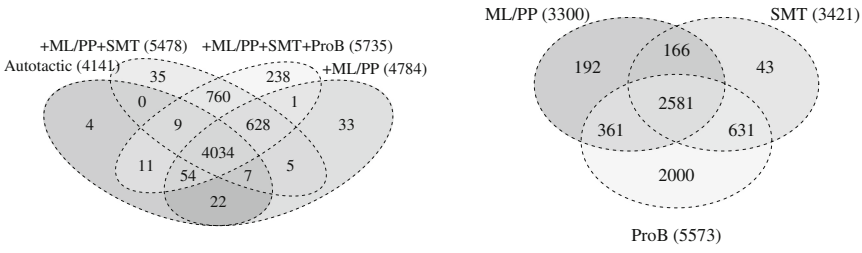
Model	Tactic alone	+ML/PP	+ML/PP+ SMT	+ML/PP+ SMT+ProB
Landing gear system 1, Su et al.	0.23	0.35	0.3	0.55
Landing gear system 2, Su et al.	0.34	0.64	0.74	0.79
Landing gear system 3, Su et al.	8.29	9.71	11.08	6.01
CAN bus, Colley	5.29	5.93	6.03	7.13
Graph coloring, Andriamiarina et al.	1.48	2.56	7.44	8.04
Landing gear system, Hansen et al.	0	2.1	2.7	0.2
Landing gear system, Mammar et al.	1.68	2.02	2.05	2.39
Landing gear system, Andre et al.	11.64	11.89	11.92	7.01
Pacemaker, Neeraj Kumar Singh	0	0.1	0.04	0.4
Stuttgart 21 interlocking, Wiegard	11.7	13.26	13.2	9.84

Table 3. Results of running provers alone (without pre-processing by Rodin)

Model	# POs	ML/PP	SMT	ProB	
				prove	disprove
Landing gear system 1, Su et al.	2328	1396	1477	2311	0
Landing gear system 2, Su et al.	1188	341	567	1176	0
Landing gear system 3, Su et al.	341	99	146	290	0
CAN bus, Colley	534	481	282	276	0
Graph coloring, Andriamiarina et al.	254	90	97	0	0
Landing gear system, Hansen et al.	74	70	59	74	0
Landing gear system, Mammar et al.	433	227	257	400	0
Landing gear system, Andre et al.	619	189	268	567	5
Pacemaker, Neeraj Kumar Singh	370	356	224	354	0
Stuttgart 21 interlocking, Wiegard	202	51	44	125	2

proof obligations are discharged by which prover. Table 2 shows the runtimes of the different provers for all proof obligations and for discharged proof obligations individually. Note that for the Stuttgart 21 model and the Andre et al. model, ProB found several unprovable proof obligations, i.e., errors in the model as can be seen in Table 3. E.g., for Stuttgart 21 ProB found a counter-example for two proof obligations, while it found five counter-examples in the landing gear model. This is very useful feedback to the developer of the model, and the initial purpose of the ProB disprover.

The diagram in Fig. 3 shows the gain of using ProB in addition to the other decision procedures. Compared to the SMT Tactic, adding ProB leads to an additional 304 (238+1+11+54) proof obligations being discharged. However, due to the time consumption by ProB, 47 (35+7+5) proof obligations cannot be discharged anymore. With a higher time-out, these could again be proven. The second diagram in Fig. 3 shows how the individual provers alone contribute: Each



(a) Tactics (b) Provers alone

Fig. 3. Visualization of the full benchmark results

of them has a set of proof obligations that cannot be solved by any of the others (192 for ML/PP, 43 for SMT and 2000 for ProB).

Except for the graph coloring algorithm ProB performs surprisingly well. The graph coloring algorithm uses unbounded sets, meaning that some of the proof obligations cannot be proven using constraint solving and enumeration.

As can be seen in Table 1, adding ProB improves the results of automatic proving for all other models. In some cases, such as the landing gears, the improvement is substantial (cf., Fig. 4). The reason for the rather big improvement is that these models only use enumerated sets, booleans and integers as base types. In these cases ProB can produce elaborate case distinctions, combined with constraint solving to narrow down the search space. This type of proof is not supported by the classical provers ML and PP. Generally, the proof obligations that pose problems to ProB are certain well-definedness proof obligations. For instance, function application requires to proof that the parameter is in the domain of the function. Usually this leads to expensive enumeration of the possible parameter values.

For some of the models, using ProB slows down the prove process. As shown in Table 2 ProB's runtime is above average for some proof obligations, while it considerably speeds up other proof attempts. We suspect that this is due to the multiple constraint solver calls ProB performs on different sets of hypotheses as shown in Fig. 1. Also, ProB is looking for proofs and counter-examples. This often means that ProB will continue the computation, even after it has realized that no proof is possible (in the hope of finding a counter-example).

It is also interesting to note that, on their own, the ML and PP provers do not fare quite so well as in Table 1: they require pre-processing and tactic support to be fully effective: See Table 3 containing the results without any pre-processing.

All models except the Landing Gear System by Mammar et al. show the same behavior: The rate of discharged proof obligations drops significantly if Rodin's default tactics are not applied. Adding SMT solvers or ProB does not replace the tactics either.

In contrast, the model by Mammar et al. shows the opposite behavior: without pre-processing, more proof obligations can be discharged. This is probably due to the timeouts leaving less time for the actual prover, if we include a pre-processing phase. In future, we want to examine whether better pre-processing can improve the performance of the ProB disprover.

The same effect can be observed in Table 4. Here, the performance of the provers on different kinds of proof obligations is given. For most kinds, ProB does perform quite well when compared to ML/PP and the SMT solvers, especially for guard strengthening proofs, theorem proofs and well-definedness proofs. For feasibility and finiteness proof obligations, on the other hand, ProB fares less well.

Unexpected Performance of SMT. To our surprise, the SMT solvers did not perform as well as we expected when compared to ProB. For certain kinds like guard strengthening or initialization in Table 4, the SMT solvers prove less proof obligations than ML/PP or ProB. We suspect that this is due to the translation from Event-B to SMT-LIB:

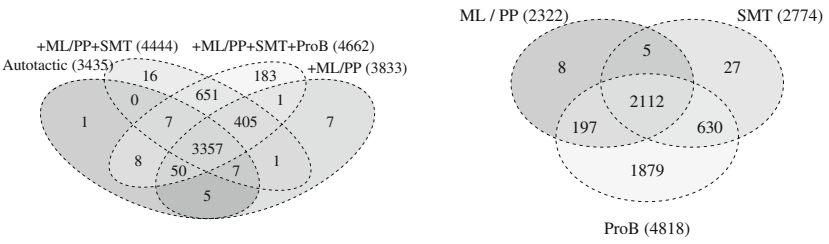
Table 4. Performance of provers on different kinds of proof obligations

Kind of PO	# POs	ML/PP	SMT	ProB
Feasibility of non-det. action	59	53 (89.8 %)	40 (67.8 %)	44 (74.6 %)
Guard strengthening	300	27 (9.0 %)	13 (4.3 %)	258 (86.0 %)
Invariant preservation	4938	2877 (58.3 %)	3111 (63.0 %)	4488 (90.9 %)
Action simulation	153	118 (77.1 %)	108 (70.6 %)	134 (87.6 %)
Theorem	97	13 (13.4 %)	29 (29.9 %)	66 (68.0 %)
Well definedness	779	200 (25.7 %)	109 (14.0 %)	570 (73.2 %)
	6326	3288 (52.0 %)	3410 (53.9 %)	5560 (87.9 %)

- The λ-based approach [14,15] does not support sets of sets. Thus, a whole class of proof obligations cannot be solved by it. Therefore, the SMT plugin uses the second approach presented in [14,15] as the default:
- The *ppTrans* approach [19] translates set theory to predicate calculus. The resulting SMT-LIB problem is then enriched by the predicate calculus version of certain set-theoretic axioms.

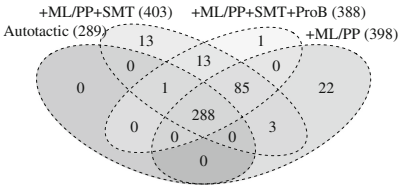
Newer releases of SMT-Solvers like CVC4 [5] support finite sets natively as an extension to the SMT-LIB language [27]. Thus, certain classes of proof obligations could be passed to the SMT-Solvers directly instead using one of the approaches mentioned above. We assume that this would increase the number of proof obligations that could be discharged successfully. In summary, while the SMT plugin has been very successful, we recommend critically examining the current SMT-LIB translation and believe there is scope for considerable improvement by using an alternate translation.

Inconsistency in Hypothesis Detection. The inconsistency detection of Sect. 2.4 found also various contradictions in the theorems (at lower refinement levels) of the Stuttgart 21 model. It also highlighted an issue in the first development of the ABZ landing gear from [26]. The ProB disprover was flagging, e.g., the proof obligation `treat_hndl_up_112/inv1/INV` in the machine LPN4 as

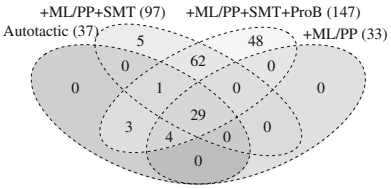


(a) Tactics (b) Provers alone

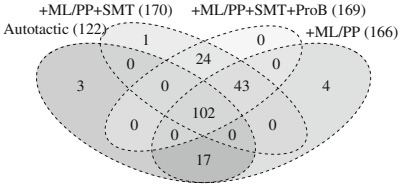
Fig. 4. Visualization of the benchmark results. Part 1: Landing gear systems



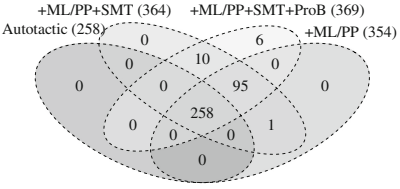
(a) Colley, CAN Bus



(b) Wiegard, Stuttgart 21



(c) Andriamiarina and Mèry, Graph Coloring Algorithm



(d) Singh, Pacemaker

Fig. 5. Visualization of the benchmark results. Part 2: Miscellaneous models

containing a contradiction in the hypothesis. The ProB unsat core algorithm found out the following root cause:

```
 close_EV = FALSE & open_EV = FALSE & door = op2cl &
((open_EV = FALSE & close_EV = FALSE) => door = cl)
```

The seen context LPNC0 contains the axiom `partition(D, {cl}, {cl2op}, {op2cl}, {op})`.[8] The first line comes from the guard of the event `treat_hndl_up`, the second line is the invariant `inv1` from LPN4. In other words, the disprover has detected that this event can never be executed given the invariant. A similar issue was detected for several other events.[9]

When (not) to Use the ProB Disprover. In summary, we present the following insights on when to use the ProB disprover (+) and when not to (−):

+ Used solely as a disprover, ProB can prevent futile interactive proof attempts. This is always worthwhile.
+ The inconsistency detection is very useful for finding subtle modelling errors.

[8] For technical reasons this axiom is not yet included in the unsat core; partition axioms are never removed from the core by the current algorithm.

[9] In LPN4 of [26]: `treat_hndl_up_122`, `treat_hndl_up_132`, `treat_hndl_dn_112`, `treat_hndl_dn_122`, `treat_hndl_dn_132`.

+ On models such as the ABZ landing gear models (Fig. 4), which rely heavily on enumerated sets, booleans and/or bounded integers as base types, ProB performs very well.
+ The Stuttgart 21 model shows that explicit data, e.g., track layouts or time tables, can often be used effectively by ProB. Often, this results in a proof by an elaborate case distinction.
+ ProB performs reasonably well on unbounded intervals, when interval reasoning can be applied. This occurs for example in the lower refinement levels of the ABZ case study models or the pacemaker model.
– As soon as the proof goal references deferred sets (e.g., in the graph coloring model), no proof can be done by construction of the disprover (see Fig. 1).
– When unbounded datastructures are used, ProB cannot exhaustively enumerate cases and is much less powerful. This happens for example in the CAN bus model, that represents a buffer as an unbounded partial function from $\mathbb{N}$ to $\mathbb{Z}$.

4 Discussion and Conclusion

One motivation for the experiments conducted in this paper was the empirical evaluation of our constraint solver, more precisely its capability to detect inconsistencies (a successful proof with the disprover requires finding a contradiction without enumerating unbounded variables; see Fig. 1). Finding inconsistencies is important for many other features of ProB, e.g., detecting disabled events during animation. Furthermore, it is useful for constraint-based validation, such as deadlock checking [17], where it avoids the constraint solver exploring infeasible alternatives. In the context of model-based testing, it enables ProB to detect uncoverable alternatives, and not spend time trying to find test cases for them.

An important issue is the soundness of the ProB disprover. In [8] we have presented the various measures we are taking to validate ProB's results in general. In addition, we have developed an SMT-LIB [6] importer for ProB and have applied our disprover to a large number of SMT-LIB benchmarks, checking that no "false theorems" are proven. For this paper, we have also double checked many of the proof obligations which were only provable by ProB, to ensure that they are indeed provable. As the Venn diagrams in Figs. 4 and 5 show, a large number of proof obligations can be proven by two or even three different provers. As the three provers rely on completely different technologies and have been developed by independent teams, we can have a very high confidence that those proof obligations are indeed provable.

We have demonstrated that constraint-based proof in general, and ProB in particular, is capable of discharging proof obligations that currently cannot be proven using Rodin's auto tactic and the SMT solvers. Our prover typically deals well with a different kind of proof obligations than the other provers, and is thus an orthogonal extension rather than a replacement. Rodin's auto tactic performs well in the realm of set theoretic constructs and relational expressions, some of which cannot be easily represented in the SMT syntax. SMT on the other

hand performs well on arithmetic expressions, where the auto tactics often fail. ProB finally covers predicates over enumerated sets, explicit data and explicit computations and has a good support for integer arithmetic over finite domains.

However, for models which make heavy use of deferred sets, such as the graph coloring algorithm model, the ProB disprover can currently mainly play its role as disprover. More precisely, for any proof obligation which involves deferred sets and where no precise value of the cardinality of the deferred set is known, the disprover can only return either a counter-example or the result "unknown".

In future, we plan to improve the treatment of deferred sets in ProB, and to have the constraint solver determine the cardinalities of those sets while solving.

We also plan to conduct experimental evaluations for ProB within Atelier B, and compare with efforts such as [13] or the BWare project [16]. First results on industrial case studies within Alstom are already very promising.

We think that the ProB Disprover is a valuable extension to the existing set of provers, because it can increase the number of proof obligations that are automatically discharged, thus saving time and money. Overall, the outcome of the empirical evaluation was a positive surprise, as ProB's main domain of application is finding concrete counter-examples, not discharging proof obligations. In particular, the fact that the number of discharged proof obligations (5573 in Fig. 3 (b)), for the models under consideration, is better than that of the two SMT solvers of the SMT plugin (3421 in Fig. 3 (b)) was completely unexpected.

Acknowledgements. We would like to thank the various developers for giving us access to their Event-B models, and for discussions and feedback: Jean-Raymond Abrial, Andre, Attiogbe, John Colley, Régine Laleau, Luis-Fernando Mejia, Lanoix, Amel Mammar, Dominique Méry, Neeraj Kumar Singh, Wen Su.

References

1. Abrial, J.-R.: The B-Book. Cambridge University Press, New York (1996)
2. Abrial, J.-R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press, New York (2010)
3. Abrial, J.-R., Butler, M., Hallerstede, S., Voisin, L.: An open extensible tool environment for Event-B. In: Liu, Z., Kleinberg, R.D. (eds.) ICFEM 2006. LNCS, vol. 4260, pp. 588–605. Springer, Heidelberg (2006)
4. André, P., Attiogbé, C., Lanoix, A.: Modelling and analysing the landing gear system: a solution with Event-B/Rodin. http://www.lina.sciences.univ-nantes.fr/aelos/softwares/LGS-ABZ2014/index.php. Solution to ABZ-2014, Accessed: 17 March 2014
5. Barrett, C., Conway, C.L., Deters, M., Hadarean, L., Jovanović, D., King, T., Reynolds, A., Tinelli, C.: CVC4. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 171–177. Springer, Heidelberg (2011)
6. Barrett, C., Stump, A., Tinelli, C.: The SMT-LIB Standard: Version 2.0. In: Gupta, A., Kroening, D. (eds.) Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (2010)

7. Barrett, C.W., Tinelli, C.: CVC3. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 298–302. Springer, Heidelberg (2007)
8. Bendisposto, J., Krings, S., Leuschel, M.: Who watches the watchers: Validating the prob validation tool. In: Proceedings of the 1st Workshop on Formal-IDE, EPTCS XYZ, 2014, Electronic Proceedings in Theoretical Computer Science (2014)
9. Boniol, F., Wiels, V.: The landing gear system case study. In: Boniol, F., Wiels, V., Ait Ameur, Y., Schewe, K.-D. (eds.) ABZ 2014. CCIS, vol. 433, pp. 1–18. Springer, Heidelberg (2014)
10. Bouton, T., Caminha B. de Oliveira, D., Déharbe, D., Fontaine, P.: `veriT`: an open, trustable and efficient SMT-Solver. In: Schmidt, R.A. (ed.) CADE-22. LNCS, vol. 5663, pp. 151–156. Springer, Heidelberg (2009)
11. Carlsson, M., Ottosson, G.: An open-ended finite domain constraint solver. In: Hartel, P.H., Kuchen, H. (eds.) PLILP 1997. LNCS, vol. 1292. Springer, Heidelberg (1997)
12. ClearSy. Atelier B, User and Reference Manuals. Aix-en-Provence, France (2009). http://www.atelierb.eu/
13. Conchon, S., Iguernelala, M.: Tuning the Alt-Ergo SMT solver for B proof obligations. In: Ait Ameur, Y., Schewe, K.-D. (eds.) ABZ 2014. LNCS, vol. 8477, pp. 294–297. Springer, Heidelberg (2014)
14. Déharbe, D.: Automatic verification for a class of proof obligations with SMT-solvers. Proceedings ASM **2010**, 217–230 (2010)
15. Déharbe, D., Fontaine, P., Guyot, Y., Voisin, L.: SMT solvers for rodin. In: Derrick, J., Fitzgerald, J., Gnesi, S., Khurshid, S., Leuschel, M., Reeves, S., Riccobene, E. (eds.) ABZ 2012. LNCS, vol. 7316, pp. 194–207. Springer, Heidelberg (2012)
16. Delahaye, D., Dubois, C., Marché, C., Mentré, D.: The `BWare` project: building a proof platform for the automated verification of `B` proof obligations. In: Ait Ameur, Y., Schewe, K.-D. (eds.) ABZ 2014. LNCS, vol. 8477, pp. 290–293. Springer, Heidelberg (2014)
17. Hallerstede, S., Leuschel, M.: Constraint-based deadlock checking of high-level specifications. TPLP **11**(4–5), 767–782 (2011)
18. Hansen, D., Ladenberger, L., Wiegard, H., Bendisposto, J., Leuschel, M.: Validation of the ABZ landing gear system using ProB. In: Boniol, F., Wiels, V., Ait Ameur, Y., Schewe, K.-D. (eds.) ABZ 2014. CCIS, vol. 433, pp. 66–79. Springer, Heidelberg (2014)
19. Konrad, M., Voisin, L.: Translation from set-theory to predicate calculus. Technical report, ETH Zurich (2011)
20. Leuschel, M., Bendisposto, J., Dobrikov, I., Krings, S., Plagge, D.: From animation to data validation: the prob constraint solver 10. In: Boulanger, J.-L. (ed.) Formal Methods Applied to ComplexSystems: Implementation of the B Method, Chap. 14, pp. 427–446. Wiley ISTE, Hoboken (2014)
21. Leuschel, M., Butler, M.: ProB: A model checker for B. In: Araki, K., Gnesi, S., Mandrioli, D. (eds.) FME 2003. LNCS, vol. 2805. Springer, Heidelberg (2003)
22. Leuschel, M., Butler, M.: ProB: An automated analysis toolset for the B method. Software Tools for Technology Transfer (STTT) **10**(2), 185–203 (2008)
23. Ligot, O., Bendisposto, J., Leuschel, M.: Debugging Event-B Models using the ProB Disprover Plug-in. In: Proceedings AFADL 2007, June (2007)
24. Mammar, A., Laleau, R.: Modeling a landing gear system in Event-B. In: Boniol, F., Wiels, V., Ait Ameur, Y., Schewe, K.-D. (eds.) ABZ 2014. CCIS, vol. 433, pp. 80–94. Springer, Heidelberg (2014)
25. Méry, D., Singh, N.K.: Formal specification of medical systems by proof-based refinement. ACM Trans. Embed. Comput. Syst. **12**(1), 15:1–15:25 (2013)

26. Su, W., Abrial, J.-R.: Aircraft landing gear system: approaches with Event-B to the modeling of an industrial system. In: Boniol, F., Wiels, V., Ait Ameur, Y., Schewe, K.-D. (eds.) ABZ 2014. CCIS, vol. 433, pp. 19–35. Springer, Heidelberg (2014)
27. Weissenbacher, G., Kröning, D., Rümmer, P.: A proposal for a theory of finite sets, lists, and maps for the smt-lib standard. In: Proceedings of the 7th International Workshop on Satisfiability Modulo Theories, SMT 2009 (2009)