Turning Failure into Proof: Evaluating the ProB Disprover

Sebastian Krings, Jens Bendisposto and Michael Leuschel

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

Abstract. The ProB disprover uses constraint solving to try and find counter examples to 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 compare the ProB Prover with the standard automatic provers for B and Event-B, such as ml, pp and the Rodin SMT plug-in. We demonstrate that ProB is able to deal with classes of proof obligations that are not easily discharged by other provers. As benchmarks we use medium sized specifications such as solutions to the ABZ 2014 case study, a CAN bus specification and a railway system.

1 Introduction and Motivation

Both the B-method and its successor Event-B [1] 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. The proof obligations often include set theoretic theorems and claims. Many provers such as "ml" of Atelier-B are able to discharge certain proof obligations automatically. In former work [13] we already 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.

We made the observation that in some cases, namely if we never encounter infinite sets nor deferred sets[1] whose cardinality remains unbounded, the absence of a counter example is actually a proof. In [13] we thus suggested as future work to implement an analysis that checks if the absence of a counter example is a valid proof. This work has been finalized in the recent months: ProB now keeps track of infinite set 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.

** Part of this research has been sponsored by the EU funded FP7 project 287563 (ADVANCE).

[1] Deferred sets are sets which are not given upfront by enumerating their elements. They are unbound sets which can become bounded by further constraints.

In [13] we have also identified the need to empirically evaluate the disprover. In this paper we will focus on this research goal: the empirical evaluation of our constraint-based approach to checking proof obligations, in particular when compared to the existing provers available for B.

2 Technique

When working on a proof obligation, Rodin keeps track of two sets of hypotheses: the set of all hypothesis available to proof the target goal and a user-selected subset. The idea behind this is to be able to shrink the search space of automatic provers by omitting hypothesis that should not be used in a proof attempt. 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 can not lead to false positives because limiting the number of available hypothesis can not 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. In order not to confuse the user with invalid counter-examples, we only try to disprove a sequent using all hypotheses.

Figure 1 outlines how we proceed:

1. We 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 a counter-example to Rodin and show it inside the proof tree as shown in Figure 2. If a contradiction is detected, either by analyzing the predicate or by enumerating exhaustively without finding a solution, the initial sequent is proven, because no counter-example exists.
2. If the constraint solver is unable to prove or disprove the predicate, we reduce the number of hypotheses to the user-selected hypotheses. After reducing to a subset of the hypotheses we will not report counter examples to avoid false negatives as discussed:
   - A contradiction detected with the reduced set of hypotheses is still a valid proof as reducing the number of hypotheses might introduce further counter-examples but not remove them.
   - If we find a solution, we report a possible counter-example. However, we do not prevent a following proof effort.
   - Otherwise we return without a result.

The ProB constraint solver supports sets in different ways. First of all, all set theoretic features of the B language are available to formulate constraints. This includes, among others, subset, strict subset, membership, union and intersection as well as cardinality of sets.

The solver is based on constraint-propagation and resorts to enumeration if no further propagation is possible. While doing so, the solver tracks where and why the elements of a set have to be enumerated. It is able to distinguish

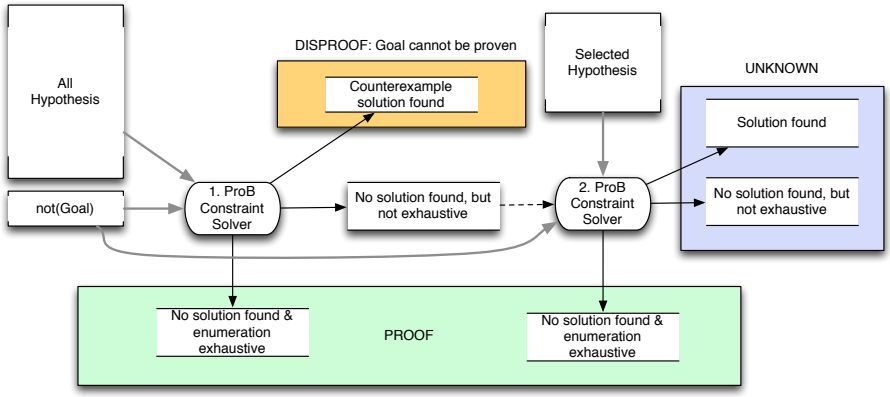


Fig. 1. Disproving Algorithm

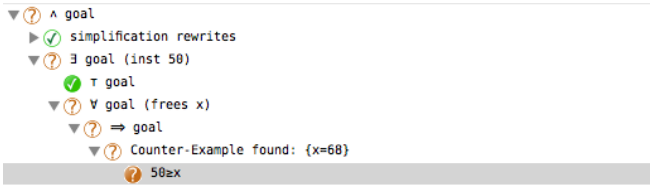


Fig. 2. Counter-Example inside the Rodin Proof Tree

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 examples, where we suppose all free variables to be existentially quantified:

- $i \in \{1, 2, 1024, 2048\} \wedge i > 2 \ \wedge \ \neg(i \ mod \ 2 \neq 0)$:
  ProB finds two solutions ($i = 1024$ and $i = 2048$) and no infinite enumeration has occurred as ProB has narrowed down the interval of i to 3..2048 before enumeration has started. As such, we can conclude that $G = i \ mod \ 2 \neq 0$ is *not* a logical consequence of the hypotheses $H_1 = i \in \{1, 2, 1024, 2048\}$ and $H_2 = i > 2$. The same solutions could be found by a CLP(FD) query.[3]
- $i > 20 \ \wedge \ \neg(i \ mod \ 2 \neq 0)$:
  ProB finds a solution ($i = 22$), but infinite enumeration has occurred in

[2] This includes quantification, negation and arbitrarily nested combinations of them.
[3] SICStus Prolog: `list_to_fdset([1,2,1024,2048],FDSet), I in_set FDSet, I #>2, I mod 2 #\= 0 #<=> 0, labeling([],[I]).`

the sense that the possible values of i lie in the interval $22..\infty$. However, in this context this is not an issue, as a solution has been found. As such, we can conclude that $i \; mod \; 2 \neq 0$ is not a logical consequence of $i > 20$. This time there is no CLP(FD) query that returns a solution. As there is no finite domain attached to i, labeling can not be performed. In contrast, ProB is able to (partially) enumerate the infinite domain of i in order to find a solution.

- $i \in \{1, 2, 1024, 2048\} \wedge i > 2 \; \wedge \; \neg(i \; mod \; 2 = 0)$:
  ProB finds no solution and no infinite enumeration has occurred. As such, we have proven that $i \; mod \; 2 = 0$ follows logically from $i \in \{1, 2, 1024, 2048\} \wedge i > 2$. A CLP(FD) query also confirms, that there is no solution.
- $i > 20 \; \wedge \; \neg(i \; mod \; 2 = 0 \vee i \; mod \; 1001 \neq 800)$:
  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 $801..\infty$, 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. [4] Again, CLP(FD) is unable to solve the query due to the infinite domain of i.

As mentioned in the introduction, we will not go into further technical details in this paper.

3 Empirical Evaluation and Comparison

For our empirical evaluation we compare ProB to several other provers available for the Rodin platform [2], i.e., Rodin's automatic tactic and the SMT plug-in [9,10]. Our comparison shows the benefit gained from using ProB as a prover. Each additional obligation that is discharged in this comparison actually saves time and money.

3.1 Experimental Setup

- The automatic tactic applies a number of rewriting rules and decision procedures to the proof tree. For instance, a decision procedure checks if the goal is listed in the set of hypotheses and thus discharged. It also uses the PP and ML provers from AtelierB. The automatic tactic is applied until a fixpoint is reached.
- The SMT plug-in [9,10] applies two different SMT solvers (veriT [8] and CVC3 [5]) to the original goal, after some pre-processing.
- The disprover tactic applies three trivial decision procedures (check if goal is $\top$, check if the hypotheses contain $\bot$ and if the goal appears in the list of hypotheses). Afterwards the disprover is applied to the goal.

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

For our experiments, we have used Rodin 2.8, version 2.0.1 of the Atelier B provers plugin and version 1.1.0.e126305 of the SMT Solvers Plugin, with the bundled version 2.4.1 of CVC3 and the bundled development version of veriT. We have used a timeout of 1 second for each SMT solver, run in succession. ProB was used in version 1.3.7-beta10, connected through the disprover plugin version 2.4.4.201403152244. Again, a timeout of 1 second was used for each constraint solving attempt with a maximum of two attempts per proof obligation (see Figure 1). Both the CLP(FD) and the CHR-based solvers of ProB were activated. All benchmarks were run on a MacBook Pro featuring a 2,6 GHz Intel Core i7 CPU and 8 GB 1600 MHz DDR3 memory. The CPU includes 4 cores, yet we ran at most two proof attempts at once. We used a plugin[5] for the Rodin platform that applies the user- or pre-defined proof tactics to selected proof obligations.

As models for our benchmarks we used the following models:

- Answers to the ABZ-2014 case study [7]. The case study models a landing gear system. Beside our own version [12], we also used the three models by Su and Abrial [17], a model by André, Attiogbé and Lanoix [3] as well as a model by Mammar and Laleau [14].
- A model of the Stuttgart 21 Railway station interlocking by Wiegard, derived from the interlocking model in chapter 17 of [1] with added timing and performance modeling.
- A model of a controller area network (CAN) bus. A CAN Bus is used in vehicles for direct communication between components without a central processor. The model was 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 Neeraj Kumar Singh [15].
- A model formalizing a number of set theoretical laws; generated for regression tests.

3.2 Results and Analysis

The results of the benchmarks are shown in Table 1 and Figures 4 and 5. Table 1 shows the total number of proof obligations discharged, as well as a column showing 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 proof obligations are discharged by which prover. Except for the graph coloring algorithm and the set laws example ProB performs surprisingly well.

The graph coloring algorithm uses unbounded sets, that means that some of the proof obligations cannot be proven using constraint solving and enumeration.

[5] The source code of the plugin can be found at `https://github.com/wysiib/ProverEvaluationPlugin`. An update site for installation inside Rodin is available at `http://nightly.cobra.cs.uni-duesseldorf.de/rodin_provereval/`.

Table 1. Benchmark results: proof obligations discharged for various developments

Model	# POs	ML/PP	SMT	ProB	% excl. ProB	% incl. ProB
Landing Gear System 1, Su, et. al.	2328	2171	2312	2275	99.57	99.79
Landing Gear System 2, Su, et. al.	1188	845	1140	1165	97.22	99.49
Landing Gear System 3, Su, et. al.	341	201	187	251	74.78	85.63
CAN Bus, Colley	542	501	490	320	95.02	95.2
Graph Coloring, Andriamiarina, et. al.	254	226	116	19	96.06	96.06
Landing Gear System, Hansen, et. al.	74	72	63	74	100	100
Landing Gear System, Mammar, et. al.	433	347	385	334	95.15	98.15
Landing Gear System, Andre, et. al.	619	466	400	459	79.81	90.63
Pacemaker, Neeraj Kumar Singh	370	360	358	351	98.38	100
Stuttgart 21 interlocking, Wiegard	202	57	94	184	55.45	93.56
Set laws, Leuschel	67	67	67	62	100	100

ProB is only able to prove some very trivial invariants, such as $\forall n \cdot n \in S \Leftrightarrow n \in dom(R)$ for the initialization $S := \emptyset \mid\mid R := \emptyset$. The other unfavorable case, the set laws, is very similar. The model contains five invariants that contain infinite sets and cannot be proven using ProB.

As can be seen in the last two columns of Table 1, ProB improves the results of automatic proving in all other developments. In some cases, such as the cases shown in Figure 5(b), 4(c) and 4(e) the improvement is rather big. The reason for the big improvement is that these models only use enumerated sets and integers. 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 the ProB disprover are well-definedness proof obligations.

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 specific pre-processing to be effective. In Table 2 are the results for two models without any pre-processing (except for collecting hypotheses using the lasso tool):

Table 2. Results without pre-processing by Rodin

Model	# POs	Provers			SMT	ProB
		ML	PP	ML/PP		
Landing Gear System, Mammar et al	433	284	127	284	385	341
Landing Gear System, Andre et al	619	560	81	567	400	511
Pacemaker, Neeraj Kumar Singh	370	344	187	352	328	350

As can be seen, for the first model ML on its own discharges just 284 (45.9 %) proof obligations. PP discharges just 127 (20.5 %) of the proof obligations. The SMT solver also benefits considerably from pre-processing: without it, it discharges "just" 385 (62.2 %) of the proof obligations. The third model shows

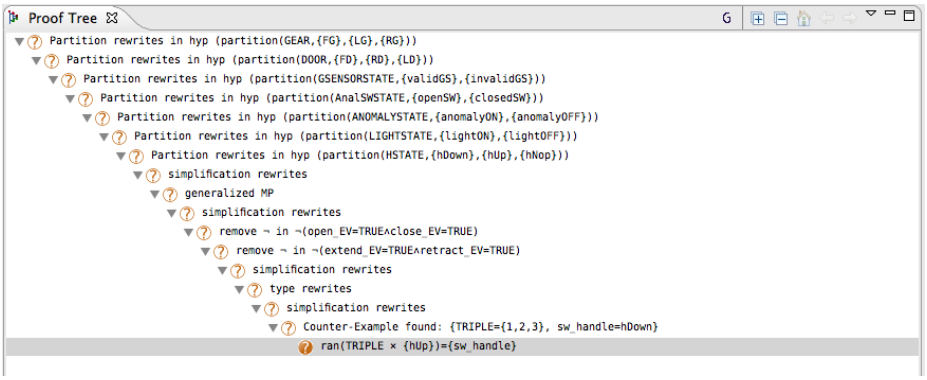


Fig. 3. Counter-Example for proof obligation of Landing Gear System by Andre et al.

quite similar declines if pre-processing is omitted. However, the second model shows the opposite behavior: without pre-processing, more proof obligations can be discharged. This is 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 first landing gear system by Andre et al. contains unprovable proof obligations, where the disprover finds counter examples (e.g., the proof obligation `cockP_handleUp/onHand/INV` in the model `LandingSysDP_SWITCH_A`). This is very useful feedback to the developer of the model, and the initial purpose of the ProB disprover. Figure 3 shows the counter-example inside the Rodin proof tree.

Finally, for the Landing Gear System by Mammar et al., the developers had trouble discharging a few proof obligations using the other provers (including the SMT plugin). The ProB disprover was able to discharge them; one of the proof obligations was a well-definedness proof obligation.

4 Discussion and Conclusion

A secondary 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 an inconsistency without enumerating unbounded variables; see Fig. 1). Finding inconsistencies is important for detecting disabled events during animation, and more importantly for constraint-based validation, such as constraint-based deadlock checking [11]: it avoids the constraint solver exploring unsuccessful alternatives. In the context of model-based testing, it enables one to detect uncoverable alternatives, and not spending time trying to find test cases to cover them.

One important issue is the soundness of the ProB disprover. In [6] we have presented the various measures we are taking to validate ProB's results. In

addition, we have developed a SMT-LIB [4] 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 Figures 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 ProB is capable to discharge 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 colouring algorithm model (see Table 1), 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. This should also enable the disprover to act as a prover for more proof obligations involving deferred sets.

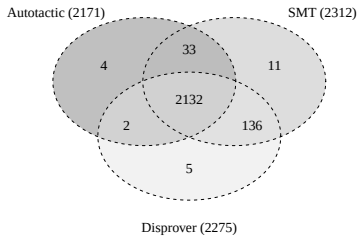
We think that the ProB Disprover is a valuable extension to Rodin's set of provers, because it can increase the number of proof obligations that are automatically discharged, thus saving time and money. Overall, the outcome of the empirical evaluation was a positive surprise, as ProB's main domain of application is finding concrete counter examples, not discharging proof obligations. In particular, the fact that the number of discharged proof obligations, for the models under consideration in Table 1, is comparable to that of the SMT plugin with its two SMT solvers was unexpected. In future, we also plan to use our SAT backend [16] for the ProB disprover, and evaluate its performance.

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, Lanoix, Amel Mammar, Dominique Méry, Neeraj Kumar Singh, Wen Su.

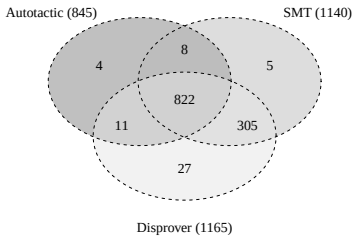
References

1. J.-R. Abrial. *Modeling in Event-B: System and Software Engineering.* Cambridge University Press, 2010.

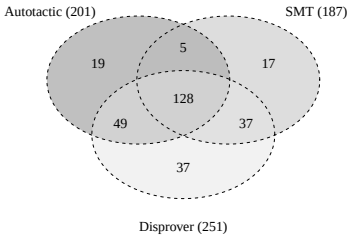
2. J.-R. Abrial, M. Butler, and S. Hallerstede. An open extensible tool environment for Event-B. In Z. Liu and J. He, editors, *Proceedings ICFEM'06*, LNCS 4260, pages 588–605. Springer-Verlag, 2006.
3. André, Attiogbé, and Lanoix. Modelling and Analysing the Landing Gear System: a Solution with Event-B/Rodin. `http://www.lina.sciences.univ-nantes.fr/aelos/softwares/LGS-ABZ2014/index.php`. Solution to ABZ-2014, Accessed: 2014-03-17.
4. C. Barrett, A. Stump, and C. Tinelli. The SMT-LIB Standard: Version 2.0. In A. Gupta and D. Kroening, editors, *Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (Edinburgh, UK)*, 2010.
5. C. Barrett and C. Tinelli. CVC3. In W. Damm and H. Hermanns, editors, *Proceedings of the* 19^{th} *International Conference on Computer Aided Verification (CAV '07)*, volume 4590 of *Lecture Notes in Computer Science*, pages 298–302. Springer-Verlag, July 2007. Berlin, Germany.
6. J. Bendisposto, S. Krings, and M. Leuschel. 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.
7. Boniol and Wiels. Landing gear system. `http://www.irit.fr/ABZ2014/landing_system.pdf`. Accessed: 2014-03-17.
8. T. Bouton, D. C. B. de Oliveira, D. Déharbe, and P. Fontaine. verit: an open, trustable and efficient smt-solver. In R. A. Schmidt, editor, *Proc. Conference on Automated Deduction (CADE)*, Lecture Notes in Computer Science, pages 151–156. Springer-Verlag, 2009.
9. D. Déharbe. Automatic Verification for a Class of Proof Obligations with SMT-Solvers. In M. Frappier, U. Glässer, S. Khurshid, R. Laleau, and S. Reeves, editors, *Proceedings ASM 2010*, LNCS 5977, pages 217–230. Springer, 2010.
10. D. Deharbe, P. Fontaine, Y. Guyot, and L. Voisin. SMT solvers for Rodin. In *Proceedings ABZ'2012*, LNCS 7316, pages 194–207. Springer, 2012.
11. S. Hallerstede and M. Leuschel. Constraint-based deadlock checking of high-level specifications. *TPLP*, 11(4–5):767–782, 2011.
12. Hansen, Ladenberger, Wiegard, Bendisposto, and Leuschel. Validation of the ABZ Landing Gear System using ProB. `http://www.stups.uni-duesseldorf.de/ProB/index.php5/ABZ14`. Solution to ABZ-2014 case study, Accessed: 2014-03-17.
13. O. Ligot, J. Bendisposto, and M. Leuschel. Debugging Event-B Models using the ProB Disprover Plug-in. *Proceedings AFADL'07*, Juni 2007.
14. Mammar and Laleau. Modeling a Landing Gear System in Event-B. `http://www-public.it-sudparis.eu/~mammar_a/LandingGearsSystem.html`. Solution to the ABZ-2014 Case Study, Accessed: 2014-03-17.
15. D. Méry and N. K. Singh. Formal specification of medical systems by proof-based refinement. *ACM Trans. Embed. Comput. Syst.*, 12(1):15:1–15:25, Jan. 2013.
16. D. Plagge and M. Leuschel. Validating B, Z and TLA+ using ProB and Kodkod. In D. Giannakopoulou and D. Méry, editors, *Proceedings FM'2012*, LNCS 7436, pages 372–386. Springer, 2012.
17. Su and Abrial. Aircraft Landing Gear System: Approaches with Event-B to the Modeling of an Industrial System. `http://www.lab205.org/home/#!/case-landing`. Solution to the ABZ-2014 Case Study.



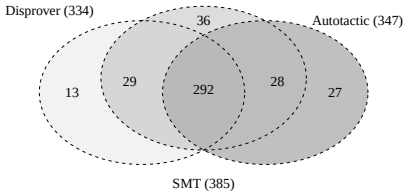
(a) Su and Abrial, version 1



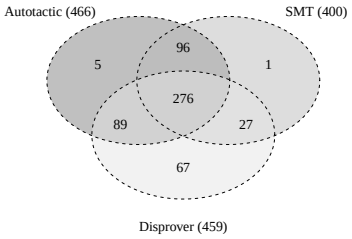
(b) Su and Abrial, version 2



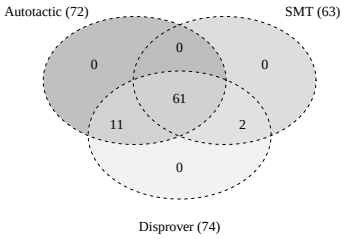
(c) Su and Abrial, version 3



(d) Mammar and Laleau

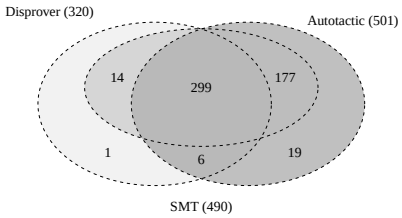


(e) André, Attiogbé and Lanoix

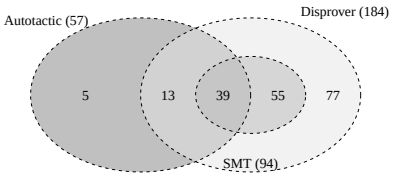


(f) Hansen, Ladenberger, Wiegard, Bendisposto and Leuschel

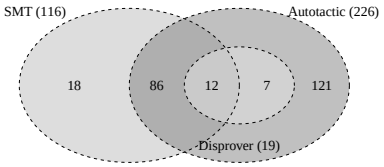
Fig. 4. Visualization of the benchmark results. Part 1: Landing gear system



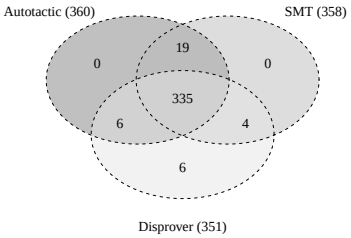
(a) Colley, CAN Bus



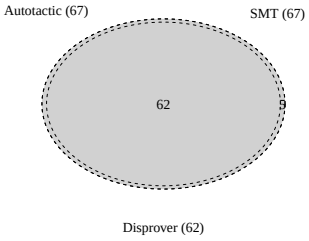
(b) Wiegard, Stuttgart 21



(c) Andriamiarina and Mèry, Graph Coloring Algorithm



(d) Singh, Pacemaker



(e) Leuschel, Set Laws

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