# SMT Solvers for Validation of B and Event-B models

Sebastian Krings and Michael Leuschel

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

**Abstract.** We present an integration of the constraint solving kernel of the ProB model checker with the SMT solver Z3. We apply the combined solver to B and Event-B predicates, featuring higher-order datatypes and constructs like set comprehensions. To do so we rely on the finite set logic of Z3 and provide a new translation from B to Z3, better suited for constraint solving. Predicates can then be solved by the two solvers working hand in hand: constraints are set up in both solvers simultaneously and (intermediate) results are transferred. We thus combine a constraint logic programming based solver with a DPLL(T) based solver into a single procedure. The improved constraint solver finds application in many validation tasks, from animation of implicit specifications, to test case generation, bounded and symbolic model checking on to disproving of proof obligations. We conclude with an empirical evaluation of our approach focusing on two dimensions: comparing low and high-level encodings of B as well as comparing pure ProB to ProB combined with Z3.

**Keywords:** B-Method, Event-B, SMT, Animation

## 1 Introduction and Motivation

B [1] and its successor Event-B [2] are two specification languages for the formal development of software and systems following the correct-by-construction approach. Both languages are rooted in set-theory and support different higher order data types like relations, functions and sequences. ProB [20,19] is a model checker for both languages featuring explicit state model checking as well as different constraint based techniques [13,18] for the analysis of specifications.

Originally, the ProB kernel has been tailored towards satisfiable formulas, acting primarily as a model finder [20,19]. Recent additions to ProB have extended the kernel in a different direction. With the introduction of ProB-based (dis-)proving of Event-B proof obligations, detecting the unsatisfiability of predicates shifted into focus [15].

ProB's kernel is developed in SICStus Prolog [8]. The integer part of the solver is mostly based on the CLP(FD) library. Custom extensions and solvers are implemented for sets, relations and records. Furthermore, support for quantifiers has been added on top of CLP(FD). The different solvers are integrated

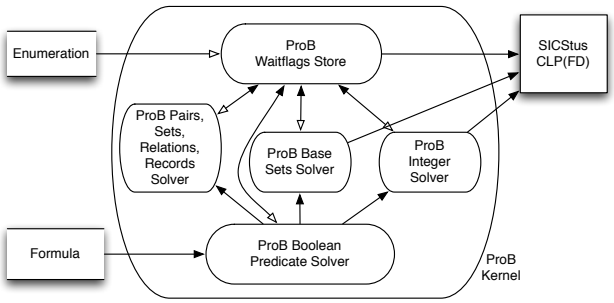


**Fig. 1.** ProB Kernel Overview

using "waitflags" to control which constraints should be tackled. Truth values between solvers are communicated using reification variables. Figure 1 gives an overview.

This approach is fundamentally different from the DPLL(T) [12] approach employed by modern SMT solvers like Z3 [9]. In [15] we already compared both approaches for Event-B proof obligations and outlined that neither is able to outperform the other: there are a considerable number of proof obligations that can only be solved by one of them. Hence, our idea is to combine the particular strengths into a single solving procedure. In Section 1.1 we will show some examples for strengths and weaknesses and argue towards our integrated approach.

Our new translation from B to Z3 and its integration is included in the latest nightly release of ProB. Information regarding installation and usage is available at:

`http://stups.hhu.de/ProB/Using_ProB_with_Z3`.

### 1.1 Small Experiments

To outline some of the weaknesses of the CLP(FD) based solving kernel, have a look at the following predicate: $X > 3 \wedge X < 7 \wedge X < Y \wedge Y < X$. Classic CLP(FD) style domain propagation first sets up the domains $4 \mathinner{..} 6$ for $X$ and $-\infty \mathinner{..} \infty$ for $Y$. In a second step, all values that cannot be part of a solution are removed from the domains. Both domains end up being empty. Hence, the predicate is detected as unsatisfiable. As soon as we drop one of the constraints on $X$, CLP(FD) is unable to do so and has to resort to enumeration. For example, the predicate $X < Y \wedge Y < X$ can not be proven unsatisfiable by ProB's CLP(FD) kernel alone, as both domains for $X$ and $Y$ are infinite $(-\infty \mathinner{..} \infty)$. Similarly, $X < 7 \wedge X < Y \wedge Y < X$ leads to an infinite sequence of narrowed down domains, never reaching inconsistency. Z3 on the other hand easily detects the unsatisfiability.

The CLP(FD) based solver in ProB however can handle certain higher-order constructs like set comprehensions better than the SMT solvers: look for example

at the predicate $(2 \mapsto 4) \in \{y | \exists(x).(y = (x \mapsto x + 2))\}$. It states that the pair $(2 \mapsto 4)$ is a member of the set of all pairs $y$ that are of the form $(x, x + 2)$. The predicate is identified as true by ProB. Of course the performance of Z3 highly depends on the translation. Choosing a low-level translation, the predicate can be broken down to $4 = 2 + 2$ and be solved by Z3. If we stay on the high-level of set logics, the set comprehension has to be described using universal quantification. If translated this way, Z3 runs into a timeout.

Additionally, the CLP(FD) based solver performs better for model finding tasks that involve non-linear integer constraints. As an example take the verbal arithmetic puzzle to find (non-equal) digits $K, I, S, P, A, O, N$ such that $KISS * KISS = PASSION$. In B this can be written as $(1000 * K + 100 * I + 10 * S + S) * (1000 * K + 100 * I + 10 * S + S) = 1000000 * P + 100000 * A + 10000 * S + 1000 * S + 100 * I + 10 * O + N$. As each letter should represent a single digit, constraints like $0 \leq K \leq 9$ are added for all the variables. Finally, we add pairwise disequalities for all variables. The resulting predicate is solved by ProB in milliseconds, while Z3 answers unknown.

In the following sections we suggest a possible integration between the CLP(FD) and SMT approaches, trying to gain the advantages of both.

## 2 New High-Level Translation of B to Z3

The following section will explain both our new translation from B to Z3 and how we integrated Z3 into ProB in order to solve constraints given in B or Event-B. First, in Section 2.1 we outline a normal form for B that avoids certain constructs that are hard to translate. Primarily, this is achieved by replacing several expressions by equivalent ones using different operators. Following, in Section 2.2 we translate constraints given in normalized B into the (set-)logic of Z3. Lastly, Section 3 explains how ProB's kernel and the SMT solver are integrated in order to combine both solvers.

### 2.1 Normalizing B / Event-B

B and Event-B feature many constructs that are not directly available in Z3's input language. In preparation of the translation from B to SMT in Section 2.2, we use rewrite rules to transform a B predicate into a normal form that is easier to translate. All these transformation rules are meant to be applied repeatedly until a fixpoint is reached.

In a first step, we replace certain negated operators available in B by the negation of the regular operator. For instance, we replace $x \notin y$ by $\neg(x \in y)$. In addition, we have to rewrite set operations involving strict subsets to subsets and (dis-)equalities. See Table 1 for the operators and their translations.

Currently, the set logics of SMT solvers have no direct support for intervals or the bounded B integer sets `NAT, NAT1, INT`. We thus rewrite constraints featuring membership in one of these to a conjunction of disequalities, e.g.,

$$x \in 1..5 \Leftrightarrow 1 \leq x \wedge x \leq 5.$$

**Table 1.** Normalization of Operators

| B | Normalized B |
|---|---|
| $E \neq S$ | $\neg(E = S)$ |
| $E \notin S$ | $\neg(E \in S)$ |
| $E \not\subset S$ | $\neg(E \subset S)$ |
| $E \not\subseteq S$ | $\neg(E \subseteq S)$ |
| $E \subset S$ | $\neg(E = S) \wedge (E \subseteq S)$ |

Membership in unions, intersections or set differences of these are handled by decomposing into multiple conjuncts or disjuncts respectively, e.g.,

$$x \in -2..5 \cap \texttt{NAT} \Leftrightarrow (-2 \leq x \wedge x \leq 5) \wedge (0 \leq x \wedge x \leq \texttt{MAXINT}).$$

ProB represents relations and functions as sets of tuples. Usually, the set is computed exhaustively. For certain relations or functions, e.g., infinite ones, ProB tries to keep the set symbolic. Furthermore, B allows set theoretic operators to be applied to functions as well. For these two reasons, we cannot simply express B functions as uninterpreted functions in SMT-LIB. We represent functions in SMT-LIB the same way we do in ProB. This makes it necessary to rewrite some B expressions on functions. For instance, we rewrite the function application using a temporary variable:

$$f = \{(1 \mapsto 4), (2 \mapsto 2)\} \wedge x = f(1)$$

becomes

$$f = \{(1 \mapsto 4), (2 \mapsto 2)\} \wedge \exists t.x = t \wedge (1 \mapsto t) : f.$$

During normalization, we have to keep in mind that well-definedness conditions of a predicate might change. In the given examples, if we request the function value of $f$ at 3, the predicate is not well-defined:

$$f = \{(1 \mapsto 4), (2 \mapsto 2)\} \wedge x = f(3)$$

We have applied the function $f$ outside of its domain. In contrast,

$$f = \{(1 \mapsto 4), (2 \mapsto 2)\} \wedge \exists t.x = t \wedge (3 \mapsto t) : f.$$

is well-defined and evaluates to false. In several cases, we add well-defined conditions later on. We show an example, division, in Section 2.2. Note that Rodin creates a separate proof obligation for well-definedness. Hence, one can assume well-definedness to be handled by those proof obligations.

Several other operators like domain(restriction) or range(restriction) can be rewritten to set comprehensions. For example, the following equality holds for the range of a function $f$:

$$\text{ran}(f) = \{y | \exists x.(x \mapsto y) \in f\}.$$

More definitions of (Event-)B operators in terms of set comprehension can be found in the "reference" books on B [1,2] and [25].

B features record datatypes comparable to those supported by Z3. However, using Z3 record types have to be introduced and typed before constraints can be applied to the fields. In normalized B, the declaration of a constrained record is hence split in the declaration of a general record conjoined with a predicate constraining the fields. A record membership expression like

$$r : struct(f1 : 11..20, f2 : 12..30)$$

becomes

$$\text{type of r} \wedge r'f1 \geq 11 \wedge 20 \geq r'f1 \wedge r'f2 \geq 12 \wedge 30 \geq r'f2.$$

Some functions included in B, like the two arithmetic functions min and max or the cardinality of a set, are not directly available in SMT-LIB. We hence add temporary variables and supply certain axioms as we did to encode function application. For instance, the expression $\min(S)$ is replaced by variable $t$ and the following additional constraints are added:

- $\forall m.m \in S \Rightarrow t \leq m$, i.e., the temporary variable is less or equal to all members of the set.
- $\exists m.m \in S \wedge t = s$, i.e., $t$ is equal to one of the members of $S$.

We encode max using the same pattern. For the cardinality, we add a constraint stating that $c$ is the cardinality of $S$ if there exists a bijection between the interval $1..c$ and $S$. For the empty set, this holds for any $c \leq 0$. Hence, we add $c \geq 0$ to Z3, resulting in $\text{card}(\varnothing) = 0$.

The choice of axioms supplied to Z3 in order to define the B functions influences the performance. We could provide more properties of max, e.g.,

$$\max(S1) > \max(S2) \Rightarrow \forall c.c \in S2 \wedge \exists s.s \in S1 \wedge s > c.$$

Additional axioms might aid Z3 in detecting unsatisfiable predicates. However, they might also decrease performance as they have to be considered during reasoning.

The rules above transform a B predicate into an equivalent B predicate. However, we could go even further, depending on how we employ Z3: For animation and (explicit state) model checking, we have to use an equivalent formula, as we rely on the models. In contrast, for certain symbolic model checking algorithms or proof attempts, we could use rewriting rules that transform a B predicate into an equisatisfiable predicate. The added freedom could be used to tailor the formula towards the solvers' strengths. We will address this in future work.

While nearly all complicated B constructs can be rewritten to set comprehensions, not all of the resulting predicates can be solved by Z3. So far, we did not have any success with the following operators:

– The *general union*, *general intersection*, *general sum* and *general product*. For instance, the general union of $U \in \mathbb{P}(\mathbb{P}(S))$ could be rewritten as $union(U) = \{x | x \in S \wedge (\exists s.s \in U \wedge x \in s)\}$. However, the existential quantification inside the set comprehension leads to highly involved constraints later on.
– The construction of (non-empty) powersets. Again we could translate $\mathbb{P}(X) = \{s | s \subseteq X\}$.
– The iteration and closure operators of classical B.

### 2.2 Translation Rules

We feed the normalized constraints generated in the previous section into the C / C++ APIs of Z3. In particular we use logics including support for sets. Z3 realizes those using the techniques described in [23].

Any logic including integer arithmetic, sets and quantifiers already covers most of the expressions occurring in our normalized constraints. Thus, we can pass most of the constraints unmodified. There are however some exceptions:

– Some common operators have different semantics in B and SMT-LIB.
– SMT-LIB as well as Z3 do not support set comprehensions natively. We will translate those by using a universal quantification constraining all members of a set variable.
– User-given sets have to be mapped to SMT-LIB sorts.

For an approach that is based on translation to be both sound and complete we have to ensure that semantical differences are taken into account. In particular, B features a distinct concept of well-definedness, i.e., operators may only be applied under certain conditions. This contrasts with SMT-LIB treating operators as total functions that always return a result. Additionally, the results of applying certain operators differ as well.

Integer division is a prominent example: B uses a division that rounds towards zero. In contrast, SMT-LIB semantics define a division rounding towards $-\infty$. Furthermore, B does not allow division by zero while for SMT solvers division is a total function, e.g., for the predicate $x = 1/0$ Z3 returns the solution $x = 0$. In order to overcome these differences, we set up $x = a/b$ using SMT-LIB's if-then-else as

$$\text{ite}(a > 0, a/b, \text{ite}(b > 0, (a/b) + 1, (a/b) - 1)) \wedge b \neq 0.$$

For the sake of brevity we can not fully discuss the sematical differences between B and SMT-LIB in this article.

Now, let us have a look at the translation of set comprehensions. A B expression like

$$\neg(r \in \{x | x \mod 2 = 1\})$$

is submitted to Z3 using a temporary variable and axiomatizing the set comprehension. The resulting constraint is

$$\neg(\exists tmp.(r \in tmp \wedge \forall v.v \in tmp \Leftrightarrow v \mod 2 = 1)).$$

So far we do not provide any additional hints like instantiation triggers to Z3.

In addition to given types likes `INTEGER`, the B method features user defined types represented as deferred or enumerated sets. We translate those to custom SMT-LIB sorts. For enumerated sets we additionally introduce the identifiers and enforce their disequality using an additional constraint. Z3 natively supports sorts with given cardinality. Hence, if the cardinality of a user-given type can be computed statically by ProB we can submit said cardinality to Z3.

## 3 Integration of Solvers

We investigated different modes of using Z3 together with the ProB kernel:

- Use it alone without relying on ProB. This approach was quickly abandoned due to the (currently) untranslatable predicates outlined in Sections 2.1 and 2.2. Additionally, some translations have to resort to quantification that hinders proof efforts and model finding.
- Use Z3 solely for falsification of B predicates. If we only rely on the SMT solvers for the detection of unsatisfiability, we can safely skip untranslatable parts of the predicate without risking unsound results (as those parts will be checked by ProB's solver). However, many predicates cannot be disproven once important parts are missing.
- We could employ a cooperative approach where parts of a predicate are given to one or both of the SMT solvers, while other parts are handled by the ProB kernel. In this case, we would translate partial assignments back and forth between the two solvers in order to communicate intermediate results.
- Lastly, we could use a fully integrated approach where the whole predicate is given to the ProB kernel and as much as is translatable is given to the SMT solvers. In addition to partial assignments we could transport information about inferred or learned clauses or unsatisfiable cores back and forth.

The first approach was quickly discarded, because the SMT solvers alone are often too weak to solve interesting predicates. This is mostly due to cumbersome translations of higher-order B expressions like set cardinality. The same holds true for the second approach. Even though the SMT solvers are able to falsify several predicates that ProB cannot falsify (see Section 1.1), much is left to be desired. Hence, we investigated the integrated approaches more thoroughly.

The third approach is comparable to the one taken in [24], translating B to SAT. The key problem to this approach is to decide which predicate to translate and submit to Z3 and which ones to keep in ProB. In [24] the authors used a greedy approach: every predicate that can be translate will be translated.

However, we integrated the two solvers further and set up constraints in both simultaneously. We delay the call to Z3 until after the deterministic propagation phase of ProB and also submit the information inferred so far. Additionally, we use the unsat core found by Z3 to control backtracking on the ProB Prolog side and to lift ProB from backtracking to backjumping. Details on both techniques are given below.

```
Data: Predicate P, (partial) State S
Result: fails iff P is unsat, succeeds with model iff P is sat; might time out
procedure boolean solve(P, S)
    set_up_clpfd_variables(S)
    set_up_smt_variables(S)
    while exists conjunct C in P that has not been set up do
        D = to_clpfd_solver(C)          // domains D from clpfd propagation
        smt_result = to_smt_solver(C,D)          // transfer C and domains
        if smt_result = unsat then
            backjump using unsat core
        end
    end
    while exists unbound variable V in S do
        clpfd_labeling(V)                                  // binds V to value
        smt_result = to_smt_solver(V)  // V now bound: transfer new value
        if smt_result = unsat then
            backjump using unsat core
        end
    end
    return S with all variables labeled
```

**Algorithm 1:** Integrated Constraint Solver

**Transferring CLP(FD) Domains to the SMT Solvers** As can be seen in Algorithm 1 communication with the SMT solver starts after the deterministic propagation phase. During this phase, ProB tries to deterministically infer knowledge about the values of the variables in a predicate. For instance, from $X > 3 \wedge Y > X$ ProB infers $Y > 4$. The underlying propagation rules are not limited to arithmetic but support further B constructs like set theory. Before a predicate is submitted to Z3, all the statically inferred information is added to it.

**Controlled Backjumping Using the Unsat Core** In case Z3 detects unsatisfiability, we can use Z3's unsat core computation in order to perform backjumping inside ProB's kernel. The unsat core contains a subset of the conjuncts $C$ taken from $P$ as outline in Algorithm 1. Note that this subset does not necessarily contain the conjunct submitted last. Inside ProB's kernel we can now backjump until at least one of the conjuncts inside the unsat core has been removed from both the SMT solver and the CLP(FD) solver. After the backjump, ProB can choose a different path inside case distinctions or decide on different heuristics. Thus, the backjump has cut of parts of the search space ProB would have explored otherwise.

**Table 2.** Results of running Provers

| Model | # POs SMT | HL-SMT | | ProB | | ProB/SMT | |
|---|---|---|---|---|---|---|---|
| | | prove | disprove | prove | disprove | prove | disprove |
| Landing Gear System 1, Su, et. al. | 2328 | 1478 | 2196 | 0 | 2311 | 0 | 2303 | 0 |
| Landing Gear System 2, Su, et. al. | 1188 | 548 | 741 | 0 | 1176 | 0 | 1152 | 0 |
| Landing Gear System 3, Su, et. al. | 341 | 171 | 77 | 0 | 290 | 0 | 262 | 0 |
| CAN Bus, Colley | 534 | 296 | 316 | 0 | 276 | 0 | 340 | 1 |
| Graph Coloring, Andriamiarina, et. al. | 254 | 119 | 51 | 0 | 0 | 0 | 51 | 0 |
| Landing Gear System, Hansen, et. al. | 74 | 59 | 55 | 0 | 74 | 0 | 74 | 0 |
| Landing Gear System, Mammar, et. al. | 433 | 265 | 212 | 0 | 400 | 0 | 413 | 0 |
| Landing Gear System, André, et. al. | 619 | 263 | 77 | 0 | 567 | 5 | 533 | 4 |
| Pacemaker, Neeraj Kumar Singh | 370 | 198 | 369 | 0 | 354 | 0 | 370 | 0 |
| Stuttgart 21 interlocking, Wiegard | 202 | 46 | 18 | 0 | 125 | 2 | 123 | 0 |

## 4 Limitations

One key limitation of our approach is related to the type system of B. There is no strict differentiation between functions, sets and sequences. For instance, one can apply the set union operator to two functions leading to a result that might not be a function.

For the same to be allowed in the SMT-LIB translation, we had to use a common representation: we express relations and functions as sets of pairs connecting input and output values; sequences are encoded as sets of pairs consisting of the sequence index and the value.

Using this common base representation, all B and Event-B operators can be encoded. However, we cannot use more sophisticated SMT-LIB representations anymore. In particular, sequences could have been mapped to SMT-LIB arrays, resulting in improved performance due to the usage of specialized decision procedures.

Another limitation is the missing support for set cardinality in Z3's set logic. Although it was part of the initial proposal for the SMT-LIB finite set theory [16] it has not yet been implemented in Z3. We thus encode $c = \text{card}(S)$ as

$$\exists t.t \in S \rightarrowtail\!\!\!\!\!\rightarrow [1, c]$$

i.e., we search for a total bijection from $S$ to the interval $[1, c]$. This encoding is quite cumbersome and often leads to Z3 answering "unknown". Cardinality constraints however have to be used in the translation of some B operators, e.g., to compute the next index of a sequence upon concatenation. Hence, those can often not be solved as well.

## 5 Empirical Results

In this section we will evaluate two different aspects. First, we want to know how our new high-level translation of set theory in Z3 compares to the more

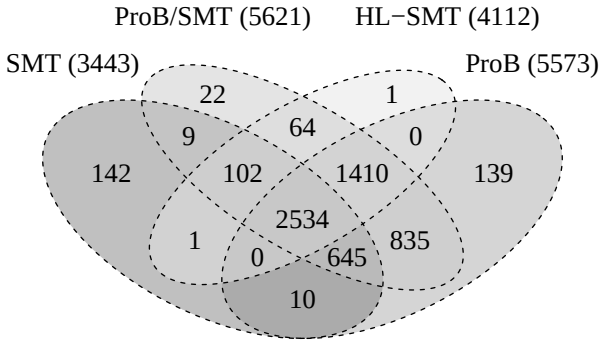


**Fig. 2.** Performance on Proof Obligations

low-level approach of the SMT translation outlined in [10,11]. Second, we want to evaluate if it is it worthwhile to integrate Z3 into ProB and to communicate back and forth. In order to find out, we compare the integrated solution to Z3 and ProB on their own.

We benchmarked the following configurations:

- **SMT**, the SMT solvers plugin for Rodin as presented in [10,11],
- **HL-SMT**, our high-level translation from Event-B to SMT featuring Z3's set theory, alone without ProB's solver,
- **ProB**, a plain version of ProB's constraint solving kernel, and
- **ProB/SMT**, ProB's constraint solving kernel integrated with Z3.

For better comparability, we used the same set of benchmarks already employed in [15]:

- Answers to the ABZ-2014 landing gear case study [6]. Beside our own version [14], 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 [21].
- A model of the Stuttgart 21 Railway station interlocking by Wiegard, derived from chapter 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 [22].

For the benchmarks, we have used Rodin 3.2, version 2.1.0 of the Atelier B provers plugin and version 1.2.1 of the SMT plugin. For better comparability,

we did not use the bundled SMT solvers CVC3 and veriT. Instead, we relied on Z3 version 4.4.1 as used in the ProB integration as well. ProB was used in version 1.5.1-beta3, connected through the disprover plugin version 3.0.8. We used a global timeout of 25 seconds for a single proof attempt.

All benchmarks were run on a MacBook Pro featuring a 2.6 GHz i7 CPU and 8 GB of RAM. We did not parallelize the benchmarks in order to avoid issues due to hyper-threading or scheduling. Benchmarks were run using a dedicated evaluation plugin[1] for the Rodin platform. The data is presented as follows:

- Figure 2 shows a Venn diagram comparing the number of discharged proof obligations by each of the configurations mentioned above.
- Table 2 shows how the individual configurations perform on the different models. In particular it distinguishes between proof and disprove.
- Table 3 shows how the individual configurations perform on different kinds of proof obligations.

Regarding the different performance of the high-level vs. the low-level SMT translation we have mixed results. Judging by the total numbers, the high-level approach is superior: as can be seen in Fig. 2 it is able to discharge 4112 proof obligations, while the low-level approach only discharges 3443. However, there is also a considerable amount of proof obligations that can be discharged with the low-level approach but not with the high-level one. Since the original SMT plugin does not support disprove of POs, we cannot say anything about the performance. The high-level approach is unable to disprove a single of the defective obligations.

Comparing ProB solo and together with Z3 paints a similar picture. The integrated solution is superior but the margin is small. Again, 149 proof obligations cannot be discharged anymore once the SMT integration is enabled. Virtually all of these result in a timeout afterwards. Since a global timeout is used and Z3 takes up to much time ProB misses the solution. We could indeed use a local timeout for the integrated SMT solver. However, we did not find a sensible heuristic to decide when to give time to Z3 vs. giving it to the ProB kernel.

Regarding disproving, integrating Z3 into ProB lead to the discovery of a new counter-example. Despite our usage of the CAN Bus model in [15] the error went unnoticed till now. Yet again, some counter-examples previously found cannot be discovered by the integrated solver in the given timeframe.

Table 2 outlines for which models we see better or worse performance for the high-level SMT translation. In particular the landing gear systems and the Stuttgart 21 interlocking models show a decline in successfully discharged POs when compared with the low-level SMT translation. This models feature a considerable amount of concrete data that can easily be translated using the low-level approach. We assume that some of these POs can be discharged on the

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

**Table 3.** Performance of provers on different kinds of proof obligations

| Kind of PO | # POs | SMT | HL-SMT | ProB | ProB/SMT |
|---|---|---|---|---|---|
| Feasibility of non-det. action | 59 | 40 (67.8 %) | 52 (88.1 %) | 44 (74.6 %) | 57 (96.6 %) |
| Guard strengthening | 300 | 13 (4.3 %) | 139 (46.3 %) | 258 (86.0 %) | 254 (84.7 %) |
| Invariant preservation | 4938 | 3106 (62.9 %) | 3741 (75.8 %) | 4488 (90.9 %) | 4552 (92.2 %) |
| Natural number for a numeric variant | 6 | 5 (83.3 %) | 6 (100.0 %) | 4 (66.7 %) | 6 (100.0 %) |
| Action simulation | 153 | 104 (68.0 %) | 86 (56.2 %) | 134 (87.6 %) | 142 (92.8 %) |
| Theorem | 97 | 29 (29.9 %) | 26 (26.8 %) | 66 (68.0 %) | 62 (63.9 %) |
| Decreasing of variant | 6 | 6 (100.0 %) | 6 (100.0 %) | 6 (100.0 %) | 6 (100.0 %) |
| Well definedness | 779 | 140 (18.0 %) | 56 (7.2 %) | 570 (73.2 %) | 539 (69.2 %) |
| Feasibility of a witness | 1 | 0 (0.0 %) | 0 (0.0 %) | 1 (100.0 %) | 1 (100.0 %) |
| Well definedness of a witness | 4 | 0 (0.0 %) | 0 (0.0 %) | 2 (50.0 %) | 2 (50.0 %) |
| | 6343 | 3443 (54.3 %) | 4112 (64.8 %) | 5573 (87.9 %) | 5621 (88.6 %) |

boolean level, without any higher-order reasoning. Table 2 also shows that these are the models where ProB alone works well.

The high-level SMT approach, both with and without ProB integration performs better for more abstract models like the CAN Bus, the graph coloring algorithm and the pacemaker model. This stresses our assumption that integration the high-level SMT translation into ProB is worthwhile as they represent orthogonal technologies that could benefit from one another.

## 6 Related Work

As mentioned above, in [10,11] the authors present an integration of SMT solvers into Rodin [3], an IDE for Event-B development. In this scenario, the SMT solvers are used as provers in order to discharge Event-B proof obligations. The authors investigate two different ways of translating Event-B to SMT-LIB.

For SMT solvers in general they suggest the *ppTrans* approach. Here, set theory and arithmetic are broken down into first-order formulas using uninterpreted functions for membership, etc. On the one hand, this approach is more flexible than the one presented in this paper: it does not rely on the API of a specific SMT solver. On the other hand, the resulting formulas only approximate the Event-B semantics, as operators are replaced by uninterpreted functions. The authors thus add certain set theoretic axioms to the SMT problem in order to recover from this.

A second approach, called $\lambda$-*based* relies on an extension to SMT-LIB provided by the veriT solver [7]. Set theoretic constructs are then translated into $\lambda$-expressions. The major shortcoming of this approach is that sets of sets cannot be handled.

Many of the rewrite rules presented here are similar to those in [10,11]. The key difference is that we rely on the given set theory of Z3 instead of translating further into first-order logic.

In addition to other SMT-based approaches, there are different ways of solving B and Event-B predicates. ProB itself mainly relies on constraint logic programming. There is also the formerly mentioned backend [24] translating B

to Kodkod [27]. Kodkod then uses a SAT solver to find solutions to the given formulas.

## 7 Future Work

For the future, we have different directions in mind. First of all, we would like to investigate whether using an equisatisfiable translation instead of an equivalent one is of use. In particular for approaches like proving or disproving as discussed in [15] we expect improved performance.

We also want to tighten the integration of the SMT solvers and ProB. Currently we are transporting partial assignments and we use the unsat core to control backjumping on the Prolog side. In future, we want to investigate, whether we can access and use clauses learned on the SMT side in order to set up further constraints on the Prolog side. For instance, we want to investigate whether we can use interpolants or conflict clauses in case of unsatisfiable predicates.

Regarding our translation to SMT-LIB, the benchmarks show that in particular the usage of quantifiers can be improved. One possibility to do so is to further investigate how to set instantiation triggers for comprehensions typically occurring in our scenarios. In [17] the authors already outlined a general approach that can serve as a starting point. Another option is to try to reduce the amount of quantifiers we use. This could be achieved by providing a custom theory to the SMT solvers, i.e., including inference rules for min and max that avoid the quantifiers introduced in Section 2.2. Changing the set of axioms we supply to Z3 in order to define min and max is certainly another direction that should be evaluated.

Another technique we want to implement should help us to overcome some of the limitations discussed in Section 4. As mentioned, the B type system allows to use set operators on sequences. Hence, we had to encode sequence using the a representation as sets of pairs. A static check could investigate, how operators are applied in a B machine. It could determine, if sequences are only used with sequence operators. In this case, we could employ a more efficient translation to SMT-LIB, e.g., encode them as arrays.

Regarding benchmarks and applications, we would like to move from solving predicates to explicit state model checking and later to symbolic model checking and constrained based validation techniques.

## 8 Discussion and Conclusion

One motivation for the integration of SMT solvers into ProB was to overcome the weaknesses we spotted in our previous work [15]: ProB should be enabled to handle infinite domains and detection of unsatisfiability should be improved.

With the suggested high-level translation of B to SMT-LIB both goals could be achieved. The integrated solution is able to discharge more proof obligations than ProB alone. In many cases, translation into the high-level (set) logics of

Z3 seems advantageous over a low-level translation to predicate logic. Indeed, in our experimental evaluation on Event-B proof obligations, our new high-level translation discharges 4112 proof obligations in total, out of which 1475 cannot be discharged by the previous SMT translation [10,11].

Our evaluation also showed, that there is not only a gain in the number of proof obligations: the low-level translation discharges 806 proof obligations that are not discharged by our new translation. Yet, it is not clear when to employ a high-level and when to employ a low-level approach. A practical solution is to use both in a solver portfolio.

It remains yet to be seen, how SMT solvers like Z3 will evolve regarding high-level theories. The current version of the SMT-LIB standard only features a "possible declaration for a theory of sets and relations" [5]. How and if different possibilities are realized will certainly influence the impact SMT solvers have in the formal methods community.

Summarizing, we provided new ways to tackle the complexity of constraints in B and Event-B. We provided a new high-level translation of B to Z3's input language, which can be used on its own or integrated into ProB's solver. This high-level SMT based solver appears to be an orthogonal addition to the other solvers, solving many constraints that could not be solved by the previous low-level translation and is better suited at finding models. Our evaluation also confirms that the integration of the ProB solver with Z3 provides the best overall result, discharging 5621 proof obligations. We hope that these new capabilities open up new applications, from synthesis to improved symbolic validation techniques such as bounded model checking.

**Acknowledgements** We would like to thank the reviewers of iFM'2016 for their useful feedback. In addition, we thank Christoph Schmidt for implementing parts of the translation.

## References

1. J.-R. Abrial. *The B-Book: Assigning Programs to Meanings*. Cambridge University Press, 1996.
2. J.-R. Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2010.
3. 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*, LNCS, pages 588–605. Springer, 2006.
4. André, Attiogbé, and Lanoix. Modelling and Analysing the Landing Gear System: a Solution with Event-B/Rodin. Solution to ABZ 2014 Case Study.
5. C. Barrett, P. Fontaine, and C. Tinelli. The SMT-LIB Standard: Version 2.5. Technical report, Department of Computer Science, The University of Iowa, 2015. Available at `www.SMT-LIB.org`.
6. F. Boniol and V. Wiels. The landing gear system case study. In *ABZ 2014: The Landing Gear Case Study*, volume 433 of *CCIS*, pages 1–18. Springer, 2014.
7. 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, *Proceedings CADE*, LNCS. Springer, 2009.

8. M. Carlsson and P. Mildner. Sicstus prolog—the first 25 years. *Theory and Practice of Logic Programming*, 12(1-2):35–66, 2012.
9. L. De Moura and N. Bjørner. Z3: An efficient smt solver. In *Proceedings TACAS*, LNCS, pages 337–340, Berlin, Heidelberg, 2008. Springer.
10. D. Déharbe, P. Fontaine, Y. Guyot, and L. Voisin. SMT Solvers for Rodin. In J. Derrick, J. Fitzgerald, S. Gnesi, S. Khurshid, M. Leuschel, S. Reeves, and E. Riccobene, editors, *Proceedings ABZ*, volume 7316 of *LNCS*, pages 194–207. Springer, 2012.
11. D. Déharbe, P. Fontaine, Y. Guyot, and L. Voisin. Integrating SMT solvers in Rodin. *Science of Computer Programming*, 94, Part 2(0):130 – 143, 2014.
12. H. Ganzinger, G. Hagen, R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Dpll(t): Fast decision procedures. In R. Alur and D. Peled, editors, *Computer Aided Verification*, volume 3114 of *LNCS*, pages 175–188. Springer, 2004.
13. S. Hallerstede and M. Leuschel. Constraint-based deadlock checking of high-level specifications. *Theory and Practice of Logic Programming*, 11(4–5):767–782, 2011.
14. D. Hansen, L. Ladenberger, H. Wiegard, J. Bendisposto, and M. Leuschel. Validation of the abz landing gear system using ProB. In *ABZ 2014: The Landing Gear Case Study*, volume 433 of *CCIS*, pages 66–79. Springer, 2014.
15. S. Krings, J. Bendisposto, and M. Leuschel. From Failure to Proof: The ProB Disprover for B and Event-B. In *Proceedings SEFM*, LNCS. Springer, 2015.
16. D. Kröning, P. Rümmer, and G. Weissenbacher. A proposal for a theory of finite sets, lists, and maps for the smt-lib standard. In *Informal proceedings SMT Workshop*, 2009.
17. K. R. M. Leino and R. Monahan. Reasoning about comprehensions with first-order SMT solvers. In *Proceedings ACM SAC*, pages 615–622, 2009.
18. M. Leuschel, J. Bendisposto, I. Dobrikov, S. Krings, and D. Plagge. From animation to data validation: The prob constraint solver 10 years on. In J.-L. Boulanger, editor, *Formal Methods Applied to Complex Systems: Implementation of the B Method*, chapter Chapter 14, pages 427–446. Wiley ISTE, Hoboken, NJ, 2014.
19. M. Leuschel and M. Butler. ProB: A model checker for B. In *Proceedings FME*, LNCS, pages 855–874. Springer, 2003.
20. M. Leuschel and M. Butler. ProB: an automated analysis toolset for the B method. *Int. J. Softw. Tools Technol. Transf.*, 10(2):185–203, Feb. 2008.
21. A. Mammar and R. Laleau. Modeling a landing gear system in event-b. In *ABZ 2014: The Landing Gear Case Study*, volume 433 of *CCIS*, pages 80–94. Springer, 2014.
22. 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.
23. L. M. D. Moura and N. Bjørner. Generalized, efficient array decision procedures. In *Formal Methods in Computer-Aided Design*, pages 45–52, 2009.
24. D. Plagge and M. Leuschel. Validating B, Z and TLA+ using ProB and Kodkod. In D. Giannakopoulou and D. Méry, editors, *Proceedings FM*, LNCS, pages 372–386. Springer, 2012.
25. S. Schneider. *The B-method: An Introduction*. Cornerstones of computing. Palgrave, 2001.
26. W. Su and J.-R. Abrial. Aircraft landing gear system: Approaches with event-b to the modeling of an industrial system. In *ABZ 2014: The Landing Gear Case Study*, volume 433 of *CCIS*, pages 19–35. Springer, 2014.
27. E. Torlak and D. Jackson. Kodkod: A relational model finder. In O. Grumberg and M. Huth, editors, *TACAS'07*, LNCS 4424, pages 632–647. Springer, 2007.