Towards a Shared Specification Repository

Philipp Körner ✉, Michael Leuschel ✉, and Jannik Dunkelau

Institut für Informatik, Universität Düsseldorf
Universitätsstr. 1, D-40225 Düsseldorf, Germany
{p.koerner,michael.leuschel,jannik.dunkelau}@uni-duesseldorf.de

Abstract. Many formal methods research communities lack a shared set of benchmarks. As a result, many research articles in the past have evaluated new techniques on specifications that are specifically tailored to the problem or not publicly available. While this is great for proving the concept in question, it does not offer any insights on how it performs on real-world examples. Additionally, with machine learning techniques gaining more popularity, a larger set of public specifications is required. In this paper, we present our public set of B machines and urge contribution. As we think this to be an issue in other communities in scope of the ABZ as well, we are also interested in specifications expressed in other formalisms, for example Alloy, TLA$^+$ or Z.

1 Introduction and Motivation

Our group in Düsseldorf has collected since 2003 thousands of B and Event-B machines: our ProB repository contains around 13 000 machines, of which more than 3500 are publicly available. The examples are used for ProB's regression, performance and feature tests. Those public examples contain some duplicates, as they are compiled from different sources: e.g., from tickets in our bug tracker, teaching, literature, case studies, or student projects.

Naturally, not all machines are relevant to all research questions: infinite state spaces might be interesting in order to evaluate symbolic model checking techniques [11], whereas large yet finite state spaces are the important class for distributed model checking [10]. Other use cases, such as data validation [7] work by executing a model along one particular, linear path, while others, like constraint solving problems, sometimes work on machines without variables, consisting of a single state. Most recently, machine learning (ML) techniques are applied to model checking or synthesis as well, and require a large number of specifications, e.g., in order to extract and re-combine predicates [6]. Even with access to numerous machines, it is time-consuming and cumbersome to identify machines to use for benchmarking, especially since only a small amount of data can be presented in a typical research article. Without any doubt, other research groups have their individual set of B machines they use for testing and evaluation. Thus, we propose that individual sets of benchmarks from different parts of the community are combined into a global, shared repository. With this paper, we start this endeavour, and create an index of our specifications as described in Section 2. Benefits include:

- Benchmarks are publicly available and experiments can be replicated easily.
- Performance comparisons of several tools in different versions can be drawn.
- Suitable benchmarks can be quickly identified.
- Examples for translations between formalisms or ML are available.
- Particularly successful examples can be shared for teaching.

While we are most involved in the B and Event-B community, we think that similar issues are present in other communities which make up the ABZ conference. Thus, we explicitly want to invite everyone to contribute specifications written in other formalisms as well. The repository is located at:

https://github.com/hhu-stups/specifications

2 Proposed Index

Since our initial set of models is rather large, it is vital that a sufficient amount of meta-information is attached to the models. For this, we suggest usage of edn[1], a serialisation format with parsers available in most mainstream programming languages. For each specification, some basic information should be offered:

- Which formalism is this specification written in?
- A SHA-256 hash code to identify duplicates, and to ensure reproducibility of experiments regarding the specification.
- Number of deferred sets, enumerated sets, constants, state variables and operations / events, number of included machines, etc.
- Number of states and state transitions in the machine (if known).[2]
- Presence of invariant violations, deadlocks, etc. (if the property is known).
- Optional link to another (previous) model (e.g., a correction or evolution).

The information above is known to never change, but can be extended once further properties are considered. Additional information depending on the tool, its configuration or the use case altogether can be included as well, such as temporal properties (e.g., expressed in LTL or CTL) which are expected to hold or to be violated, tool name and version/revision which is able to parse or execute the specification, or settings, walltime and memory usage required for application of a technique such as model checking.

[1] Extensible Data Notation, see: https://github.com/edn-format/edn

[2] Note that different tools count the number of transitions and states slightly differently. it might be necessary to keep track of the number of initial states and, e.g., the virtual constants setup states of prob. then, one can derive the expected statistics for other validation tools. some settings can also influence the number of states, e.g., the default scope for deferred sets or maximum number of transitions per operations. in that case, it is preferable not to specify a number of states, but rather include that number in a specific run of the tool (see below), that also includes the settings needed for replication.

Optional Fields. Naturally, this data must also be extensible via optional fields. For instance, additional information due to a new use case can be gathered, e.g., the amount of states when using state space reduction techniques. As runtime might depend on the hardware it was ran on, relevant data should be included as well. They also allow extension of the information, e.g., for further tools such as Atelier-B [4] or handling of entirely different file formats, e.g., Rodin [1] archives. In order to select suitable set of specifications, one can simply apply a filter predicate testing the formalism or dialect of it. Furthermore, optinal fields enable links between different machines (e.g., due to refinement or different parameter instantiation) and to external information, such as references to articles describing the model, descriptions of the models as well as the author(s) and their contact information. Finally, certain metrics do not make sense for specific use cases of a formalism, or cannot be applied to other formalisms at all. Thus, such data must not be a mandatory field (but may be mandatory for a given formalism)[3].

Filtering Specification. As previously mentioned, we use edn for the meta information because this format can easily be processed. A short example written in Clojure is given in Listing 1.1. There, all files containing meta-information in the directory are located (ll. 1–5). Then, they are read in and filtered (ll. 7–15). The expression starting in l. 9 returns a list of all file names of specifications written in the B formalism that are known to have a state space of at least 100 000 states. At the time of writing, there are 45 such machines. This example shows that finding specifications based on certain criteria is fairly easy and necessary for verification tool maintainers.

Table 1 provides an overview of the information of B machines currently present in the repository, compiled after running each machine with a timeout of 30 minutes in the ProB model checker.

On Updating Versions. We strongly argue that the published version of a specification *must not be replaced.* Once they are online, they may be used by any researcher. Even though git clearly documents the history of a file, it would be unclear which version was used as a benchmark or presented in an article. If mistakes were spotted, new versions can be submitted *as a modified copy.*

3 Conclusions, Related and Future Work

We firmly believe that a shared repository of specifications will benefit all communities coming together at ABZ. Aside from making benchmarks available for replication, it can assist courses teaching the formal methods. Furthermore, it builds the foundation for exciting new research that relies on such a dataset.

[3] It would be sensible to define different standard formats for different formalisms. These can be automatically enforced in a CI pipeline, e.g., by Clojure Spec [5], before pull requests are accepted.

```
(def META-INF-DIR (java.io.File. "../meta-information"))

;; get a sequence of all meta-information files in the directory
(def meta-files (remove (fn [file] (.isDirectory file))
                        (file-seq META-INF-DIR)))

(defn read-meta-file [f] (read-string (slurp f)))

(->> meta-files
     (map read-meta-file)
     (filter (fn [data]
               (and (= (:formalism data) :b)
                    (number? (:number-of-states data))
                    (> (:number-of-states data) 100000))))
     (map :file))
```

Listing 1.1: Finding Specifications Based on Their Information

Similar issues have been found in other communities. This led to the creation of central benchmarking sets, e.g., BEEM for models written in DVE [13], or the PRISM benchmark suite [12] for models written in PRISM. Yet, to our knowledge, it is not possible to contribute to these databases. This has led to criticism that, e.g., not many models that are large enough are featured. Also, a fixed set of benchmarks is not a viable approach in the B community, that creatively uses the B language in order to solve very different types of problems.

In other communities, such as SMT and SAT solving, shared benchmark sets are established for many years [3,8]. They both grow via community contributions and are the foundation for solver competitions [2,9]. SMT-LIB in particular is a success story, containing more than 100 000 benchmarks. There are many other examples for competitions and problem collections, e.g., SV-COMP[4], TPLP[5] [15], which we cannot exhaustively list here due to page limitations.

An interesting question we could not answer in this paper is to what extent our examples match the reality of (confidential) industrial specifications. An answer requires to take a closer look at the data that is available to us. When considering state space size, number of variables and operations as well as idioms used, e.g., usage of program counters or certain data structures, it might be possible to label some public machines accordingly.

Furthermore, research papers often contain links to download pages not only for benchmarks, but also tools themselves. Some tools presented years ago are hard or near impossible to find now. Some conferences, e.g., POPL, established artifact evaluation committees, yet making artifacts permanently available often

[4] https://sv-comp.sosy-lab.org/2020/

[5] Which inspired the second author to generate another library, Dozens of Problems for Partial Deduction https://github.com/leuschel/DPPD.

Table 1: Overview of available machine meta data with a timeout of 30 min.

Errors on Load			310
Formalism		763 Event-B	2886 Classical B
Deadlock found	1080 yes	1576 no	683 timeout
Invariant violated	255 yes	2498 no	586 timeout
	max	*avg*	*usage in # machines*
States	1 000 002	8743	2624
Transitions	5 570 544	53 296	2624
Included Machines	13	1.18	3339
State Variables	10 000	7.49	2282
Operations	2000	6.00	2497
Deferred Sets	50	0.44	669
Enumerated Sets	19	0.79	1310
Invariants	10 000	9.39	1958
Constants	10 000	8.63	2090
Properties	12 015	17.51	2094
Static Assertions	188	1.46	646
Dynamic Assertions	54	0.20	200
Definitions	374	2.75	1265

is optional. ACM conferences offer different badges[6] depending on availability, replicability, etc. A similar, *mandatory* repository containing at least one binary version or even the source code of tools presented at conferences might prove useful to the research community as well. Worth mentioning here is the StarExec platform [14], that allows storage and execution of tools and benchmark problems, which may serve this effort to a satisfactory extent already.

In order for the presented endeavour to be successful, the effort of the entire community is required and their contributions to this repository will be appreciated.

Acknowledgement. Computational support and infrastructure was provided by the "Centre for Information and Media Technology" (ZIM) at the University of Düsseldorf (Germany). We thank the many persons who contributed to the repository (a list is available at the project's website).

References

1. J.-R. Abrial, M. Butler, S. Hallerstede, and L. Voisin. An Open Extensible Tool Environment for Event-B. In *Proceedings ICFEM*, volume 4260 of *LNCS*, pages 588–605. Springer, 2006.
2. C. Barrett, L. De Moura, and A. Stump. SMT-COMP: Satisfiability modulo theories competition. In *Proceedings CAV'05*, volume 3576 of *LNCS*, pages 20–23. Springer, 2005.

[6] Cf. https://www.acm.org/publications/policies/artifact-review-badging

3. C. Barrett, A. Stump, and C. Tinelli. The SMT-LIB Standard – Version 2.0. In *Proceedings SMT'10*, July 2010.
4. ClearSy. *Atelier B, User and Reference Manuals*. Aix-en-Provence, France, 2016. Available at http://www.atelierb.eu/.
5. Clojure Spec Guide. https://clojure.org/guides/spec. Accessed: 2020-03-12.
6. J. Dunkelau, S. Krings, and J. Schmidt. Automated Backend Selection for ProB Using Deep Learning. In *Proceedings NFM'19*, volume 11460 of *LNCS*, pages 130–147. Springer, 2019.
7. D. Hansen, D. Schneider, and M. Leuschel. Using B and ProB for Data Validation Projects. In *Proceedings ABZ'16*, volume 9675 of *LNCS*, pages 167–182. Springer, 2016.
8. H. H. Hoos and T. Stützle. SATLIB: An online resource for research on SAT. In *SAT2000*, pages 283–292. IOS Press, 2000.
9. M. Järvisalo, D. Le Berre, O. Roussel, and L. Simon. The international SAT solver competitions. *Ai Magazine*, 33(1):89–92, 2012.
10. P. Körner and J. Bendisposto. Distributed Model Checking Using ProB. In *Proceedings NFM'18*, volume 10811 of *LNCS*. Springer, 2018.
11. S. Krings. *Towards Infinite-State Symbolic Model Checking for B and Event-B*. PhD thesis, Universitäts-und Landesbibliothek der HHU Düsseldorf, 2017.
12. M. Kwiatkowska, G. Norman, and D. Parker. The PRISM benchmark suite. In *Proceedings QEST'12*, pages 203–204. IEEE CS press, Sept. 2012.
13. R. Pelánek. BEEM: benchmarks for explicit model checkers. In *Proceedings SPIN'07*, volume 4595 of *LNCS*, pages 263–267. Springer, 2007.
14. A. Stump, G. Sutcliffe, and C. Tinelli. StarExec: A cross-community infrastructure for logic solving. In *Proceedings IJCAI'14*, volume 8562 of *LNCS*, pages 367–373. Springer, 2014.
15. G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure. From CNF to TH0, TPTP v6.4.0. *Journal of Automated Reasoning*, 59(4):483–502, 2017.