Event-B and Cloud Provers

Alexei Iliasov[1] Paulius Stankaitis[2] David Ebo Adjepon-Yamoah[3]

Centre for Software Reliability, School of Computing Science, Newcastle University, UK,
`(alexei.iliasov,paulius.stankaitis,d.e.adjepon-yamoah)@ncl.ac.uk`

Abstract: We discuss the whys and hows of remotely managing a collection of automated theorem provers supporting verification of Event-B models in the Rodin Platform[4]. We report on the current state of the work, our general aspirations and a range of technical obstacles encountered so far.

1 Overview

Event-B [1] is one of the more popular notations for model-based, proof driven specification. It offers a fairly high-level mathematical language based on FOL and ZF set theory and an economical yet expressive modelling notation centred around the notion of an atomic event - a form of before-after predicate. Leaving aside the methodological qualities of Event-B, one can regard an Event-B model as a high-level notation from which a number of *proof obligations* may be automatically derived. Proof obligations seek to establish properties like the preservation of invariant and satisfaction of refinement obligations. A model is deemed correct when all proof obligations are successfully discharged.

Event-B employs syntactic schemas for constructing its safety and refinement proof obligations. A number of provers, some tailor-made and some external, attempt to automatically discharge an open proof obligation. Failing this, a user may provide proof hints (but not a complete proof script) to direct a combination of built-in rewrite procedures and automatic provers.

There was a concerted effort, funded by a succession of EU research projects, to make Event-B and its toolkit appealing and competitive in an industrial setting. One of the lessons of this mainly positive exercise is the general aversion of industrial users to interactive proofs. It is possible in principle (although hardly realistic in the context) to learn, through experience and determination, the ways of underlying verification tools and master refinement and decomposition to minimize proof effort. The methodological implications of avoiding manual proofs are far more serious: building a good model is always a trial and error process; any non-trivial design necessitates re-starting from scratch or doing considerable model refactoring (i.e., re-arranging refinement layers). This means redoing manual proofs which makes time spent proving dead-end efforts seem like pointlessly wasted. Hence, proof-shy engineers too often do not make good use of the formal specification stage as they tend to hold on to an initial, often incoherent design.

Proof economy is thus one of the worthy goals to explore. We propose to attack the problem from several directions at once.

First, we want to change the way proofs are done, at least in an industrial setting. Instead of doing an interactive proof - something that is an inherently one-off effort - we shall invite users to write and proof a schematic lemma that is strong enough to discharge an open proof obligation. Such a lemma may not refer to any model variables or types and is, in essence, a property supporting axiomatization of Event-B mathematical language. If such a lemma cannot be found or seems to difficult to prove, the model must be changed. Experience suggests that a modelling project is likely to have a fairly distinctive usage of data types and mathematical constructs; we conjecture that this leads to a distinctive set of supporting lemmas. We also hypothesise that such distinctnesses is pronounced and dictated by modeller's experience and background as well as model subject domain. We have also observed that the style of informal requirements - structured text, hierarchical diagram, structural diagram - has an impact on a modelling style.

A schematic lemma is a tangible and persistent outcome of any modelling effort, even an abortive one. The 'distinctnesses' hypothesis suggests that such a lemma is likely to be useful in a next iteration and, by an extension, there is a point when all relevant lemmas are collected; then modelling, in a given context, becomes completely free of interactive proofs. We intend to demonstrate that such a point is reachable at least for a number of non-trivial existing models.

One problem we foresee is the accumulation of lemmas of which majority could be irrelevant outside of some narrow context. Adding an extra lemma in a proof context generally makes proof more difficult. It could conceivably reach a point where nothing may be proven due to the sheer number of supporting properties. Some form of filtering is thus paramount. From the outset, we make a set of support lemmas bound to a specific application and private to a group of developers. To filter out irrelevant lemmas within such context we are looking into state-of-the-art techniques in hypothesis filtering. At the very basic level, we start with filtering by user-defined template expressions where a lemma is included only if the goal or some hypothesis predicate matches a given template.

Our second direction of attack is an extensive use of cheap computational resources made available by data centres and cloud computing [3]. Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g. networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. Cloud computing provides different deployment models: Infrastructure-as-a-service (IaaS), Platform-as-a-service (PaaS) and Software-as-a-service (SaaS). Generally, many functionalities are currently being provided as a service under a broad term of "Everything-as-a-service (XaaS)" on the cloud. Provers-as-a-service is a natural direction given that provers are CPU and memory intensive; at the same time, running a collection of (distinct) provers on the same conjecture is a trivial and fairly effective way to speed up proof given plentiful resources. Usability perception of interactive modelling methods such as Event-B is sensitive to peak performance when a burst of activity (new invariant) is followed by a relatively long period of idling (modeller thinking and entering model). The cloud paradigm, where only the actual CPU time is rented, seems well suited to such scenario. Also, the cloud's feature of scalability plays a critical role in this situation. Virtualization, which is a key underlying concept of cloud computing enables the installation of multiple virtual machines on the same physical machine. Virtualization supports scaling through load balancing. Here, a service running on a provisioned virtual machine scales either up or down to handle varying amounts of requests using a live migration process.

From the technical perspective one low hanging fruit was the absence of interface between Event-B and TPTP provers. Thus hosting a collection TPTP provers on a cloud and providing an interface to them seemed a promising direction. To simplify translation we decided to use Why3 [2] umbrella prover that offers a far more palatable input notation and also supports SMT-LIB provers. A plug-in to the Rodin Platform was realised to map between the Event-B mathematical language and Why3 *theory* input notation (we don't make use of its other part - a modelling language notation). The syntactic part of the translation is trivial: just one Tom/Java class. The bulk of the effort is in the axioms and lemmas defining the properties of the numerous Event-B set-theoretic constructs. We are already at a stage where there is a working prototype able to discharge (via provers like SPASS and Alt-Ergo) a number of properties that previously required interactive proof. At the same time, we realize that axiomatization of a complex language like Event-B is likely to be an ever open problem. It is apparent that different provers prefer differing styles of operator definitions: some perform better with an inductive style (the size of an empty set is zero, adding one element to a set increases its size by one) others prefer regress to already known concepts (here exists a bijection such that ...). Since we don't know how to define an optimal axiomatization, even for any one given prover, we offer an open translator with which a user may define, with as many cross-checks as practically reasonable, a custom embedding of Event-B into Why3.

Despite provided users with axiomatization option, some general axioms are provided by us. In early development stages it was noticed that with some provers (e.g. Z3) machine quickly runs out of memory, thus axiomatization process of WHY3 functions was highly constrained by state space. Secondly, lack of axiomatization experience resulted in a proving anything case, where missing finiteness statement or use of bi-implication instead of implication resulted in contradicting axiom, which then allowed to prove anything. Simple, though not a completely reassuring solution is to have and check contradicting dummy-lemma at the end of your library. Operator axiomatization was done in similar manner to [ref], where all Event - B set operators, where defined membership form.

Analysing undischarged proof obligations of different Event - B models was a next step in axiomatization process. It was noticed that a lot of undischarged goals required fairly simple, but very model - specific lemmas. Therefore, discovered missing lemmas had a high potential to be redundant in another context. In order to not expand state space dramatically some generalization of lemmas is necessary or otherwise as previously mentioned some state-of-the-art hypothesis filtering.

Doing proofs on a cloud opens possibilities that we believe were previously not explored, outside, perhaps, prover contests results. The cloud service keeps a detailed record of each proof attempt along with (possibly obfuscated) proof obligations, supporting lemmas and translation rules. There is a fairly extensive library of Event-B models constructed over the past 15 years and these are a ready of source of proof obligations. Some of these come from academia and some from industry. We are now starting to put models through our prover plug-in in order to collect some tens of thousands of proof obligations. One immediate point of interest is whether one can train a classification algorithm to make useful prediction of relative prover performance. If such a prediction can yield statistically significant results, prover call order may be optimized to minimize resource utilization while retaining or improving average proof time.

References

[1] J.-R. Abrial. *Modelling in Event-B*. Cambridge University Press, 2010.

[2] François Bobot, Jean-Christophe Filliâtre, Claude Marché, and Andrei Paskevich. Why3: Shepherd your herd of provers. In *Boogie 2011: First International Workshop on Intermediate Verification Languages*, pages 53–64, August 2011.

[3] Y. Jadeja and K. Modi. Cloud computing - concepts, architecture and challenges. In *Computing, Electronics and Electrical Technologies (ICCEET), 2012 International Conference on*, pages 877–880, March 2012.

[4] The RODIN platform. Online at http://rodin-b-sharp.sourceforge.net/.