On the Organisation of Program Verification Competitions

Marieke Huisman[1], Vladimir Klebanov[2], and Rosemary Monahan[3]

[1] University of Twente, The Netherlands
[2] Karlsruhe Institute of Technology, Germany
[3] National University of Ireland Maynooth, Ireland

Abstract. In this paper, we discuss the challenges that have to be addressed when organising program verification competitions. Our focus is on competitions for verification systems where the participants both formalise an informally stated requirement and (typically) provide some guidance for the tool to show it. The paper draws its insights from our experiences with organising a program verification competition at FoVeOOS 2011. We discuss in particular the following aspects: challenge selection, on-site versus online organisation, team composition and judging. We conclude with a list of recommendations for future competition organisers.

1 Introduction

As verification competitions are becoming more popular we are gaining experience on how to organise them. There have been three competitions to-date focusing on a particular form of program verification. In this paper, by program verification, we mean a formal verification process, where a human user contributes in two ways: (a) by formalising an informally stated requirement specification for a program, and (b) by providing (if necessary) some guidance to the verification system to show formally the conformance of the program to the requirement. This setup is due to the strong properties being shown and the heterogeneity of the verification system landscape. It makes such a competition quite different from other competitions in verification (e.g., SV-COMP [1]) or automated reasoning (e.g., CASC [7]), where the formal requirement is identical for all teams and fixed in advance, and no user guidance in showing it is accepted. In our context, organisers have to deal with a whole new range of issues, such as judging the adequacy of the requirement formalisation.

Although verification competitions up until now have varied in their organisation, all of them succeeded in bringing together the verification community to compare their tools and techniques. Hence, it is important that these competitions become a regular event, perhaps co-located with the same conference every year in order to increase participation and build momentum. Through our participation, as both organisers and competitors, we realise that competitions are not mature, and we often make "imperfect" arrangements in order to

increase participation and build momentum. Therefore, such competitions are also a learning process from an organisational viewpoint. In the remainder of this paper we share our experience of organising a verification competition at FoVeOOS 2011. We begin with an overview of verification competitions held so-far. Then, we discuss the challenge selection, on-site versus online organisation, team composition and judging. Finally, we present a list of recommendations for the organisers of future verification competitions.

2 History of Program Verification Competitions

The first program verification competition[4] [5] was an informal event, held during the VSTTE 2010 conference as a prelude to more formal competitions at future meetings. The competition was organised by Natarajan Shankar, SRI International, and Peter Müller, ETH Zürich, who were assisted by Gary Leavens, University of Central Florida, in the judging. The challenges involved simple data types that are supported by most verification tools for sequential or functional programs. The teams, of up to three people, were given five verification exercises with informal specifications, test cases, and pseudo code. The task was to prepare a reproducible verification of executable code relative to a formalisation of the specifications using one or more verification tools. The allotted time was two hours, and the solutions were judged for completeness and elegance as well as correctness.

The second competition was initiated by the COST Action IC0701 [2], whose topic is advancing formal verification of object-oriented software. Organised by Marieke Huisman, University of Twente, Vladimir Klebanov, Karlsruhe Institute of Technology, and Rosemary Monahan, National University of Ireland, Maynooth, the competition aimed to evaluate the usability of verification tools in a relatively controlled experiment that could be easily repeated by others. This competition was inspired by the first (in fact, both Vladimir Klebanov and Rosemary Monahan participated in the first competition), and had a similar format: up to three people forming a team, all participants physically present, and teams using any verification system of their choice. The event took place the afternoon prior to the FoVeOOS 2011 conference. Three challenges were given in natural language and required a solution that consisted of a formal specification and an implementation, where the specification was formally verified with respect to its implementation. In contrast to the VSTTE event, a fixed time slot was assigned for each of the challenges provided. This setup was chosen in order to increase precision of the tool comparisons.

In both of these verification competitions, team registration was not required in advance so participation was quite informal, with student teams especially encouraged. This setup proved to be successful with eleven teams participating in the VSTTE competition and six teams participating in the FoVeOOS competition. It is interesting to note that in each competition every team used

[4] In the sense defined in the introduction.

one verification tool and each tool was represented once. There was no explicit ranking of solutions or a winner announcement. The judging panel manually inspected the solutions and pointed out strengths and weaknesses according to the criteria of completeness, elegance, and automation; these subjective results were presented during the conferences to foster discussions among the participants. In both cases a post-competition paper provided the chance for further discussion and revision of competition solutions.

The third competition [4] had a different format to the previous two: it was an online competition in which participants had 48 hours to attempt five problems that were presented on the conference website. Any programming language, specification language, and verification tool was allowed in the solution. The competition, affiliated with VSTTE 2012 and organised by Jean-Christophe Filliâtre, CNRS, Andrei Paskevich, University of Paris-Sud 11, and Aaron Stump, University of Iowa, attracted 29 teams (79 participants total) using 22 verification tools. Each problem included several sub-tasks, e.g., safety, termination, behavioural correctness, etc., and each sub-task was given a number of points. Submissions from teams of up to four people, were ranked according to the total sum of points they scored. The competition resulted in the award of two gold medals, two silver medals, and two bronze medals. Within each medal class, the teams were tied for points, with the gold medal teams earning perfect scores of 600 points.

3 Challenge Selection

An important step in the organisation of a verification competition is the selection of challenges. Many of the verification challenges posed in competitions so far have been variations of typical "text book" exercises. While posing more open problems is an aspiration, these problems make it difficult to compare solutions and may be daunting to new participants. Here, we discuss the importance of the selection of competition challenges with the aim of making the competitions accessible to all levels of participants, and in particular, making the event attractive to newcomers to the area.

Having a pool of past competition problems in a repository like Verify This [3] assists the challenge selection as one can vary existing problems or can extend the problems to obtain similar or more advanced challenges. On the one hand, such a repository would be a perfect test case for tool developers and a perfect training base for new users. On the other, we want to avoid tool builders tailoring their tools towards the competition database simply to win competitions rather than contributing to the wider verification challenge.

3.1 Challenge Variety

Competition challenges should not (dis)favour a particular tool or approach if at all possible. Verification competitions held so far did not feature tracks or divisions, so quite different tools were pitted against each other. In our opinion,

the challenges issued so far have been favouring tools that target functional programming languages. Tools that target object-oriented languages were in general at a disadvantage.

While introducing tracks or divisions, increases the organisers' effort and requires a bigger participant critical mass, we suggest that the organisers define a set of core challenges, which all teams address, and several "speciality" tracks, where teams can choose the set of challenges that best match the stronger features of their tool. However, it is good to have a nonempty set of core challenges that are attempted by all participants, because one of the important goals of any competition is the comparison of the different solutions.

Additionally, we believe that a good (core) challenge set should be distinctive, i.e., only a few teams should be able to solve each challenge. At the moment, we do not think that there is enough experience with verification competitions, but we believe that eventually the problems should be so distinctive that even strong teams might not be able to solve all challenges (within the given time). Furthermore, we advocate individual time slots for each challenge.

3.2 Challenge Sources

It is important that the challenges are attempted in advance to determine specification pitfalls and to determine the time that should be allocated to solving each challenge. The drawback is that this reduces the pool of people that can participate in a competition.

To obtain a better challenge set with more variety, a possibility is to ask participants to contribute a challenge, with a worked-out solution and an estimate of the required time. This should be something that they consider can be done very well (fast, elegant etc.) with their approach, and would be challenging for other tools. We believe that this will force participants to explicitly consider the strong features of their own tool. It will also help to balance the challenges so that they are not all targeting the same language and problem set. Challenges (and solutions) should be submitted well in advance, to allow the organisers to check that the solution is actually solvable, and does not just use a "trick" that only the tool developers know about.

However it does not seem a good idea to make challenge submission obligatory, as this might prevent non-developer teams from participating, and it would make last minute participation complicated. Instead it would be a better idea to reward challenge submission. For example, bonus points will be awarded if the standard solution is "better" than all submitted solutions.

3.3 Importance of Small Challenges

With certain regularity, we face expectations that competitions should feature larger and more complex challenges. In fact, this has been almost the predominant dimension along which progress of verification as a whole has been evaluated: how large/complex a system can be formally verified? We would like to

argue that this view needs to be supplemented with a different one involving the verification of small, highly controlled challenges.

Two observations lead us to this opinion. First, the larger the challenge, the more difficult/expensive it is to reproduce it. It is a significant advantage if the competition situation can be re-enacted by anybody with access to a verification system and a few hours to dedicate to the task. Larger time demands significantly hamper penetration in the notoriously short-for-time work environment. Second, when working on larger challenges, it is more difficult to keep track of net time and effort spent, as other day-to-day activities (be it sleeping, teaching, or other work) interfere.

It is true that certain tool capabilities that are essential for working on large projects (hierarchical development, abstractions in-the-large, proof and change management) are difficult to test with small challenges. At the same time, a number of larger comparative case studies in formal development and verification have already been carried out. Here we name examples such as the "production cell" case study [6] and the Mondex case study [8].

What is missing is an on-going effort to evaluate *usability* of verification systems, i.e., the amount of work that can be carried out by an average user (preferably not the system's designer) in a fixed amount of time. We conclude that competitions with small focused challenges are an appropriate vehicle for this.

4 On-site versus Online Competitions

To-date, verification competitions have been mainly on-site events, with all team members participating in a common location, for the duration of the competition. These events are typically between two and four hours long with a selection of small challenges to be completed within the allocated timeframe. The location of an on-site competition must provide adequate space for participants, with sufficient caffeine and sugar supplies, and without disturbance from others. It is an advantage to have all teams working in close proximity as this adds to the enthusiasm and adrenaline; with teams reacting when competitors rejoice as a challenge is solved, or lament as the verification doesn't work out as nicely as expected.

From the organisers' perspective the advantage of such a setting is the ease of ensuring that teams participate in accordance with the competition rules (number of teams members involved etc.). A further advantage is that the organisers are available to notice, and clarify, any mis-understandings that arise. From the participants' perspective, the major advantage is the opportunity to interact with users and developers of competing tools after the competition. The momentum, built up through this interaction regarding alternative solution strategies, has led to tool comparisons in a number of conference publications [5] [2]. Another observation is that on-site competition with fixed time slots encourages co-operation between team members. This is due to the urgency of a well-planned solution which solves the challenge in a limited time.

The major disadvantage of an on-site competition is the cost and effort required to get all participants present. Both co-locating the competition with a conference on a related topic so that participants are already on-site and the provision of funding for student team participation have proved to be fruitful strategies.

Online competitions, like those used in many programming competitions, allow for greater participation as teams may participate without travelling. They allow for competitions of longer duration and hence challenges of a larger size. We believe that on-site and online competitions complement each other and should co-exist. For example, larger problems are more suited to off-site challenges that could be issued for tool developers whereas smaller problems are more suited to students/tool users rather than developers.

In either competition setup, we suggest that the interaction between teams after the competition could be increased through the provision of live recordings of the competition (a suggestion, for which we thank Gerhard Schellhorn). Monitoring a team's interaction with a tool could reveal strategies and tips for tool users as well as aiding tool evaluation and increasing interest in verification competitions itself. Of course, the interests of judges and spectators must be balanced with the privacy preferences of participants.

5 Team Composition

With all the verification competitions that have been held so far, one of the dominating questions has been on how to control for the human factor, since it is not meaningful to test the verification system alone. Without proper control, there is a risk that competitions will be dominated by "super experts"—tool developers with many years of experience. They are aware of all the ins-and-outs of the tools, and can even make small changes to the tool during the competition. They also know how to tweak the specifications so that they are easily expressed in the tool's input language and are easily accepted by the tool.

Several ideas exist on how to ensure that super expert users do not skew the competition to their advantage: one could allow only teams with non-expert users (as suggested by Erik Poll: forbid any participants with a Master's degree); one could force expert users to use a tool that they are not very familiar with; one could have mixed teams with users of different tools; or one could forbid tool developers to participate (except as judges).

Unfortunately, all these suggestions seem to have practical problems (how to get enough participants that are not tool builders or experts; program verification tools often have a steep learning curve; and manuals are not always available). Therefore, we believe that the best workable solution is to consider team composition and tool maturity when judging the solutions. In addition, for future competitions we will explicitly encourage several teams using the same tool to participate, allowing user competitions within the overall tool competition.

To increase the variety of participants, we believe that there should be some reward for taking part in the competition. In particular, if you are not a tool developer, then why would you bother participating in a competition? If there is a winner announced, you can put this on your CV. However, competitions could have so many categories that almost all tools and participants can be judged so that they win a prize. Organising competitions, where participants are invited to contribute to a post-competition publication about the challenges could also provide a motivation.

6 Judging

Judging verification competition solutions is challenging, but it is also very exciting. Solutions are typically judged for their correctness, their completeness and their elegance. While tools may verify if a given implementation is correct with respect to the given specification, determining if a solution is complete and elegant is not so straightforward. Presentation of proofs, degree of automation in verification, annotation overhead, and the extent of verification (e.g., partial vs. total correctness, etc.) are some of the further considerations.

In the first two verification competitions the judges manually inspected the solutions providing subjective results at a presentation during the co-located conference. A follow up conference paper allowed the participants to clean up and revise their solutions for public consumption. In the third, a scoring scheme was applied to each solution and submissions were ranked according to the total sum of points they scored. However, it was noted upfront that "a certain degree of subjectivity in judgement is inevitable and should be considered as part of the game."

Tools, solutions and team member abilities vary greatly, so there are many parameters that play a role when measuring the quality of the solutions. One strategy to aid the judging process is to categorise the tools according to their characteristics and maturity, classifying the results based on these categories. Usability should be measured, qualitatively until better metrics can be found.

6.1 The Role of the Tools in Judging

An important question is whether judging has to involve replaying the solutions in the tools. There are arguments both for and against this. The main goals related to the requirement of tool replay are: punishing fraud, tool unsoundness, and specification inadequacy. While we discount the first issue, in the current state of affairs, the others, especially the third, are of great importance. It is important to keep in mind that there is no canonical requirement formalisation, and tool replay does not bring an ultimate judgement.

If the tool produces an explicit proof object, inspecting it may expose both unsoundness and specification inadequacy. Otherwise, the only way a tool may help is mutation testing. If after changing a part of the requirement (including the program being verified), the tool can still show conformance, there is *a*

probability that the changed part contributes nothing to the problem. This *may* indicate an issue with the tool soundness or requirement adequacy (or both).

The biggest argument against tool replay is the effort, both in installing and running the tools and carrying out the solution analysis as described above. There exists many versions of many verification tools, which can be installed on many different platforms, each using many different plug-ins (all having many versions). Knowing the exact tools that will be used in the competition in advance, especially if a large number of tools participate, is essential. Taking these arguments together, at the current level of competition maturity we would not advocate tool replay, unless doubts in the quality of a solution are present, or if the replay in the tool promises significant benefits in judging the solution (e.g., by advanced proof presentation). Replay could be made easier in the future, if tool developers make their tools available via a web interface, or if virtual machine images could be provided.

6.2 Understanding the Argument

In order to judge the completeness or elegance of a solution it is necessary to understand the argument behind it. Unfortunately, program verification arguments are notoriously difficult to communicate. This applied both to systems that expose an explicit proof object (i.e., a derivation in a certain calculus) and to systems where the user only works with the annotated source code and does not see the logical reasoning behind it.

The explicit proof object is typically too fine-grained, while the annotated source code often does not make the argument structure explicit. Moreover, whenever a tool silently infers a particular fact (a termination measure, for instance), it reduces the burden on the user but may appear as a gap in reasoning to an outsider. In any case, a good portion of knowledge about the background theory implemented in a tool is needed to understand a solution.

A team's approach to solving a problem is often one that the adjudicator themselves would not have used. While this is normal for any problem-solving scenario, we have noted that the solution presented is often a result of the particular strengths and weaknesses of the verification tool used.

While the overhead of adjudicating solutions in many different formalisms is quite high, the benefits are many. While adjudicators will not be an expert in every tool, it is our experience that expert non-users of tools can largely understand the various solutions. Examining solutions for the same challenges in many different verification environments is extremely educational, and experiencing different approaches to solving the challenges (tool-driven or user-driven) is also a fun component of the process.

The enthusiasm of participants, both in terms of the tools that they use and the solutions that they develop is also uplifting. Explicitly scheduling an explanation session where the team members talk an adjudicator through the solution (possible with on-site competitions only) would take full advantage of this enthusiasm and assist the judges in developing a complete understanding of the teams' solutions. Above all, we believe that participants should be encouraged

to clean up their solutions and interact after the competition, to discuss their submissions and to compare the strengths and weaknesses of each tool.

7 List of Recommendations for Organisers

To conclude, we end this paper with a list of recommendations for future verification competition organisers. These recommendations arise from our experience of participating and organising verification competitions, and from our interactions with other competition particpants. We believe that these recommendations will contribute to improved verification competitions in the future.

- Associate the competition with a well-established, regular event.
- Encourage newcomers to participate in the competitions.
- Set up a repository of challenges.
- Remember the goals of competitions, and do not disregard small challenges.
- Ask participants to contribute challenges, and reward them for this.
- On-site verification competitions have their place, do not make all competitions online.
- Encourage discussion between participants about their solutions.
- Record teams during competition participation.
- Judge teams depending on the maturity of their tools and the experience of team members.
- Let team members explain their solutions to the judges.
- Encourage multiple teams using the same tool to participate.
- Invited participants to contribute to a post-competition publication.
- Rotate organisation and participation.

We look forward to further verification competitions and are confident that, as they mature, they will become a major contributor to benchmarking verification tools, improving their capabilities, and extending their usability.

References

1. D. Beyer. Competition on software verification (SV-COMP). In C. Flanagan and B. König, editors, *Proceedings of the 18th International Conference on Tools and Algorithms for the Construction and of Analysis Systems (TACAS 2012)*, volume 7214 of *LNCS*, pages 504–524. Springer-Verlag, Heidelberg, 2012.
2. T. Bormer, M. Brockschmidt, D. Distefano, G. Ernst, J.-C. Filliâtre, R. Grigore, M. Huisman, V. Klebanov, C. Marché, R. Monahan, W. Mostowski, N. Polikarpova, C. Scheben, G. Schellhorn, B. Tofan, J. Tschannen, and M. Ulbrich. The COST IC0701 verification competition 2011. In B. Beckert, F. Damiani, and D. Gurov, editors, *International Conference on Formal Verification of Object-Oriented Systems (FoVeOOS 2011)*, LNCS. Springer, 2012. To appear.
3. COST Action IC0701. Verification problem repository. `www.verifythis.org`.
4. J.-C. Filliâtre, A. Paskevich, and A. Stump. The 2nd Verified Software Competition: Experience report. In A. Biere, B. Beckert, V. Klebanov, and G. Sutcliffe, editors, *Proceedings of the 1st International Workshop on Comparative Empirical Evaluation of Reasoning Systems (COMPARE 2012)*, 2012.

5. V. Klebanov, P. Müller, N. Shankar, G. T. Leavens, V. Wüstholz, E. Alkassar, R. Arthan, D. Bronish, R. Chapman, E. Cohen, M. Hillebrand, B. Jacobs, K. R. M. Leino, R. Monahan, F. Piessens, N. Polikarpova, T. Ridge, J. Smans, S. Tobies, T. Tuerk, M. Ulbrich, and B. Weiß. The 1st Verified Software Competition: Experience report. In M. Butler and W. Schulte, editors, *Proceedings, 17th International Symposium on Formal Methods (FM)*, volume 6664 of *LNCS*. Springer, 2011. Materials available at `www.vscomp.org`.
6. C. Lewerentz and T. Lindner. Case study "production cell": A comparative study in formal specification and verification. In M. Broy and S. Jähnichen, editors, *KORSO: Methods, Languages, and Tools for the Construction of Correct Software*, volume 1009 of *LNCS*, pages 388–416. Springer, 1995.
7. G. Sutcliffe and C. Suttner. The state of CASC. *AI Communications*, 19(1):35–48, 2006.
8. J. Woodcock. First steps in the Verified Software Grand Challenge. *Computer*, 39(10):57–64, 2006.