AFRL-RI-RS-TR-2012-290

# CROWD-SOURCED PROGRAM VERIFICATION

UNIVERSITY OF WASHINGTON

*DECEMBER 2012*

FINAL TECHNICAL REPORT

**AIR FORCE RESEARCH LABORATORY**
**INFORMATION DIRECTORATE**

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

# NOTICE AND SIGNATURE PAGE

AFRL-RI-RS-TR-2012-290 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE DIRECTOR:

**/ S /**

ROBERT L. KAMINSKI
Work Unit Manager

**/ S /**

WARREN H. DEBANY, JR.
Technical Advisor, Information
Exploitation & Operations Division
Information Directorate

| REPORT DOCUMENTATION PAGE | *Form Approved* OMB No. 0704-0188 |
|---|---|

| | | |
|---|---|---|
| **1. REPORT DATE** *(DD-MM-YYYY)* DECEMBER 2012 | **2. REPORT TYPE** FINAL TECHNICAL REPORT | **3. DATES COVERED** *(From - To)* APR 2011 – JUN 2012 |

| | |
|---|---|
| **4. TITLE AND SUBTITLE** CROWD-SOURCED PROGRAM VERIFICATION | **5a. CONTRACT NUMBER** FA8750-11-2-0221 |
| | **5b. GRANT NUMBER** N/A |
| | **5c. PROGRAM ELEMENT NUMBER** 62303E |
| **6. AUTHOR(S)** Michael Ernst and Zoran Popović | **5d. PROJECT NUMBER** BM90 |
| | **5e. TASK NUMBER** WA |
| | **5f. WORK UNIT NUMBER** SH |
| **7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)** UNIVERSITY OF WASHINGTON<br>OFFICE OF SPONSORED PROGRAMS<br>4333 BROOKLYN AVE NE<br>SEATTLE WA 98195-0001 | **8. PERFORMING ORGANIZATION REPORT NUMBER** N/A |
| **9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)** DARPA<br>675 North Randolph St<br>Arlington, VA 22203<br>Air Force Research Laboratory/RIG<br>525 Brooks Road<br>Rome NY 13441-4505 | **10. SPONSOR/MONITOR'S ACRONYM(S)** N/A |
| | **11. SPONSORING/MONITORING AGENCY REPORT NUMBER** AFRL-RI-RS-TR-2012-290 |

**12. DISTRIBUTION AVAILABILITY STATEMENT**
Approved for Public Release; Distribution Unlimited. This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09.

**13. SUPPLEMENTARY NOTES**

**14. ABSTRACT**
This effort investigated a radical new approach to program verification by transforming the problem domain into one in which non-experts could interact with a program through the framework of a video game. For the investigation, the contractor constructed a prototype of a crowd-sourced verification system that takes as input a given program and produces as output a proof of correctness of that program. The system translates a select verification property into a game. The completion of the prototype suggested that research along this avenue is worth continuing.

**15. SUBJECT TERMS**
Program verification, Proof of Correctness, Software Development

| **16. SECURITY CLASSIFICATION OF:** | | | **17. LIMITATION OF ABSTRACT** | **18. NUMBER OF PAGES** | **19a. NAME OF RESPONSIBLE PERSON** ROBERT L. KAMINSKI |
|---|---|---|---|---|---|
| **a. REPORT** U | **b. ABSTRACT** U | **c. THIS PAGE** U | UU | 13 | **19b. TELEPONE NUMBER (*Include area code)*** NA |

## Table of Contents

# Crowd-Sourced Program Verification
# Final Performance Report

## 1 Summary

In this project, we attempted a radical new approach to program verification by transforming the problem domain into one in which non-experts could interact with a program through the framework of a video game. For the investigation, we constructed a prototype of a crowd-sourced verification system that takes as input a given program and produces as output a proof of correctness of that program. The system translates a select verification property into a game.

The completion of the prototype suggested that research along this avenue is worth continuing. Items for further exploration include the development of additional type systems (the prototype works solely for null pointer exceptions), expanding on the program translation and game rendering systems to accommodate input programs of a much larger size, and implementing systems around players to incentivize play and drive engagement.

## 2 Introduction

Our increasing dependence on software makes it imperative to find more effective and efficient mechanisms for improving software reliability. Formal verification is an important part of this effort, but since its practice is currently restricted to specially trained engineers, it remains difficult and expensive. However, if we were able to transform ordinary computer users into ones capable of performing verification tasks, we could achieve a dramatic reduction in the cost of producing verified code.

We proposed a one-year technical evaluation (“seedling”) to determine whether the concept of crowd-sourced program verification, by translation of a verification problem into a puzzle game, is feasible and worthy of further development.

We described a game that visualized the problem space into a series of interconnected pipes of variable width. Small or large balls roll down these pipes, and the player can set each pipe to be either narrow or wide. The player’s goal is to ensure that all the balls can traverse the network of pipes.

The network of pipes in the game are directly generated from the flow (similar to dataflow) properties of a program. The pipes represent program variables, their widths represent types, and the balls represent approximations to run-time values. Player settings for the pipes’ widths directly correspond to type annotations in the program that can be mechanically checked and provide a proof of partial correctness. The system, when complete, will translate the final configuration of the game board to a proof.

## 3 Methods, Assumptions, and Procedures

With this project, our ultimate goal was to make software verification available, for free, via crowd-sourcing to an audience that enjoys the challenge of playing a game. Even if the crowd is not able to fully verify a program, partial verification still has great value, permitting the valuable time of highly-skilled engineers to be focused on the most important parts of the program. This approach is unlike, though complementary to, other work that aims to make verification easier for trained computer scientists. Our work also had a different goal than dynamic analyses that detect errors at run time and terminate the program before more damage is done. By contrast, this work sought to detect certain errors statically, at compile time, so that they are impossible in the field, no matter what data the program is run on.

In our approach, the game player is unknowingly accomplishing the task of *type inference*: creating an annotation for each variable declaration in the program that gives additional information about that variable's type. Type inference tools exist that can infer type annotations for correct un-annotated code. Whereas type *checking* is largely a solved problem, type *inference* emphatically is not, especially with regard to programs that are not verifiable.

The crowdsourcing approach is complementary with the development of better inference tools. In particular, we can run the best available practical inference tool before the player starts, and use that as the initial configuration that the player attempts to improve. This leverages human intuition exactly where it is needed– every formal analysis has a limited vocabulary of properties that it can express, which limits the properties that it can prove. It is an unavoidable fact, related to the un-decidability of the halting problem, that for every conservative (sound) program analysis, there are programs that never go wrong at runtime but that the program analysis rejects as potentially erroneous. Our belief is that humans will be able to overcome, or at least pinpoint, these limitations via higher-level reasoning, pattern-matching, and heuristics to guide themselves through very large search spaces.

In order to leverage this human problem solving, however, we must motivate large groups of people to want to play the game. This remains as critical future work now that we have demonstrated the feasibility of the approach. Here, the most important guiding principle in our game development process is our iterative approach. In other words, we cannot assume that we have gotten it right the first– or second– time. Continuing iteration based on analysis of gameplay data allows us to tune the game design to increase player engagement. In some sense, we will use randomized sensitivity experiments to guide the development of the experience in order to produce the experience that maximizes the rate of return to the game (a common objective for all video games). We rely on the ability to rapidly prototype a number of possibilities and use the analysis of student play to guide subsequent design improvements.

## 4 Results and Discussion

We began from an initial prototype, in which a small (100 line) hard-coded program of a single file was successfully "played" as a game. From there, we worked on developing new game mechanics (aspects of gameplay that are apparent to the game player, such as new icons on the game board) that paved the way for ingesting and representing more complex programs:

- A way to handle maps (hash tables) in the underlying program, via a "pin-striping" of the chutes and the balls, and a special lookup that requires the pin-striping to match.
- A way to handle casts, or loopholes in the type system, via a "buzzsaw" that converts a ball from large to small so that it fits in more pipes
- A way to handle verification of multi-class (multi-file) programs, via multiple "levels" in a "world," and a world map that permits switching between the levels

For the first several months of the project, translation from a program to the game was done by hand, and there was no translation from the game back to a proof for the program. We hand-translated a few larger programs, including a portion of our system's own implementation, into games.  This yielded insight into algorithms to automate the process, as well as data structure representations, file formats, and game mechanics.

From there, we were able to design a translation algorithm from programs to games. We began prototyping the automated translation of programs to games (the initial algorithm did not handle flow-sensitive type refinement). However, we did not finish prototyping the full scope of translation from programs to games.

We created an algorithm for laying out game boards that takes as input a set of constraints, and produces as output the exact locations of each pipe and intersection that the player sees on the game board. Our first implementation created many unnecessary edge crossings, and was seen as confusing and unattractive. We threw away the old, manual, ad hoc approach to laying out game boards and replaced it with calls to GraphViz. The output is not perfect, but is a great improvement even at this early stage.

Early versions of the Flash game did not perform as well as expected. We improved this by introducing new algorithms and implementations for graph analysis, for instance to handle recursion in the game board. We also refactored and improved the overall implementation. We improved scalability; the game used to be painfully slow for even modest programs of 100 methods. We made many more layout improvements (that also simplified the layout code).

Additionally, the user interface initially handled only a single level of limited size, but we modified it over the course of development to accommodate a level of arbitrary size.

Navigation among boards is handled via a horizontal scrollable pane of thumbnails. A World Map was added that permits the user to navigate among levels in the world.

We designed and implemented a file format for games, which permits layouts to be persisted across sessions and enables integration of multiple tools. Over the course of the project, we enhanced the file format to represent more information about the game and the player's interaction with it.

We worked continuously on one particularly challenging problem: the handling of map key properties (that is, that a value is a key in a given Map/Dictionary object). We designed a representation for the special behavior of the Map.get routine. Given

```
Map<K, V> m;
```

this expression

```
m.get(myKey)
```

returns a non-null value exactly if V is a non-null type *and* myKey is a key in map m. This required two extensions: a game mechanic to represent this complex condition, and tracking, for every value and every map, whether the value is a key in that map.

We came up with a solution in which pin-striping represents whether a value is a key in a map (most values aren't, for most maps), and in which there is a new widget used only for calls to Map.get. We implemented our game-side representation for map keys and Map.get and it now is possible to play a game board that uses the new features, but our system does not yet work end-to-end: the tools do not yet convert a program that uses Map.get into the file format that we have defined and that the game uses.

We devised multiple new algorithms for collision detection (when a large ball tries to roll into a narrow pipe). One is a fast, local algorithm that depends only on characteristics of the pipes: what contents a pipe might contain. Another is a global algorithm that effectively simulates the board to determine what contents a pipe actually does contain. This global flow-sensitive algorithm is more precise and permits some game configurations that would be rejected by the simpler local algorithm. We have implemented the simple algorithm. We plan to refactor our code and some point to make collision detection pluggable and changeable upon demand.

We performed initial, limited user testing to see whether the game is understandable, solvable, and fun, and we made many improvements as a result. More testing is required.

We designed a system by which new "skins" for the game can be added. The game mechanics will be identical in each case, but the visual representations differ. A first example is a "traffic" skin in which, instead of wide and narrow balls, there are one or two lanes of traffic. Merging into a smaller number of lanes causes a traffic jam, analogously to a large ball trying to roll into a narrow pipe. We implemented this skin art into the game; the minimal coding required validated our implementation approach so far. (The traffic game is missing a few features that the pipe-and-ball version has, such as map key support.) Preliminary tests indicate that users find the new skin more visually attractive, since cars are moving continuously. Future studies will determine whether that motion is helpful or distracting to players.

We created a preliminary website at which a user can upload a program, play a game, and get an annotated version back. (Most game players would never see this— they will simply be presented with a game. But this shows how the entire system would work.)

We created a new type system and derived a game from it. The type system tracks tainting, such as the use of untrusted inputs.

We created an algorithm that pre-computes results for player actions on the game board.

We added support for exceptions to the laws of physics, to accommodate certain bugs or casts.

We made some good progress on flow-sensitivity, which permits a single variable to have different types in different parts of its scope. This makes the type system and game much more precise, reducing false positives (the need for buzzsaws). However, it has proved much more challenging than we first anticipated, and did not make it into the scope of this project. Flow sensitivity continues to be worked on as part of the Crowd-Sourced Formal Verification (CSFV) project.

## 5 Conclusions

The prototype demonstrates that it is possible to create a game out of a verification task and suggests that research along this avenue is worth continuing. Of course, much more work needs to be done in order to build a robust and complete system for crowd-sourced verification. Items for further exploration include:

- **Expanding translation from programs to games to larger, real-world programs**: much work is required on two fronts to get the system ingesting and deploying larger programs. The first is the translation process, which needs additional attention in order to handle the various cases present in large programs. The second is the visual and gameplay representation of those programs.
- **The development of additional type systems**: the prototype works solely for null pointer exceptions, with additional work done on tainting; this could be extended to a large number of type systems if each is translated into a game.
- **Implementing systems around players to incentivize play and drive engagement**: the crowdsourcing aspect of the system will only work if there is an active community of players engaged with the game; while we have ideas for achieving this through timed

challenges, leaderboards, etc. partially informed from our experience on creating Foldit, we have not tried them with our program verification game.

- **Different "visual metaphors" for gameplay**: since the games solver and the visualization of the output of the games solver are separated, we could create multiple visualizations for each ingested program. Currently we have developed "pipes and balls" and "traffic," with the idea that each has particular strengths and weaknesses. With additional development, we could crate additional visual metaphors for gameplay that might appeal to different demographics or which might be more appropriate for one type system over another. We could also test each of these visual metaphors for engagement and select the best one based on playtest data.
- **Other approaches (both gaming and otherwise) to crowd sourced verification**: Another possible future direction would be to observe the best players and use machine learning or other techniques to mimic their strategies.

In the course of our development, we found that an extremely close collaboration and integration between the program verification and game portions of the system is necessary. In order for the game to accurately represent the state of the underlying program, it must operate in a way that may not be a game designer's ideal. Similarly, in order to create an engaging game that players return to of their own volition, mechanics and progression systems must be created that coexist with the underlying logic and do not invalidate the accuracy of the results. Future efforts along this area of inquiry should emphasize the collaboration between these two sometimes-oppositional forces influencing the design of the final product.

## 6 Publications

**"Verification games: Making verification fun"** by Werner Dietl, Stephanie Dietzel, Michael D. Ernst, Nathaniel Mote, Brian Walker, Seth Cooper, Timothy Pavlik, and Zoran Popovic. In *FTfJP'2012: 14th Workshop on Formal Techniques for Java-like Programs,* (Beijing, China), June 12, 2012.

**"A type system for regular expressions"** by Eric Spishak, Werner Dietl, and Michael D. Ernst. In *FTfJP'2012: 14th Workshop on Formal Techniques for Java-like Programs,* (Beijing, China), June 12, 2012.

**"Type Annotations specification (JSR 308)"** by Michael D. Ernst. Oct 2011.

**"Building and using pluggable type-checkers"** by Werner Dietl, Stephanie Dietzel, Michael D. Ernst, Kıvanç Muşlu, and Todd Schiller. In *ICSE'11, Proceedings of the 33rd International Conference on Software Engineering*, (Waikiki, Hawaii, USA), May 25-27, 2011, pp. 681-690.

## 7 LIST OF REFERENCES

[1] Erik Andersen, Yun-En Liu, Ethan Apter, Fran¸cois Boucher-Genesse, and Zoran Popovi´c. Gameplay analysis through state projection. In Proceedings of the Fifth International Con- ference on the Foundations of Digital Games, FDG ’10, pages 1–8, New York, NY, USA, 2010. ACM.

[2] Erik Andersen, Yun-En Liu, Rich Snider, Roy Szeto, and Zoran Popovi´c. Placing a value on aesthetics in online casual games. In Proceedings of the 2011 annual conference on Human factors in computing systems, CHI ’11, pages 1275–1278, New York, NY, USA, 2011. ACM.

[3] Erik Andersen, Yun-En Liu, Richard Snider, Roy Szeto, Seth Cooper, and Zoran Popovi´c. On the harmfulness of secondary game objectives. In Proceedings of the 6th International Conference on Foundations of Digital Games, FDG ’11, pages 30–37, New York, NY, USA, 2011. ACM.

[4] Erik Andersen, Eleanor O’Rourke, Yun-En Liu, Rich Snider, Jeff Lowdermilk, David Truong, Seth Cooper, and Zoran Popovi´c. The impact of tutorials on games of varying complexity. In Proceedings of the 2012 annual conference on Human factors in computing systems, CHI ’12, 2012. To appear.

[5] Yun-En Liu, Erik Andersen, Richard Snider, Seth Cooper, and Zoran Popovi´c. Feature-based projections for effective playtrace analysis. In Proceedings of the 6th International Conference on Foundations of Digital Games, FDG ’11, pages 69–76, New York, NY, USA, 2011. ACM.

## 8 LIST OF ACRONYMS

| | |
|---|---|
| AFRL | Air Force Research Laboratory |
| CSFV | Crowd-Sourced Formal Verification |
| DARPA | Defense Advanced Research Projects Agency |