CrowdMine: Towards Crowdsourced Human-Assisted Verification

Wenchao Li
UC Berkeley
wenchaol@eecs.berkeley.edu

Sanjit A. Seshia
UC Berkeley
sseshia@eecs.berkeley.edu

Somesh Jha
UW Madison
jha@cs.wisc.edu

ABSTRACT

We propose the use of crowdsourcing and human computation to help solve difficult problems in verification and debugging that can benefit from human insight. As a specific scenario, we explain how non-expert humans can assist in the verification process by finding patterns in portions of simulation or execution traces which are represented as images. Such patterns can be used in a variety of ways, including assertion-based verification, improving coverage, bug localization, and error explanation. Several related issues are discussed, including privacy and incentive mechanisms.

Categories and Subject Descriptors

B.7.2 [**Design Aids**]: Verification; H.1.2 [**User/Machine Systems**]: Human factors

General Terms

Algorithms, Verification, Human Factors

Keywords

Specification, verification, crowdsourcing, human computation

1. INTRODUCTION

The field of electronic design automation (EDA), in general, and formal verification, in particular, has relentlessly pushed for automation. For several problems, this is indeed the right strategy. But for many problems human insight and involvement remain invaluable. Consider, for example, the process of verifying a design. First of all, one needs to write a specification, typically in the form of properties (assertions) or a reference model. Second, one must create an environment model, typically in the form of constraints on the inputs or a state machine description. Next, one runs the verifier, such as a model checker, which is usually thought of as a "push-button" technique. While this is largely true, human insight is not entirely absent; e.g., one might need to supply hints to the verifier in the form of suitable abstraction techniques or (templates for) inductive invariants. If the verifier returns with a counterexample trace, one must debug the design by localizing the cause of error in time (relevant part of the trace) and space (relevant part of the design). Finally, the process of repairing the design to eliminate the bug is also one that needs human input. To summarize, even after decades of work on automating the verification process, we continue to need human insight in a variety of tasks, including writing specifications, creating models, guiding the verification engine, debugging and error localization, and repair.

This paper takes the position that while we cannot completely remove human insight from the verification process, we can change the way humans provide insight to the verifier. Today, such input typically comes from expert verification engineers, trained in the tools of their field. But such experts are few and expensive. And even experts have a hard time answering questions such as: When are we done verifying? Have we written enough properties? Where is the bug? And so on. We contend that the experts and automated tools can be assisted in the verification process by a large crowd of non-expert humans performing simple, repetitive tasks. Each task involves a pattern recognition or other cognitive operations that humans are typically good at. The main technical challenges are to identify steps in the verification process where human insight is critical, find ways to transform these steps into tasks that non-expert humans can perform, and combine the results to resolve those steps in the verification process. As preliminary evidence to show that these challenges can be met, we present a system called CrowdMine for finding specifications from traces based on pattern recognition by humans.

The idea of tapping into a crowd of humans to assist in a computational task is not new. *Crowdsourcing* is the act of taking a job traditionally performed by a designated agent (usually an employee) and outsourcing it to an undefined, generally large group of people in the form of an open call [4]. *Human computation* is a paradigm for utilizing human processing power to solve problems that computers cannot yet solve [10]. (See Quinn and Bederson [7] for a more detailed description of these and related terms.) Our proposal is to use a combination of crowdsourcing and human computation to improve the state-of-the-art in verification. The availability of tools like Amazon's Mechanical Turk [1] and TurKit [6] make such a combination easier to deploy today.

In recent years, others have also advocated the use of crowdsourcing and human computation in design and verification, both for hardware and software. DeOrio and Bertacco [2] propose having humans assist in solving NP-complete problems arising in EDA, such as Boolean satisfiability (SAT) solving. Schiller and Ernst [8] propose the use of crowdsourcing and human computation for solving problems in software engineering, including software verification. The important difference between our proposal and these works is that we target steps in the verification process that *already* require human input, and which we think are unlikely to be automated entirely (similar to hard AI problems in the class of passing the Turing test, but unlike many NP-hard problems). We seek to leverage crowdsourcing and human computation to scale up the productivity in these steps manyfold.

To summarize, this paper makes the following contributions:

- Advocate the use of crowdsourcing and human computation for sub-tasks in verification that require human insight;
- Demonstrate the idea through CrowdMine, a novel game devised for finding patterns from system traces that can suggest likely specifications (Section 2), and
- Sketch out the landscape of similar applications (Section 3).

2. CROWDMINE

Many existing behavioral or specification mining techniques rely on the use of templates [3, 5]. Hence, it is the user's responsibiltiy to come up with a good set of templates. This process requires expert insight and is often incomplete. The main idea of CrowdMine is to tap into the human ability to recognize patterns in images to assist the process of mining specifications. For example, a trace can be visualized as a 2D image, where the rows are signals and the columns are cycles. CrowdMine first transforms segments of a trace into images and then queries a non-expert crowd to identify common patterns in those images. We have designed a game, described below, that incorporates these ideas.

2.1 Game Design

The purpose of our game is to discover other interesting patterns that do not match any of the pre-defined templates.

Game Design:

1. The player is presented with a set of two images along with

DAC 2012, June 3-7, 2012, San Francisco, California, USA.

$m (1 \leq m \leq 3)$ patterns, as shown in Figure 1. Each pattern is a collection of squares which are not necessarily adjacent.

2. The player is asked to identify a pattern that is common in the two images but does not match any of the m patterns given. Additional constraints for the pattern can also be specified.
3. If the identified pattern is indeed common in the two images, points will be awarded. The number of points awarded is equal to the number of squares in the identified pattern. This scoring function directs the player's attention to more complex patterns.
4. A time limit of T (e.g. 15) seconds is enforced for each play.

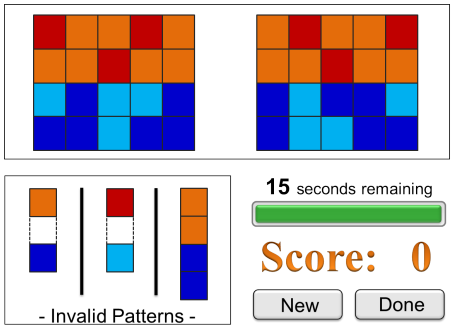

Figure 1: Example game interface: the task is to find a pattern that is common in the top two images but is not one of the invalid patterns.

Backend:

1. First, we sample a small set of n (e.g. 4) signals of interest and assign them an arbitrary order. Next, we randomly select two segments of k (e.g. 5) cycles in the trace and project them onto the n signals. This generates the two sub-traces for the game.
2. A template-based pattern mining algorithm is run to find common patterns that exist in these two sub-traces. Three of these mined patterns are selected in random as invalid patterns.
3. When each play finishes, the pattern selected by the player is checked if it is indeed a common pattern in both images.
4. A correct pattern is added to a database for further refinement.

GUI design:

- Different color ranges are used to encode different types of signals, e.g. input signals vs. output signals.
- Different shades of color are used to encode different signal values, e.g. dark for 0 and light for 1 for binary signals.
- The player can click on a square to select it. The selected square will be highlighted.
- When the player clicks "Done", the collection of selected squares in each image is considered as the identified pattern.

The example above is a 2-input and 2-output round-robin arbiter. We use this simple example to illustrate our workflow. The bottom two rows are the two request signals req_0 and req_1. The top two rows are the two response signals $resp_0$ and $resp_1$. The meaning of the patterns given are described below.

- (A): "When req_0 is low, $resp_0$ is low."
- (B): "When req_1 is high, $resp_1$ is high."
- (C): "All signals are low in one cycle."

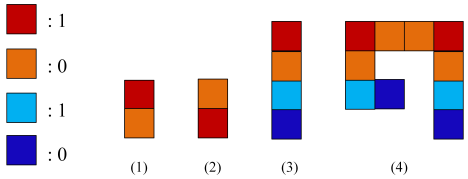

Figure 2: Color Code and Example Player-Identified Patterns

Some player-identified patterns are shown in Figure 2.

- (1) and (2): "Only one response is high at any cycle."
- (3): "When req_1 is high and there is no competing req_0, $resp_1$ is high at the same cycle."
- (4): This is the largest common pattern in the two images. Note that this pattern is an artifact of the two sub-traces chosen rather than a universal behavior that characterizes the arbiter. We can rank identified patterns by frequency of occurrence, thus filtering artifacts that appear rarely across many pairs of sub-traces.

2.2 Discussion

Privacy. Our design is particularly attractive for companies that value confidentiality because the internals of the circuit are not revealed. For IP protection, the mapping of sub-traces to images should be kept confidential. This mapping include the correspondence of signals in the circuit, the color code, and any additional transformation on the subtraces. Randomization can also be used in selecting sub-traces and the mapping to images. Finally, *secret sharing* methods such as the *threshold schemes* developed by Shamir [9] are particularly relevant in this context.

Incentives. Three mechanisms are possible. (1) *Necessity:* Authentication systems such as reCAPTCHA [10] embed queries into a human challenge with partially known answers. Our game design can be augmented for this purpose. For example, two plays are presented to the user in series in which the answer is known for one of the plays. (2) *Enjoyment:* Our game design can be viewed as a puzzle game and the player derives enjoyment by solving it. In addition, the scoring-based system invites human competition and can attract a larger crowd. (3) *Profit:* Platforms such as Amazon's Mechanical Turk [1] provide a *for-profit* medium for crowdsourcing any human intelligence task. We plan to deploy our game on the Mechanical Turk to evaluate the proposed approach.

Human-Computer Collaboration. It is possible to combine algorithmic techniques with inputs from humans to achieve something better than what can be accomplished by either solely humans or a completely automated approach. In our setting, the human-identified patterns can be further refined (e.g. ranked) to produce the most relevant ones based on feedback from the back-end verification and debugging processes. They can also be used in automated tasks such as bug localization [5].

3. LOOKING AHEAD

We believe several games similar to CrowdMine can be created and applied to a range of applications in verification, debugging, and related areas. For example, one can improve coverage of a design by properties (or tests) by highlighting parts of a trace corresponding to variables not covered by (enough) properties, and users can be provided incentives to find patterns involving those parts. Properties generated by a system like CrowdMine can be hypothesized as auxiliary inductive invariants to speed up verification. Human-observed patterns in spurious counterexamples could potentially enable better abstraction-refinement in model checking. Finally, the process of debugging has similarities to investigating a crime scene (!) — the "crime" is the manifestation of the error (the failure), and one seeks to find a cause-and-effect chain that explains how the failure happened; this analogy suggests a natural game that could be formulated for non-expert humans to assist in debugging.

Acknowledgements. This research was supported in part by NSF grant CNS-0644436, an Alfred P. Sloan Research Fellowship, and the Gigascale Systems Research Center, one of six research centers funded under the Focus Center Research Program (FCRP), a Semiconductor Research Corporation entity.

4. REFERENCES

[1] Amazon's mechanical turk. `www.mturk.com/mturk/welcome`.
[2] A. DeOrio and V. Bertacco. Human computing for EDA. In *Design Automation Conference (DAC)*, pages 621–622, 2009.
[3] M. D. Ernst, J. H. Perkins, P. J. Guo, S. McCamant, C. Pacheco, M. S. Tschantz, and C. Xiao. The daikon system for dynamic detection of likely invariants. *Sci. Comput. Program.*, 69(1-3):35–45, 2007.
[4] J. Howe. Crowdsourcing: A definition. http://crowdsourcing.typepad.com.
[5] W. Li, A. Forin, and S. A. Seshia. Scalable specification mining for verification and diagnosis. In *Proceedings of the Design Automation Conference (DAC)*, pages 755–760, June 2010.
[6] G. Little, L. B. Chilton, M. Goldman, and R. C. Miller. TurKit: Human computation algorithms on mechanical turk. In *Proc. 23nd ACM symposium on User interface software and technology (UIST)*, pages 57–66, 2010.
[7] A. J. Quinn and B. B. Bederson. Human computation: A survey and taxonomy of a growing field. In *ACM Conference on Human Factors in Computer Systems (CHI)*, 2011.
[8] T. W. Schiller and M. D. Ernst. Rethinking the economics of software engineering. In *In Workshop on the Future of Software Engineering Research*, pages 325–330, 2010.
[9] A. Shamir. How to share a secret. *Commun. ACM*, 22:612–613, November 1979.
[10] L. von Ahn. *Human Computation*. PhD thesis, Carnegie Mellon University, December 2005.