Web Verification: Perspective and Challenges

Shriram Krishnamurthi [1]

Computer Science Department
Brown University
Providence, RI, USA

Abstract

The Web poses novel and interesting problems for both programming language design and verification—and their intersection. This paper provides a personal outline of one thread of work on this topic.

Key words: Web applications, temporal verification, access control, program analysis

1 What is a Web Site?

The term "Web site" contains a hidden ambiguity. Is a site a static entity, to be viewed as a program source, or is it a dynamic entity, to be viewed through the lens of user behavior? This distinction significantly impacts what it means to analyze and verify a Web site. All the traditional trade-offs between static and dynamic analyses apply: a static analysis can quantify over all program behaviors, but will usually be less specific; a dynamic analysis can only offer guarantees relative to the behaviors it has examined, but the additional contextual information can often yield more informative answers. This distinction potentially matters more on the Web, due to the nature of Web interactions.

2 Web Interactions

In a console or even a GUI application, a user cannot choose to go back or forward, to clone a window and submit responses from both clones, and so on. These user interaction capabilities distinguish Web applications from many other kinds of interactive programs. Indeed, many Web sites are notorious for

[1] This work is partially funded by NSF grants CCR-0305949 and CCF-0447509.

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

their poor handling of such interactions. For instance, on some travel Web sites, viewing the list of hotels, viewing one choice in a new window, examining a second in another new window, then switching back to make a reservation for the first hotel, will result in a reservation at the second hotel [5].

A Web application must not only be sensitive to the possibility of these actions, it must often detect them without help from the browser (which does not report every user action). Furthermore, the availability of rendering platforms is likely to spark innovation, meaning the set of browsers—and, consequently, of interaction behaviors—will grow over time. This makes Web site analysis especially exciting and challenging.

3 Web Verification

My work has focused on static analyses of Web programs. Specifically, I have studied Web applications from two complementary perspectives. All this work has been driven by a desire to build a robust application that has value in its own right, in addition to serving as a generator of research problems.

3.1 A Driving Application

The concrete application is Continue [7,9], a Web-based application for conference paper management. Continue is similar in spirit to several programs in this genre (though it has had considerably more investment into its user interface quality), so familiarity with one of those applications is sufficient for understanding it at a high level. It has several useful features not found in many other conference applications, covering soliciting sub-reviews, helping chairs with assignments, and changing user identity. My goal is to create an application that is not only usable, but has also been verified along as many dimensions as necessary for sufficient reliability. After all, when a community has the creation of papers—as an expression of research—as a primary goal, the safe handling of those papers should be considered mission-critical!

3.2 Temporal Behavior

Web applications must satisfy many temporal specifications. For instance, on a travel reservation site, a user expects that the hotel they reserve is the same hotel that they selected—even if they chose to investigate other hotels along the way. In a virtual bookstore, a user might have a "shopping cart" into which they place their selections. They—or, at least, the store!—would want that every book placed in the cart is purchased upon final check-out. (In fact, the actual property would be more subtle: the books purchased must be all those that were placed in the cart *and not subsequently removed*, creating an additional level of temporal quantification.) There are several similar expectations of "reasonable" behavior in Continue.

The statement of temporal properties naturally suggests the use of a model checker [1]. This proves to be somewhat complex in practice. A naive use of model checking will not, for instance, capture some of the interaction-induced errors mentioned in section 2. Why not? Because the natural model that one would construct from the Web application fails to capture the many behaviors that users can perform through the browser; colloquially speaking, nowhere in the source code of a Web application does it say, "Here the user clicks the Back button".

The problem of building accurate models is further hampered by the problem of accounting for the many kinds of Web interactions. Not only do browsers offer a plethora of choices, even the popular browsers have different feature sets—and this doesn't account for the possibility of additional features in the future.

To support the many interaction features of browsers, we employ prior work that presents a core model of Web interactions [5]. This model presents a small set of Web primitives that are sufficient for modeling all the known Web interaction forms, and should cover many new ones as well. Given this abstraction, we have been studying the problem of building a model checker that can handle the subtleties of the Web [10]. Note that this is not a model checker *only* for Web-specific interactions, but rather one that will *also* account for Web interactions: that is, if the program violates a property independently of any Web interactions, the checker will find those also.

The property language in this work is subtle and, therefore, interesting. Specifically, properties need to be able to refer to elements of Web pages. To index these elements, we refrain both from parsing HTML (a thankless activity!) and from using static distance coordinates (which would be too brittle). Instead, we expect the developer to tag individual page elements using Cascading Style Sheet (CSS) tags. These are not only part of most developers' vocabulary, often the developer has already tagged interesting page elements to highlight visually. While such ideas are not scientifically deep, I believe they are essential for successfully deploying formal methods.

3.3 Information Safety and Visibility

Verifying a program for temporal behavior isn't enough. In a conference server, it is at least as important to ensure both the safety and availability of information: e.g., program committee members can see reviews they should see, and can't see ones that they shouldn't. These properties generally fall under the rubric of *access control*. Once we discovered actual information access bugs in Continue (which have since been fixed!), we embarked on a study of access control policy specification and verification.

Access control has gained fresh popularity owing to the widespread availability of information on the Web. In particular, because many Web applications are interfaces over databases and provide the same data in different

circumstances to different users, access control has increasingly become *role-based*. Industrial standards such as XACML [3], which are effectively rule-based languages, are being employed to describe—and their evaluation engines are being used to enforce—such policies.

Our work in this area [2] has focused on two problems for a restricted (but nevertheless useful) subset of XACML. First, naturally, is the question of whether a policy meets some set of properties; this is the traditional verification question. The second is more intriguing. Given the simplicity of these policy languages, it is easy to patch problems and quickly check that the new policy does what it was intended—on a specific input. The patch may, however, have both exposed private data, or made necessary information unavailable. This danger is exacerbated by the declarative nature of these policy languages, for changes can have very non-local impact. As a result, a simple syntactic difference is no longer sufficient; users require some form of *semantic* differencing. This is the second problem our work addresses.

It is worth noting that the problems surrounding information access—especially the danger of leakage—make a compelling case for static analyses: no reviewer wants to hear that their confidential comments were leaked due to a lack of good test cases. Wading through false positives is certainly onerous; to be effective, this cost must be kept minimal. Nevertheless, this is an instance where the universally quantified guarantees that a static analysis can provide are worth reasonable costs.

4 The Structure of Web Programs

Focusing on Web *programs* (as static entities) raises an interesting subtlety. To obey the stateless nature of the Web, the structure of Web applications has traditionally been "inverted", resembling programs written in continuation-passing style [4,8,12]. Furthermore, important information is communicated using hidden fields and other channels that are traditionally ignored by static analyses. A traditional static analysis would, therefore, approximate a great deal of the important information (particularly the values referred to in properties). The resulting models would simply not be useful for further analysis.

Not surprisingly, the same problems that affect analyses also plague developers. There has thus recently been a trend towards using continuation-based primitives in the source program, which can be handled either by a specialized server [6,12] or on a traditional server after compilation [11]. This means, for instance, that lexical bindings remain as such in the source, rather than being transformed into hidden fields or other external storage. By avoiding this program inversion, this form of source program is therefore a better input to an existing program analysis.

5 Some Research Problems

There are numerous open research problems in this area. What follows is only a small and eclectic sampling.

Some of the most interesting ones have to do with access control. For instance, most of the access control verification work deals solely with the policy. But to be truly effective, it must also take into account the program's behavior relative to the policy. (In an extreme case, if an application were to rigorously consult a policy engine but always ignore its response, no amount of policy verification would be useful. While this particular behavior may appear extreme, it is not inconceivable during testing, and a lack of good test suites will fail to uncover all the places where this prototype failed to grow from a script into a program.)

Access control policies also need to address the temporal behavior of these applications. While some research has studied temporal policies, it is unclear how well these results apply to the less structured world of the Web, where a program potentially has several entry points and users can engage in complicated interactions that the program cannot prevent.

One other important aspect of Web applications is that they are increasingly no longer "on the Web". A growing number of Web sites now make extensive use of client-side scripting languages, especially JavaScript, to implement numerous user operations. In particular, whereas the use of JavaScript tended to be limited to echoing browser operations or performing consistency checks before transmitting data over the wire, now a non-trivial part of the application source is downloaded with the page. This creates a challenge and opportunity for cross-language analyses.

Acknowledgments

I thank the several co-authors with whom I've had the pleasure of conducting this research. I especially thank Pete Hopkins, who helped transform Continue from a very good prototype into a true product. Even his bugs are more interesting than most people's programs.

References

[1] Clarke, E., O. Grumberg and D. Peled, "Model Checking," MIT Press, 2000.

[2] Fisler, K., S. Krishnamurthi, L. A. Meyerovich and M. C. Tschantz, *Verification and change-impact analysis of access-control policies*, in: *International Conference on Software Engineering*, 2005.

[3] Godik, S. and T. M. (editors), *eXtensible Access Control Markup Language, version 1.1* (2003).

[4] Graham, P., *Beating the averages* (2001), `http://www.paulgraham.com/avg.html`.

[5] Graunke, P. T., R. B. Findler, S. Krishnamurthi and M. Felleisen, *Modeling Web interactions*, in: *European Symposium on Programming*, 2003, pp. 238–252.

[6] Graunke, P. T., S. Krishnamurthi, S. van der Hoeven and M. Felleisen, *Programming the Web with high-level programming languages*, in: *European Symposium on Programming*, 2001, pp. 122–136.

[7] Hopkins, P. W., *Enabling complex UI in Web applications with send/suspend/dispatch*, in: *Scheme Workshop*, 2003.

[8] Hughes, J., *Generalising monads to arrows*, Science of Computer Programming **37** (2000), pp. 67–111.

[9] Krishnamurthi, S., *The* Continue *server*, in: *Symposium on the Practical Aspects of Declarative Languages*, 2003, pp. 2–16.

[10] Licata, D. R. and S. Krishnamurthi, *Verifying interactive Web programs*, in: *IEEE International Symposium on Automated Software Engineering*, 2004, pp. 164–173.

[11] Matthews, J., R. B. Findler, P. T. Graunke, S. Krishnamurthi and M. Felleisen, *Automatically restructuring programs for the Web*, Automated Software Engineering: An International Journal (2003).

[12] Queinnec, C., *The influence of browsers on evaluators or, continuations to program web servers*, in: *ACM SIGPLAN International Conference on Functional Programming*, 2000, pp. 23–33.