Research.js: Evaluating Research Tool Usability on the Web

Joel Galenson, Cindy Rubio-González, Sarah Chasins, Liang Gong
University of California, Berkeley
{joel,rubio,schasins,gongliang13}@cs.berkeley.edu

Abstract

Many research projects are publicly available but rarely used due to the difficulty of building and installing them. We propose that researchers compile their projects to JavaScript and put them online to make them more accessible to new users and thus facilitate large-scale online usability studies.

1. Motivation

When researchers create new programming languages and tools, they sometimes choose to release them publicly. Unfortunately, released projects are often complicated to install, which makes it difficult to find enough users to conduct large-scale usability studies. Yet releasing tools that are accessible to users is typically difficult for researchers. We propose building an infrastructure that makes public releases easy for both potential users and researchers, thereby facilitating large-scale user studies.

Unfortunately, even when languages and tools are publicly available, they are often difficult to build and install. A recent conference encouraged authors to submit artifacts and stipulated that the setup should take less than 30 minutes [6]. For some potential users, including reviewers and researchers who intend to use or evaluate the artifacts, this may be a reasonable amount of time. However, this level of time commitment is likely to prevent a potential user with a passing interest from trying a programming language or tool. In order to keep these users interested and collect data about their usage, we must lower the barriers to entry. Making it easy for these moderately interested parties to participate is key to achieving truly large-scale user studies.

In recent years, the web has become an increasingly powerful platform, one on which applications are easily portable. Some research projects have made online versions of their tools available, making them easy to explore (e.g., [3, 7]).

PLATEAU '14, October 21, 2014, Portland, OR, USA..

ACM 978-1-4503-2277-5/14/10. . . $15.00.
http://dx.doi.org/10.1145/2688204.2688217

To make research tools available online, current practice requires researchers to set up and run their own servers, which run their tools on users' inputs. Setting up such a server can be a difficult, time-consuming, and costly process. Because of load restrictions, this approach imposes a limit on the number of concurrent users. Further, most research tools are not designed to be secure, which makes it dangerous to run them on a server that accepts user inputs. Some researchers also distribute their tools as virtual machine images, but these are very heavyweight.

We propose helping researchers compile their tools to JavaScript, thereby allowing anyone with a web browser to use them. This would enable researchers to make their tools available online without running their own secure servers. The relatively-lightweight client-side JavaScript would complete all processing, eliminating researchers' security concerns. Further, the tools can be distributed within static webpages, which can easily be hosted on any number of external services, so researchers need not run servers themselves. With this approach, users could browse language and tool demos as easily as they browse web pages, trying tools before deciding to install them locally. If users consent, their usage data could be sent to the researchers, allowing each visitor to serve as a participant in a user study. The collected data could then be leveraged to evaluate and ultimately improve usability. In essence, the online release becomes a large-scale remote user study.

The research community has already started advocating releases. Conferences increasingly encourage authors to submit artifacts [1] to allow others to evaluate and build on their contributions. While this too is a laudable goal, we focus not on reproducing evaluations but on the potential of public releases to bring large-scale remote user studies within reach.

2. Proposed Approach

We propose building an infrastructure that makes it easy to compile existing projects to JavaScript and optionally collect usage data. We plan to leverage existing tools to translate programs into JavaScript. One such tool is Emscripten [15], which compiles C/C++ code and LLVM [14] bitcode to JavaScript. There are other projects that compile many languages, including Scala, Haskell, and Python, directly to JavaScript [2, 5].

	Changes required			
Name	Source	Build	Compiles	Runs
MiniSat [13]	0 (0)	1 (1)	✔	✔
Lingeling [10]	1 (1)	0 (0)	✔	✔
Boolector [11]	0 (0)	0 (0)	✔	✔
Hugs [4]	1 (1)	1 (1)	✔	✔
Z3 [12]	1 (1)	1 (1)	✔	✘
LLVM+Clang [14]	12 (5)	3 (3)	✔	✘

Table 1. The projects we have attempted to compile with Emscripten, the number of changes required to compile them (the number of lines of code and files changed), and whether they successfully compile and run.

An alternative plan for running research tools in a web browser is to use interpreters written in JavaScript [9] without compiling the project under evaluation. As a last resort, projects could be run in a JavaScript PC emulator [8].

In our infrastructure, we also plan to collect and analyze traces automatically (with users' consent) to help evaluate projects' usability and guide future improvements.

There are several issues to consider when translating research projects into JavaScript and running in a browser.

Closed-Source Binaries. Many projects use closed-source binaries. It may be possible to compile assembly code to JavaScript or decompile binaries into a higher-level language that could be compiled directly. Alternatively, we could provide an option to obfuscate the generated JavaScript code.

Libraries. A research project might depend on many external libraries, each of which would have to be compiled with the above techniques. We believe that pre-compiling and establishing a central repository that hosts all known ported libraries would help alleviate this problem.

Performance. Running a research project in a browser is slower than running it natively. Fortunately, performance is not generally critical for evaluating usability. Thus we prioritize compatibility and ease-of-use over performance.

3. Exploration and Early Experience

In our prototype, we use Emscripten [15], which is designed to allow execution at close to native speeds. Many projects have been ported to JavaScript with Emscripten, including Unreal Engine 3, LaTeX, Lua, Python, and parts of LLVM and Emscripten [2, 5]. It is robust and, with its emphasis on porting games to the web, performs well. We have used it to compile the tools, such as compilers and SAT/SMT solvers, listed in Table 1. Only one project required changing more than a single line of code.[1] Our prototype does not yet automatically generate code to collect tool usage statistics.

To demonstrate the feasibility of the proposed approach, we have made the changes required to compile each project with Emscripten publicly available at `https://github.com/jgalenson/research.js`, which links to demos for MiniSat, Boolector, and Hugs.

We analyzed the performance of our JavaScript versions of MiniSat, Lingeling, and Boolector on a few benchmarks hand-chosen from SATLIB and SMT-LIB 2. On average, the Emscripten-compiled versions are 3 times slower than native, which we believe is sufficient for our purposes. To show that performance is likely to improve further over time, we compiled and tested MiniSat with eighteen-month-old versions of Emscripten and Firefox and found that it was 11 times slower than native. Using our prototype, compile times are 2-7 times slower and file sizes of Emscripten-compiled projects are less than 2 times larger.

[1] Most of the source changes involved modifying `#include` statements.

Acknowledgments

We would like to thank Christos Stergiou, Nishant Totla, and Cuong Nguyen for their useful advice. We also appreciate the valuable comments from anonymous reviewers and our shepherd Joshua Sunshine for improving the paper.

References

[1] Artifact Evaluation for Software Conferences. `http://www.artifact-eval.org/`. Accessed: 09/10/2014.

[2] Emscripten. `https://github.com/kripken/emscripten/wiki`. Accessed: 11/22/2013.

[3] Flapjax. `http://www.flapjax-lang.org/`. Accessed: 07/03/2013.

[4] Hugs. `http://www.haskell.org/hugs/`. Accessed: 11/22/2013.

[5] List of languages that compile to JS. `https://github.com/jashkenas/coffee-script/wiki/List-of-languages-that-compile-to-JS`. Accessed: 11/22/2013.

[6] OOPSLA Artifacts. `http://splashcon.org/2013/cfp/665`. Accessed: 07/03/2013.

[7] rise4fun. `http://rise4fun.com/`. Accessed: 07/03/2013.

[8] JavaScript PC Emulator. `http://bellard.org/jslinux/`. Accessed: 07/03/2013.

[9] repl.it. `http://repl.it/`. Accessed: 07/03/2013.

[10] A. Biere. Lingeling, plingeling, picosat and precosat at sat race 2010. *FMV Report Series Technical Report*, 10(1), 2010.

[11] R. Brummayer and A. Biere. Boolector: An efficient smt solver for bit-vectors and arrays. In *TACAS*, pages 174–177. Springer, 2009.

[12] L. De Moura and N. Bjørner. Z3: An efficient smt solver. In *TACAS*, pages 337–340. Springer, 2008.

[13] N. Eén and N. Sörensson. An extensible sat-solver. In *Theory and Applications of Satisfiability Testing*, pages 502–518. Springer, 2004.

[14] C. Lattner and V. Adve. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *CGO'04*, Palo Alto, California, 2004.

[15] A. Zakai. Emscripten: an llvm-to-javascript compiler. In *SPLASH '11*, pages 301–312, 2011.