Abstract

Many research projects are publicly available but rarely used due to the difficulty of building and installing them. As web browsers have grown in popularity, they have become an excellent platform for offering demos. We propose that researchers compile their projects to JavaScript and put them online to make them more accessible to first-time users and thus facilitate large-scale online usability studies.

1. Motivation

When computer scientists create new programming languages and tools in the course of their research, they sometimes choose to release them publicly. The research community has advocated this practice because it allows other researchers to evaluate the languages and tools and build on their contributions. For example, some conferences are now encouraging authors to submit artifacts to be evaluated alongside their papers [1, 3, 7]. While this is a laudable goal in its own right, public releases may make an even greater impact by allowing researchers to conduct large-scale evaluations of the usability of their languages and tools.

Unfortunately, even when languages and tools are publicly available, they are often difficult to build and run. Locating all dependencies can demand a significant investment of time and effort. A recent conference encouraged authors to submit artifacts and stipulated that the setup should take less than 30 minutes [7]. While 30 minutes may be a reasonable amount of time for a reviewer, that sort of time commitment is enough to prevent a programmer with a passing interest from trying a programming language or tool. In order to keep these programmers interested and achieve truly large-scale user studies, we must lower the barriers to entry.

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, which has made it straightforward for others to explore them (e.g., [4, 8]).

To make research tools available online, the current practice requires researchers to set up and run their own servers. This can be a difficult, time-consuming, and costly process; because of load restrictions, it also imposes a limit on the number of concurrent users. In addition, most research tools are not designed to be secure, which makes it dangerous to run them on a server that accepts arbitrary inputs.

We propose to help researchers compile their tools to JavaScript, thereby allowing anyone with a web browser to use them. This would enable researchers to make their tools usable online without running their own secure servers. It would also allow users to browse through language and tool demos as easily as they browse through web pages. 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 be leveraged to improve usability.

2. Proposed Approach

We propose building an infrastructure that makes it easy to compile an existing project to JavaScript and optionally collect usage data.

We plan to leverage existing tools to translate programs into JavaScript. One such tool is Emscripten [17], which compiles LLVM [16] bitcode to JavaScript and contains a wrapper that allows it to compile C/C++ code directly to JavaScript. There are other projects for many languages, including Scala, Haskell, and Python, that compile them directly to JavaScript [2, 6].

An alternative plan for running research tools in a web browser is to use interpreters written in JavaScript [11] without compiling the project under evaluation. As a last resort, projects could be run in a JavaScript PC emulator [9].
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 rely on closed-source binaries. It might be possible to compile assembly code to JavaScript or decompile the binaries into a higher-level language that could then be compiled normally. 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 [17], which targets asm.js [10], a low-level subset of JavaScript that 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, 6]. It is robust, and given its emphasis on porting games to the web, performs well. We have used it to compile tools such as compilers and SAT/SMT solvers, which are listed in Table 1. All but one of these projects compiled with Emscripten after changing at most two lines. Our prototype does not yet include the option 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](https://github.com/jgalenson/research.js). This link also contains demos for MiniSat, Boolector, and Hugs.

We analyzed the performance of our JavaScript versions of MiniSat, Lingeling, and Boolector on a few benchmarks. On average, the Emscripten-compiled versions are 2.4–5.5x 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 six-month-old versions of Emscripten and Firefox and found that it was 11.4x slower than native. Using our prototype, compile times are 2-7x slower and file sizes of Emscripten-compiled projects are less than 3x larger.

### Acknowledgments

We would like to thank Christos Stergiou, Nishant Totla, and Cuong Nguyen for their useful advice.

### References


