Research.js: Sharing Your Research on the Web

Full Presentation

Joel Galenson, Cuong Nguyen, Cindy Rubio-González, Christos Stergiou, Nishant Totla

University of California, Berkeley

{joel,nacuong,rubio,chster,nishant}@cs.berkeley.edu

Abstract

Most research tools are publicly available but rarely used due to the difficulty of building them, which hinders the sharing of ideas. However, web browsers have recently become an excellent platform for giving portable demos. Researchers could enable their tools to run on a web browser by compiling them into JavaScript. Making tools accessible on the web would facilitate trying new tools without the overhead of configuring and building them.

1. Motivation

When Computer Scientists create tools in the course of their research, they often publicly release them. This is a useful practice, as it makes their findings reproducible, allows other researchers to evaluate and build on their ideas, and potentially enables real users to use a practical tool. Conferences have begun to encourage authors to submit artifacts to be evaluated along with their papers [1, 3, 8].

Unfortunately, even when such tools are publicly available, they are often difficult to build. Finding all the dependencies needed to build a program can require significant effort, and when a binary is available, it can be difficult to figure out how to use it. A recent conference that encouraged authors to submit artifacts stipulated that the setup should take less than 30 minutes [8], which is still a significant amount of time for reviewers to spend on a single paper.

Recently, the web has become a powerful application platform that makes it easy to run programs in a portable fashion. Some research projects have made online versions of their tools available, which has made it easy for others to try them out (e.g., [4, 9]).

Unfortunately, the current practice for making research tools available online requires that researchers set up and run their own servers. This can be a difficult and costly process that also imposes a limit on the number of concurrent users to reduce load.[1] In addition, security is a problem with researchers running code on their own servers. Most research tools are not designed to be secure, so running them with arbitrary inputs from the web could be dangerous.

We propose facilitating researchers to compile their tools to JavaScript, which would allow anyone with a web browser to use them. This would allow researchers to make their tools usable online without running their own server or limiting the number of users. Since our approach compiles code to JavaScript that runs on the client, security is not a problem.

[1] We do, however, agree that it is a significant step forward over current practice; here we are proposing an alternative approach that could have additional advantages.

2. Approach

We propose building an infrastructure that makes it easy both to take an existing project and compile it to JavaScript and to put the JavaScript onto a webpage for users to visit.

2.1 Translation to JavaScript

We plan to leverage existing tools that translate programs into JavaScript. For example, Emscripten [18] compiles LLVM [17] bitcode to JavaScript. It contains wrappers around Clang (LLVM's C/C++ frontend) that allow it to compile C/C++ code directly to JavaScript. Furthermore, interpreters for many languages, including Python and Ruby, have been compiled with Emscripten and could be used to run code written in those languages. Finally, separate projects for many other languages, including Scala, Haskell, and OCaml, aim to generate JavaScript code directly [2, 7]. As a last resort, projects could be run in a JavaScript PC emulator [11].

For this paper, we focus on Emscripten. It targets asm.js [12], 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 the Unreal Engine 3 [10], LaTeX [6], Lua, Python, JavaScript, and parts of LLVM and Emscripten themselves [2, 7]. It is thus reasonably robust, and given its emphasis on porting games to the web, performs reasonably well.

2.2 Other Issues to Consider

There are many issues to consider when translating research projects into JavaScript. We discuss some of them below.

Binaries Many projects rely on closed-source binaries. In theory, it might be possible to compile assembly code to

Name	Changes required		Compile	Run
	Source	Make		
MiniSat [16]	0(0)	1(1)	✔	✔
Lingeling [13]	1(1)	0(0)	✔	✔
Boolector [14]	0(0)	0(0)	✔	✔
Hugs [5]	0(0)	2(2)	✔	✔
Z3 [15]	1(1)	1(1)	✔	✘
LLVM+Clang [17]	12(5)	3(3)	✔	✘

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

LLVM or decompile the binaries into a higher-level language that could then be compiled normally. A web-based emulator or sandbox for native code could allow the use of binaries at the cost of significant extra complexity. We also note that it is possible that the creator of the binary could produce an obfuscated JavaScript file that would serve as a binary for the web.

Libraries Projects often use many libraries, each of which would have to be compiled with the above technique. This is not a theoretical concern, as we would like to be able to compile all code, but it is a practical one, as compiling one research project might involve compiling many libraries used as dependencies. We believe that a central repository that hosted all known ported libraries would help alleviate this problem.

Performance A number of the techniques mentioned above, such as emulating shared-memory multithreading and running code in dynamic languages by running their intepreters in JavaScript, will likely be much slower than running the code natively. Luckily, performance is not critical for our domain, as most research tools are not optimized and online demos do not usually need to be as fast as possible. Thus unlike the games that asm.js targets, we will likely prioritize compatibility and ease-of-use over performance.

3. Experience

Since our goal is to make it easy for researchers to port their tools to JavaScript, we have started trying to use Emscripten to compile tools such as compilers and SAT/SMT solvers. We list all the projects we have compiled and their statuses in Table 1. All but one of these projects compiled with Emscripten after changing at most two lines of code, which shows that this process is often easier than other alternatives.

To encourage others to compile their own work to JavaScript, we have made the changes required to get each project to compile with Emscripten publicly available at `https://github.com/jgalenson/research.js`. This link also contains demos for MiniSat [16], Boolector [14], and Hugs [13]. We now briefly discuss the status of each project.

We have analyzed the performance of our compiled 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. Furthermore, to show that performance is likely to improve further with more 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.

We have also analyzed the build times and filesizes of Emscripten-compiled projects. Compile times are 2-7x slower and filesizes are 0.2-3x larger.

References

[1] ECOOP Artifacts. `http://ecoop13-aec.cs.brown.edu/`. Accessed: 07/03/2013.

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

[3] FSE Artifacts Report. `http://cs.brown.edu/~sk/Memos/Conference-Artifact-Evaluation/`. Accessed: 07/03/2013.

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

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

[6] LaTeX. `http://manuels.github.com/texlive.js/website/`. Accessed: 07/03/2013.

[7] 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.

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

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

[10] Unreal Engine 3. `http://www.unrealengine.com/html5/`. Accessed: 07/03/2013.

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

[12] asm.js. `http://asmjs.org/`. Accessed: 07/03/2013.

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

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

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

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

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

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