MatJuice

Vincent Foley-Bourgon

COMP-621 - Winter 2014
McGill University

April 2014

Outline

- Overview of MatJuice
- Demo
- Technical details
- Future work
- Questions

Overview

What is MatJuice?

- A source-to-source compiler from MATLAB to JavaScript[1];
- Uses the McLab framework;
- Outputs readable[2] JavaScript code;
- Aims for a correct translation;

[1] My two favorite languages in the world! /s
[2] Nicely indented and formatted.

What is MatJuice?

- A source-to-source compiler from MATLAB to JavaScript[1];
- Uses the McLab framework;
- Outputs readable[2] JavaScript code;
- Aims for a correct translation;
  - Whenever we need to choose between simplicity and performance, we pick the former;
  - We'll introduce optimizations once we have a solid and fairly complete implementation.

[1] My two favorite languages in the world! /s

[2] Nicely indented and formatted.

Why MATLAB?

- Widely used by scientists and engineers[3];
- Presents "interesting" challenges from a compiler point of view[4];
- Benefit from all the work that went into McLab.

[3] Estimated 2M users

[4] Remember Nishanth's presentation?

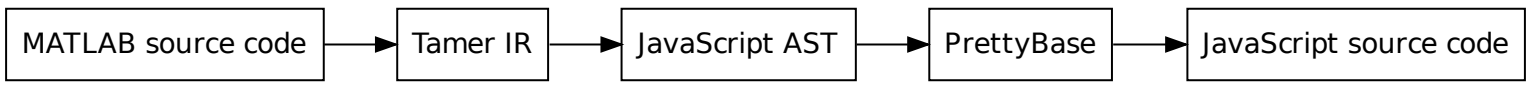
Why JavaScript?

- Most widely available language;
- JavaScript engines keep improving and performance is now very good;
- Allow MATLAB users to put their work on the web and integrate with web services.

Demo

Technical details

Compiler flow



Compiler flow

Matlab source code

```
function s = signint(n)
  if n < 0
    s = -1
  elseif n > 0
    s = 1
  else
    s = 0
  end
end
```

Compiler flow

Tamer code

```
function  [s] = signint(n)
  mc_t4 = 0;
  [mc_t3] = lt(n, mc_t4);
  if mc_t3
    mc_t0 = 1;
    [s] = uminus(mc_t0);
  else
    mc_t2 = 0;
    [mc_t1] = gt(n, mc_t2);
    if mc_t1
      s = 1;
    else
      s = 0;
    end
  end
end
```

Compiler flow

JavaScript source code

```
function signint(n) {
    var mc_t0, mc_t1, mc_t2, s, mc_t3, mc_t4;
    mc_t4 = 0;
    mc_t3 = lt(n, mc_t4);
    if (mc_t3) {
        mc_t0 = 1;
        s = uminus(mc_t0);
    }
    else {
        mc_t2 = 0;
        mc_t1 = gt(n, mc_t2);
        if (mc_t1) {
            s = 1;
        }
        else {
            s = 0;
        }
    }
    return s;
}
```

JavaScript AST

Why use another IR instead of outputting JavaScript directly?

- More modular: concerns such as proper indentation are moved down the pipeline;
- Easier to manipulate a tree than raw text;
  - For example, finding all the lhs variables and adding *var* declarations.
- In the future, we can apply JavaScript-specific optimizations.

JavaScript AST

- Written with JastAdd;
- Described in a high-level grammar language;
- Automatically translated to Java code;
- Aspect system is used to convert AST to *PrettyBase* tree.

JavaScript AST

- Nodes for the different JavaScript expressions and statements necessary to translate Tamer;
- Doesn't respect the JavaScript grammar, made simpler.

JavaScript AST

Conversion from Tamer to JavaScript AST

- Tamer has a visitor pattern interface;

JavaScript AST

Conversion from Tamer to JavaScript AST

- Tamer has a visitor pattern interface;
- I don't use it.

JavaScript AST

Conversion from Tamer to JavaScript AST

- Tamer has a visitor pattern interface;
- I don't use it.

The *tirAnalyze* method returns *void*; that makes it harder than necessary to accumulate the result of translating children nodes.

I use mutually recursive methods, which makes the code much easier to write and read.

Pretty printing

Another IR?! Yes!!

- Language agnostic: could be targeted by other backends;
- Small number of nodes (4); conversion to string is very short (< 30 lines);
- Expose high-level combinators (e.g. *parenthesized*, *separatedBy*, etc.).

Pretty printing

- Design shamelessly copied from *Peyton-Jones and Lester*[5];
- $<$ 150 lines of code;
- Created before the project was started; design and API didn't need to be changed.

[5] `http://research.microsoft.com/en-us/um/people/simonpj/papers/pj-lester-book/`

Benchmarks

Benchmarks

- MATLAB 2013a, Firefox 28, Chrome 33;
- Times are in seconds;
- Average over 10 runs.

[6] Array of size 4096

Benchmarks

- MATLAB 2013a, Firefox 28, Chrome 33;
- Times are in seconds;
- Average over 10 runs.

	MATLAB	JS (FF)	JS (Cr)	MatJuice (FF)	MatJuice (Cr)
collatz(100000)	0.5601	0.8388	0.0323	9.0951	0.0523

[6] Array of size 4096

Benchmarks

- MATLAB 2013a, Firefox 28, Chrome 33;
- Times are in seconds;
- Average over 10 runs.

	MATLAB	JS (FF)	JS (Cr)	MatJuice (FF)	MatJuice (Cr)
collatz(100000)	0.5601	0.8388	0.0323	9.0951	0.0523
bubbleSort[6]	0.4609	2.1369	0.6017	6.7054	0.6211

[6] Array of size 4096

Future

Future

- Fix incorrect semantics (e.g. pass-by-value);
- Some optimizations, e.g.: copy-on-write arrays;
- Statically transforming *plus(x,y)* into $x+y$ when x and y are scalars;
- More built-ins + framework to support different kinds of arguments;
  - Design a declarative DSL to avoid writing everything by hand;
  - Use static analysis information to specialize calls to the proper function;
- Fix underlying Tamer framework to properly support recursive functions.

Future

http://github.com/sable/mclab/tree/javascript-backend

Questions?

Bonus!

Figuring out the proper translation to balance browser performance is going to be tricky.

Copying an array

```
# Method 1
var i = a.length;
while (i--) { b[i] = a[i]; }

# Method 2
var b = a.concat();
```

Copying an array

ops/second (higher is better)

	Firefox	Chrome
Method 1	1.6M	1.1M
Method 2	0.6M	1.6 M