CScript: Implementing C in JavaScript

Daniel Feltey

December 12, 2012

CScript

CScript is . . .

- An educational tool for learning programming in C
- A *mostly* specifcation compliant implementation of C in JavaScript
- An interesting project in compiler/interpreter design

Why?

Learning to program online:

- tryruby.org
- codecademy.com
- Nothing for learning C interactively

I need to teach high school students a C based language.

Available options:

- Deal with platform differences
- Learn an IDE first
- Build an interactive tool

Implementing C: The Toolbox

- Parser generators
  - CFG vs. PEG
  - Jison vs PEG.js
- Continuations
  - Continuation Passing Style
  - Compiling with continuations
  - CESK machine
- Compiler vs. Interpreter

Parser Generators

Context Free Grammars

- Bison/Yacc/Jison
- Happy

Parsing Expression Grammars

- PEG.js
- Pappy

Parser Generator Syntax

Context Free Grammars

```
assign_expr
    : conditional_expr
    | unary_expr assign_op assign_expr
```

Parsing Expression Grammars

```
assign_expr
    = conditional_expr
    / unary_expr assign_op assign_expr
```

CFG vs. PEG

Context Free Grammars

- Bottom up parsing
- LR, LALR algorithms
- Ambiguity
- Usually require pre-lexed input

Parsing Expression Grammars

- Top down parsing
- Linear time
- *Unambiguous*
- No need for a lexer
- Problems with left recursion

PEG: The Good

- Ordered choice eliminates ambiguity
- Left recursion can be eliminated
- Linear time parsing
- No need for pre-lexed input
- The C grammar is easy to adapt to a PEG

PEG: The Bad

- Rule order matters
  - **do** vs. **double**
- Left recursion elimination can be tricky
  - Can change associativity
- Space use proportional to input
  - The downside to linear time parsing
- Sometimes pre-lexed input would be nice

Left Recursion

Context Free Grammars

```
add_expr
 : cast_expr
 | add_expr + mult_expr
 | add_expr - mult_expr
```

Parsing Expression Grammars

```
add_expr
 = cast_expr (("+" / "-") mult_expr)*
```

Continuations

- According to wikipedia Continuations are an “Abstract representation of the control state of a program”
- Continuation Passing Style
  - IL for programming languages
- Compiling with Continuations
  - Desugaring to function and continuation calls
  - Desugaring to just function calls
- CESK Machine
  - Control Environment Store (K)ontinuation
  - One way to implement interpreters

Compiler vs. Interpreter

Compile to JavaScript

- CoffeeScript
- Elm
- Fay

Interpret in JavaScript

- Various Schemes
- O'Browser: OCaml bytecode interpreter in JavaScript

Challenges

Paradigm shift

- C is imperative
- JavaScript is Object-oriented, Imperative, Functional

Source, Intermediate, Target, and Implementation Languages

- Semantic differences can be challenging
- Compiler vs. Interpreter
- C -> Functionl IL may be forcing square peg into round hole

Similar Projects

Emscripten

- Compiles LLVM bitcode into JavaScript
- Using this would still require a C -> LLVM frontend

Clue

- Not web based
- Last updated over four years ago
- CScript is based on a C11 draft

The Future

- Finishing the parser and adding semantic actions
- Building the interpreter/compiler
- Putting the pieces together
- Dealing with the ugly parts of C

Contributing

- CScript is open source and all code is available at GitHub.com/dfeltey/CScript
- I'm very open to suggestions/opinions/contributions
- I hope that CScript could become a valuable learning/teaching tool

References

References:

- bford.info/packrat
- blog.might.net
- altJS.org