20—Meta-circular Evaluator

CS1101S: Programming Methodology

Martin Henz

November 2, 2012

Generated on Friday 2 November, 2012, 15:45

1. T-Diagrams
2. Interpreting JavaScript
3. True JavaScript
4. Lazy JavaScript

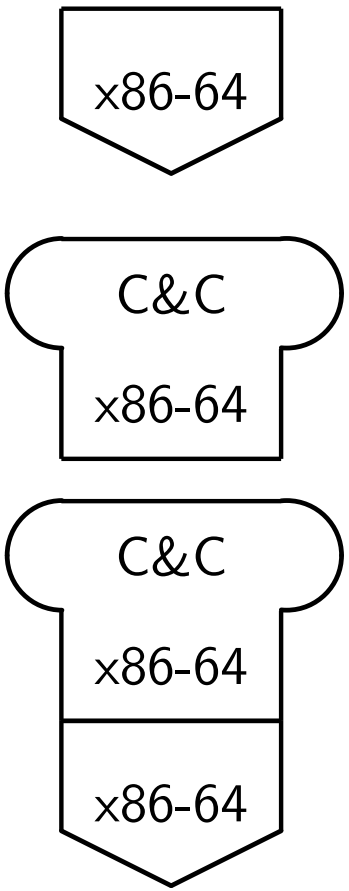
1 T-Diagrams

2 Interpreting JavaScript

3 True JavaScript

4 Lazy JavaScript

T-Diagrams

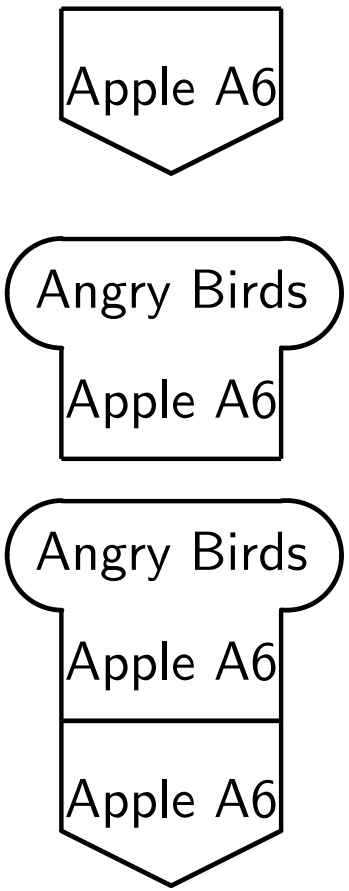


x86-64 Processor

Program "C&C" (x86-64 code)

"C&C" running on x86-64

Running an app on iPhone 5



Apple A6, the processor of iPhone 5

Program "Angry Birds" (A6 build)

"Angry Birds" running on iPhone 5

1 T-Diagrams

2 Interpreting JavaScript

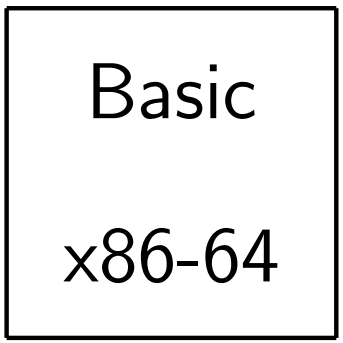
3 True JavaScript

4 Lazy JavaScript

Interpreter

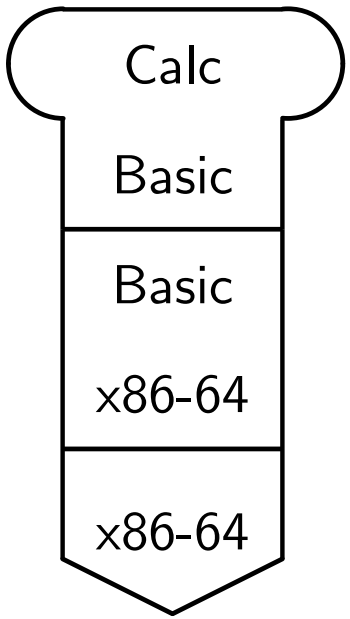
- Interpreter is program that executes another program
- The interpreter's *source language* is the language in which the interpreter is written
- The interpreter's *target language* is the language in which the programs are written which the interpreter can execute

Interpreters



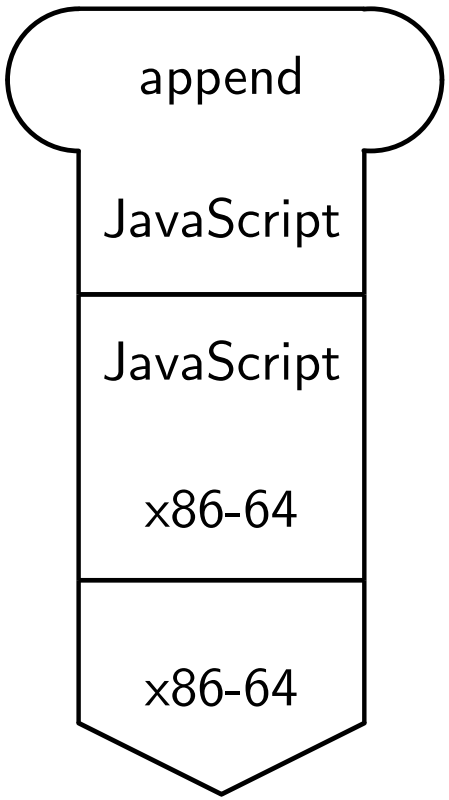
Interpreter for Basic, written in x86-64 machine code

Interpreting a Program



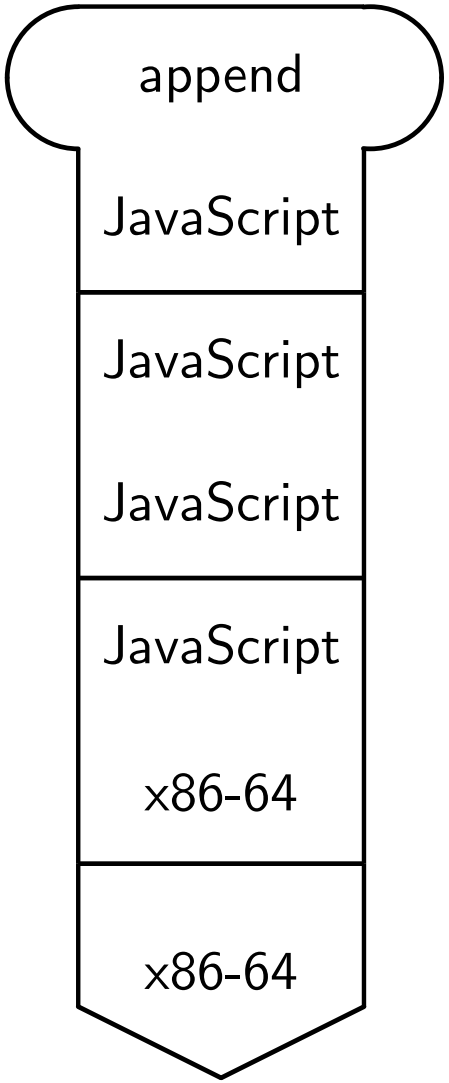
Basic program "Calc"
running on x86-64 using interpretation

Running JavaScript in Firefox



JavaScript "append" program running in Firefox

Running the meta-circular interpreter in Firefox

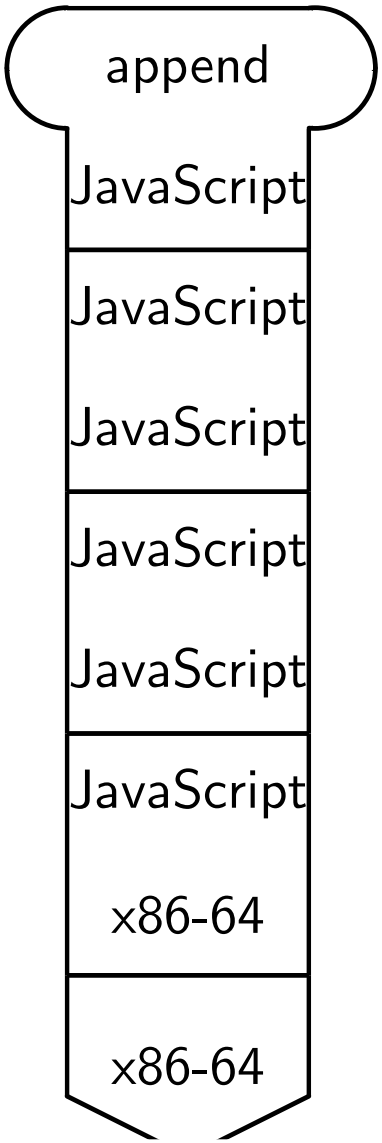


JavaScript "append" program running in Firefox using parse_and_evaluate ("... append (...);"); in Web Console.

Gaps between JavaScript and Meta-circular Interpreter

- Assignment to undeclared variables
- Variable declarations in function bodies
- Treatment of `undefined` in Firefox
- Missing features: `new`, `while`, etc
  Ambition: Add enough features to be able to intepret the interpreter

Meta-meta



JavaScript "append" program running in Firefox using

```
parse_and_evaluate(
 "function evaluate(...);..."+
 "parse_and_evaluate("+
 "   '...append(...);'"+
 ");"
);
```

in Web Console.

Assignment to undeclared variables

Idea

Modify set_variable_value such that the variable is added at toplevel (global environment)

Variable declarations in function bodies

Idea

Whenever a function is being defined, collect the variable declarations of the body, and save the list of declared variables in the function value

Modify function application

When functions are applied, extend the environment by a binding of the declared variables to `undefined`.

Recall Streams

Central idea of streams

Place data in head; represent the rest of the data (tail) by a function that is called when the data is needed

Why stop at streams?

We could aim for a general principle of "laziness": Compute values only when they are needed.

Lazy Functional Programming

Central idea

Execute applications of user-defined functions such that argument expressions are not evaluated, but placed in functions

When will the functions ever be evaluated?

When we apply a primitive operation, such as "+" to them, or when we "force" evaluation to get the overall result of the program.

Implementation

```
function list_of_values(exps,env) {
    if (no_operands(exps)) return [];
    else return pair(make_thunk(
                        first_operand(exps),env),
                     list_of_values(
                        rest_operands(exps),env));
}
```

Implementation

```
function evaluate(stmt,env) {
    ...
    else if (is_variable(stmt)) {
        var val=lookup_variable_value(
                    variable_name(stmt),
                    env);
        if (is_thunk(val) {
            return force_thunk(val);
        } else {
            return val;
} } }
```

Background

Lazy languages exist

A number of programming languages use lazy functional programming as the general evaluation mechanism: Haskell, Miranda

Lambda-calculus

The semantics of such languages is defined using a strategy to evaluate lambda calculus expressions, called *normal order reduction*