Studio 09-D

Metacircular Evaluator

Jia Xiaodong

5 Nov 2019

What is it?

- *Meta*

What is it?

- *Meta*
    - About itself.

What is it?

- *Meta*
    - About itself.
- *Circular*

What is it?

- *Meta*
  - About itself.
- *Circular*
  - The language is implemented in itself (self-hosting).

What is it?

- *Meta*
  - About itself.
- *Circular*
  - The language is implemented in itself (self-hosting).
- *Evaluator*

Programming

- Leaving the abstract: how does a computer work?

Programming

- Leaving the abstract: how does a computer work?
- They operate on a pre-defined language — machine code.

Programming

- Leaving the abstract: how does a computer work?
- They operate on a pre-defined language — machine code.
  - Normal desktops and laptops: `x86`

Programming

- Leaving the abstract: how does a computer work?
- They operate on a pre-defined language — machine code.
    - Normal desktops and laptops: `x86`
    - Mobile devices: `ARM`

Programming

- Leaving the abstract: how does a computer work?
- They operate on a pre-defined language — machine code.
  - Normal desktops and laptops: `x86`
  - Mobile devices: `ARM`
  - eg.: 893C2500000000

Programming

- Leaving the abstract: how does a computer work?
- They operate on a pre-defined language — machine code.
  - Normal desktops and laptops: `x86`
  - Mobile devices: `ARM`
  - eg.: 893C2500000000
  - Can be read with tools eg. `xxd`, `hexdump`.

Programming

- Leaving the abstract: how does a computer work?
- They operate on a pre-defined language — machine code.
    - Normal desktops and laptops: `x86`
    - Mobile devices: `ARM`
    - eg.: 893C2500000000
    - Can be read with tools eg. `xxd`, `hexdump`.
- Not very friendly for humans.

Programming

- Leaving the abstract: how does a computer work?
- They operate on a pre-defined language — machine code.
  - Normal desktops and laptops: `x86`
  - Mobile devices: `ARM`
  - eg.: 893C2500000000
  - Can be read with tools eg. `xxd`, `hexdump`.
- Not very friendly for humans.
  - Assembly: macros

Programming

- Leaving the abstract: how does a computer work?
- They operate on a pre-defined language — machine code.
    - Normal desktops and laptops: `x86`
    - Mobile devices: `ARM`
    - eg.: 893C2500000000
    - Can be read with tools eg. `xxd`, `hexdump`.
- Not very friendly for humans.
    - Assembly: macros
    - eg.: `mov $0x1,%edi`

Programming

- Leaving the abstract: how does a computer work?
- They operate on a pre-defined language — machine code.
  - Normal desktops and laptops: `x86`
  - Mobile devices: `ARM`
  - eg.: 893C2500000000
  - Can be read with tools eg. `xxd`, `hexdump`.
- Not very friendly for humans.
  - Assembly: macros
  - eg.: `mov $0x1,%edi`
  - Can be read with tools eg. gdb.

Programming

- Leaving the abstract: how does a computer work?
- They operate on a pre-defined language — machine code.
  - Normal desktops and laptops: `x86`
  - Mobile devices: `ARM`
  - eg.: 893C2500000000
  - Can be read with tools eg. `xxd`, `hexdump`.
- Not very friendly for humans.
  - Assembly: macros
  - eg.: `mov $0x1,%edi`
  - Can be read with tools eg. `gdb`.
- High level programming languages:

Programming

- Leaving the abstract: how does a computer work?
- They operate on a pre-defined language — machine code.
  - Normal desktops and laptops: `x86`
  - Mobile devices: `ARM`
  - eg.: 893C2500000000
  - Can be read with tools eg. `xxd`, `hexdump`.
- Not very friendly for humans.
  - Assembly: macros
  - eg.: `mov $0x1,%edi`
  - Can be read with tools eg. gdb.
- High level programming languages:
  - `return 1;`

Programming

- Leaving the abstract: how does a computer work?
- They operate on a pre-defined language — machine code.
  - Normal desktops and laptops: `x86`
  - Mobile devices: `ARM`
  - eg.: 893C2500000000
  - Can be read with tools eg. `xxd`, `hexdump`.
- Not very friendly for humans.
  - Assembly: macros
  - eg.: `mov $0x1,%edi`
  - Can be read with tools eg. gdb.
- High level programming languages:
  - `return 1;`
  - Can be read with tools eg. your eyes

Parsing

- Parsing is the way to comprehend a language.

Parsing

- Parsing is the way to comprehend a language.
- The programming language is defined with strict rules.

Parsing

- Parsing is the way to comprehend a language.
- The programming language is defined with strict rules.
- The language can be decomposed into elements with these rules.

Parsing

- Parsing is the way to comprehend a language.
- The programming language is defined with strict rules.
- The language can be decomposed into elements with these rules.
- Can be seen with `parse(str)` in Source.

Evaluation

- Irreducible things: done.

Evaluation

- Irreducible things: done.
- Reducible things: evaluate statement by statement.

Evaluation

- Irreducible things: done.
- Reducible things: evaluate statement by statement.
- See lecture slides for step-by-step walkthrough.

S12 Q1

Implement function definition hoisting in the MCE.

S12 Q1

Implement function definition hoisting in the MCE.

```
function reorder_statements(stmts) {
    function split_statements(stmts) {
        if (is_null(stmts)) {
            return pair(null, null);
        } else {
            const first_statement = head(stmts);
            const split_rest = split_statements(tail(stmts));
            return is_function_declaration(first_statement)
                   ? pair(pair(first_statement, head(split_rest)),
                          tail(split_rest))
                   : pair(head(split_rest),
                          pair(first_statement, tail(split_rest)));
        }
    }
    const split = split_statements(stmts);
    return append(head(split), tail(split));
}
```

S12 Q2

Make the MCE detect undeclared names.

S12 Q2

Make the MCE detect undeclared names.

```
function evaluate(component, env) {
    return is_literal(component)
        ? literal_value(component)
        : is_name(component)
        ? lookup_symbol_value(
        symbol_of_name(component),
        env)
        : is_application(component)
        ? apply(
        evaluate(
            function_expression(component),
            env),
        list_of_values(
            arg_expressions(component),
            env))
        : is_operator_combination(component)
        ? evaluate(
            operator_combination_to_application(component),
            env)
```

```
function check_names(component, env) {
    is_literal(component)
        ? "ok"
        : is_name(component)
        ? lookup_symbol_value(symbol_of_name(component), env)
        : is_application(component)
        ? check_names(
          make_sequence(
              pair(function_expression(component),
              arg_expressions(component))),
          env)
        : is_operator_combination(component)
        ? check_names(
          operator_combination_to_application(component),
          env)
```