

Structure and Interpretation of Computer Programs

R10 Metacircular Evaluator

Getting Started

For your convenience, the final evaluator given in class [is available here](#).

It may be useful to look at the parser output when working on some of the following problems. You can enter `display_list(parse("<statement>"))`; to do so.

The evaluator linked above includes the following convenient function:

```
function parse_and_evaluate(program) {
    return evaluate(make_block(parse(program)),
                    the_global_environment);
}
```

When applied to a given string input, it performs the following steps:

- parse input
- wrap program in a block which we called “implicit top-level block” when discussing the environment model
- evaluate the implicit top-level block using the global environment.

Note that besides single and double quotes, JavaScript (and Source) provide a backtick that can stretch over multiple lines. This is convenient when the program strings that we give to the parser become longer. Example:

```
parse_and_evaluate(`
function factorial(n) {
    n === 1
    ? 1
    : n * factorial(n - 1);
}
factorial(5);`);
```

Note that the interpreted language covered in class does not have `return` statements.

Problems:

1. The target languages of the evaluators given in the lecture do not provide an exponentiation operator. Modify the evaluator program for the calculator language such that the operator `%` is now the exponentiation operator, i.e. `b % e` evaluates to the value b^e .

Does `b % e % n` evaluate to b^{e^n} ? Why? If not, what needs to be changed in the evaluator?

2. The evaluator currently does not support the “lazy” logical composition operations; the following program leads to an error `Unknown component` in the metacircular evaluator, but evaluates to false in Source.

```
false && 0();
```

Modify the evaluator such that lazy logical composition operations are supported as described in the lectures.

Depending on your approach, you might find the following functions useful, [also available here](#).

```
// the syntax predicate
function is_logical_composition(component) {
    return is_tagged_list(component, "logical_composition");
}
// selectors
function logical_symbol(comp) {
    return list_ref(comp, 1);
}
function logical_composition_first_component(comp) {
    return list_ref(comp, 2);
}
function logical_composition_second_component(comp) {
    return list_ref(comp, 3);
}
// helper to make a conditional expression
function make_conditional_expression(pred, cons, alt) {
    return list("conditional_expression", pred, cons, alt);
}
// helper to make a literal value
function make_literal(value) {
    return list("literal", value);
}
```

3. The evaluator currently does not support the function `parse`, and therefore we cannot write a meta-metacircular evaluator!

- Add the function `parse` such that the following program evaluates to a list as expected:

```
parse_and_evaluate(`parse('parse("1;");');`);
// returns
// list("application", list("name", "parse"), list(list("literal", "1;")))
```

- In the program above you find four semicolons. Explain the meaning of each of them, in reverse order.
- (If you have a day to spare and like this sort of thing) Apply the evaluator to itself, i.e. modify/extend your evaluator sufficiently so that it can be truly metacircular: You should be able to evaluate the evaluator as follows:

```
// here goes your evaluator
parse_and_evaluate(`// here goes your evaluator again
                    parse_and_evaluate("(x => x + 1)(7);");`);
```

Report your findings to your tutor: What problems did you encounter? How did you solve them?