CS 61A
Fall 2023

Interpreters, More Scheme

Discussion 11: November 8, 2023

Getting Started

Someone from your group should join Discord.

To get help from a TA, send a message to the `discuss-queue` channel with the @discuss tag and your discussion group number.

If you have only 1 or 2 people in your group, you can join the other group in the room with you.

This discussion is designed to last **50 minutes or less**. If your group works for 50 minutes and you haven't finished yet, you can just skip to the end, fill out the attendance form, and wrap up. After that, you're free to go, or you can discuss the homework or project with your group.

Q1: Ice Breaker

Everybody say your name, and then figure out who is planning to travel outside of the Bay Area the soonest. They're your group's facilitator for the day.

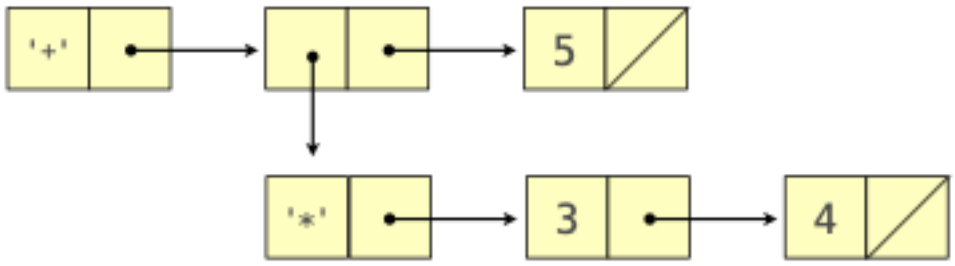
Facilitator: If you ever don't want to be facilitator anymore, just ask: "Would someone else be facilitator?" You can even do that now.

Representing Lists

A Scheme call expression is a Scheme list that is represented using a `Pair` instance in Python.

For example, the call expression `(+ (* 3 4) 5)` is represented as:

```
Pair('+', Pair(Pair('*', Pair(3, Pair(4, nil))), Pair(5, nil)))
```



(+ (* 3 4) 5)

The `Pair` class and `nil` object are defined in pair.py of the Scheme project.

```
class Pair:
    "A Scheme list is a Pair in which rest is a Pair or nil."
    def __init__(self, first, rest):
        self.first = first
        self.rest = rest

    ... # There are also __str__, __repr__, and map methods, omitted here.
```

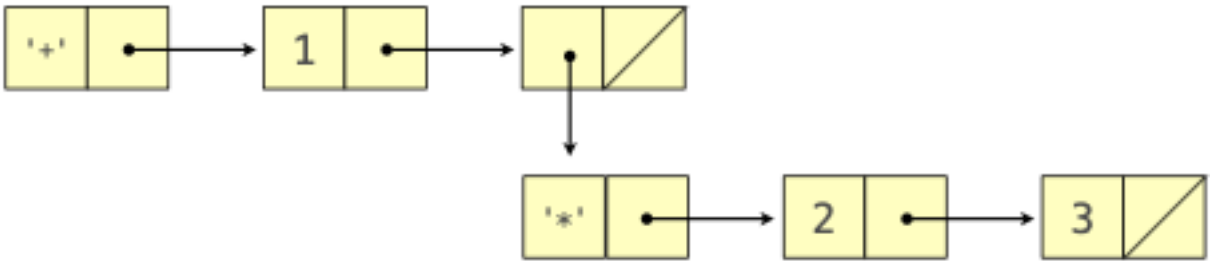
Q2: Representing Expressions

Write the Scheme expression in Scheme syntax represented by each `Pair` below. Try drawing the linked list diagram too. The first one is done for you.

```
Pair('+', Pair(Pair('*', Pair(3, Pair(4, nil))), Pair(5, nil)))
```

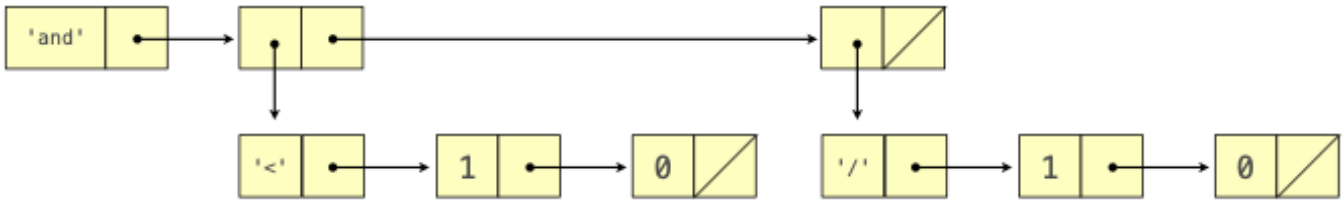
Answer: `(+ (* 3 4) 5)`

```
>>> Pair('+', Pair(1, Pair(Pair('*', Pair(2, Pair(3, nil))), nil)))
```



`(+ 1 (* 2 3))`

```
>>> Pair('and', Pair(Pair('<', Pair(1, Pair(0, nil))), Pair(Pair('/', Pair(1, Pair(0, nil
    ))), nil)))
```



`(and (< 1 0) (/ 1 0))`

Facilitator: Ask your group, "What does `(and (< 1 0) (/ 1 0))` evaluate to?" Once you all agree, ask someone to post your answer in your group's Discord channel's text chat.

Evaluation

To evaluate the expression `(+ (* 3 4) 5)` using the Project 4 interpreter, `scheme_eval` is called on the following expressions (in this order):

1. `(+ (* 3 4) 5)`
2. `+`
3. `(* 3 4)`
4. `*`
5. `3`
6. `4`
7. `5`

Discussion time: Describe to each other why `*` is evaluated and what it evaluates to.

Facilitator: Decide when discussion has gone on long enough (maybe 3-5 minutes); when you think the group has a good answer already or is not making progress, say: "Let's look at the answer now."

The `*` is evaluated because it is the operator sub-expression of `(* 3 4)`, which is an operand sub-expression of `(+ (* 3 4) 5)`.

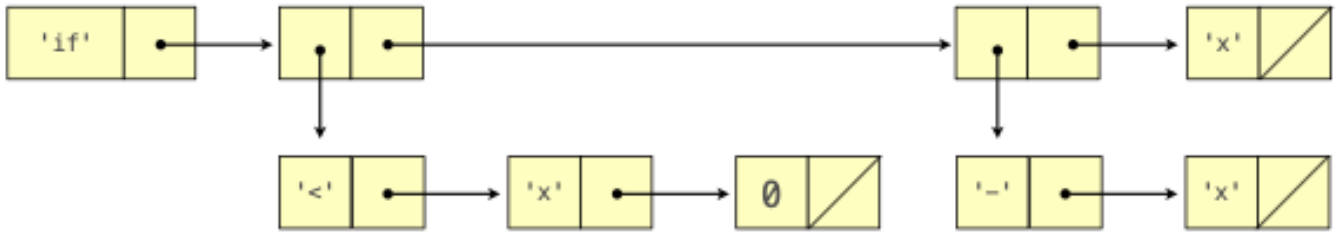
By default, `*` evaluates to a procedure that multiplies its arguments together. But `*` could be redefined at any time, and so the symbol `*` must be evaluated each time it is used in order to look up its current value.

```
scm> (* 2 3)  ; Now it multiplies
6
scm> (define * +)
*
scm> (* 2 3)  ; Now it adds
5
```

An `if` expression is also a Scheme list represented using a `Pair` instance.

For example, `(if (< x 0) (- x) x)` is represented as:

```
Pair('if', Pair(Pair('<', Pair('x', Pair(0, nil))), Pair(Pair('-', Pair('x', nil)), Pair('x', nil))))
```



To evaluate this expression in an environment in which `x` is bound to 2 (and `<` and `-` have their default values), `scheme_eval` is called on the following expressions (in this order): 1. `(if (< x 0) (- x) x)` 1. `(< x 0)` 1. `<` 1. `x` 1. `0` 1. `x`

Presentation time: Come up with a short explanation of why neither `if` nor `-` are evaluated even though they both appear in `(if (< x 0) (- x) x)`. Once your group agrees on an answer (or wants help), send a message to the `#discuss-queue` channel with the `@discuss` tag, your discussion group number, and the message "If you please!"

Note: This worksheet is a problem bank—most TAs will not cover all the problems in discussion section.

and a member of the course staff will join your voice channel to hear your explanation and give feedback. The facilitator can decide who will present your explanation (or ask for volunteers).

The symbol `if` is not evaluated because it is the start of a special form, not part of a call expression. The symbols that introduce special forms (`and`, `if`, `lambda`, etc.) are never evaluated.

The symbol `-` is not evaluated, nor is the whole sub-expression `(- x)` that it appears in, because `(< x 0)` evaluates to `#f`. **If you're still not certain why some parts are evaluated and some aren't, ask the course staff.**

Q3: Evaluation

Which of the following are evaluated when `scheme_eval` is called on `(if (< x 0) (- x) (if (= x -2) 100 y))` in an environment in which `x` is bound to -2? (Assume `<`, `-`, and `=` have their default values.)

- `if`
- `<`
- `=`
- `x`
- `y`
- `0`
- `-2`
- `100`
- `-`
- `(`
- `)`

With `x` bound to -2, `(< x 0)` evaluates to `#t`, and so `(- x)` will be evaluated, but `(if (= x 1) 100 x)` will not. The operator and operands of a call expression are evaluated for every call expression that is evaluated. `(< x 0)` and `(- x)` are both call expressions.

Facilitator: Ask someone to tag `@discuss` and list your group's answers in your group's Discord channel's text chat. For example, write `if < 0 )` if you think those are the four that get evaluated. A member of the course staff will review your answer and give feedback.

Q4: Print Evaluated Expressions

Define `print_evals`, which takes a Scheme expression `expr` that contains only numbers, `+`, `*`, and parentheses. It prints all of the expressions that are evaluated during the evaluation of `expr`. They are printed in the order that they are passed to `scheme_eval`.

Note: Calling `print` on a `Pair` instance will print the Scheme expression it represents.

```
>>> print(Pair('+', Pair(Pair('*', Pair(3, Pair(4, nil))), Pair(5, nil))))
(+ (* 3 4) 5)
```

```
def print_evals(expr):
    """Print the expressions that are evaluated while evaluating expr.

    expr: a Scheme expression containing only (, ), +, *, and numbers.

    >>> nested_expr = Pair('+', Pair(Pair('*', Pair(3, Pair(4, nil))), Pair(5, nil)))
    >>> print_evals(nested_expr)
    (+ (* 3 4) 5)
    +
    (* 3 4)
    *
    3
    4
    5
    >>> print_evals(Pair('*', Pair(6, Pair(7, Pair(nested_expr, Pair(8, nil))))))
    (* 6 7 (+ (* 3 4) 5) 8)
    *
    6
    7
    (+ (* 3 4) 5)
    +
    (* 3 4)
    *
    3
    4
    5
    8
    """
    if not isinstance(expr, Pair):
        print(expr)
    else:
        print(expr)
        while expr is not nil:
            print_evals(expr.first)
            expr = expr.rest
```

If `expr` is not a pair, then it is a number or `'+'` or `'*'`. In all of these cases, the `expr` should be printed to indicate that it would be evaluated.

If `expr` is a pair, then it is a call expression. Print it. Then, the operator and operands are evaluated. These are the elements in the list `expr`. So, iterate through `expr` (using either a `while` statement or `expr.map(...)`) and call `print_evals` on each element.

Document the Occasion

Please all fill out the attendance form (one submission per person per week).