Interpreters and Macros

Abhinav Ashar
CS 61A: Structure and Interpretation of Computer Programs

January 18, 2019

1 Interpreters

a) 3 steps to a function call: evaluate the operator, evaluate the operand, and apply the result of the evaluated operator to the result of the evaluated operands

b) In the statement above, steps 1 and 2 are both +1 for scheme eval, and step 3 is +1 for scheme apply

c) For the first step, always scheme eval the entire expression

d) Do not call scheme eval on a special form keyword (Ex. if)

e) Short circuiting can still apply

f) When evaluating, go deep by going all the way through one evaluation before moving onto the next evaluation
Example:
(+ (+ 1 2) 3)
Order of evaluation: entire expression, +, (+ 1 2), + (the one in (+ 1 2)), 1, 2, 3

g) The **let** keyword locally assigns values and then uses these local values in some expression that the end of the let statement

h) The *comma* is used to unquote or evaluate a statement

2 Macros

a) Allows you to avoid evaluating inputs so you can evaluate the expressions only under certain conditions

b) Macros can allow you to create syntax in Scheme that normally would not exist (Ex. for loops)

c) define-macro avoids the need of an eval call that is necessary at the end for non-macro functions (see the CSM worksheet for an example)