Quasiquote

Last updated Spring 2019.

This guide serves as a review and extension of the Quasiquote special form covered in class. While this goes fairly in depth, please refer to lecture slides and discussions first.

Definitions and Background

Refresh on the `quote` special form

Earlier, we saw the `quote` special form, or `'` for short. The `quote` special form takes in an expression and returns the expression without evaluating it.

For example,

```
scm> 'hello
hello
scm> (quote hello)
hello
scm> '(cons 1 (cons 2 nil))
(cons 1 (cons 2 nil))
```

Introducing the `quasiquote` special form

Similarly, `quasiquote` also takes in an expression and returns the expression without evaluating it, **unless a subexpression of expression is unquoted.** The shorthand notation for `quasiquote` is a backtick, `` ` ``.

```
(quasiquote <expr>) OR `<expr>
```

The `unquote` special form essentially "unquotes" the expression directly following it. Unquoting means to evaluate the expression. The shorthand notation for `unquote` is a comma (`,`). *NOTE: The `unquote` special form must be inside a quasiquote expression!*

Combining the two special forms above, we can do amazing things!

```
scm> (quasiquote (4 (unquote (+ 2 3)) 6))
(4 5 6)
scm> `(4 ,(+ 2 3) 6)
(4 5 6)
scm> (eval `(and ,(if (- 1 2) 3 4) (= 1 1) ,(or)))
#f
```

Looking at the last example more closely, we start with `scm> (eval `(and ,(if (- 1 2) 3 4) (= 1 1) ,(or)))`. Let's work through the `quasiquote` expression step by step.

- The first subexpression is `and`. Since `and` is not unquoted, our expression so far would be `(and`.
- The next subexpression is `,(if (- 1 2) 3 4)`. Because there is an unquote, we have to evaluate the expression that follows, `(if (- 1 2) 3 4)` which evalutes to 3. 3 is true, so we now have `(and 3`.
- The next subexpression is `(= 1 1)`. While this indeed evaluates to `#t`, notice there is no unquote. Thus, our expression so far becomes `(and 3 (= 1 1)`.
- The last subexpression is `,(or)`. The unquote tells us to evaluate `(or)`, which from last week's section, we learned evaluates to `#f`.

Thus, the `quasiquote` expression evaluates to `(and 3 (= 1 1) #f)`, which would then evaluate to #f.