S-Expressions and Trees

CS251 Programming Languages
Fall 2017, Lyn Turbak

Department of Computer Science
Wellesley College

Symbols

Lisp was invented to do **symbolic processing**.

A key Racket value is the **symbol**.

The symbol `cat` is written `(quote cat)` or `'cat`.

Symbols are values and so evaluate to themselves.

```
> 'cat
'cat
```

```
; 'thing is just an abbreviation for (quote thing)
> (quote cat)
'cat
```

Symbols similar to strings, except they're **atomic**; we don't do character manipulations on them.

S-Expressions

Lisp pioneered ***symbolic expressions***, a.k.a. ***s-expressions***, a parenthesized notation for representing trees as nested lists (compare to other tree notations, like XML or JSON).

Example:

```
'((this is (a nested)) list (that (represents a) tree))
```


Atoms

The leaves of an s-expression are atomic (indivisible) and so are called ***atoms***. In Racket, atoms include numbers, booleans, and strings in addition to symbols.

Example: `'((251 #f) ("foo bar" baz))`

Quotation with Atoms and Lists

A quoted atom `(quote atom)` (abbreviated `'atom`) denotes the atom. For atoms that are not symbols, `(quote atom)` desugars to `atom`. For example:

- `(quote 251)` desugars to `251`
- `(quote #t)` desugars to `#t`
- `(quote "Hi there!")` desugars to `"Hi there!"`

A quoted parenthesized structure `(quote (...))` (abbreviated `'(...)`) denotes a list, according to the following desugaring:

`(quote (sexp_1 … sexp_n))`
desugars to `(list (quote sexp_1) … (quote sexp_n))`

Example: What is the desugaring of the following:

```
'((17 foo #f) "bar" (list + (quote quux)))
```


A sample s-expression

We will do some exercises with this sample s-expression:

```
(define tr '((a (b c) d) e (((f) g h) i j k)))
```

Draw the tree associated with this s-expression.

Functions on s-expression trees

Write the following functions that take an s-expression tree as their only arg:

1. `(sexp-num-atoms sexp)` returns the number of atoms (leaves) in the s-expression tree `sexp`

```
> (sexp-num-atoms tr)
11
```

2. `(sexp-atoms sexp)` returns a list of the atoms (leaves) encountered in a left-to-right depth first search of the s-expression tree `sexp`.

```
> (sexp-atoms tr)
'(a b c d e f g h i j k)
```

3. `(sexp-height sexp)` returns the height of the s-expression tree `sexp`.

```
> (sexp-height tr)
4
```


An s-expression Read-Eval-Print Loop (REPL)

```
(define (sexp-repl)
  (begin (display "Please enter an s-expression:")
     (let {[(sexp (read)]} ; read prompts user for sexp
       (if (eq? sexp 'quit)
            'done
            (begin (display (list 'sexp-num-atoms:
                                  (sexp-num-atoms sexp)))
                   (newline)
                   (display (list 'sexp-atoms:
                                  (sexp-atoms sexp)))
                   (newline)
                   (display (list 'sexp-height:
                                  (sexp-height sexp)))
                   (newline)
                   (sexp-repl))))))
```

On to Metaprogramming

A ***metaprogram*** is a program that manipulates another program, such as an interpreter, compiler, type checker, assembler, etc.

In a metaprogram, how could we represent a Racket definition like this?

```
(define avg (lambda (a b) (/ (+ a b) 2)))
```