EECS 368
Programming Language Paradigms

Dr. Andy Gill

Department of Electrical Engineering & Computer Science
University of Kansas

October 8, 2015

KU

(Scheme)

We can define new atoms on the command line.

```
> (define atom "Smallish Thing")
> (define turkey "Fair Game")
> atom
"Smallish Thing"
> turkey
"Fair Game"
```

Better is putting the definitions in a file, and loading them each time.

We store the following definitions in a file.

```
(define atom "Smallish Thing")
(define turkey "Fair Game")
(define *abc$ "Silly String")
```

```
> *abc$
"Silly String"
> (atom)
procedure application: expected procedure,
            given: "Smallish Thing" (no arguments)
```

The problem is that the command line **expects** a function call, not data.

Quoting

In Java, there is a difference between `foo` and ``` ``foo'' ```.

```
String foo = "foo";
```

One is a variable, the other data.

In Scheme, we quote our data by prefixing a single quote.

- `turkey` is the name of an expression.
- `'turkey` is an expression.

KU

S-Expressions

```
SExp ::= atom
       | string
       | ' SExp
       | ( SExp* )
```

- Atoms are names that do not begin with '(', ')', '"', '‘'
- Strings are like Java Strings, starting with '"', and ending with '"'

KU

Scheme

We know know most of Scheme.

- The syntax is *really* simple
- The same syntax is used for data as well as programs

KU

```
(define atom "Smallish Thing")
(define turkey "Fair Game")
```

```
> (atom)
procedure application: expected procedure,
            given: "Smallish Thing" (no arguments)
> '(atom)
(list 'atom)
```

The problem was that the command line **expects** a function call, not data, so we used ' to denote the list as data.

```
.. no definitions ..
```

```
> 'atom
atom
> '(atom)
(list 'atom)
> '(atom turkey or)
(list 'atom 'turkey 'or)
> '(atom turkey) or
 or: bad syntax in: or
> '((atom turkey) or)
(list (list 'atom 'turkey) 'or)
```

KU

```
(((how) are) ((you) (doing so)) far)
```

How many S-expressions are in this list?

- Is () a list?

- Is () a list? Yes
- It is an empty list!

- Is () a list? Yes
- It is an empty list!

- Is () an atom?

- Is () a list? Yes
- It is an empty list!

- Is () an atom? No
- It is a list, not an atom

- Is (( ) ( ) ( ) ( )) a list?

- Is (() () () ()) a list? Yes
- It is a list of empty lists

```
(define ls1 '(a b c))
(define ls2 '((a b c) x y z))
(define ls3 'hotdog)
(define ls4 '())
```

```
> (car ls1)
a
> (car ls2)
(list 'a 'b 'c)
> (car ls3)
. . car: expects argument of type <pair>;
                                given hotdog
> (car ls4)
car: expects argument of type <pair>; given ()
```

The Law of Car

The primitive `car` is defined only for non-empty lists

```
(define ls5 '(((hotdogs)) (and) (pickle) relish))
```

```
> (car ls5)
(list (list 'hotdogs))
```

cdr (pronounced could-er) returns the rest of a list

```
> (cdr ls5)
(list (list 'and) (list 'pickle) 'relish)
```

KU

```
(define ls5 '(((hotdogs)) (and) (pickle) relish))
```

```
> (car ls5)
(list (list 'hotdogs))
> (cdr ls5)
(list (list 'and) (list 'pickle) 'relish)
> (car (car ls5))
???
```

```
(define ls5 '(((hotdogs)) (and) (pickle) relish))
```

```
> (car ls5)
(list (list 'hotdogs))
> (cdr ls5)
(list (list 'and) (list 'pickle) 'relish)
> (car (car ls5))
???
```

```
(list 'hotdogs)
```

Notice the difference between

```
(list (list 'hotdogs)) and (list 'hotdogs)
```

KU

```
(define ls3 'hotdog)
(define ls4 '())
(define ls5 '(((hotdogs)) (and) (pickle) relish))
```

```
> (cdr ls3)
cdr: expects argument of type <pair>;
                                given hotdog
> (cdr ls4)
cdr: expects argument of type <pair>;
                                given ()
> (car (cdr (cdr ls5)))
???
```

```
(define ls3 'hotdog)
(define ls4 '())
(define ls5 '(((hotdogs)) (and) (pickle) relish))
```

```
> (cdr ls3)
cdr: expects argument of type <pair>;
                                given hotdog
> (cdr ls4)
cdr: expects argument of type <pair>;
                                given ()
> (car (cdr (cdr ls5)))
???
```

```
(list 'pickle)
```

KU

The Law of Cdr

The primitive `cdr` is defined only for non-empty lists, and the `cdr` of any non-empty list is always another list.

KU

Glossary of acronyms:

CAR Originally meant "Contents of Address portion of Register", which is what CAR actually did on the IBM 704.

CDR Originally meant "Contents of Decrement portion of Register", which is what CDR actually did on the IBM 704. Pronounced "Cudder" /kUdd@r/ (as in "a cow chews its cdr"). The first syllable is pronounced like "could".

(from online lisp FAQ on `www.faqs.org`)

KU

cons command

```
(define ls1 '(a b c))
(define ls2 '((a b c) x y z))
(define ls3 'hotdog)
(define ls4 '())
(define ls5 '(((hotdogs)) (and) (pickle) relish))
```

```
> (cons ls3 ls1)
(list 'hotdog 'a 'b 'c)
> (cons ls1 ls1)
(list (list 'a 'b 'c) 'a 'b 'c)
> (cons ls4 ls1)
(list '() 'a 'b 'c)
> (cons ls4 ls3)
???
```

KU