CSC 520, Spring 2020

Principles of Programming Languages

Michelle Strout

Version: 1

February 3, 2020

This Week's Plan

- **Last time: Metatheory enables us to prove things about ALL programs in a language**
- **This week: Can prove algebraic laws from operational semantics**
  - Higher-level of abstraction
  - Algebraic laws can help guide recursive implementations
- **Context will be recursion and composition in uScheme**
  - In-depth study of recursive functions
  - Two recursive data structures: the list and the S-expression
  - More powerful ways of putting functions together

Algebraic Laws to Recursive Functions

- **To discover recursive functions, write algebraic laws:**

```
sum 0 = 0
sum n = n + sum(n-1)
```

- **Which side of the equality gets smaller?**

- **Code:**

```
(define sum (n)
    (if (= n 0) 0 (+ n (sum (- n 1)))))
```

- **Another example:**

```
exp x 0 = 1
exp x (n+1) = ?
```

For a new language, five powerful questions:

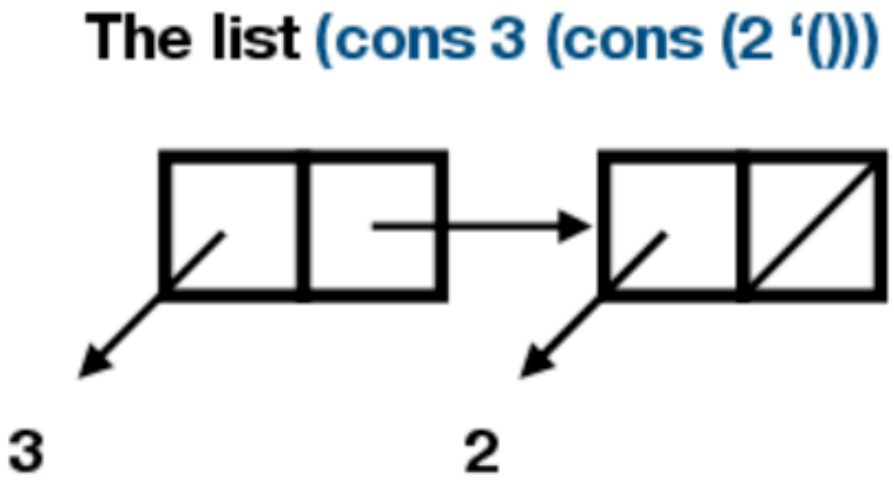
- **You can ask these questions about any language**
  1. What is the abstract syntax? Syntax categories?
  2. What are the values?
  3. What environments are there? What are names mapped to?
  4. How are terms evaluated?
  5. What's in the initial basis? Primitives and otherwise, what is built in?

- **Introduction to Scheme**
  - Question 2. What are the values?
  - Two new kinds of data:
    - The function closure: the key to "first-class" functions
    - Pointer to automatically managed cons cell

Graphically

- **Two cons cells**

```
(cons 3 (cons 2 '()))
```



Scheme Values

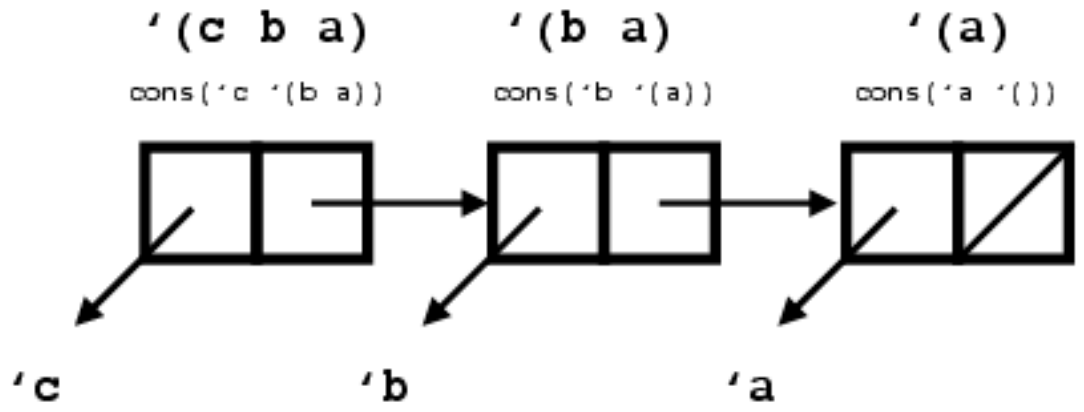
- **Values are S-expressions**
- **An S-expression is either:**
  - A symbol ‘GouldSimpson ‘UofA
  - A literal integer 0 77
  - A literal Boolean #t #f
  - (cons v1 v2), where v1 and v2 are S-expressions
- **A list of S-expressions is either**
  - The empty list ‘()
  - (cons v1 v2), where v1 is an S-expression and v2 is a list of S-expressions

S-Expression Operators

- **Like any other abstract data type, S-Expressions have:**
  - **Creators that create new values of the type '()**
  - **Producers that make new values from existing values (cons s s')**
  - **Mutators that change values of the type (not in uScheme)**
  - **Observers that examine values of the type**
    - **number?**
    - **symbol?**
    - **boolean?**
    - **null?**
    - **pair?**
    - **car?**
    - **cdr?**

Examples of S-Expression operators

```
(cons 'a '())          also written '(a)

(cons 'b '(a))         equals '(b a)

(cons 'c '(b a))       equals '(c b a)

(null? '(c b a))       equals #f

(cdr '(c b a)          equals '(b a)

(car '(c b a)          equals 'c
```



Your turn!

- **What is the representation of**

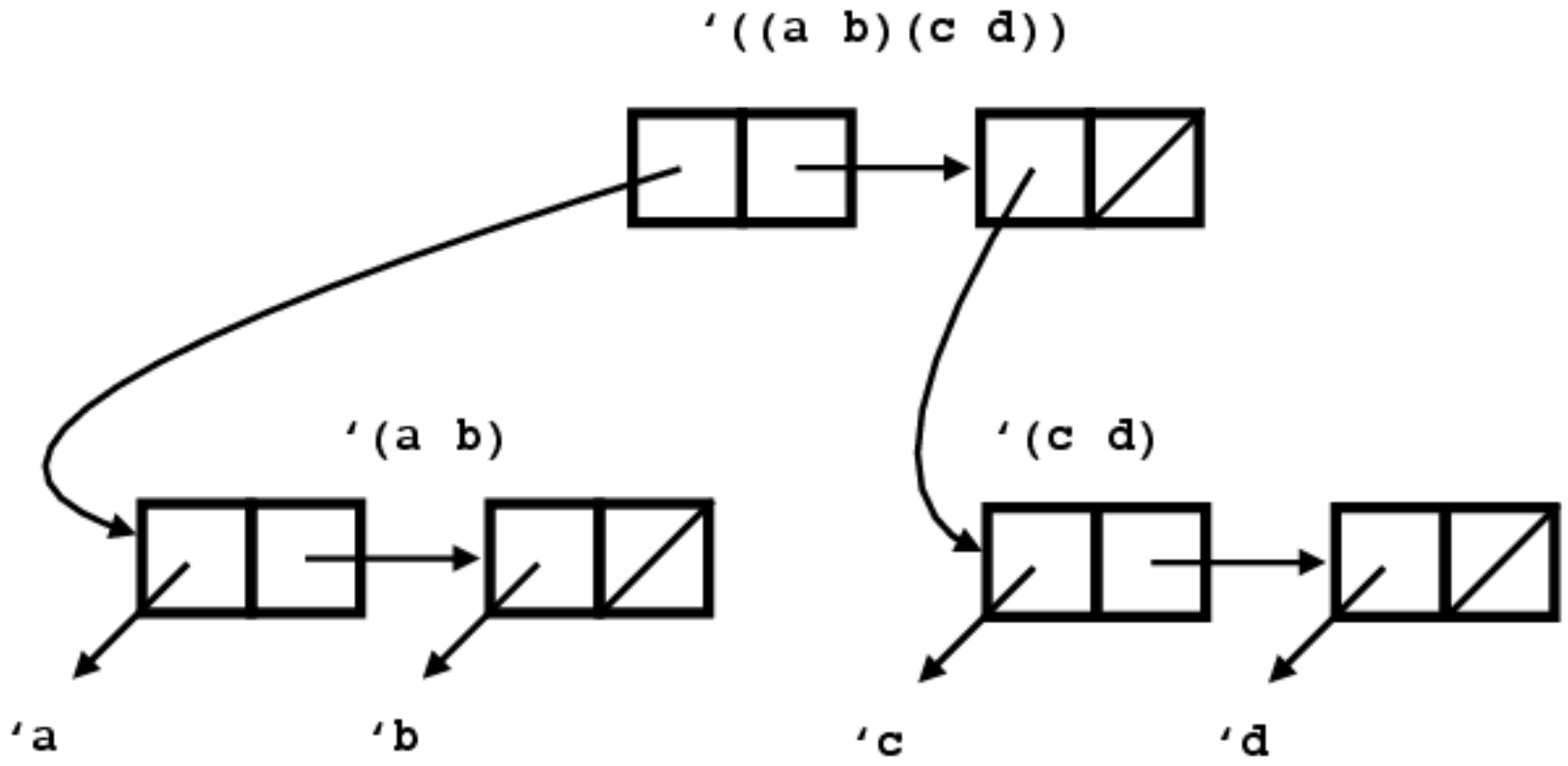
```
'((a b) (c d))
```

- **Which can be alternatively written**

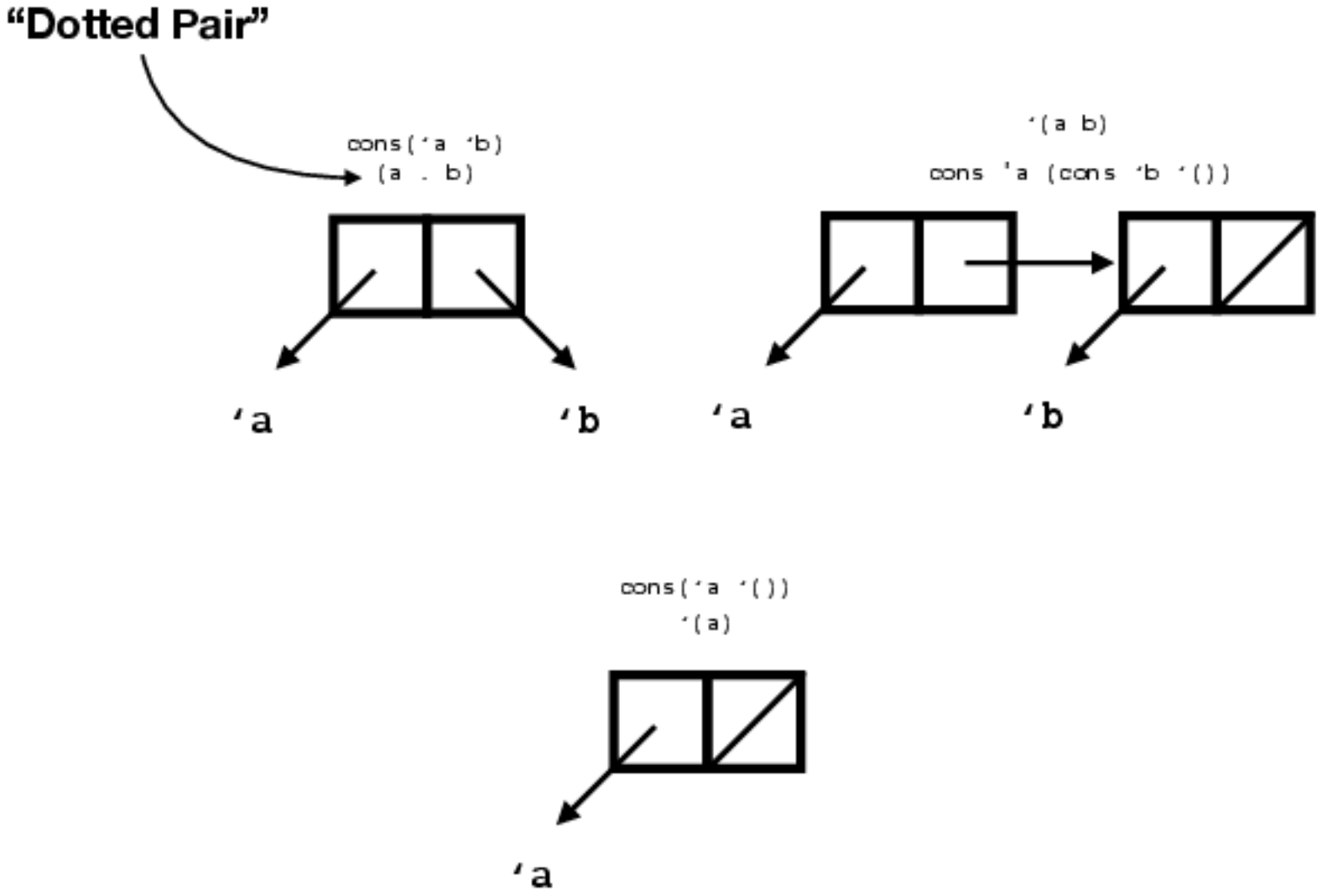
```
(cons (cons 'a (cons 'b '()))
      (cons (cons 'c (cons 'd '())) '()))
```

```
'((a b) (c d))
```

```
(cons (cons 'a (cons 'b '()))
      (cons (cons 'c (cons 'd '())) '()))
```



```
(cons `a `b)
```



Next

- **Lists**
  - Are a subset of S-Expressions, what is an S-Expr that isn't a list?
  - Can be defined via a recursive equation or by inference rules
- **Algebraic Laws for writing functions**
- **The cons cost model**
- **The method of accumulating parameters**

Lists defined inductively

$LIST(Z)$ is the smallest set satisfying this equation:

$$\mathrm{LIST}(Z) = \{\texttt{'()}\} \cup \{\texttt{(cons}\ z\ zs\texttt{)} \mid z \in Z, zs \in \mathrm{LIST}(Z)\}$$

Equivalently, $LIST(Z)$ is defined by these rules:

$$\frac{}{\texttt{'()} \in List(Z)}\ (\textsc{Empty})$$

$$\frac{z \in Z \qquad zs \in List(Z)}{\texttt{(cons}\ z\ zs\texttt{)} \in List(Z)}\ (\textsc{Cons})$$

Lists

- **Constructors**

```
'()    cons
```

- **Observers**

```
null? pair? car cdr
```

- **Why are lists useful**
  - Sequences are a frequently used abstraction
  - Can easily approximate a set
  - Can implement finite maps with association lists (aka)
  - You don't have to manage memory
  - Immutable data structures, instead build a new one

Review: Algebraic laws of lists

- **You fill in these right-hand sides:**

```
(null? '()) =
(null? (cons v vs)) =
(car (cons v vs)) =
(cdr (cons v vs)) =

(length '()) =
(length (cons v vs)) =
```