Lisp Data Structures

Lisp Good Points

- ◊ Consistent structure for data and programs
  - » **Both are lists**
- ◊ Clean design for ‘Pure’ Lisp
  - » **Common Lisp not so clean – lot of operational features**
- ◊ Promotes modular programming through lots of small functions.

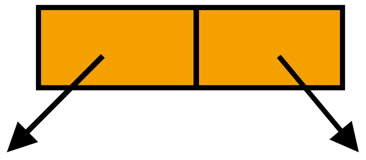
Lisp Bad Points

- ◊ Excessive use of parenthesis can make it difficult to understand
  - » **Lots of Insignificant Silly Parenthesis**
- ◊ Prefix for all operators makes arithmetic clumsy
  - » **But for everything else matches procedure calls**
- ◊ Lambda calculus underpinning can be difficult for beginners to understand

Data Structures

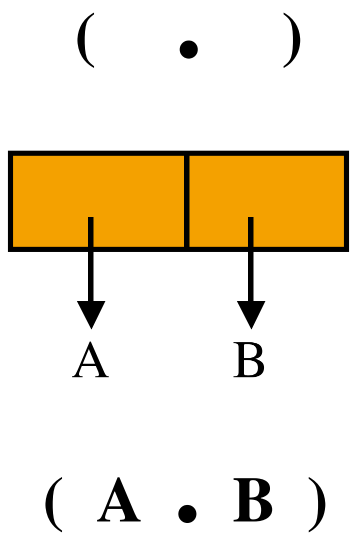
◊ Atoms
  - » **Essentially simple, but ...**
  - » **can have a complex internal structure**
    - > **see notes on symbols**

◊ Lists
  - » **Actually binary trees**
  - » **Only binary cells**

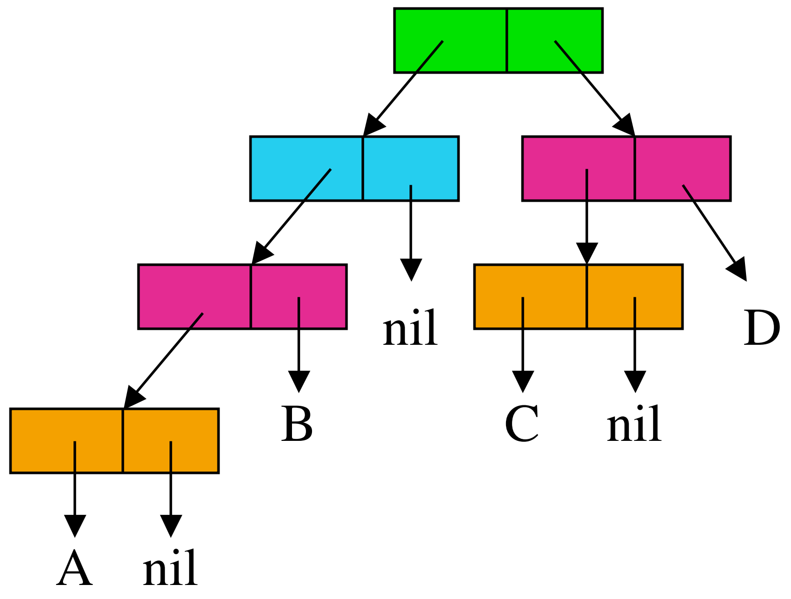


Dotted Notation

- Most general notation
- Directly encodes list structures
- Parenthesis pair denote a cell with a dot separating the two parts of the cell
- Recursive definition



Dotted notation example



```
((((A.nil).B).nil).((C.nil).D))
```

```
((((A.nil).B).nil).((C.nil).D))
```

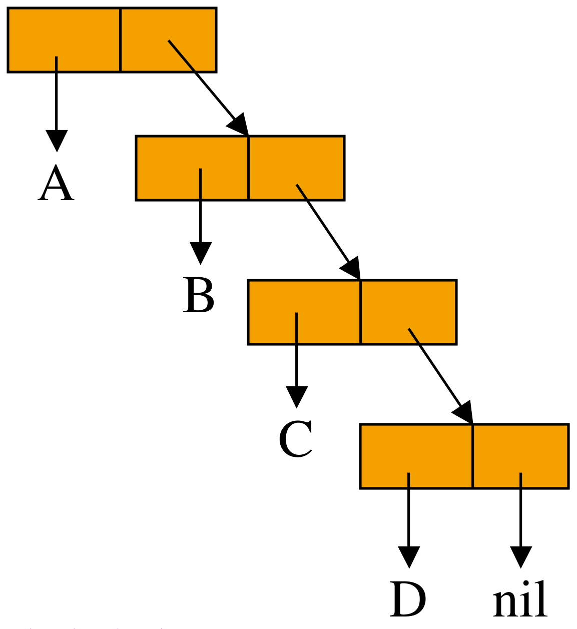
S-expressions

- Most common structure is a list
- Simplify by removing the 'redundant' dots and parenthesis

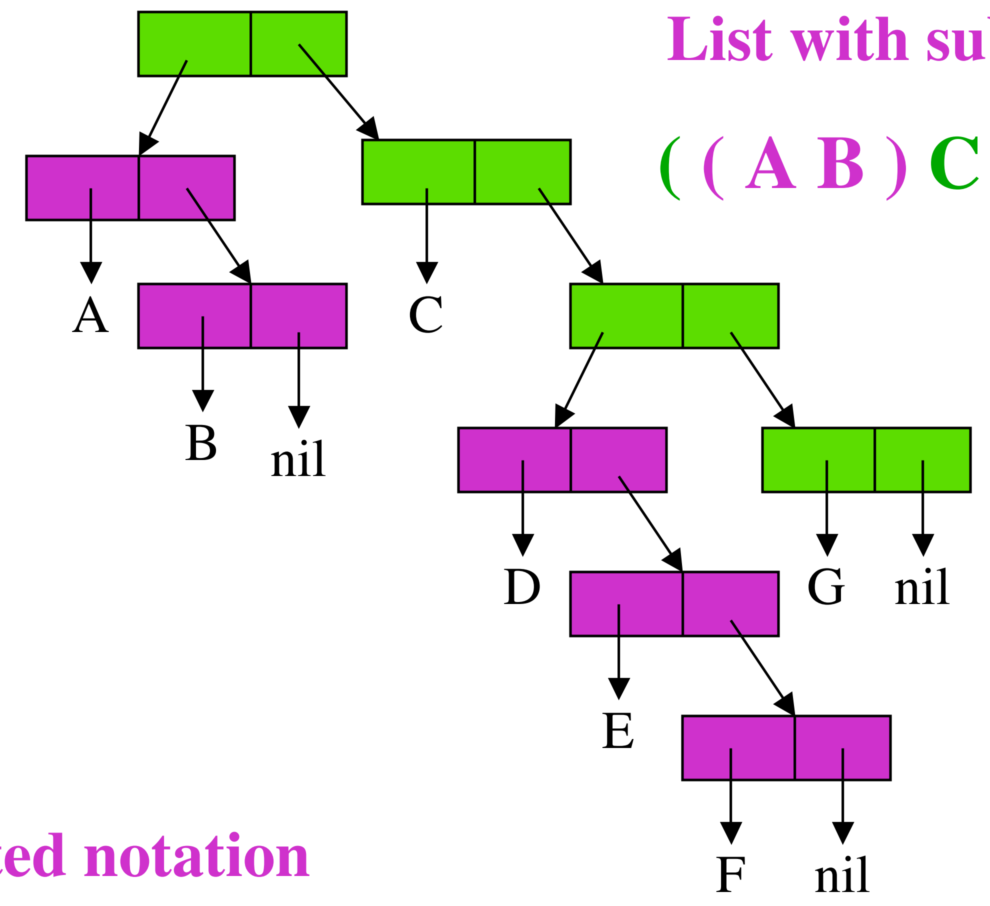
(A B C D)

- Instead of

(A . (B . (C . (D . nil))))



S-expressions – example 1



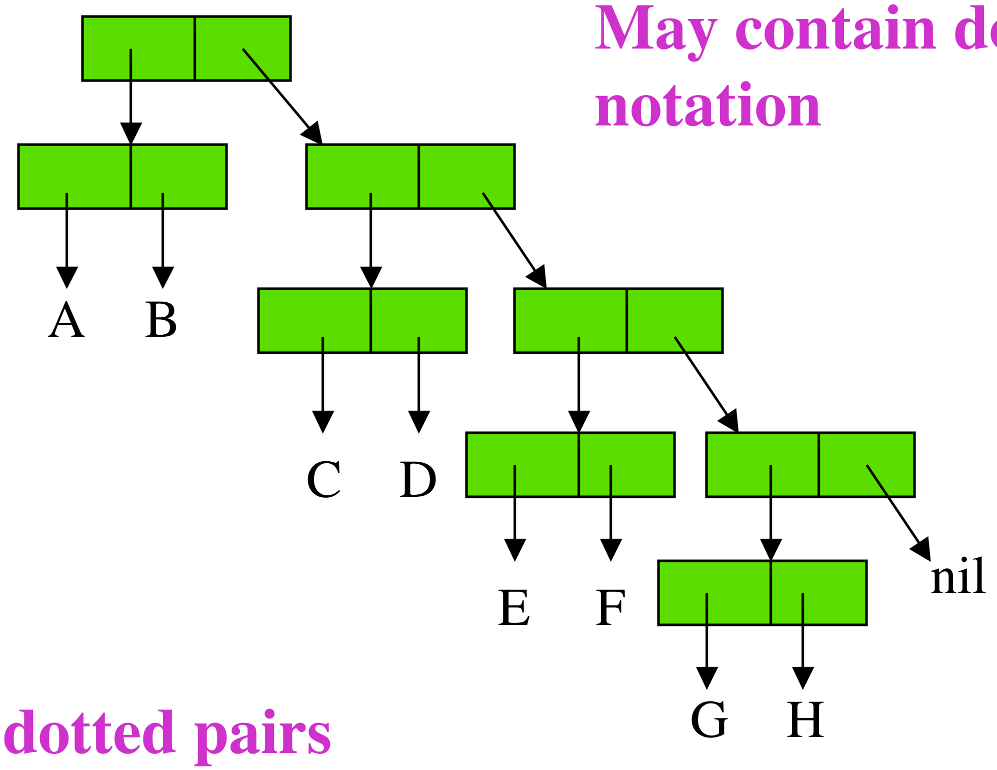
List with sublists

((A B) C (D E F) G)

Dotted notation

((A . (B . nil)) . (C . ((D . (E . (F . nil))) . (G . nil))))

S-expressions – example 2



May contain dotted notation

List of dotted pairs

((A • B) (C • D) (E • F) (G • H))

S-expressions cont'd

- ◊ The empty list is

  nil = **( )**

- ◊ It is both an **atom** and a **list** !

Dotted => S-expression

Apply the following rules from **right to left**

1 **Replace** **. nil)** **with** **)**

- » **end of a list**
- » **(A . nil) ----> (A)**

2 **Replace** **. (...)** **with** **‘space’ ...**

- » **end of a list**
- » **(A . (B . C)) ---> (A B . C)**
- » **What is the length of the above list?**

Example dotted => S-expr

Can apply rule 1 in three places

```
( ( A . ( B . nil ) ) . ( C . ( ( D . ( E . ( F . nil ) ) ) . ( G . nil ) ) ) )
( ( A . ( B ) ) . ( C . ( ( D . ( E . ( F ) ) ) . ( G ) ) ) )
```

Can apply rule 2 in three places

```
( ( A . ( B ) ) . ( C . ( ( D . ( E . ( F ) ) ) . ( G ) ) ) )
( ( A  B ) . ( C . ( ( D . ( E  F ) )  G ) ) )
```

Successive applications of rule 2

```
( ( A  B ) . ( C . ( ( D . ( E  F ) )  G ) ) )
( ( A  B ) . ( C . ( ( D  E  F )  G ) ) )
( ( A  B ) . ( C ( D  E  F )  G ) )
( ( A  B )  C ( D  E  F )  G )
```