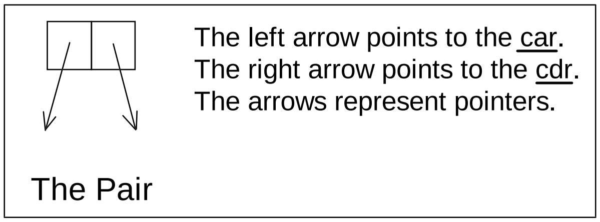
By Joshua Cantrell
jjc@cory.berkeley.edu

# Box & Pointer Diagrams

## What is a box? What is a pointer?

A box in a box & pointer diagram is formed by drawing a square. Two squares next to each other describes the object formed by using the constructor, *cons*. The orientation should be horizontal for clarity. Each square contains a pointer that points to something. The left pointer points to the *car* and the right pointer points to the *cdr*. The whole thing is called a *pair*[1].

A pointer in a box & pointer diagram is formed by drawing an arrow. This arrow should typically be drawn from the center of a square to clearly show that it starts from that box and pointing elsewhere. A simple box with pointers not pointing to anything is shown below:



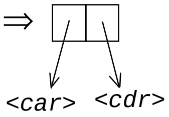
The left arrow points to the car.
The right arrow points to the cdr.
The arrows represent pointers.

The Pair

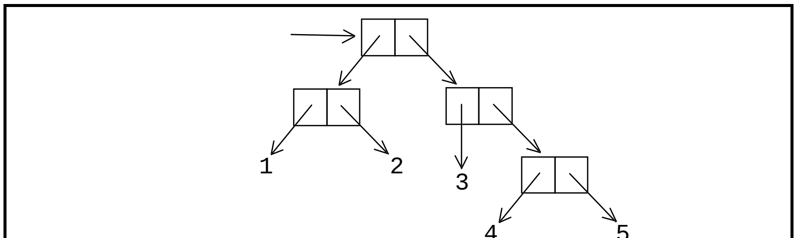
## Drawing Simple Box & Pointer Diagrams

The simplest type of box & pointer diagrams to draw are those created with the use of *cons*. One cons results in the creation of one pair. The first argument is what becomes the *car* and the second argument is what becomes the cdr.

(cons <*car*> <*cdr*>) ⇒



The pointers can point to anything, so they can also point to other pairs. This relationship can be used to form large structures containing large amounts of data.



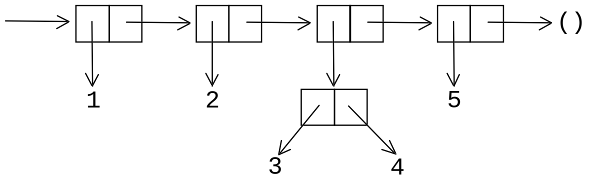
```
(cons (cons 1 2) (cons 3 (cons 4 5)))
```

A common mistake is to think that a pointer points to a specific location in a pair. This is not the case, and pointers can only point *at* a pair, not in a pair. The arrows only need to point at one of the two squares that form a pair.

[1] It's called a pair because it has two pointers (similar to saying you have a pair of shoes means you have two shoes).

## Drawing Box & Pointer Diagrams of a Scheme List

A Scheme List is a structure made up of pairs. Each element in the list is the *car* of a pair and each *cdr* of a pair points to a sub-list. The last pair's cdr points to the empty list which is not a pair.



`(list 1 2 (cons 3 4) 5)`

## Text Representation of Box & Pointer Diagrams

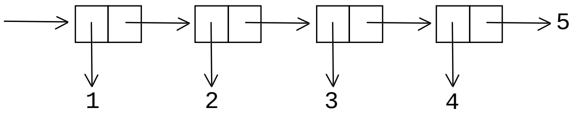
In Scheme, box & pointer diagrams have text representation since most interaction is done through text, not graphics. The way Scheme draws the text is highly related to the way pairs form lists. For example, a list is represented as elements separated by spaces and surrounded by parentheses:

```
(list 1 2 3 4 5) ⇒ (1 2 3 4 5)
```

If a list was built using cons, it would still be represented as a list:

```
(cons 1 (cons 2 (cons 3 (cons 4 (cons 5 '()))))) ⇒ (1 2 3 4 5)
```

If a bunch of pairs don't form a list, then an oddity appears in the way it is displayed. The only structure of pairs that doesn't form a list is one who's last pair's cdr is not an empty list:



`(cons 1 (cons 2 (cons 3 (cons 4 5))))`

```
(cons 1 (cons 2 (cons 3 (cons 4 5)))) ⇒ (1 2 3 4 . 5)
```

A *dot* is placed where a close parenthesis would have been if the list's last pair's cdr had been an empty list, and after this dot, the thing that has replaced the empty list is displayed. A single pair is the same as a list except that it would be the last pair without an empty list, so it's displayed as follows[2]:

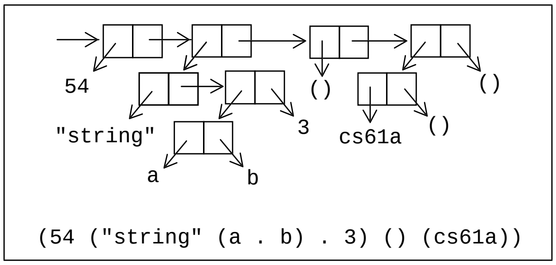
```
(cons <car> <cdr>) ⇒ (<car> . <cdr>)
```

[2] It's sometimes referred to as a *dotted-pair*.

## Drawing a Text Representation Given a Box & Pointer Diagram

The easiest way to draw the text representation of a bunch of pairs and lists is to first draw its box & pointer diagram. After that's been accomplished, it's simply a matter of correctly interpreting the structure of the diagram. The following steps should help you draw the text representation of a box & pointer diagram:

1. If the thing being displayed is a pair, display an open parenthesis, otherwise display the text representation of the thing.
2. Display the text representation of the car of the pair.
3. If the cdr points to another pair:
   a. Display a space to separate the elements.
   b. Go to step 2.
4. If the cdr doesn't point to an empty list:
   a. Display a space followed by a dot followed by a space.
   b. Display the text representation of the cdr of the pair.
5. Display a close parenthesis.
6. You are done.

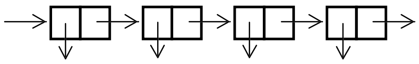


```
(54 ("string" (a . b) . 3) () (cs61a))
```

## Drawing a Box & Pointer Diagram Given a Text Representation

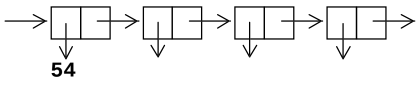
Similar to having steps from going to a text representation from a box & pointer diagram, there are steps to go in the opposite direction:

1. Count the number of elements starting from the open parenthesis until you come to a dot or close parenthesis (anything enclosed within parenthesis after the open parenthesis is considered to be an element).
2. Draw a list of pairs equal in number to the quantity of counted elements.
3. Draw each element (in box & pointer diagram format) as the car of the corresponding pairs.
4. If there is a dot before the close parenthesis:
   a. Draw the element (in box & pointer diagram format) as the cdr of the last pair.
   b. Go to step 6.
5. Draw an empty list as the cdr of the last pair.
6. You are done.



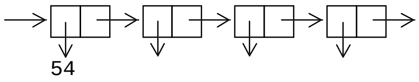
Step1&2: Count and draw 4 elements

`(54 ("string" (a . b) . 3) () (cs61a))`



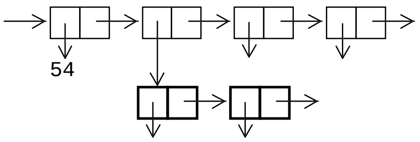
Step3-1: Draw Car of Pair #1

`(54 ("string" (a . b) . 3) () (cs61a))`



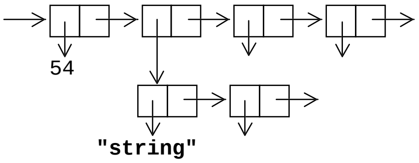
Step3-2: Draw Car of Pair #2

`(54 ("string" (a . b) . 3) () (cs61a))`



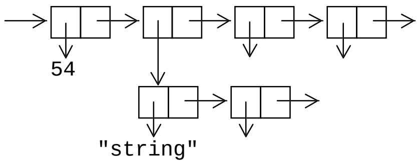
Step1&2: Count and draw 2 elements

`(54 ("string" (a . b) . 3) () (cs61a))`



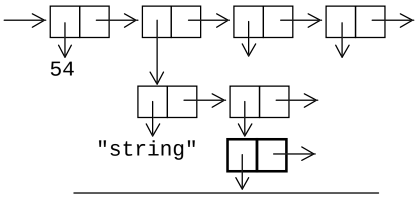
Step3-1: Draw Car of Pair #1

`(54 ("string" (a . b) . 3) () (cs61a))`



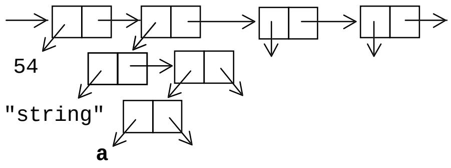
Step3-2: Draw Car of Pair #2

`(54 ("string" (a . b) . 3) () (cs61a))`



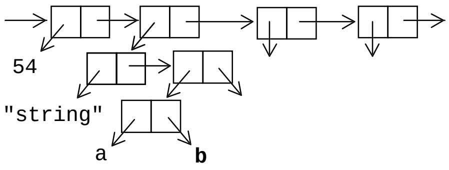
Step1&2: Count and draw 1 element

`(54 ("string" (a . b) . 3) () (cs61a))`



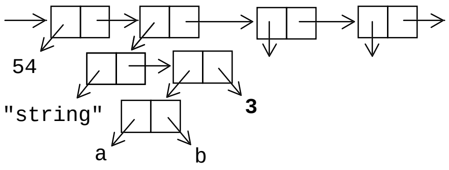
Step3-1: Draw Car of Pair #1

`(54 ("string" (a . b) . 3) () (cs61a))`



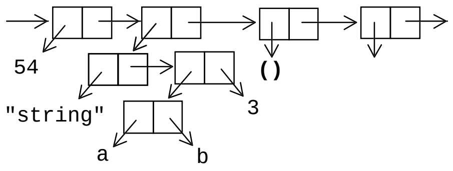
Step4a: Dot found. Draw Cdr of Last Pair.

`(54 ("string" (a . b) . 3) () (cs61a))`



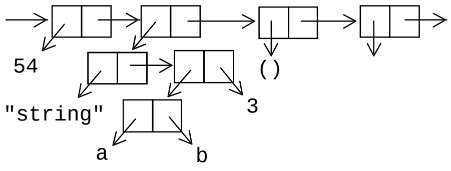
Step4a: Dot found. Draw Cdr of Last Pair.

`(54 ("string" (a . b) . 3) () (cs61a))`



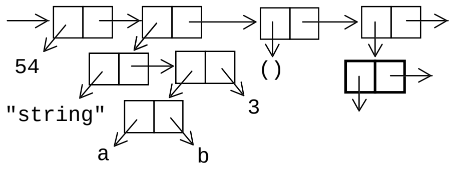
Step3-3: Draw Car of Pair #3

```
(54 ("string" (a . b) . 3) () (cs61a))
```



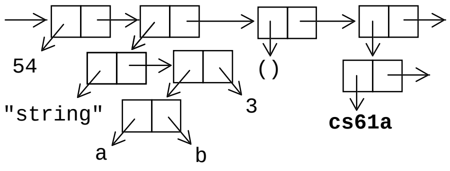
Step3-4: Draw Car of Pair #4

```
(54 ("string" (a . b) . 3) () (cs61a))
```



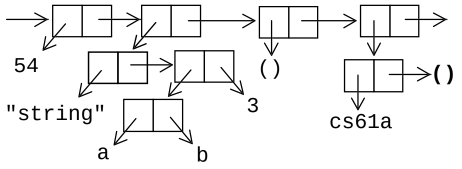
Step1&2: Count and draw 1 element

```
(54 ("string" (a . b) . 3) () (cs61a))
```



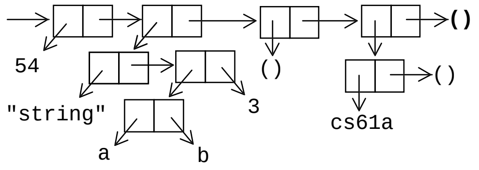
Step3-1: Draw Car of Pair #1

```
(54 ("string" (a . b) . 3) () (cs61a))
```



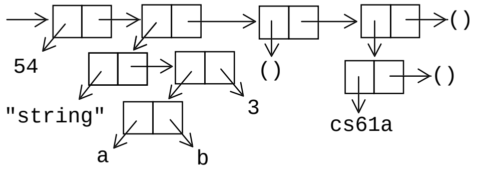
Step4&5: No dot found.  Draw Empty List as Car of last pair.

```
(54 ("string" (a . b) . 3) () (cs61a))
```



Step4&5: No dot found.  Draw Empty List as Car of last pair.

```
(54 ("string" (a . b) . 3) () (cs61a))
```



Step6: You are done!

```
(54 ("strinq" (a . b) . 3) () (cs61a))
```