4. Structures and Patterns in Functional Programming

Lecture 04 -- Structures and Patterns in Functional Programming.pdf

Order of Growth

- Introduction to the Θ notation, big O notation with constraints on both lower and upper ends.
- Growth of Space is measured in number of deferred (delayed for later computation) operations.
- Growth of Time is measured in number of primitive steps

Compound Data (cons)

`(cons <x> <y>) ==> <P>`
Returns a pair `<P>` whose:
car: `<x>`, cdr: `<y>`
Basically a linked list whose end is indicated with an empty list.

Constructor: `(cons <x> <y>) ==> <P> **Accessors:** ` car, cdr `**Predicate:**` pair? -> boolean`

Main practices while working with lists

- Consing up - to build the list,
- Cdring down - to process the elements of the list

Question:

What are the internal workings of this structure? Does cons return a pointer that points to the first element and its pointer which points to the second element?

Answer by ChatGPT:

Yes, your understanding is on the right track. In Lisp-like languages, such as Scheme, the `cons` function is a fundamental operation to create pairs, which are the building blocks of lists. Let's delve deeper into the internal workings:

1. The Pair Structure: When you use the `cons` function, it creates a pair, which is a composite data structure consisting of two parts: the `car` and the `cdr`. The `car` typically holds the first value, and the `cdr` typically holds the second value.
2. Pointers: The pair does not "contain" the two values in the traditional sense. Instead, it contains pointers to these values. If these values are themselves pairs, then the pointers

will point to other pairs, creating a linked structure.

3. **Building Lists with Pairs**: When building lists using `cons`, the convention is for the `car` to contain the current element of the list, and the `cdr` to contain a pointer to the next pair (or element) in the list. This chaining of pairs forms a linked list. The last element in the list will typically have a `cdr` value of `nil` (or its equivalent, like `empty-list`), signaling the end of the list.
4. **Representation**: A visual representation might show the `car` and `cdr` of each pair as two adjacent boxes, with arrows indicating the pointers. In a list structure, the `cdr` box of each pair would have an arrow pointing to the next pair, until the end of the list is reached.
5. **Accessing Elements**: To access the elements of a pair or a list, you'd use the functions `car` and `cdr`. The `car` function retrieves the first element of the pair (or the current element of a list), while the `cdr` function retrieves the second element of the pair (or the rest of the list).
6. **Memory Allocation**: When you use `cons` to create a pair, memory is allocated for that pair. This memory holds the pointers for the `car` and `cdr`. Because lists in Lisp-like languages are built from these pairs, they are essentially linked lists, where each node (pair) contains a value and a reference (pointer) to the next node.

In summary, the `cons` function is used to construct pairs, and by chaining these pairs together, you create lists. The structure consists of pointers to values (or other pairs), making it a dynamic and flexible way to create and manipulate complex data structures. This foundational concept is central to the design and operation of Lisp-like languages.

[Confidence Level: High] [Source: https://mitpress.mit.edu/sites/default/files/sicp/full-text/book/book-Z-H-15.html#%_sec_2.2]