Module 4a: Self-Reference

CPSC 110

Peyton Seigo

2018-09-26

Module 4a: Self-Reference

Learning goals

- Be able to use list mechanisms to construct and destruct lists.
- Be able to identify problem domain information of arbitrary size that should be represented using lists and lists of structures.
- Be able to use the HtDD, HtDF and Data Driven Templates recipes with such data.
- Be able to explain what makes a self-referential data definition well formed and identify whether a particular self-referential data definition is well-formed.
- Be able to design functions that consume and produce lists and lists of structures.
- Be able to predict and identify the correspondence between self-references in a data definition and natural recursions in functions that operate on the data.

Notes

-

Terminology

- **Arbitrary-sized information**: information that we don't know the size of in advance.
  - *A program that can display any number of cows is operating with abitrary-sized information.*

Syntax

The primitive `cons` is a two element constructor that constructs a list:

```
1 (cons x y) -> list?
2   x : any/x
3   y : list?
```

`cons` can be used to produce lists with more than one type of data; but we will not do that (our data definitions do not let us talk about that very well).

Lists have functions that are SIMILAR to `struct` selectors:

- `(first <list>)`: first element in list
- `(rest <list>)`: list with front popped off
  - *Note: `rest` expects a non-empty list*

- `(first (rest L2))`: produces element in `<list>`
  - pops element off the front of `L2`, then gets the first element in the new list
  - `(second <list>)` also exists, but popping and getting the first element as shown above is VERY useful in things like recursion and using accumulators! It's mostly useful because the procedure is generalized.
- `(empty? <list>)`: produce true if argument is the empty list
- `(length <list>)`: evaluates number of items on a list