Halting Measures for Tree-Like Structures

CS 5010 Program Design Paradigms

Lesson 6.6

© Mitchell Wand, 2016
This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

Learning Outcomes

- At the end of this lesson, the student should be able to
  - Explain the definition of a halting measure for mutually-recursive functions
  - Write a halting measure for functions on S-expressions that use the template.

Let's review halting measures for list functions

- Let's look at the template for list data and the definition of a halting measure.
- Then we'll look at the call graph for a list function and see what the halting measure looks like on the call graph.

Review: Template for List data

```
;; list-fn : ListOfX -> ??
(define (list-fn lst)
  (cond
    [(empty? lst) ...]
    [else (... (first lst)
               (list-fn (rest lst)))]))
```

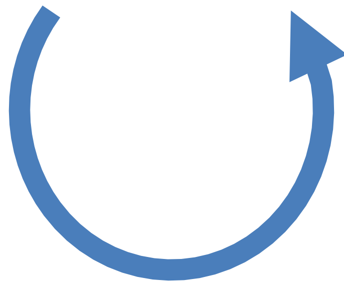
Remember the template for list data

Review: Halting Measure

- Definition: a *halting measure* for a particular function is an integer-valued quantity that can't be less than zero, and which decreases at each recursive call in that function.

Another picture: the call graph

list-fn



>

halting measure decreases

list-fn calls itself. The halting measure (the size of the argument) decreases at each call.

A computation can go around this cycle only finitely many times, because the halting measure is always a non-negative integer.

Now let's do it again for SoS and LoSS

An S-expression of Strings (SoS) is either

-- a String

-- a List of SoS's (LoSS)

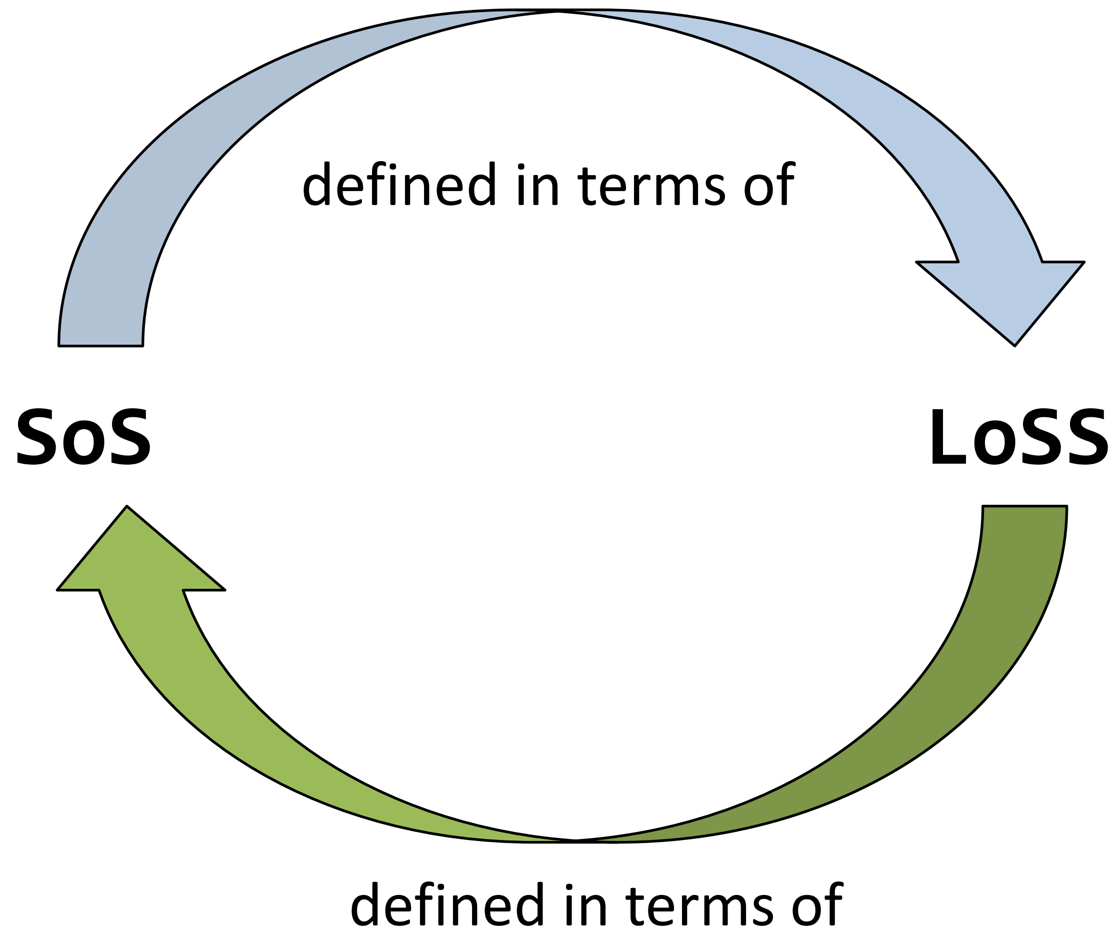
A List of SoS's (LoSS) is either

-- empty

-- (cons SoS LoSS)

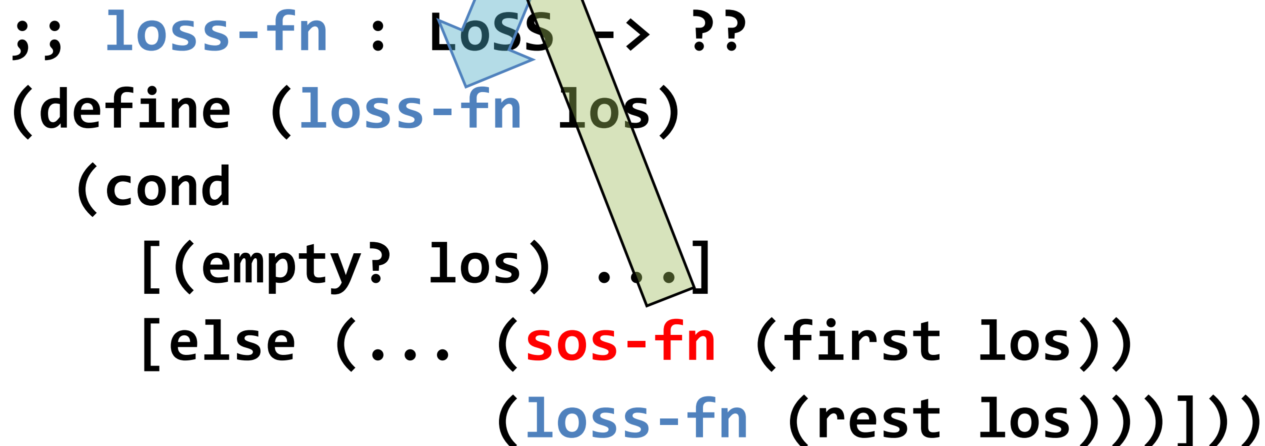
Here's the data definition again

This is mutual recursion

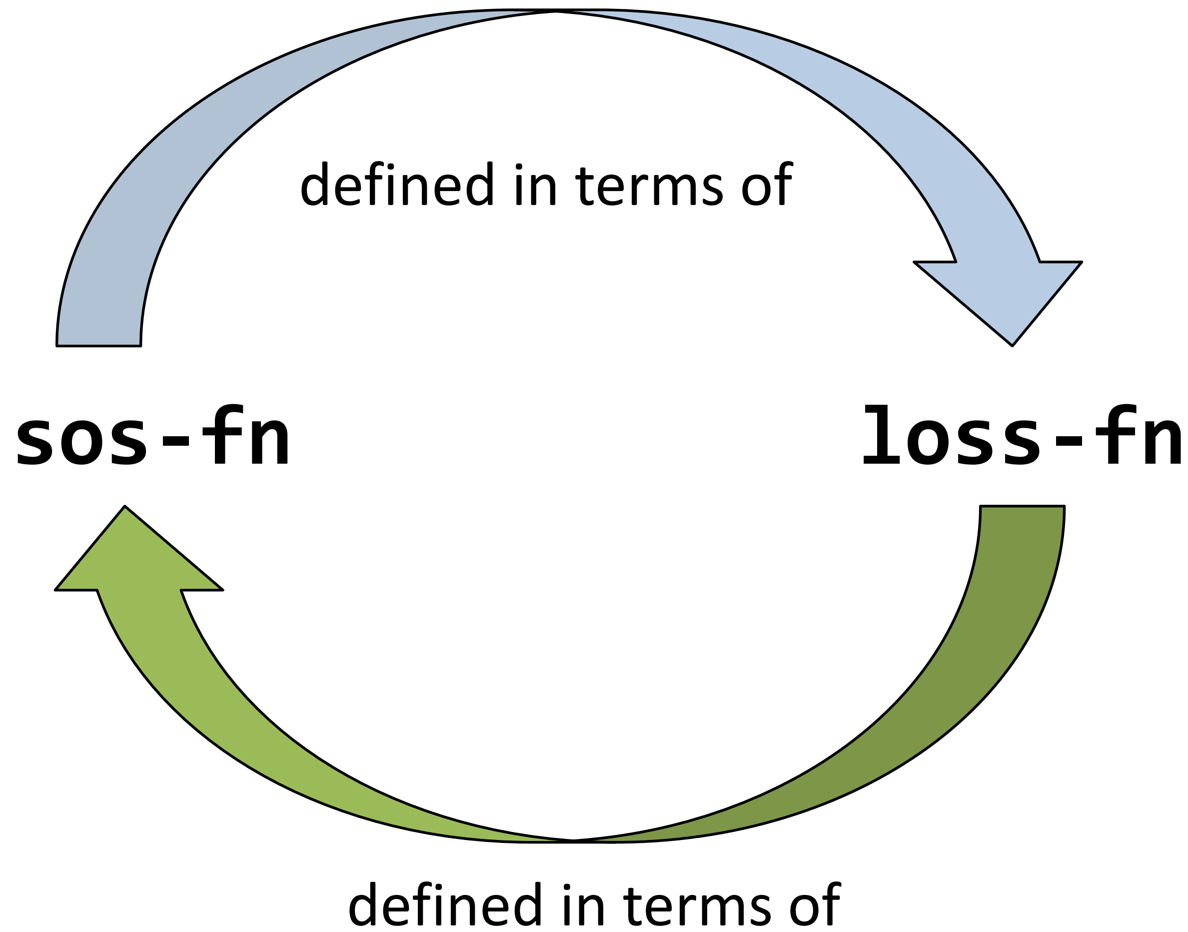


And here's the template

```
;; sos-fn : SoS -> ??
(define (sos-fn s)
  (cond
    [(string? s) ...]
    [else (loss-fn s)]))

;; loss-fn : LoSS -> ??
(define (loss-fn los)
  (cond
    [(empty? los) ...]
    [else (... (sos-fn (first los))
               (loss-fn (rest los)))]))
```



This is mutual recursion

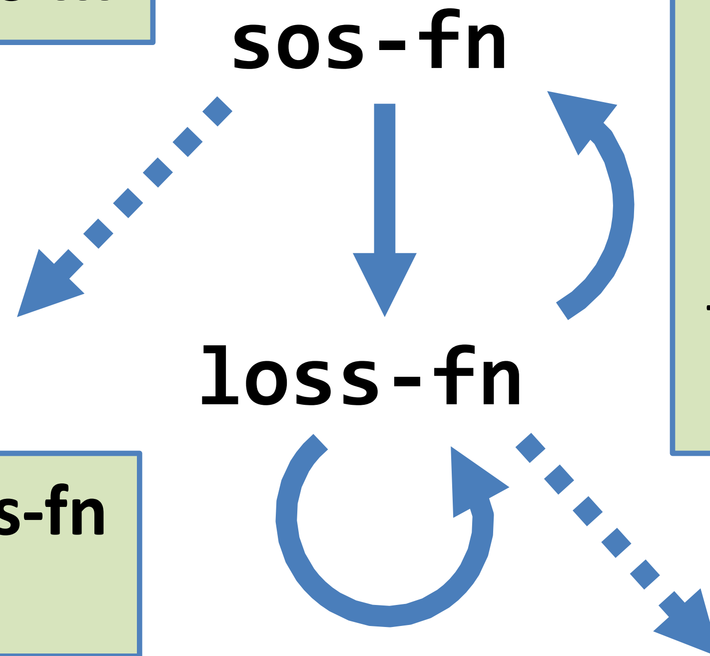


What's a good halting measure for this pair of functions?

- We claim that the size of the SoS or LoSS is a halting measure for this pair of functions.
- What do we mean by size here? Ans: the number of cons cells
- But wait, you say: when sos-fn calls loss-fn, this size of the argument doesn't decrease
- Let's look at this more closely by examining the call graph

Let's draw the call graph in more detail

sos-fn calls **loss-fn**

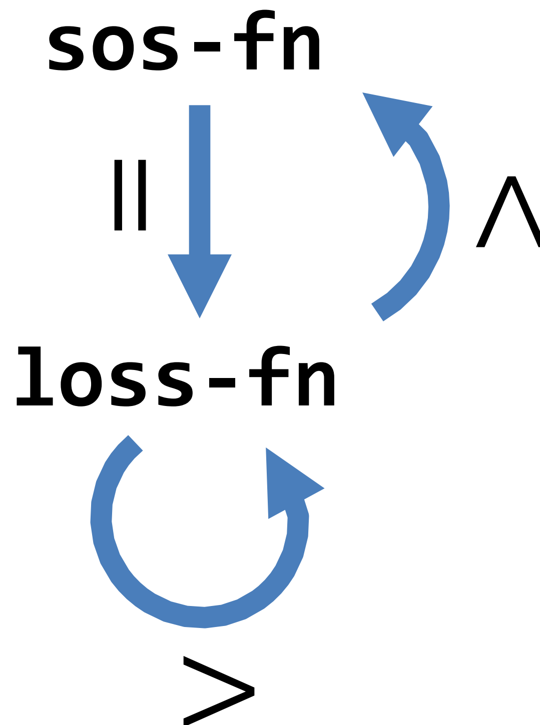


sos-fn and **loss-fn** may call other functions, but none of those functions ever call **sos-fn** or **loss-fn**

loss-fn calls **sos-fn** and **loss-fn**

Where does the halting measure decrease?

The halting measure (the size of the argument) decreases along each arrow labelled with a >, and never increases on any arrow.



So the halting measure decreases around every cycle in this graph. Since the size of the argument is a non-negative integer, a computation can make only finitely many calls in this graph.

Refined Definition of a Halting Measure

- Definition: a *halting measure* for a particular function is an integer-valued quantity that can't be less than zero, and which *decreases around every cycle in the call graph.*
- In general, you can't just look at a single function– you have to trace the call graph.
- For functions that follow the template, the size of the argument is almost always a halting measure.

A more subtle example
Descendant Trees

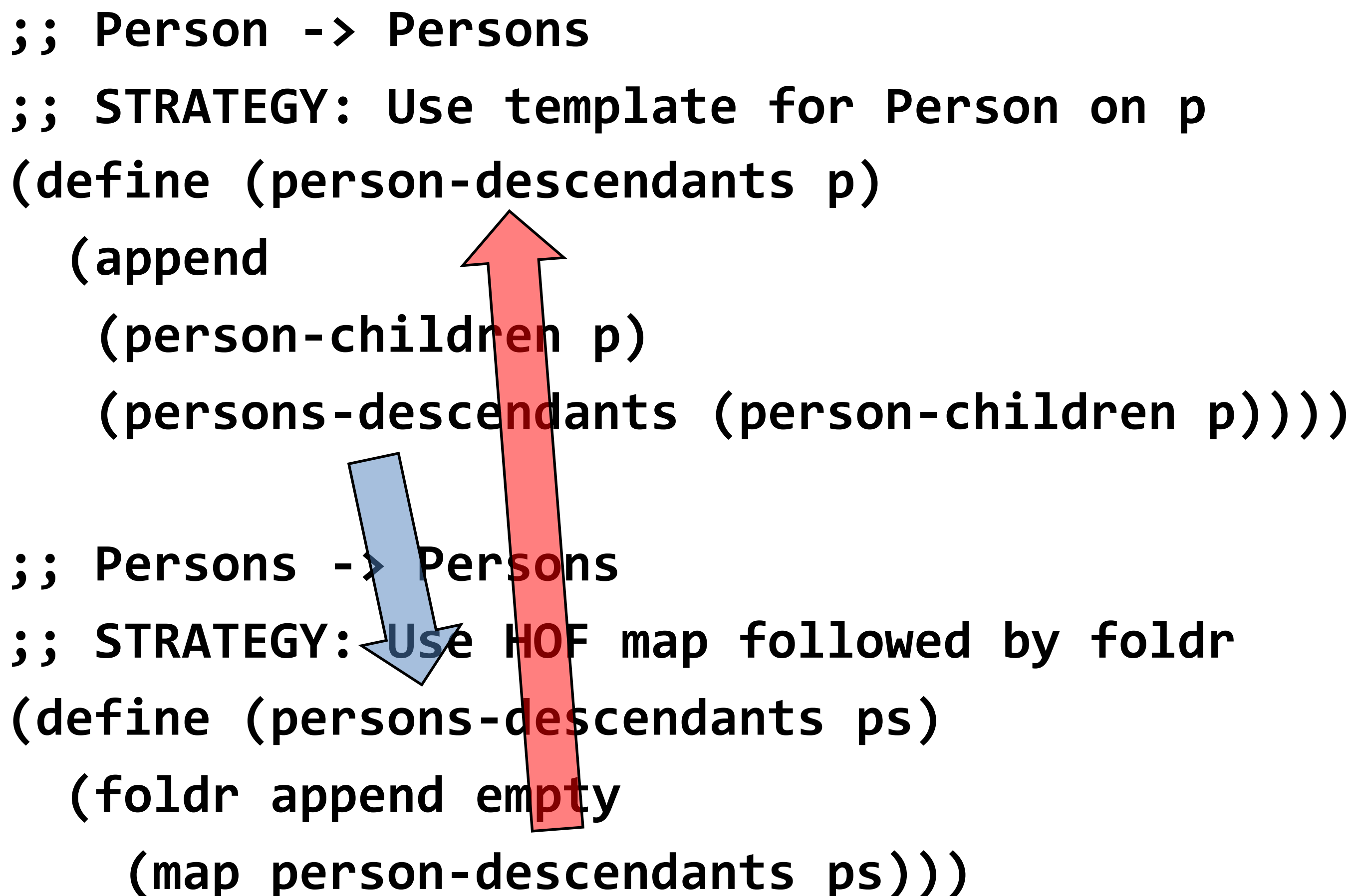
```
(define-struct person (name children))

;; A Person is a
;; (make-person String Persons)

;; A Persons is one of
;; -- empty
;; -- (cons Person Persons)
```

Two *mutually recursive* data definitions

What's a good way to measure the size of one of these?

- Ans: number of nodes in the tree, where a node is either a **make-person** or a **cons**.
- This is the standard way of measuring the size of a structure.

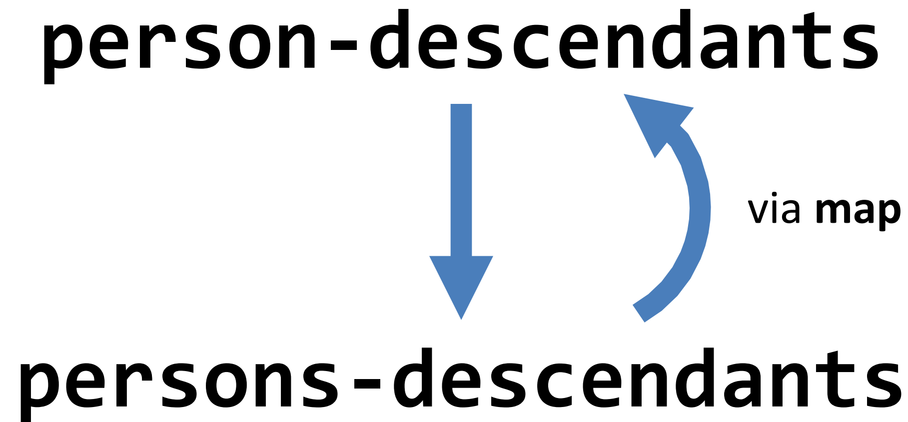
Example of a pair of functions on this data definition



```
;; Person -> Persons
;; STRATEGY: Use template for Person on p
(define (person-descendants p)
  (append
   (person-children p)
   (persons-descendants (person-children p))))

;; Persons -> Persons
;; STRATEGY: Use HOF map followed by foldr
(define (persons-descendants ps)
  (foldr append empty
    (map person-descendants ps)))
```

With HOFs, the finding the call graph may take more care. Here's an example.

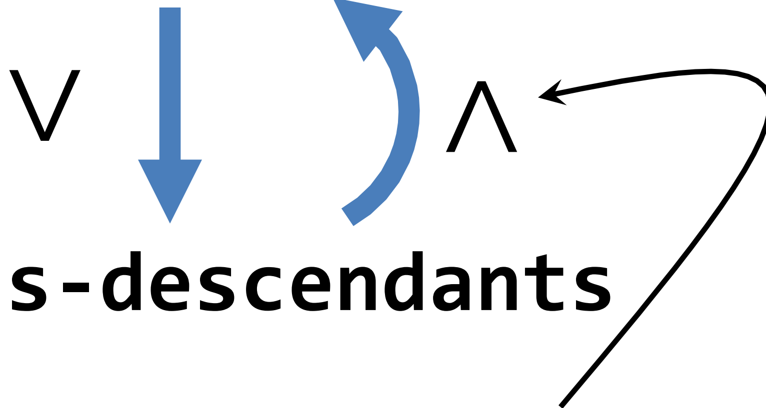
The call graph for this pair of functions



The halting measure decreases on both arrows

person-descendants

(person-children p) is smaller than p



persons-descendants

All we need is for the halting measure to decrease on every cycle, so it would be ok if one of these >'s was an =. Just so long as none of the calls *increases* the halting measure!

map calls **person-descendants** on each element of **ps**; the elements of **ps** are always smaller than **ps**

Summary

- You should now be able to:
  - Explain the definition of a halting measure for mutually-recursive functions
  - Write a halting measure for functions on S-expressions and other mutually-recursive data types that use the template.

Next Steps

- If you have questions about this lesson, ask them in class or on the Discussion Board
- Go on to the next lesson