Mutually-Recursive Data Definitions

CS 5010 Program Design Paradigms

Lesson 6.3

Mutually Recursive Data Definitions

- Sometimes two kinds of data are intertwined
- In this lesson, we'll consider an easy example: alternating lists
- An alternating list is a list whose elements alternate between numbers and strings

Learning Objectives

- At the end of this lesson, the student should be able to
  - recognize information that should be represented as an alternating list
  - write a data definition for an alternating list
  - explain why templates for alternating lists come in pairs

Alternating Lists

- Let's write a data definition for lists whose elements alternate between numbers and strings.

Data Definitions

```
;; A ListOfAlternatingNumbersAndStrings (LANS) is one of:
;; -- empty
;; -- (cons Number LASN)

;; A ListOfAlternatingStringsAndNumbers (LASN) is one of:
;; -- empty
;; -- (cons String LANS)
```

A **LANS** is a list of alternating numbers and strings, starting with a number. A **LASN** is a list of alternating numbers and strings, starting with a string. Either can be empty. Note that the rest of a non-empty **LANS** is a **LASN**, and vice-versa.

Examples

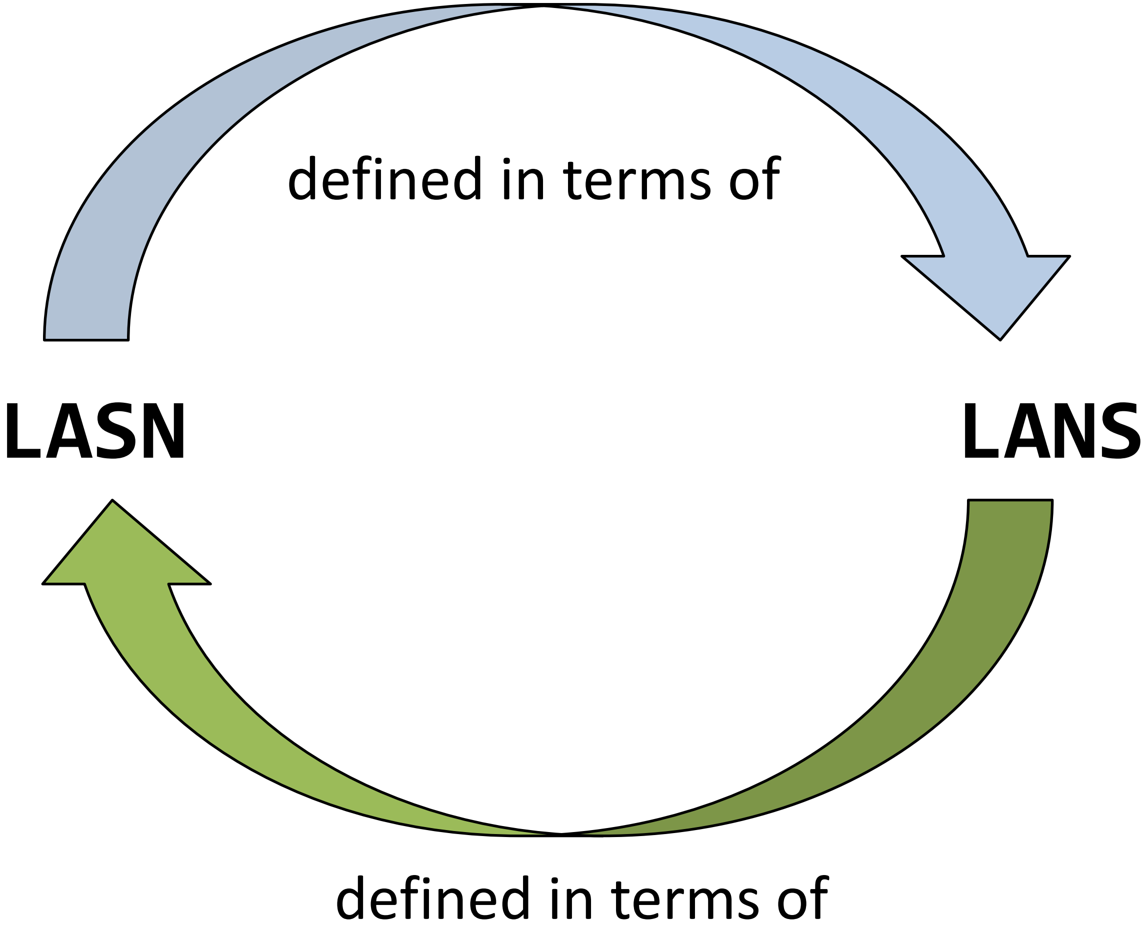
```
                                          empty     is a LASN
                                 (cons 11 empty)    is a LANS
                     (cons "foo" (cons 11 empty))   is a LASN
            (cons 23 (cons "foo" (cons 11 empty)))  is a LANS
(cons "bar" (cons 23 (cons "foo" (cons 11 empty)))) is a LASN
```

These data definitions are *mutually recursive*

```
;; A ListOfAlternatingNumbersAndStrings
   (LANS) is one of:
;; -- empty
;; -- (cons Number LASN)

;; A ListOfAlternatingStringsAndNumbers
   (LASN) is one of:
;; -- empty
;; -- (cons String LANS)
```

The definition of a **LANS** depends on **LASN**, and the definition of a **LASN** depends on **LANS**.

This is mutual recursion



defined in terms of

LASN

LANS

defined in terms of

The template recipe

Question	Answer
Does the data definition distinguish among different subclasses of data?	Your template needs as many cond clauses as subclasses that the data definition distinguishes.
How do the subclasses differ from each other?	Use the differences to formulate a condition per clause.
Do any of the clauses deal with structured values?	If so, add appropriate selector expressions to the clause.
Does the data definition use self-references?	Formulate ``natural recursions'' for the template to represent the self-references of the data definition.
Do any of the fields contain compound or mixed data?	If the value of a field is a foo, add a call to a foo-fn to use it.

The template recipe doesn't need to change

Templates come in pairs

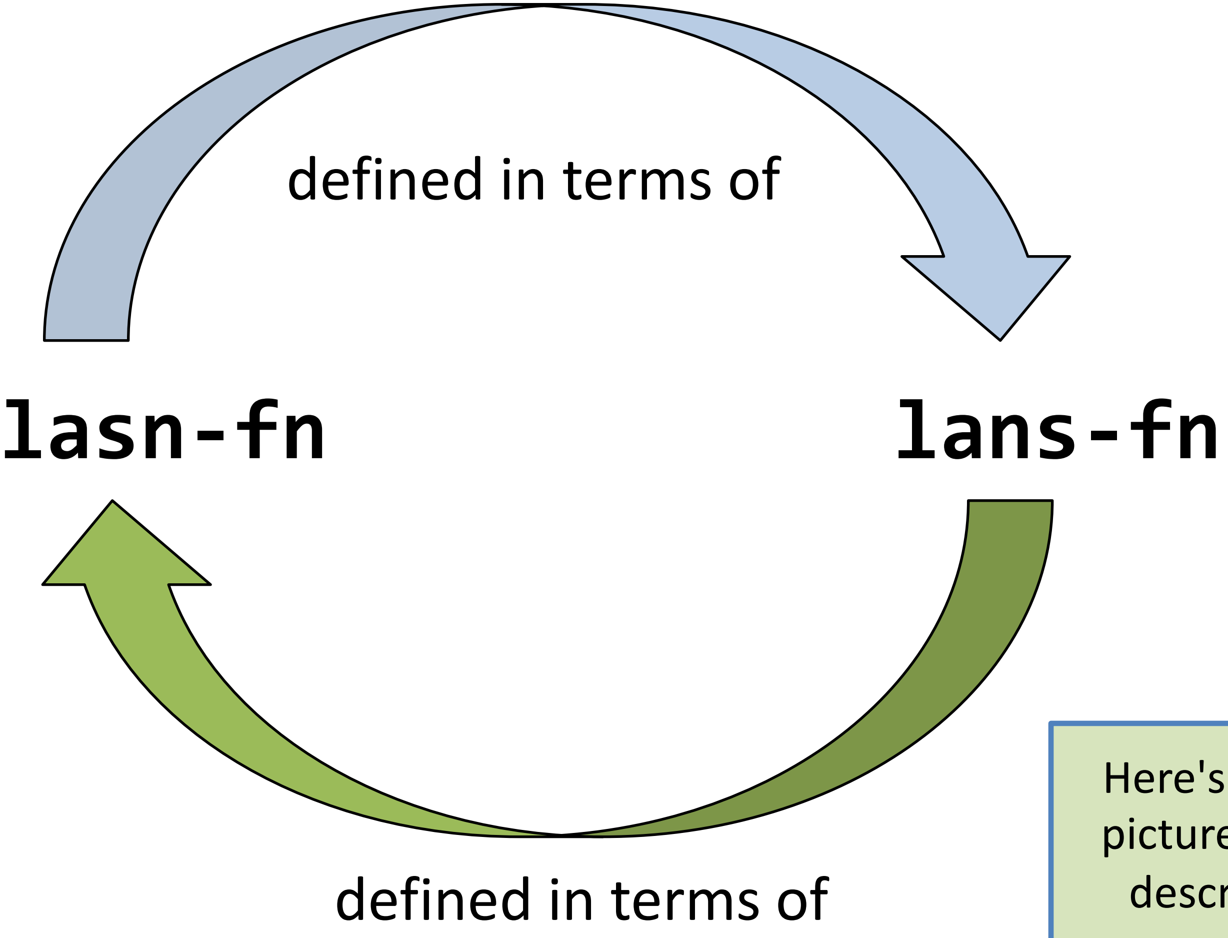
```
;; lans-fn : LANS -> ??
;; (define (lans-fn lans)
;;   (cond
;;     [(empty? lans) ...]
;;     [else (...
;;             (first lans)
;;             (lasn-fn (rest lans)))]))

;; lasn-fn : LASN -> ??
;; (define (lasn-fn lasn)
;;   (cond
;;     [(empty? lasn) ...]
;;     [else (...
;;             (first lasn)
;;             (lans-fn (rest lasn)))]))
```

Here are the templates for **LANS** and **LASN**. Observe the recursive calls, in red.

Templates are mutually recursive

```
;; lans-fn : LANS -> ??
;; (define (lans-fn lans)
;;   (cond
;;     [(empty? lans) ...]
;;     [else (...
;;             (first lans)
;;             (lasn-fn (rest lans)))]))

;; ;; lasn-fn : LASN -> ??
;; (define (lasn-fn lasn)
;;   (cond
;;     [(empty? lasn) ...]
;;     [else (...
;;             (first lasn)
;;             (lans-fn (rest lasn)))]))
```

This is mutual recursion



defined in terms of

`lasn-fn`

`lans-fn`

defined in terms of

Here's that same picture, this time describing the recursive calls in the template.

The template questions

```
;; lans-fn : LANS -> ??
;; (define (lans-fn lans)
;;   (cond
;;     [(empty? lans) ...]
;;     [else (...
;;             (first lans)
;;             (lasn-fn (rest lans)))]))

;; ;; lasn-fn : LASN -> ??
;; (define (lasn-fn lasn)
;;   (cond
;;     [(empty? lasn) ...]
;;     [else (...
;;             (first lasn)
;;             (lans-fn (rest lasn)))]))
```

What is the answer for the empty LANS?

If you knew the answer for the LASN inside the LANS, what would the answer be for the whole LANS?

What is the answer for the empty LASN?

If you knew the answer for the LANS inside the LASN, what would the answer be for the whole LASN?

As usual, we have one question for each blank in the template.

One function, one task

- Each function deals with exactly one data definition.
- So functions will come in pairs
- Write contracts and purpose statements together, **or**
- Write one, and the other one will appear as a wishlist function

Example

```
lans-sum : LANS -> Number
Returns the sum of all the numbers
  in the given Lans


lasn-sum : LASN -> Number
Returns the sum of all the numbers
  in the given Lasn
```

Here's an example of a pair of functions that should go together.

Examples

```
(lans-sum
 (cons 23
  (cons "foo"
   (cons 11 empty))))  = 34

(lasn-sum
  (cons "bar"
   (cons 23
    (cons "foo"
     (cons 11 empty))))) = 34
```

And here are some examples for our two functions. Observe that **lans-sum** is applied to a **LANS**, and **lasn-sum** is applied to a **LASN**.

Strategy and Function Definitions

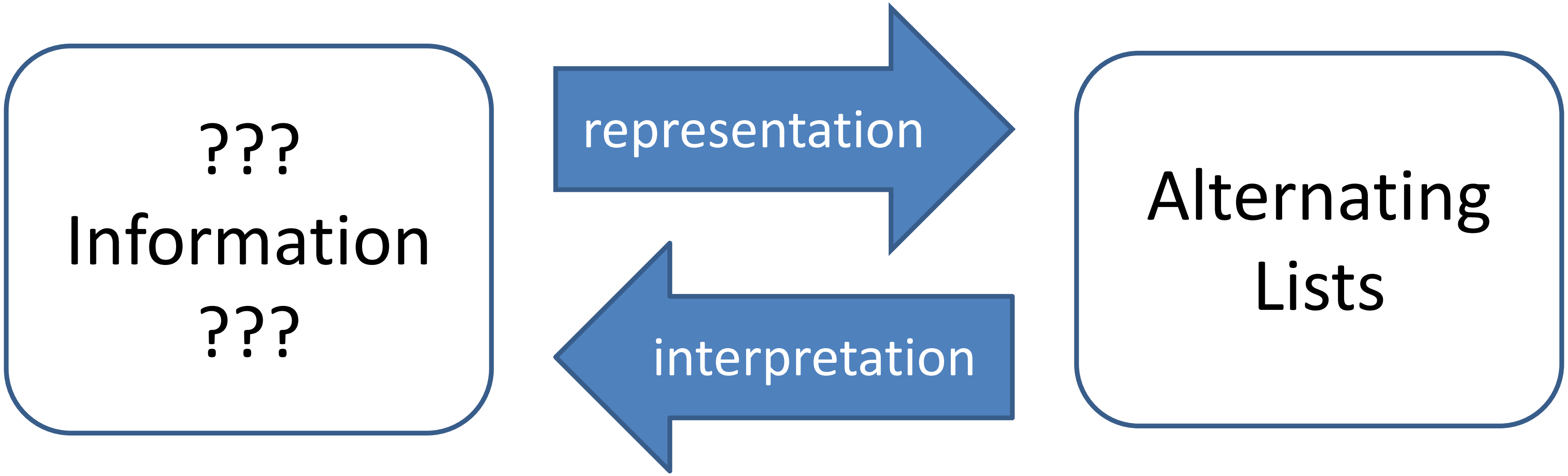
```
;; strategy: Use template for LANS and LASN
;; lans-sum : LANS -> Number
(define (lans-sum lans)
  (cond
    [(empty? lans) 0]
    [else (+
            (first lans)
            (lasn-sum (rest lans)))]))

;; lasn-sum : LASN -> Number
(define (lasn-sum lasn)
  (cond
    [(empty? lasn) 0]
    [else (lans-sum (rest lasn))]))
```

We apply the template by filling in each of the four blanks with the answer to the corresponding template question.

Halting Measure

- The two functions in the template are mutually recursive, so we need a single halting measure that will work for both functions.
- Each of the functions recurs on **(rest lst)**, so the length of the list works as a halting measure.

What are alternating lists good for?



???
Information
???

representation

interpretation

Alternating
Lists

Answer: Not much! Don't use them!

But they make a good example of mutually-recursive data definitions

Summary

- You should now be able to:
  - recognize information that should be represented as an alternating list
  - write a data definition for an alternating list
  - explain why templates for alternating lists come in pairs

Next Steps

- Study the file 06-3-lasns.rkt
- If you have questions about this lesson, ask them on the Discussion Board
- Do Guided Practice 6.3
- Go on to the next lesson