More Recursive Data Types

CS 5010 Program Design Paradigms

Lesson 4.4

Introduction

- There are other recursive data types besides lists
- Programming with these is no different:
  - write down the data definition, including interpretation and template
  - Follow the Recipe!

Learning Objectives

- At the end of this lesson you should be able to:
  - Explain what makes a recursive data definition sensible
  - Explain how the Natural Numbers definition works
  - write simple programs using the Natural Numbers template

What's interesting about lists?

- Our Lists data definitions are the first "interesting" data definitions:
- They are mixed data
- They are recursive

Question: Why did we say "data definitions" instead of data definition?"
Answer: Remember that we have a data definition **ListOfX** for each **X**

What makes a good definition for mixed data?

- The alternatives are *mutually-exclusive*
- It is easy to tell the alternatives apart
- There is one and only one way of building any value.

Example of a bad data definition

A Blue number is one of

- **an integer that is a multiple of two**
- **an integer that is a multiple of three**

These categories are not mutually exclusive

Example of a bad data definition

A Green number is one of

- **an integer that is a product of exactly two prime numbers**
- **any other integer**

These categories are mutually exclusive, but it is complicated to distinguish them

Example of a bad data definition

A Purple number is one of

- **the number 1**
- **a number of the form (+ n1 n2)**

Just knowing the value of a purple number, like **56**, doesn't tell you how it was constructed as **(+ n1 n2)** . There are many choices of **n1** and **n2** that would build **56**.

The Natural Numbers

- The natural numbers are the counting numbers:

$$0, 1, 2, 3, 4, \ldots$$

- This is just another name for the non-negative integers

A data definition for the natural numbers

```
;; A Natural Number (Nat) is one of
;; -- 0
;; -- (add1 Nat)
```

Here we use the Racket function **add1**, which adds 1 to its argument. We'll also use **sub1**, which subtracts 1 from its argument.

Examples

```
0
1  (because 1 = (add1 0))
2  (because 2 = (add1 1))
3  (because 3 = (add1 2))
4  (because 4 = (add1 3))
Etc...
```

Is this a good data definition?

- Are the alternatives *mutually exclusive?*

Answer: yes

- Is it easy to tell the alternatives apart?

Answer: yes, with the predicate **zero?**

Is this a good data definition? (2)

- Is there one and only one way of building any value?
- Answer: Yes. There's only one way to build the number n :

$$\overbrace{\texttt{(add1 (add1 (add1 (add1 ... 0))))}}^{n\text{ times}}$$

Is this a good data definition? (3)

- If we have a natural number **x** of the form **(add1 y)**, there's only one possible value of **y**. Can we find it?
- Answer: sure. If **x** = **(add1 y)**, then **y** = **(sub1 x)**.
- So **add1** is like a constructor, and **sub1** is like an observer.
- This leads us to a template:

Template

```
;; nat-fn : Nat -> ??
(define (nat-fn n)
 (cond
  [(zero? n) ...]
  [else (... n (nat-fn (sub1 n)))]))
```

double

```
;; double : Nat -> Nat
;; strategy: use template for
;;   Nat on n
(define (double n)
  (cond
    [(zero? n) 0]
    [else (+ 2 (double (sub1 n)))]))
```

sum

```
;; sum : Nat Nat -> Nat
;; strategy: use template for
;;   Nat on x
(define (sum x y)
 (cond
   [(zero? x) y]
   [else (add1 (sum (sub1 x) y))]))
```

Example

```
(sum 3 2)
= (add1 (sum 2 2))
= (add1 (add1 (sum 1 2)))
= (add1 (add1 (add1 (sum 0 2))))
= (add1 (add1 (add1 2)))
= 5
```

product

```
;; prod : Nat Nat -> Nat
;; strategy: use template for
;; Nat on y
(define (prod x y)
  (cond
    [(zero? y) 0]
    [else
      (sum x (prod x (sub1 y)))]))
```

Example

```
(prod 2 3)
= (sum 2 (prod 2 2))
= (sum 2 (sum 2 (prod 2 1)))
= (sum 2 (sum 2 (sum 2 (prod 2 0))))
= (+ 2 (+ 2 (+ 2 0)))
= 6
```

Summary

- At the end of this lesson you should be able to:
  - write down the definition for non-negative integers as a data type
  - use the template to write simple functions on the non-negative integers and other simple recursive data types.
- The Guided Practices will give you some exercise in doing this.

Next Steps

- Study 04-3-nats.rkt in the Examples file
- If you have questions about this lesson, ask them on the Discussion Board
- Do Guided Practice 4.4
- Go on to the next lesson