Case analysis

October 2, 2018

What is case analysis?

Breaking a problem into different situations:

- What do I want to do if the input is a number?
- What do I want to do if the list is empty?
- What do I want to do if the test evaluates to true?

Recursion is case analysis

- Base case
- Inductive case

Is-sorted function

The is-sorted function from Lab 1 had three cases:

- The list contained strings
- The list contained numbers
- The list contained a mix of numbers and strings (or other datatypes)

We used if/cond to check these conditions, but there is also a special case-matching language feature: match

Match

```
> (match 5
      (5    “five”)                ; Check if x is 5
      (10   “ten”)                 ; Check if x is 10
      (20    “twenty”))            ; Check if x is 20

“five”
```

Special match syntax: (? exp pat)

(? *exp pattern*) is a special feature of match.
It checks whether *exp* applied to *pattern* is true.

This is useful for type-checking, since *pattern* refers to the value of the matched item, not its type.

Special match syntax: _

_ is the match equivalent of else in a conditional: it matches any expression.

You should only use _ in your last case, since otherwise, none of your other cases will be evaluated.

Special match syntax: …

You can omit named sub-expressions in a case using …

Ellipsis acts like the Kleene star (*) in regular expressions.

```
 (match lst
    ((list 1)  “length 1”)
    ((list x … 10) “length 10”))
```

Exercise: check for duplicates

Write a function that takes a list of strings and checks whether the first item in the string ever re-occurs:

```
> (dups? '("cat" "is" "cat"))
#t
> (dups? '("cat" "says" "meow"))
#f
```

Exercise: generic add

Write a generic addition function using match:

If given a list of strings, your function should join them together into a single string.

If given a list of numbers, your function should sum them together.

If given any other kind of list, your function should return void.

Returning functions

The right-hand-side of match cases can return any kind of Racket expression, including functions.

```
(match x
     (0 +)
     (1 *))
```