Principles of Programming, Fall 2011
Practice 4
Programming with interface, Recursive Programming

Programming Research Lab,@SNU
Seungjung Lee, Youngseok Lee

Oct 14, 2011

The purpose of this practice:

- Tagging type into the data and using it
- Programming inner implementation of the data
- Programming inner implementation in various ways
- Programming only using interface even if there are several implementation
- Practicing to make a recursive program

1. Complex Number (SICP 2.4.1)

   Let's make complex number type. Complex number can be represented as two ways. One is (x, y) rectangular coordinate and another one is (r, θ) polar representation .

   Both representation has same shape of data. The implementation is more easier when you put a tag in the data to figure out which representation the data use. Putting `'rect`, `'polar` tag with your data is one of the example.

   Let's make a complex number type which uses rectangular representation. A function `c-rect-make` takes two numbers: real part and imaginary part.

`is-c-rect?`	:	$\alpha \rightarrow bool$
`c-rect-make`	:	$number \times number \rightarrow c\text{-}rect$
`c-rect-real`	:	$c\text{-}rect \rightarrow number$
`c-rect-imaginary`	:	$c\text{-}rect \rightarrow number$

Let's make a complex number type which uses polar representation. A function `c-polar-make` takes two numbers: angle and radius.

`is-c-polar?`	:	$\alpha \to bool$
`c-polar-make`	:	$number \times number \to c\text{-}polar$
`c-polar-angle`	:	$c\text{-}polar \to number$
`c-polar-radius`	:	$c\text{-}polar \to number$

Now, we can make general functions that takes complex type as any input. The function doesn't depend on inner implementation. These functions are must be only made by above 8 functions.

`c-real`	:	$complex \to number$
`c-imaginary`	:	$complex \to number$
`c-angle`	:	$complex \to number$
`c-radius`	:	$complex \to number$
`c-add`	:	$complex \times complex \to complex$
`c-mul`	:	$complex \times complex \to complex$
`c-conjugate`	:	$complext \to complex$

As a default, result of these functions are the complex number type of rectangular representation.

2. (SICP ex. 2.42) The "eight-queens puzzle" asks how to place eight queens on a chessboard so that no queen is in check from any other (i.e., no two queens are in the same row, column, or diagonal). One possible solution is shown in figure 1. One way to solve the puzzle is to work across the board, placing a queen in each column. Once we have placed $k-1$ queens, we must place the kth queen in a position where it does not check any of the queens already on the board. We can formulate this approach recursively: Assume that we have already generated the sequence of all possible ways to place $k-1$ queens in the first $k-1$ columns of the board. For each of these ways, generate an extended set of positions by placing a queen in each row of the kth column. Now filter these, keeping only the positions for which the queen in the kth column is safe with respect to the other queens. This produces the sequence of all ways to place k queens in the first k columns. By continuing this process, we will produce not only one solution, but all solutions to the puzzle.

   We implement this solution as a procedure `queens`, which returns a sequence of all solutions to the problem of placing n queens on an $n \times n$ chessboard. `queens` has an internal procedure `queen-cols` that returns the sequence of all ways to place queens in the first k columns of the board.

```
(define (queens bs)
  (define (queen-cols k)
    (if (< k 0)
        (list empty-b)
```

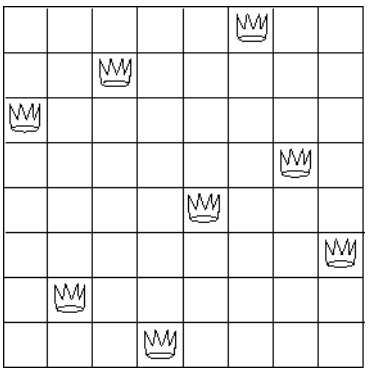

Figure 1: a solution to the eight-queens puzzle

```
        (filter
         (lambda (p) (safe? p))
         (accumulate append
                     null
                     (map <??>
                          (queen-cols (- k 1)))))))
  (queen-cols (- bs 1)))
```

In this procedure `empty-b` represents an empty set of positions. The procedure `safe?` determines for a set of positions, whether the queen in the kth column is safe with respect to the others. (Note that we need only check whether the new queen is safe – the other queens are already guaranteed safe with respect to each other.) Fill in the missing expressions in the above definition.