Lecture 23: OCaml Clarifications
10:00 AM, Oct 29, 2018

Contents

1 Tuples

1.1 Write a tuple type: ................................................................. 1

1.2 Write an expression whose value has that type: ....................... 1

1.3 Write a pattern that matches a tuple: ..................................... 1

1.4 Write an expression whose value is a (nonrecursive) procedure that takes a pair of
ints and returns true if those ints have the same remainder when divided by 7: ... 1

1.5 Write a definition that binds the identifier equivalent `mod` 7 to the above procedure: 2

2 Lists

2.1 Write a list type: ................................................................. 2

2.2 Write an expression whose value has that type: ....................... 2

2.3 Write a pattern that matches a list: ....................................... 3

2.4 Write an expression whose value is a (nonrecursive) procedure that takes a list and
and returns the second element of that list: .................................. 3

2.5 Write a definition that binds the identifier `get_second` to the above procedure: ... 3

2.6 Rewrite the definition so that the type of the procedure assigned to the identifier is
such that the list elements are required to be strings: ....................... 3

1 Tuples

1.1 Write a tuple type:

Potential answers:

- int * string
- char * char list
- int -> int, bool

1.2 Write an expression whose value has that type:

Corresponding to our potential answers from above:

- (8+1, "hi")
Potential answers: (false, [true;false;true]), (1, [1;2;3]), ([1;2;3], [[1;2];[3;4]])

1.3 Write a pattern that matches a tuple:
- \(x, s\)
- \((9, _)\)
- \((_, _)\)

1.4 Write an expression whose value is a (nonrecursive) procedure that takes a pair of ints and returns true if those ints have the same remainder when divided by 7:

Our first draft looked something like this:

```ocaml
function (a,b) ->
    match a, b with
    x, y -> if x mod 7 = y mod 7
            then true
            else false
```

How can we improve this function? For starters, we do not need a match expression within the function body because \(a\) and \(b\) are already bound to the input arguments. All the match expression accomplishes is “renaming” these input arguments to \(x\) and \(y\). Further, we should always try to avoid having literal booleans in our procedures. Instead, we can just return the output of the equals statement. With all of this in mind, here is our revised function:

```ocaml
function (a,b) ->
    a mod 7 = b mod 7
```

**Question:** What would the type annotation for this function look like?

**Answer:** `int * int -> bool`

1.5 Write a definition that binds the identifier equivalent_mod_7 to the above procedure:

```ocaml
let equivalent_mod_7 : int * int -> bool =
    function (a,b) -> a mod 7 = b mod 7
```
2 Lists

2.1 Write a list type:

Potential answers:

- int list
- bool list
- string list list
- 'a list list

2.2 Write an expression whose value has that type:

Corresponding to our potential answers from above:

- 1::[4;3*2;5+1]
- [true; false; true]
- [["a"; "b"]; ["Hello"; "students"]]
- [[]]

2.3 Write a pattern that matches a list:

- [x; (y:string)] matches a string list of size 2
- hd::tl matches a list of any type
- [] matches the empty list
- [a] matches a list with one element

2.4 Write an expression whose value is a (nonrecursive) procedure that takes a list and and returns the second element of that list:

```ocaml
function _::_:b::_ -> b
```

You also could create something like this:

```ocaml
function a::b::rest -> b
```

However, since we never use a or rest, it is better to just leave those as wildcards as we did in the first example.

The minimum size list that this function can take in is 2. If we try to run this function on a
list with fewer than two elements, we will get a match failure exception. If the list has exactly two elements, then rest will represent the empty list.

*Question:* What would the type annotation for this function look like?

*Answer:* 'a list -> 'a

2.5 Write a definition that binds the identifier `get_second` to the above procedure:

```ocaml
let get_second : 'a list -> 'a =
  function _::second::_ -> second
```

2.6 Rewrite the definition so that the type of the procedure assigned to the identifier is such that the list elements are required to be strings:

```ocaml
let get_second : string list -> string =
  function _::second::_ -> second
```

Please let us know if you find any mistakes, inconsistencies, or confusing language in this or any other CS 17 document by filling out the anonymous feedback form: [http://cs.brown.edu/courses/csci0170/feedback](http://cs.brown.edu/courses/csci0170/feedback).