SOFTWARE VERIFICATION AND COMPUTER PROOF (lesson 1, part 2)

Enrico Tassi – Inria Sophia-Antipolis

Functional programming

1. Programs are data
2. Types provide guidance for building and destructing data

Programs are data

- There is no “return” keyword: expression = statement
- The program that always returns 4 is:

  4
- This program also returns (or better computes to) 4:

  if (false || true) then 2 + 2 else 7

  if true then 2 + 2 else 7

  2 + 2

  4
- This program also computes to 4:

  2 + (if leb 7 2 then 4 else 2)

  2 + (if false then 4 else 2)

  2 + 2

  4

Types: build and destruct

type	build	destruct
bool	true : bool false : bool	match b with | true => ... | false => ... end
nat	O : nat S : nat → nat	match n with | O => ... | S n1 => … use n1 ... end
list nat	nil : list nat cons : nat → list nat → list nat	match l with | nil => … | cons hd tail =>.. use hd and tl … end

Types: build (example)

```
Definition a_natural_number : nat :=
  O          (* ok *)
  S          (* no, S : nat → nat *)
  S O        (* ok *)
  S (S O)    (* ok *)

Definition another_natural_number : nat := S a_natural_number.
```

Types: destruct (example 1)

Execution of a “match … with … end”:

match (S (S O)) with
| O => …
| S n => …
end.

1. the term under exam (S (S O)) is compare with every branch

1.1 first branch:

O => … this branch is not taken …
S (S O)

1.2 second branch. Remark: n is a name that will be *bound*.

S n => …. the branch is taken, and n takes (S O)
S (S O)

Types: destruct (example 2)

```
match (S (S O)) with
| O => ...
| S n => ... what is the value of n here? ....
end.
```

Let's run this program:

```
  match (2 + 1) with O => O | S n1 =>  n1 end
  match 3 with O => O | S n1 =>  n1 end
  match S (S (S O)) with O => O | S n1 => n1 end
  S (S O)
```

Types: build and destruct

type	build	destruct
option nat	None : option nat Some : nat → option nat	match o with | None => ... | Some n => … use n ... end
prod A B (A * B)	pair : A → B → (A * B) (a, b) if (a: A) and (b : B)	match p with | pair a b => … use a and b ... end

Types: build and destruct (example)

```
match Some 3 with
| None => ...
| Some n => ... what is the value of n here? ....
end.

Definition option_add1 (on : option nat) : option nat :=
  match on with
  | None => None
  | Some n =>  Some (n + 1)
  end.
```

Last example

From the exercises of yesterday:

3 + 5 + (slnat (3 :: 4 :: nil))

3 + 5 + (3 + (slnat (4 :: nil)))

8 + (3 + 4 + (slnat nil))

8 + (3 + 4 + O)

8 + 7

15

```
Fixpoint slnat (l : list nat) :=
  match l with
  | nil => O
  | x :: xs => x + (slnat xs)
  end.
```

A few more exercises.

- define a function “map” that takes function (f : nat → nat) and a list (l : list nat). It returns a list that is obtained by applying f to each element of l (preserving the order).

E.g.

(map (fun x : nat => x * 2) (1 :: 2 :: 3 :: nil))

(2 :: 4 :: 6 :: nil)

- define a function “append” that takes two lists and returns a list containing all the elements of the first list followed by the elements of the second list.

E.g.

(append (1 :: 2 :: nil) (3 :: 4 :: nil))

(1 :: 2 :: 3 :: 4 :: nil)