Recursion

Ziyan Maraikar

June 29, 2014

Table of Contents

The syntax of if

`if` e_b `then` e_T `else` e_F

The result of an *if expression*[1] is the value of e_T when the boolean condition e_b is `true` or the value of e_F otherwise.

```
let sign n =
if n>0 then 1
else if n=0 then 0
else -1
```

Remember that equality is written `=`. Don't use the `==` operator, which has a different meaning.

[1] unlike the C if statement

IF SEMANTICS

`if` `true` `then` e_T `else` e_F

$\equiv e_T$

`if` `false` `then` e_T `else` e_F

$\equiv e_F$

Exercise

- Write a function `abs` that prints the absolute value of a number. Show the evaluation of `abs -10`.
- Write a function that takes a mark and prints the corresponding grade. The grades are A:100–80, B: 60–79, C: 40–59, and F: less than 40.

Table of Contents

Inductive definitions

In mathematics we often encounter functions that are defined in terms of themselves. For example the factorial function is defined as,

$$n! = \begin{cases} 1 & n < 2 \\ n \times (n-1)! & \text{otherwise} \end{cases}$$

Recursive functions

Recursive definitions in Ocaml are introduced using *let rec*.

```
let rec fact n =
  if n<2 then 1 else n * fact (n-1) ;;
```

If you forget *rec* you get an "unbound value" error.

Termination

To ensure that recursion *terminates*, a recursive definition must

- ★ have a trivial or *base case* whose value is given explicitly.
- ★ the *inductive case* must be defined in terms of the function applied to a value *closer* to the base case value.

Exercise

$$
fib\, n = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ fib(n-1) + fib(n-2) & \text{otherwise} \end{cases}
$$

Table of Contents

- ★ There are no new rules required for evaluating recursive functions.
- ★ Substitute the (recursive) function definition until the base case is reached.
- ★ Don't forget to keep the intermediate vaules during substitution.

Factorial Evaluation

```
fact 3 ≡ if 3<2 then 1 else 3 ∗ fact (3−1)
≡ 3 ∗ fact 2
≡ 3 ∗ (if 2<2 then 1 else 2 ∗ fact (2−1))
≡ 3 ∗ 2 ∗ fact 1
≡ 3 ∗ 2 ∗ (if 1<2 then 1 else 1 ∗ fact (1−1))
≡ 3 ∗ 2 ∗ 1
```

Note how we keep the intermediate values $3 * 2 * \ldots$.

Exercise

Show the evaluation of `fib 3` using the following definition.

```
let rec fib n =
  if n=0 then 0
  else if n=1 then 1
  else fib (n-1) + fib (n-2)
```

Summary of Concepts

- Conditional expressions
- Recursive function definitions
- Base case and inductive case
- Ensuring termination
- Evaluating recursive functions