CSE 341: Programming Languages

Dan Grossman

Winter 2008

Lecture 3— Let bindings, options

Let bindings

Motivation: Functions without local variables can be poor style and/or really inefficient.

Syntax: `let b1 b2 ... bn in e end` where each `bi` is a *binding*.

Typing rules: Type-check each `bi` and `e` in context including previous bindings. Type of whole expression is type of `e`.

Evaluation rules: Evaluate each `bi` and `e` in environment including previous bindings. Value of whole expression is result of evaluating `e`.

Elegant design worth repeating:

- Let-expressions can appear anywhere an expression can.
- Let-expressions can have any kind of binding.
  - Local functions can refer to any bindings *in scope*.

More than style

Exercise: hand-evaluate `bad_max` and `good_max` for lists `[1,2]` `[1,2,3]`, and `[3,2,1]`.

Summary and general pattern

Major progress: recursive functions, pairs, lists, let-expressions

Each has a syntax, typing rules, evaluation rules.

Functions, pairs, and lists are very different, but we can describe them in the same way:

- How do you create values? (function definition, pair expressions, empty-list and `::`)
- How do you use values? (function application, `#1` and `#2`, `null`, `hd`, and `tl`)

This (and conditionals) is enough for your homework though:

- `andalso` and `orelse` help
- You need *options* (next slide)
- Soon: much better ways to use pairs and lists (pattern-matching)

Options

"Options are like lists that can have at most one element."

- Create a `t option` with `NONE` or `SOME e` where `e` has type `t`.
- Use a `t option` with `isSome` and `valOf`

Why not just use (more general) lists? An interesting style trade-off:

- Options better express purpose, enforce invariants on callers, maybe faster.
- But cannot use functions for lists already written.

Let bindings and nesting

Here are a couple of ML examples (of increasing complexity):

```
val x = [2,3,4];
val squid = let val x=100 in x+x end;
```

What is the value of `squid` (and why?)

```
val a = 4;
val b = 10;
val clam = let val a=100; val b=a+3 in a+b end;
```

What is the value of `clam` (and why?)

Let bindings and nesting continued

```
fun octopus (m : int, n : int) =
    let val m=100 in m+n end;
```

What is the value of `octopus(4,5)` (and why?)

Let bindings – Mini-exercises

What is the value of `x` after each of the following sequences of ML code is executed?

1.

```
val x = let val y=42 in y+3 end;
```

2.

```
val x = 100;
val y = let val x=[2,3,4] in x@x end;
```