CSE 341:
Programming Languages

Winter 2005
Lecture 3—Let bindings, options, and benefits of no mutation
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 \texttt{bad\_max} and \texttt{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 \texttt{t option} with \texttt{NONE} or \texttt{SOME e} where \texttt{e} has type \texttt{t}.
- Use a \texttt{t option} with \texttt{isSome} and \texttt{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.
You want to change something?

There is no way to mutate (assign to) a binding, pair component, or list element.

How could the lack of a feature make programming easier?

In this case:

- Amount of sharing is indistinguishable
  - Aliasing irrelevant to correctness!

- Bindings are invariant across function application
  - Mutation breaks compositional reasoning, a (the?) intellectual tool of engineering