CSE 341: Programming Languages

Autumn 2005

Lecture 12 — Modules and Abstract Types

Modules

Large programs benefit from more structure than a list of bindings.

Breaking into parts allows separate reasoning:

- Application-level: in terms of module (in ML, structure) invariants
- Type-checking level: in terms of module types
- Implementation level: in terms of module code-generation

By providing a *restricted* interface (in ML, a signature), there are *more* equivalent implementations in terms of the interface.

Key restrictions:

- Make bindings inaccessible
- Make types abstract (know type exists, but not its definition)

SML has a much fancier module system, but we'll stick with the basics.

Abstract types are a "top-5" feature of modern languages.

Structure basics

Syntax: `structure Name = struct` *`bindings`* `end`

If `x` is a variable, exception, type, constructor, etc. defined in `Name`, the rest of the program refers to it via `Name.x`

(You can also do `open Name`, which is often bad style, but convenient when testing.)

So far, this is just *namespace management*, which is important for large programs, but not very interesting.