

CS 6371: Advanced Programming Languages

Dr. Kevin Hamlen

Fall 2008

Announcements:

1. Be sure you've submitted assignment 1 through WebCT by 4:05pm.
2. Take assignment 2 handout (due in ONE WEEK).
3. Today:
 - A. OCaml compiler: compiling separate modules
 - B. Intro to operational semantics

Modules & Separate Compilation

- Each OCaml module “Foo” consists of two files:
 - source file: foo.ml (like foo.c in C)
 - interface file: foo.mli (like foo.h in C)
- Interface files declare all public types and variables (including function variables):

```
type 'a btree = BNull | BNode ('a * 'a btree * 'a btree)
val tree2list : 'a btree -> 'a list
```

- Matching .ml and .mli files must have the SAME root filename (e.g. foo.ml goes with foo.mli)
- We will not be writing .mli files in this class, but you will need to be able to read the ones I provide you

Accessing Module Members

- To refer to a type, type constructor, or variable from an external module named “foo.ml”, use:

```
Foo.tree2list Foo.BNull;;  
Foo.tree2list (Foo.BNode (3, Foo.BNull, Foo.BNull));;
```

- Note: Module names MUST be capitalized!
- Alternatively, use “open” at the top of your .ml file to import all identifiers from the external module into the current namespace:

```
open Foo  
  
tree2list BNull;;  
tree2list BNode (3, BNull, BNull);;
```

Separate Compilation

- Compile each module separately with:

```
ocamlc -c foo.mli  
ocamlc -c foo.ml
```

- There must exist an .mli file for each module that foo.mli and foo.ml refer to
- These commands produce foo.cmi and foo.cmo, respectively
- Link all modules together with:

```
ocamlc -o output.exe foo.cmo bar.cmo ...
```

- Omit “.exe” when compiling on Unix
- Order of arguments matters! Each .cmo file may only depend on those that appear earlier in the command line

Auto-generating Interface Files

- If you type `ocamlc -c foo.ml` and there is no `foo.mli` file in the current directory, OCaml generates both a `foo.cmi` and a `foo.cmo`
- Auto-generated `.cmi` files export all top-level types and variables from your `.ml` file

Roadmap

- Language design & development
 - most of the rest of the course
 - mathematical formalisms (semantics, type theory)
 - begin today with BNF syntax and operational semantics
 - programming assignments designed to test your understanding of the math
- Motivating questions
 - Is it possible to write an OCaml program that compiles, yet treats an integer as a string at runtime?
 - If not, can we prove that all OCaml programs are “safe”?
 - How does OCaml always infer the correct types for everything? (Is it always right?)
 - How do we add new features to a language without accidentally breaking one of the above guarantees?