CSE 341: Section 1

Tam Dang

University of Washington

September 27, 2018

Outline

Introduction

Hello! I'm Tam.

- BS/MS Student
- Took CSE 341 in Spring 2016 with Dan (it was my first CSE 300!)
- I like music, I do NLP research, I am currently learning French

Course Resources

We have lots of course resources!

Email all of us at once with `cse341-staff@cs.washington.edu`

Comprehensive Reading Notes (notes for unit 1)

- Great for review + filling in gaps in your understanding

Come to our office hours!

We're here for you :)

Installation

Setup

Extensive guide on how to install Emacs for each OS:
https://courses.cs.washington.edu/courses/cse341/18au/sml_emacs.pdf

We need three things:

- **Emacs**: The IDE this course supports
- **SML**: The standard ML language
- **SML-mode in Emacs**: A package for Emacs that makes writing ML code easier

Emacs Commands

Setup

C is **CTRL**

M (meta) is **ALT (option key for Macs)**

(ex. `C-s` means hold down CTRL and press `s`)

Emacs Commands

Setup

C is **CTRL**

M (meta) is **ALT (option key for Macs)**

(ex. `C-s` means hold down CTRL and press `s`)

- Open a file: `C-x C-f`
- Save a file: `C-x C-s`
- Escape out of the current command: `C-g`

ML workflow in Emacs

Setup

REPL (Read-Eval-Print-Loop)

- Great for running snippets of code
- Evaluates all of your commands
  - Each command has to end in a semi-colon
- Load `val` bindings from any .sml file with `use`

ML workflow in Emacs

Setup

REPL (Read-Eval-Print-Loop)

- Great for running snippets of code
- Evaluates all of your commands
  - Each command has to end in a semi-colon
- Load `val` bindings from any .sml file with `use`

Starting an SML REPL in Emacs

- **If you are editing a .sml file**: `C-c C-s + Enter` while inside the SML buffer
- **If you AREN'T editing a .sml file**: Do `M-x`, type 'sml-mode' and then hit enter

ML workflow in Emacs

Setup

REPL (Read-Eval-Print-Loop)

- Great for running snippets of code
- Evaluates all of your commands
  - Each command has to end in a semi-colon
- Load `val` bindings from any .sml file with `use`

Starting an SML REPL in Emacs

- **If you are editing a .sml file**: `C-c C-s + Enter` while inside the SML buffer
- **If you AREN'T editing a .sml file**: Do `M-x`, type 'sml-mode' and then hit enter

****You'll want to restart the REPL between loading files with** `use`******

- Loading a file multiple times / loading multiple files in the same REPL session can cause weird behavior (why?)

Shadowing

ML Concepts

`val` bindings are **immutable**

- You can't change a variable, but you can add another with the same name

```
val x = 2 (* x -> int *);
val y = 3 (* y -> int *);
val x = 1 (* x -> int *);
```

Shadowing

ML Concepts

`val` bindings are **immutable**

- You can't change a variable, but you can add another with the same name

```
val x = 2 (* x -> int *);
val y = 3 (* y -> int *);
val x = 1 (* x -> int *);
```

The most recent binding is always used (so `x` is 1)

Shadowing

ML Concepts

`val` bindings are **immutable**

- You can't change a variable, but you can add another with the same name

```
val x = 2 (* x -> int *);
val y = 3 (* y -> int *);
val x = 1 (* x -> int *);
```

The most recent binding is always used (so `x` is 1)

Shadowing is considered bad style and should be avoided

Shadowing

ML Concepts

Restarting the REPL between loading of files prevents weirdness caused by shadowing

Suppose I had a file `example.sml` containing

```
val x = 8 (* x -> int *);
val y = 2 (* y -> int *);
```

What happens after `use example.sml;` in the REPL?

Shadowing

ML Concepts

Restarting the REPL between loading of files prevents weirdness caused by shadowing

Suppose I had a file `example.sml` containing

```
val x = 8 (* x -> int *);
val y = 2 (* y -> int *);
```

What happens after `use example.sml;` in the REPL?

Without restarting the REPL, I edit `example.sml` to look like

```
val x = 8 (* x -> int *);
```

What do I get when I do `use example.sml;` in the REPL?

Debugging

ML Concepts

* Demo *

Debugging

ML Concepts

* Demo *

Errors can occur at 3 stages:

- **Syntax:** Your program is not "valid SML" (e.g. omitting a keyword)
- **Type Check:** One of the type checking rules didn't work out (e.g. mismatching types of an `if-then-else`)
- **Runtime:** Your program did something while running that it shouldn't (e.g. division by zero)

Read and think deeply about what you write!

Comparison Operators

ML Concepts

You can compare numbers in SML

These operators take two expressions that *evaluate* to `int` and give you a `bool`

= (Equality)	< (Less than)	<= (Less than or equal)
<> (Inequality)	> (Greater than)	>= (Greater than or equal)

Logical Operators

ML Concepts

You can chain 'bool's together in SML

Opearation	Syntax	Types	Evaluation
`andalso`	`e1 andalso e2`	`e1` and `e2` eval to `bool`	Same as Java's `e1 && e2`
`orelse`	`e1 orelse e2`	`e1` and `e2` eval to `bool`	Same as Java's `e1 || e2`
`not`	`not e1`	`e1` evals to `bool`	Same as Java's `!e1`