Agenda

- Introduction + Questions (7 min)
- Setup: get everything running (5 min)
- Emacs Basics (5 min)
- ML development workflow (3 min)
- Environments (10 min)
- Shadowing (5 min)
- Debugging (10 min)
- Comparison Operators (1 min)
- Boolean Operators (1-5 min)
- Testing (1-2 min)
Introduction

Josh Pollock

3rd year Math + CS major

Took 341 in Winter 2017 with James Wilcox

PLSE research advised by Zach Tatlock and Jared Roesch

I work on theorem proving, compilers, and visualization.
Introduction

An Wang

3rd year CS major

Took 341 in Spring 2017 with Dan Grossman

PLSE research advised by Ras Bodik

I work on program synthesis
Course Resources

We have a ton of course resources. Please use them!

If you get stuck or need help:

- Email the staff list! cse341-staff@cs.washington.edu

- Come to Office Hours (Every Weekday, see website)

We’re here for you
Setup

Excellent guide located on the course website:

We’re going to spend about 5 minutes setting up now (so you can follow along for the rest of section)

You need 3 things installed:

- Emacs
- SML
- SML mode for Emacs
Emacs Basics

Don’t be scared!

Commands have particular notation: C-x means hold Ctrl while pressing x

Meta key is Alt (thus M-z means hold Alt, press z)

C-x C-s is Save File

C-x C-f is Open File

C-x C-c is Exit Emacs

C-g is Escape (Abort any partial command you may have entered)
ML Development Workflow

REPL means **Read Eval Print Loop**

You can type in any ML code you want, it will evaluate it

Useful to put code in .sml file for reuse

Every command must end in a semicolon (;)

Load .sml files into REPL with `use` command
Environments

(* static environment *)
List of (id * type)

(* dynamic environment *)
List of (id * value)

<table>
<thead>
<tr>
<th>id</th>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>3</td>
</tr>
<tr>
<td>a</td>
<td>3</td>
</tr>
<tr>
<td>c</td>
<td>9</td>
</tr>
</tbody>
</table>

val a = 1;
val b = 2 + a;
val a = 3;
val c = a * b;
Sketching a Simple SML Program

```sml
val x = 34;
val y = 17;
val z = (x + y) + (y + 2);
val q = z + 1;
val abs_of_z = if z < 0 then 0 - z else z;
val abs_of_z_simpler = abs z
```
Sketching a Simple SML Program

```sml
val x = 34;
val y = 17;
val z = (x + y) + (y + 2);
val q = z + 1;
val abs_of_z = if z < 0 then 0 - z else z;
val abs_of_z_simpler = abs z
```

<table>
<thead>
<tr>
<th>id</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>
Sketching a Simple SML Program

```
val x = 34;
val y = 17;
val z = (x + y) + (y + 2);
val q = z + 1;
val abs_of_z = if z < 0 then 0 - z else z;
val abs_of_z_simpler = abs z
```

<table>
<thead>
<tr>
<th>id</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>int</td>
</tr>
</tbody>
</table>

34

int
Sketching a Simple SML Program

```sml
val x = 34;
val y = 17;
val z = (x + y) + (y + 2);
val q = z + 1;
val abs_of_z = if z < 0 then 0 - z else z;
val abs_of_z_simpler = abs z
```

<table>
<thead>
<tr>
<th>id</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>int</td>
</tr>
<tr>
<td>y</td>
<td></td>
</tr>
</tbody>
</table>

17
Sketching a Simple SML Program

```sml
class
val x = 34;
val y = 17;
val z = (x + y) + (y + 2);
val q = z + 1;
val abs_of_z = if z < 0 then 0 - z else z;
val abs_of_z_simpler = abs z
```

<table>
<thead>
<tr>
<th>id</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>int</td>
</tr>
<tr>
<td>y</td>
<td>int</td>
</tr>
</tbody>
</table>

17

int
Sketching a Simple SML Program

```sml
val x = 34;
val y = 17;
val z = (x + y) + (y + 2);
val q = z + 1;
val abs_of_z = if z < 0 then 0 – z else z;
val abs_of_z_simpler = abs z;
```

<table>
<thead>
<tr>
<th>id</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>int</td>
</tr>
<tr>
<td>y</td>
<td>int</td>
</tr>
<tr>
<td>z</td>
<td></td>
</tr>
</tbody>
</table>

\[(x + y) + (y + 2)\]
Sketching a Simple SML Program

val x = 34;
val y = 17;
val z = (x + y) + (y + 2);
val q = z + 1;
val abs_of_z= if z < 0 then 0 - z else z;
val abs_of_z_simpler= abs z

(x + y) + (y + 2)
Sketching a Simple SML Program

val x = 34;
val y = 17;
val z = (x + y) + (y + 2);
val q = z + 1;
val abs_of_z = if z < 0 then 0 - z else z;
val abs_of_z_simpler = abs z

<table>
<thead>
<tr>
<th>id</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>int</td>
</tr>
<tr>
<td>y</td>
<td>int</td>
</tr>
<tr>
<td>z</td>
<td></td>
</tr>
</tbody>
</table>
Sketching a Simple SML Program

val x = 34;
val y = 17;
val z = (x + y) + (y + 2);
val q = z + 1;
val abs_of_z = if z < 0 then 0 - z else z;
val abs_of_z_simpler = abs z

<table>
<thead>
<tr>
<th>id</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>int</td>
</tr>
<tr>
<td>y</td>
<td>int</td>
</tr>
<tr>
<td>z</td>
<td></td>
</tr>
</tbody>
</table>
Sketching a Simple SML Program

val x = 34;
val y = 17;
val z = (x + y) + (y + 2);
val q = z + 1;
val abs_of_z = if z < 0 then 0 - z else z;
val abs_of_z_simpler = abs z

<table>
<thead>
<tr>
<th>id</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>int</td>
</tr>
<tr>
<td>y</td>
<td>int</td>
</tr>
<tr>
<td>z</td>
<td></td>
</tr>
</tbody>
</table>
Sketching a Simple SML Program

```sml
val x = 34;
val y = 17;
val z = (x + y) + (y + 2);
val q = z + 1;
val abs_of_z = if z < 0 then 0 - z else z;
val abs_of_z_simpler = abs z
```

<table>
<thead>
<tr>
<th>id</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>int</td>
</tr>
<tr>
<td>y</td>
<td>int</td>
</tr>
<tr>
<td>z</td>
<td></td>
</tr>
</tbody>
</table>

\[
\begin{align*}
(x + y) + (y + 2) \\
(int + y) + (y + 2) \\
(int + int) + (y + 2) \\
int + (y + 2)
\end{align*}
\]
Sketching a Simple SML Program

```
val x = 34;
val y = 17;
val z = (x + y) + (y + 2);
val q = z + 1;
val abs_of_z = if z < 0 then 0 - z else z;
val abs_of_z_simpler = abs z
```

<table>
<thead>
<tr>
<th>id</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>int</td>
</tr>
<tr>
<td>y</td>
<td>int</td>
</tr>
<tr>
<td>z</td>
<td></td>
</tr>
</tbody>
</table>

```
(x + y) + (y + 2)
(int + y) + (y + 2)
(int + int) + (y + 2)
int + (y + 2)
```
Sketching a Simple SML Program

```sml
val x = 34;
val y = 17;
val z = (x + y) + (y + 2);
val q = z + 1;
val abs_of_z = if z < 0 then 0 - z else z;
val abs_of_z_simpler = abs z
```

<table>
<thead>
<tr>
<th>id</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>int</td>
</tr>
<tr>
<td>y</td>
<td>int</td>
</tr>
<tr>
<td>z</td>
<td></td>
</tr>
</tbody>
</table>

```
(x + y) + (y + 2)
(int + y) + (y + 2)
(int + int) + (y + 2)
int + (y + 2)
```
val x = 34;
val y = 17;
val z = (x + y) + (y + 2);
val q = z + 1;
val abs_of_z = if z < 0 then 0 - z else z;
val abs_of_z_simpler = abs z

<table>
<thead>
<tr>
<th>id</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>int</td>
</tr>
<tr>
<td>y</td>
<td>int</td>
</tr>
<tr>
<td>z</td>
<td>int</td>
</tr>
</tbody>
</table>

\[
(x + \_y) + (\_y + 2) \\
(int + \_y) + (\_y + 2) \\
(int + int) + (y + 2) \\
int + (y + 2)
\]
Sketching a Simple SML Program

val x = 34;
val y = 17;
val z = (x + y) + (y + 2);
val q = z + 1;
val abs_of_z = if z < 0 then 0 - z else z;
val abs_of_z_simpler = abs z

<table>
<thead>
<tr>
<th>id</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>int</td>
</tr>
<tr>
<td>y</td>
<td>int</td>
</tr>
<tr>
<td>z</td>
<td>int</td>
</tr>
<tr>
<td>q</td>
<td></td>
</tr>
</tbody>
</table>
Sketching a Simple SML Program

val x = 34;
val y = 17;
val z = (x + y) + (y + 2);
val q = z + 1;
val abs_of_z = if z < 0 then 0 – z else z;
val abs_of_z_simpler = abs z

<table>
<thead>
<tr>
<th>id</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>int</td>
</tr>
<tr>
<td>y</td>
<td>int</td>
</tr>
<tr>
<td>z</td>
<td>int</td>
</tr>
<tr>
<td>q</td>
<td></td>
</tr>
</tbody>
</table>
Sketching a Simple SML Program

val x = 34;
val y = 17;
val z = (x + y) + (y + 2);

val q = z + 1;
val abs_of_z = if z < 0 then 0 - z else z;
val abs_of_z_simpler = abs z

<table>
<thead>
<tr>
<th>id</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>int</td>
</tr>
<tr>
<td>y</td>
<td>int</td>
</tr>
<tr>
<td>z</td>
<td>int</td>
</tr>
<tr>
<td>q</td>
<td></td>
</tr>
</tbody>
</table>
Sketching a Simple SML Program

val x = 34;
val y = 17;
val z = (x + y) + (y + 2);
val q = z + 1;
val abs_of_z = if z < 0 then 0 - z else z;
val abs_of_z_simpler = abs z

<table>
<thead>
<tr>
<th>id</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>int</td>
</tr>
<tr>
<td>y</td>
<td>int</td>
</tr>
<tr>
<td>z</td>
<td>int</td>
</tr>
<tr>
<td>q</td>
<td>int</td>
</tr>
</tbody>
</table>

\[
z + 1
\]

\[
\text{int} + 1
\]

\[
\text{int} + \text{int}
\]

\[
\text{int}
\]
Sketching a Simple SML Program

```sml
val x = 34;
val y = 17;
val z = (x + y) + (y + 2);
val q = z + 1;
val abs_of_z = if z < 0 then 0 - z else z;
val abs_of_z_simpler = abs z
```

<table>
<thead>
<tr>
<th>id</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>int</td>
</tr>
<tr>
<td>y</td>
<td>int</td>
</tr>
<tr>
<td>z</td>
<td>int</td>
</tr>
<tr>
<td>q</td>
<td>int</td>
</tr>
<tr>
<td>abs_of_z</td>
<td></td>
</tr>
<tr>
<td>abs_of_z_simpler</td>
<td></td>
</tr>
</tbody>
</table>
Sketching a Simple SML Program

```sml
val x = 34;
val y = 17;
val z = (x + y) + (y + 2);
val q = z + 1;
val abs_of_z = if z < 0 then 0 - z else z;
val abs_of_z_simpler = abs z
```

<table>
<thead>
<tr>
<th>id</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>int</td>
</tr>
<tr>
<td>y</td>
<td>int</td>
</tr>
<tr>
<td>z</td>
<td>int</td>
</tr>
<tr>
<td>q</td>
<td>int</td>
</tr>
<tr>
<td>abs_of_z</td>
<td></td>
</tr>
</tbody>
</table>
Sketching a Simple SML Program

```
val x = 34;
val y = 17;
val z = (x + y) + (y + 2);
val q = z + 1;
val abs_of_z = if z < 0 then 0 - z else z;
val abs_of_z_simpler = abs z
```

<table>
<thead>
<tr>
<th>id</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>int</td>
</tr>
<tr>
<td>y</td>
<td>int</td>
</tr>
<tr>
<td>z</td>
<td>int</td>
</tr>
<tr>
<td>q</td>
<td>int</td>
</tr>
<tr>
<td>abs_of_z</td>
<td></td>
</tr>
<tr>
<td>abs_of_z_simpler</td>
<td></td>
</tr>
</tbody>
</table>
Shadowing

```ocaml
code
val a = 1;
val a = 3;
val b = 2;
val a = 3;
```

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

When looking for a variable definition, most recent is always used

Shadowing is usually considered bad style
Shadowing

```
val a = 1;
val b = 2;
val a = 3;
```

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

When looking for a variable definition, most recent is always used

Shadowing is usually considered bad style
Shadowing

This behavior, along with `use` in the REPL can lead to confusing effects.

Suppose I have the following program:

```plaintext
val x = 8;
val y = 2;
```

I load that into the REPL with `use`. Now, I decide to change my program, and I delete a line, giving this:

```plaintext
val x = 8;
```

I load that into the REPL without restarting the REPL. What goes wrong? (Hint: what is the value of y?)
Debugging

Errors can occur at 4 stages:
- Syntax: Your program is not “valid SML” in some (usually small and annoyingly nitpicky) way
- Type Check: One of the type checking rules didn’t work out
- Runtime: Your program did something while running that it shouldn’t
- Test: Your program breaks your tests

The best way to debug is to read what you wrote carefully, and think about it.
Comparison Operators

You can compare numbers in SML!

Each of these operators has 2 subexpressions of type `int`, and produces a `bool`.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>(Equality)</td>
</tr>
<tr>
<td>&lt;</td>
<td>(Less than)</td>
</tr>
<tr>
<td>&lt;=</td>
<td>(Less than or equal)</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>(Inequality)</td>
</tr>
<tr>
<td>&gt;</td>
<td>(Greater than)</td>
</tr>
<tr>
<td>&gt;=</td>
<td>(Greater than or equal)</td>
</tr>
</tbody>
</table>
Boolean Operators

You can also perform logical operations over booleans!

<table>
<thead>
<tr>
<th>Operation</th>
<th>Syntax</th>
<th>Type-Checking</th>
<th>Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>andalso</td>
<td>e1 andalso e2</td>
<td>If ___ then ___</td>
<td>Same as Java’s e1 &amp;&amp; e2</td>
</tr>
<tr>
<td>orelse</td>
<td>e1 orelse e2</td>
<td>If ___ then ___</td>
<td>Same as Java’s e1</td>
</tr>
<tr>
<td>not</td>
<td>not e1</td>
<td>If ___ then ___</td>
<td>Same as Java’s !e1</td>
</tr>
</tbody>
</table>

Technical note: andalso/orelse are SML builtins as they use short-circuit evaluation.
## Boolean Operators

You can also perform logical operations over `bool`s!

<table>
<thead>
<tr>
<th>Operation</th>
<th>Syntax</th>
<th>Type-Checking</th>
<th>Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>andalso</td>
<td><code>e1 andalso e2</code></td>
<td>if <code>e1</code> and <code>e2</code> have type <code>bool</code>, then <code>e1 andalso e2</code> has type <code>bool</code></td>
<td>Same as Java's <code>e1 &amp;&amp; e2</code></td>
</tr>
<tr>
<td>orelse</td>
<td><code>e1 orelse e2</code></td>
<td>if <code>e1</code> and <code>e2</code> have type <code>bool</code>, then <code>e1 orelse e2</code> has type <code>bool</code></td>
<td>Same as Java's `e1</td>
</tr>
<tr>
<td>not</td>
<td><code>not e1</code></td>
<td>if <code>e1</code> has type <code>bool</code>, then <code>not e1</code> has type <code>bool</code></td>
<td>Same as Java's <code>!e1</code></td>
</tr>
</tbody>
</table>

Technical note: `andalso/orelse` are SML builtins as they use short-circuit evaluation.
Testing

We don’t have a unit testing framework (too heavyweight for 5 weeks)

We require you to submit a test file (ungraded) for each homework

val test1 = ((4 div 4) = 1);

(* Neat trick for creating hard-fail tests: *)

val true = ((4 div 4) = 1);