1 Lecture Plan

1. Response to Piazza questions and comments.
2. Schedule of problem sets (6 of them) and prelim.
3. Enrichment topics (“beyond what Google knows”), e.g. partial types
   tie to computability theory and mathematics (sets versus types).
4. What is a type? (sets vs types) more enrichment.
5. Thoughts about problem set 1:
   - How to best define the rationals (the issue is \(n/0\) and using the
     gcd algorithm)
   - PS2 will look at using modules for rationals \(\mathbb{Q}\) and extending to
     reals \(\mathbb{R}\)
   - The gcd algorithm and its specifications
6. Bits of logic among the OCaml types, polymorphic types:
   \(\alpha \& \beta\) \hspace{1cm} \(\alpha \Rightarrow \beta\) \hspace{1cm} \(\alpha \lor \beta\) \hspace{1cm} \text{False?} \hspace{1cm} \text{True?}\)

2 Piazza questions and comments

What resources do we have for learning?

- Textbook: *Real World OCaml* (we use about 1/2)
  
  Please read all of Chapter 1 (‘assigned’ before), Chapter 2 (will use
  in Lecture 5), and Chapter 3 (List basics).

- See related lecture notes, Nate Foster builds on Dexter Kozen.
• List of other resources on course site. We used the small step semantics.

• Canonical values – discussed in Lectures 2 and 3.
  The key point is that they are defined by *eager evaluation*, 
  \((2 \times 3, \frac{5}{5})\) are not canonical, reduce to \((6,1)\).
  Likewise \([2 \times 3; \frac{5}{5}; 0 \times 7]\)
  What does \([\text{fun} \ x \rightarrow x; \ \text{fun} \ x \rightarrow 1]\) show when evaluated?

3 Schedule of problem sets and in-class prelim

<table>
<thead>
<tr>
<th>Date for</th>
<th>Due Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>PS2 Out on Wed. Feb. 17</td>
<td>March 3</td>
</tr>
<tr>
<td>PS3 Out on Fri. March 4</td>
<td>March 24</td>
</tr>
<tr>
<td>Prelim Thur. March 17, in class</td>
<td></td>
</tr>
<tr>
<td>PS4 Out on Thur. March 24</td>
<td>April 8</td>
</tr>
<tr>
<td>PS5 Out on Fri. April 8</td>
<td>April 28</td>
</tr>
<tr>
<td>PS6 Out on Thur. April 28</td>
<td>May 11 (last day of classes)</td>
</tr>
</tbody>
</table>

4 Enrichment topics (“Beyond what Google knows”)

Partial types are an example. Why is *int* partial?

  let loop n : int = loop(n + 1)

We say that this diverges on any input. So while *loop*(0) has type *int*, its value is not equal to 0, 1, -1, 2, -2...

We sometimes write \(\bot\) for a generic expression that has no canonical values.

We can even find \(\bot\) in *bool* and *unit*! Why?

5 What is a type?

For computer science this is a fundamental concept, analogous to the idea of sets in mathematics.
There are two ways that sets are discussed: intuitive and axiomatic.

**Intuitive** (informal): the empty set is a set and any collection of sets with no repetitions is a set, e.g

\[ \{\emptyset\}, \{\emptyset, \{\emptyset\}\}, \ldots \]

**Axiomatic**: Zermelo-Fraenkel with Choice (ZFC)

- A0. there is a set
- A1. equality
- A2. foundation
- A3. comprehension \( \{x : A \mid P(x)\} \)
- A4. pairing
- A5. union
- A6. replacement
- A7. infinity
- A8. power set
- A9. choice

Plus the axioms of First-Order Logic (\&, \lor, \implies, \sim, \forall, \exists), a dozen rules plus 10 axioms.

Compare to OCaml type theory:

- `unit`, `int`, `bool`, `char`, `string`, `exn`
- `\alpha \times \beta \alpha \rightarrow \beta \text{ Lα|Rβ}
- records, variants
- lists, recursive types
- asyn-package
- (refs)

45 or so rules plus computation rules.

Types are based on a computation system defined on untyped expressions. OCaml uses small step evaluation semantics.

Types are a collection of canonical values from the computation system with a notion of equality on the values. They can be reasoned about as **partial equivalence relations on expressions.**