Motivation for Dependent Type Theory

Aditya

University of Texas at Austin

January 22, 2020
Syllabus Information

• The Syllabus is here.
• You will be on the hook for presentations weekly. More on that in two weeks.
• You will have weekly homework assignments every Wednesday.
• The first two weeks of class will be contrary to the normal structure.
• The third week the organizers will demo for you the proper structure.
Homotopy Type Theory

- Two parts Homotopy and Type Theory
- For the first two weeks we will ignore Homotopy
- It is very very powerful, but we need to understand the Type Theory behind HoTT.
Type Theory

Curry Howard Correspondance

Dependent Type Theory
Let's Look at Some Code

```java
int x = 5;
```

This break down to:

```java
int x; //This part is super underrated
x = 5;
```

From Discrete that line is usually thought to mean $\exists x \in \{\text{Int}\}$. But in truth it is a judgement is $x : \text{Int}$, or $x$ is type Int.
What is Type Theory?

• It’s a language
• It sits behind our notion of computer science.
Useful Properties We Get

Let's take an example out of Haskell:

```haskell
data Shape = Circle Float Float Float | Triangle Float Float Float Float
```

Note that: \( \text{Shape} \neq \text{Circle} \cup \text{Triangle}. \)
This allows us to do pattern matching.
So we have been doing type theory, is that enough to be great?

- I want to do more though.
- I want a way to do math computably.
- This allows us to do formal verification with proof assistants.
- My code would be amazingly correct then.
Problem

- Propositions are not representable in set theory.
- Makes theorem provers really hard to make.
- 1969 The Curry Howard Correspondance relates proofs as programs and propositions as types.
- This is a huge result that lets us use type theory to build these systems.
- Coq, Agda, Lean, Arend, …
Consequences of Curry-Howard

<table>
<thead>
<tr>
<th>Classical Math</th>
<th>Type Theory</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A \implies B$</td>
<td>$A \rightarrow B$</td>
</tr>
<tr>
<td>$A \land B$</td>
<td>$A \times B$</td>
</tr>
<tr>
<td>$A \lor B$</td>
<td>$A + B$</td>
</tr>
</tbody>
</table>

Aditya Tewari
Template Types

• So type theory can allow you to do quite a bit.
• How many of you have heard of templates or Generics
• These are super useful tools that let us write powerful programs.
Let’s say I have a class `Shape`. Then we code the following:

```java
class Circle extends Shape {}
//A simple form of inheritance
template <typename T>
class Future {}
```

In C++ code similar to this would not allow `Future<Circle>` to be a subclass of `Future<Shape>`. This concept is a violation of a thing we want, which is co-variant types.
Dependent Types

- Clearly then Type Theory isn’t complete, because expressing this error requires a type version of $\forall$.
- Some verifiers get around this by using a sort of hybrid syntax (Coq).
- If we want to be formal we need Types that are Dependent on types.
- So we have $\Pi$ which we can think of as similar to $\forall$.
- We also have $\Sigma$ which we can think of as similar to $\exists$.
- These we can think of as similar to template types.
- Dependent Haskell and Rust both implement this.
- This is why Rust is so lit.
How will we use Dependent Type Theory

- This is the language of Homotopy Type Theory
- **Arend** is written in Dependent Type Theory.
- Your proofs in your presentations will be written in Dependent Type Theory.