Homework 2
Types $\cong$ Theorems

98-317: Hype for Types

Checkpoint Due: 5 Feb 2019 at 6:30 PM
Final Due: 12 Feb 2019 at 6:30PM

1 Introduction

In class, we discussed the idea that we can use types to express logical propositions, and that creating a value of a particular type corresponds to proving a proposition. In this homework, you will explore this idea further, writing some functions in SML to prove various propositions in logic.

There are two sections to this assignment - a checkpoint and a final. For each section, you'll be implementing a different structure.

2 Logic in SML

In this homework, you will be implementing two structures called `SimpleProofs` (checkpoint) and `Proofs` (final) so that they ascribe to the signatures `SIMPLE_PROOFS` and `PROOFS` respectively. The `SIMPLE_PROOFS` signature declares 7 values with types that correspond to logical propositions. The `PROOFS` signature has 6 slightly more difficult to prove types-as-propositions. In order to implement the signature, you will have to prove the propositions!

For the most part, you can decide what the behavior of the values in these structures should be; they just have to have the correct type. However, there are a few rules:

1. All of the definitions must be *values*. This prohibits code like

   ```
   val demorgan_cong = raise Fail "This typechecks!"
   ```

2. All function definitions must be *total*. That is, they must terminate on all possible inputs. This prohibits code like

   ```
   fun contrapos f = raise Fail "This typechecks!"
   ```

To help you implement these functions, we've given you a couple of definitions in the `Definitions` structure. Its signature is reproduced below.

```
signature DEFINITIONS =
sig
  datatype ('a,'b) or = INL of 'a | INR of 'b
  type void
  type 'a not = 'a -> void
  val abort : void -> 'b
end
```

The `or` type is the same as the sum type we discussed in the typechecking lecture, and is meant to represent logical or. A value of this type can either have the INL constructor and contain a value of type 'a, or have the INR constructor and contain a value of type 'b.

The `void` type is a type for which it is impossible to construct a value. This represents False. If you could somehow construct void, it would allow you to construct a value of any type. This is what `abort` does: it is a total function from void to any arbitrary type 'c. This mirrors how from False, you can conclude anything.

The `not` type is shorthand for `'a -> void`. In logic, $\neg A$ is defined to be "If I assume A, then I can reach a contradiction," so this is also how it is defined in terms of types.

You can use these definitions as much as you'd like in your code.

3 Checkpoint

For the first section of this assignment, you'll be implementing the `SimpleProofs` structure. You can compile your solutions to see if they typecheck by running

```
smlnj -m checkpoint.cm
```

To submit your assignment, run

```
make checkpoint
```

and submit the tar file to the Logic - Checkpoint assignment.

This part of the assignment is due Tuesday, 5 February at 6:30 PM.

4 Final

For the second section of this assignment, you'll be implementing the `Proofs` structure. You can compile your solutions to see if they typecheck by running

```
smlnj -m final.cm
```

To submit your assignment, run

```
make final
```

and submit the tar file to the Logic - Final assignment.

This part of the assignment is due Tuesday, 12 February at 6:30 PM.