Lecture 2

Specification Language
To recapitulate a key point made at the end of Lecture 1, most formal methods courses start
with (classical) logic as the language for precisely saying what a program should do. This
logic is used to be precise about programming problems or tasks.

This is not the best way to start nowadays. For one thing, it is very easy to specify unsolvable
problems this way and not notice.

For another reason we need only look at how programming problems are stated in program-
ing courses. We use types not logical formulas. Moreover, as we will stress in this course,
the language of types subsumes logic. Indeed it goes well beyond it. Also, types make it
clearer to see when a problem is solvable by a program.

So we start with types. An excellent textbook from 1991 does it this way as well:

_Type Theory and Functional Programming_

by Simon Thompson

University of Kent at Canterbury

In Lecture 1 we look at these types:

\[ A \rightarrow A \] the functions (computable) from type A to type A

\[ A \rightarrow (B \rightarrow A) \] the functions from type A into the type or computable functions from
type B to A

\[ (B \rightarrow C) \Rightarrow ((A \Rightarrow B) \Rightarrow (A \Rightarrow C)) \]

Exercise: say this in words

These are programming problems:

\[ A \Rightarrow A \] is the problem of finding a function in this type

We saw in Lecture 1 that the identity function will solve this. There are many ways
to write this, e.g.

\[ \text{id}(x) = x, \text{ for } x \text{ in } A \] math

\[ \text{fun} \ x \rightarrow x \] OCaml

\[ \text{function}(x : A) \ \text{return}(x) \ \text{end} \] another PL

\[ (\text{lambda} \ (x) \ x) \] Lisp/Scheme

\[ \lambda(x.x) \] Nuprl
Here are some "famous" programming problems written as axioms in propositional logic.

Hilbert’s 1922 Axioms, with his numbering:

1. $A \implies (B \implies A)$
2. $(A \implies (A \implies B)) \implies (A \implies B)$
3. $(A \implies (B \implies C)) \implies (B \implies (A \implies C))$
4. $(B \implies C) \implies ((A \implies B) \implies (A \implies C))$
5. $A \implies (\neg A \implies B)$
6. $(A \implies B) \implies (\neg A \implies B) \implies B$

Kleene’s Axioms for $\implies$, with his 1952 numbering

1a. $A \implies (B \implies A)$
1b. $(A \implies B) \implies (A \implies (B \implies C)) \implies (A \implies C)$
7. $(A \implies B) \implies (A \implies \neg B) \implies \neg A$

We can take $\neg A$ as $(A \implies \text{False})$

Pierce’s Law: $((P \implies Q) \implies P) \implies P$

We can read all of these as programming problems. Some of them can’t be solved by any computable function. Which ones can’t be?