Lecture 9

Summary of Propositional Calculus as a Programming Language

mPC corresponds to the applied simply typed λ-calculus

With the

- atomic types \( P, Q, R, \ldots \) for atomic propositions
- function types \( A \Rightarrow B \) for constructive implication
- product types \( A \times B \) for conjunction (\&)
- disjoint union types \( A + B \) for disjunction (or)

\( \iota \text{PC} \) adds one new evidence term, \( \text{any}(t) \), \( t \) a term, but no computational rules.

What is the logical meaning of \( \text{any}(t) \)?

The simplest use is for \( (\text{False} \Rightarrow A) \). The evidence term is simply \( \lambda(x.\text{any}(x)) \). The function cannot be applied to a canonical value \( t \) because False has no values.

If \( A \) is a non-empty type with canonical value \( a \), then \( \lambda(x.a) \) also belongs to the type \( (\text{False} \Rightarrow A) \). This element, \( \lambda(x.a) \) could also belong to the proposition \( (B \Rightarrow A) \) for any type \( B \).

The \( \nu \text{prl} \) exception mechanism adds a new canonical form. We will explore treating \( \text{any}(t) \) as an exception.

\[ \text{exception}(\text{name};\text{value}) \]

and a corresponding non-canonical form

\[ \text{try}(p; n; v.\text{c}(v)) \]

for \( p \) an exception.

The \( \nu \text{prl} \) computation system also includes a new evaluation rule for catching exceptions.

\[ \text{try} \ (\text{exception}(a, b); n, v.\text{c}(v)) \ \text{reduces to} \]

\[ \text{if} \ a =_{\text{token}} n \ \text{then} \ c[b/v] \]
\[ \text{else} \ \text{exception}(a, b) \]

The test \( a =_{\text{token}} n \) verifies that \( n \) is a name.

The other non-canonical forms such as apply, spread, decide, just pass along exceptions, e.g.

\[ \text{spread}(\text{exception}(a, b); x, y.\text{B}(x, y)) \downarrow \text{exception}(a, b) \]
To say that an expression has a value means reducing to a non-exception canonical form.

Normal types do not have exceptions as values. We’ll see later that partial types do have exceptions as “values”. Some programming languages, such as classic ML, catch all exceptions, using a “global catch”. This defeats their use in logic.

The next step in the study of the Propositional Calculus is to examine the so called classical PC, denoted as just PC.

In the list of inference rules from Lecture 7, the Law of Excluded Middle (LEM) is sufficient to axiomatize PC.

\[ H \vdash A \lor \sim A \text{ by } \text{magic}(A) \]

The proof term \( \text{magic}(A) \) asks the “Platonic oracle” which of \( A \) or \( A \implies \text{False} \) is “true”.

We will study in detail in lectures 11 and 12 the treatment of iPC with exceptions. This opens a new research topic for which we have some preliminary results.