This week is actually a good opportunity to look back at previous section notes and material, and make sure you are comfortable with the material. This is because we don’t expect you to be deeply familiar with the technical material on dependent types, nor are you required to be expert in Dafny or Coq.

1 Dependent Types

(a) Assume that `boolvec` has kind `(x : nat) ⇒ Type` and `init` has type `(n : nat) → bool → boolvec n`.

Show that the expression `init 5 true` has type `boolvec 5`,

That is, prove

\[ \Gamma \vdash init 5 \text{ true} : boolvec 5 \]

where

\[ \Gamma = boolvec :: (x : nat) ⇒ Type, \text{ init} : (n : nat) → bool → boolvec n. \]

Answer:

\[
\begin{array}{c}
\Gamma \vdash init : (n : nat) \rightarrow bool \rightarrow boolvec n \\
\Gamma \vdash 5 : nat \\
\end{array}
\]

\[
\begin{array}{c}
\Gamma \vdash 5 : nat \rightarrow boolvec 5 \\
\Gamma \vdash init 5 \text{ true} : boolvec 5 \\
\end{array}
\]

(b) Show that the types `boolvec (35 + 7)` and `boolvec ((λy : nat. y) 42)` are equivalent.

That is, prove that

\[ \Gamma \vdash boolvec (35 + 7) \equiv boolvec ((λy : nat. y) 42) :: Type \]

where

\[ \Gamma = boolvec :: (x : nat) ⇒ Type. \]

Answer: Let \( T_1 \) be defined as

\[
\begin{array}{c}
\Gamma \vdash boolvec \equiv boolvec :: (x : nat) ⇒ Type \\
\Gamma \vdash 35 + 7 \equiv 42 :: nat \\
\end{array}
\]

\[ \Gamma \vdash boolvec (35 + 7) \equiv boolvec 42 :: Type \]

and let \( T_2 \) be defined as

\[
\begin{array}{c}
\Gamma \vdash y : nat \vdash y : nat \\
\Gamma \vdash 42 : nat \\
\end{array}
\]

\[ \Gamma \vdash (λy : nat. y) 42 \equiv 42 :: nat \]

\[ \Gamma \vdash 42 \equiv (λy : nat. y) 42 :: nat \]

\[ \Gamma \vdash boolvec 42 \equiv boolvec ((λy : nat. y) 42) :: Type \]
Dependent Types
Section and Practice Problems

\[
\begin{array}{c}
\Gamma \vdash boolvec (35 + 7) \equiv boolvec 42 :: Type \\
\Gamma \vdash boolvec 42 \equiv boolvec ((\lambda y : nat. y) 42) :: Type \\
\Gamma \vdash boolvec (35 + 7) \equiv boolvec ((\lambda y : nat. y) 42) :: Type
\end{array}
\]

where here \( T_1 \) is similar to \( T_2 \) and left as an exercise to the reader.

(c) Suppose we had a function \texttt{double} that takes a \texttt{boolvec} and returns a \texttt{boolvec} that is twice the length. Write an appropriate type for \texttt{double}. (Note that you will need make sure that the type of the \texttt{boolvec} argument is well formed! Hint: take a look at the type of \texttt{join}, mentioned in the Lecture 20 notes, for inspiration.)

Answer:

\[
(n : nat) \to boolvec n \to boolvec (n + n)
\]

Note that we need to take a natural number \( n \) as an argument, in order for us to specify the type of the second argument (i.e., a boolean vector of length \( n \), \texttt{boolvec} \( n \)).

If we wrote \texttt{boolvec} \( n \to boolvec (n + n) \), then \( n \) is free and the type isn’t well formed. Note that \texttt{boolvec} \( \to boolvec \) is not well-kinded.

2 Coq and Dafny (Optional!)

If you are interested, you can play around with Dafny online at [https://rise4fun.com/dafny](https://rise4fun.com/dafny). A tutorial (on which the class demo was based) is available at [https://rise4fun.com/Dafny/tutorial/](https://rise4fun.com/Dafny/tutorial/).

The Coq website is [https://coq.inria.fr/](https://coq.inria.fr/) The easiest way to install Coq is via opam, OCaml’s package manager. See [https://coq.inria.fr/opam/www/using.html](https://coq.inria.fr/opam/www/using.html). In lecture, Prof. Chong was using Proof General (an extension to Emacs) to interact with Coq: [https://proofgeneral.github.io/](https://proofgeneral.github.io/).

The Software Foundations series [https://softwarefoundations.cis.upenn.edu/](https://softwarefoundations.cis.upenn.edu/) is a programming-languages oriented introduction to using Coq.