Harvard School of Engineering and Applied Sciences — CS 152: Programming Languages

Environment Semantics; Axiomatic Semantics; Dependent Types Section and Practice Problems

Week 11: Tue April 4–Fri April 7, 2023

---

1 Environment Semantics

For Homework 5, the monadic interpreter you will be using uses environment semantics, that is, the operational semantics of the language uses a map from variables to values instead of performing substitution. This is a quick primer on environment semantics.

An environment ρ maps variables to values. We define a large-step operational semantics for the lambda calculus using an environment semantics. A configuration is a pair $\langle e, \rho \rangle$ where expression e is the expression to compute and ρ is an environment. Intuitively, we will always ensure that any free variables in e are mapped to values by environment ρ.

The evaluation of functions deserves special mention. Configuration $\langle \lambda x.\, e, \rho \rangle$ is a function $\lambda x.\, e$, defined in environment ρ, and evaluates to the *closure* $(\lambda x.\, e, \rho)$. A closure consists of code along with values for all free variables that appear in the code.

The syntax for the language is given below. Note that closures are included as possible values and expressions, and that a function $\lambda x.\, e$ is *not* a value (since we use closures to represent the result of evaluating a function definition).

$$
\begin{aligned}
e &::= x \mid n \mid e_1 + e_2 \mid \lambda x.\, e \mid e_1\ e_2 \mid (\lambda x.\, e, \rho) \\
v &::= n \mid (\lambda x.\, e, \rho)
\end{aligned}
$$

Note than when we apply a function, we evaluate the function body using the environment from the closure (i.e., the lexical environment, ρ_{lex}), as opposed to the environment in use at the function application (the dynamic environment).

$$
\frac{}{\langle x, \rho \rangle \Downarrow \rho(x)} \qquad \frac{}{\langle n, \rho \rangle \Downarrow n} \qquad \frac{\langle e_1, \rho \rangle \Downarrow n_1 \quad \langle e_2, \rho \rangle \Downarrow n_2}{\langle e_1 + e_2, \rho \rangle \Downarrow n}\, n = n_1 + n_2
$$

$$
\frac{}{\langle \lambda x.\, e, \rho \rangle \Downarrow (\lambda x.\, e, \rho)} \qquad \frac{\langle e_1, \rho \rangle \Downarrow (\lambda x.\, e, \rho_{lex}) \quad \langle e_2, \rho \rangle \Downarrow v_2 \quad \langle e, \rho_{lex}[x \mapsto v_2] \rangle \Downarrow v}{\langle e_1\ e_2, \rho \rangle \Downarrow v}
$$

For convenience, we define a rule for let expressions.

$$
\frac{\langle e_1, \rho \rangle \Downarrow v_1 \quad \langle e_2, \rho[x \mapsto v_1] \rangle \Downarrow v_2}{\langle \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2, \rho \rangle \Downarrow v_2}
$$

(a) Evaluate the program $\mathsf{let}\ f = (\mathsf{let}\ a = 5\ \mathsf{in}\ \lambda x.\, a + x)\ \mathsf{in}\ f\ 6$. Note the closure that f is bound to.

Answer: *Here is a derivation of the program.*

$$
\dfrac{\dfrac{\dfrac{}{\langle 5, \emptyset \rangle \Downarrow 5} \quad \dfrac{}{\langle \lambda x.\, a + x, [a \mapsto 5] \rangle \Downarrow (\lambda x.\, a + x, [a \mapsto 5])}}{\langle \mathit{let}\ a = 5\ \mathit{in}\ \lambda x.\, a + x), \emptyset \rangle \Downarrow (\lambda x.\, a + x, [a \mapsto 5])} \quad \dfrac{\vdots}{\langle f\ 6, [f \mapsto (\lambda x.\, a + x, [a \mapsto 5])] \rangle \Downarrow 11}}{\langle \mathit{let}\ f = (\mathit{let}\ a = 5\ \mathit{in}\ \lambda x.\, a + x)\ \mathit{in}\ f\ 6, \emptyset \rangle \Downarrow 11}
$$

where the missing derivation is as follows (and where $\rho_0 = [f \mapsto (\lambda x.\, a + x, [a \mapsto 5])]$ *and* $\rho_1 = [a \mapsto 5, x \mapsto 6]$*)*

$$\dfrac{\dfrac{}{\langle f, \rho_0\rangle \Downarrow (\lambda x.\, a + x, [a \mapsto 5])} \quad \dfrac{}{\langle 6, \rho_0\rangle \Downarrow 6} \quad \dfrac{\dfrac{}{\langle a, \rho_1\rangle \Downarrow 5} \quad \dfrac{}{\langle x, \rho_1\rangle \Downarrow 6}}{\langle a + x, \rho_1\rangle \Downarrow 11}}{\langle f\ 6, \rho_0\rangle \Downarrow 11}$$

Note that f *is bound to the closure* $(\lambda x.\, a + x, [a \mapsto 5])$*. That is. the function* $\lambda x.\, a + x$ *has a lexical environment* $[a \mapsto 5]$*: when the function was defined, the variable* a *was bound to* 5*. Note that when the function is used (*$f\ 6$*), the environment does not bind* a *at all.*

(b) Suppose we replaced the rule for application with the following rule:

$$\dfrac{\langle e_1, \rho\rangle \Downarrow (\lambda x.\, e, \rho_{lex}) \quad \langle e_2, \rho\rangle \Downarrow v_2 \quad \langle e, \rho[x \mapsto v_2]\rangle \Downarrow v}{\langle e_1\ e_2, \rho\rangle \Downarrow v}$$

That is, we use the dynamic environment to evaluate the function body instead of the lexical environment.

What would happen if you evaluated the program $\mathsf{let}\ f = (\mathsf{let}\ a = 5\ \mathsf{in}\ \lambda x.\, a + x)\ \mathsf{in}\ f\ 6$ with this modified semantics?

Answer: *As noted in the answer to the previous question,* f *is bound to the closure* $(\lambda x.\, a + x, [a \mapsto 5])$*, i.e., the lexical environment for the function* $\lambda x.\, a + x$ *is* $[a \mapsto 5]$*: when the function was defined, the variable* a *was bound to* 5*. When the function is used (*$f\ 6$*), the dynamic environment does not bind* a *at all. So that means that evaluation of* $\lambda x.\, a + x$ *will get stuck. In particular, it will try to evaluate expression* $a + x$ *in environment* $[f \mapsto (\lambda x.\, a + x, [a \mapsto 5]), x \mapsto 6]$ *(that is, the dynamic environment at the call site extended with* x *mapped to* 6*), and so won't be able to evaluate the variable* a*.*

2 Axiomatic semantics

(a) Consider the program

$$c \equiv \mathsf{bar} := \mathsf{foo}; \textbf{while}\ \mathsf{foo} > 0\ \textbf{do}\ (\mathsf{bar} := \mathsf{bar} + 1; \mathsf{foo} := \mathsf{foo} - 1).$$

Write a Hoare triple $\{P\}\ c\ \{Q\}$ that expresses that the final value of bar is two times the initial value of foo.

Answer:

$$\{v = \mathsf{foo}\}\ \mathsf{bar} := \mathsf{foo}; \textbf{while}\ \mathsf{foo} > 0\ \textbf{do}\ (\mathsf{bar} := \mathsf{bar} + 1; \mathsf{foo} := \mathsf{foo} - 1)\ \{\mathsf{bar} = 2 \times v\}$$

Note that v *is a logical variable, and we are using it to provide a name for the initial value of* foo*. Note also that the Hoare triple could have said more things about the program. For example, the post condition could have included that* foo *is equal to zero.*

(b) Prove the following Hoare triples. That is, using the inference rules from Section 1.3 of Lecture 19, find proof trees with the appropriate conclusions.

(i) $\vdash \{\mathsf{baz} = 25\}\ \mathsf{baz} := \mathsf{baz} + 17\ \{\mathsf{baz} = 42\}$

Answer:

$$\textsc{Cons}\ \frac{\vDash \mathsf{baz} = 25 \Rightarrow \mathsf{baz} + 17 = 42 \qquad \textsc{Asg.}\ \dfrac{}{\vdash \{\mathsf{baz} + 17 = 42\}\ \mathsf{baz} := \mathsf{baz} + 17\ \{\mathsf{baz} = 42\}}}{\vdash \{\mathsf{baz} = 25\}\ \mathsf{baz} := \mathsf{baz} + 17\ \{\mathsf{baz} = 42\}}$$

(ii) $\vdash \{\textbf{true}\}\ \mathsf{baz} := 22; \mathsf{quux} := 20\ \{\mathsf{baz} + \mathsf{quux} = 42\}$

Answer:

$$\textsc{Consq.}\ \frac{\vDash \textbf{true} \Rightarrow 22 + 20 = 42 \qquad \textsc{Seq.}\ \dfrac{\textsc{Asg.}\ \dfrac{}{\{22 + 20 = 42\}\ \mathsf{baz} := 22\ \{\mathsf{baz} + 20 = 42\}} \qquad \textsc{Asg.}\ \dfrac{}{\{\mathsf{baz} + 20 = 42\}\ \mathsf{quux} := 20\ \{\mathsf{baz} + \mathsf{quux} = 42\}}}{\{22 + 20 = 42\}\ \mathsf{baz} := 22; \mathsf{quux} := 20\ \{\mathsf{baz} + \mathsf{quux} = 42\}}}{\{\textbf{true}\}\ \mathsf{baz} := 22; \mathsf{quux} := 20\ \{\mathsf{baz} + \mathsf{quux} = 42\}}$$

(iii) $\vdash \{\mathsf{baz} + \mathsf{quux} = 42\}\ \mathsf{baz} := \mathsf{baz} - 5; \mathsf{quux} := \mathsf{quux} + 5\ \{\mathsf{baz} + \mathsf{quux} = 42\}$

Answer: *Let* $c \equiv \mathsf{baz} := \mathsf{baz} - 5; \mathsf{quux} := \mathsf{quux} + 5$.

$$\textsc{Consq.}\ \frac{\vDash \mathsf{baz} + \mathsf{quux} = 42 \Rightarrow \mathsf{baz} - 5 + \mathsf{quux} + 5 = 42 \qquad \dfrac{\vdots}{\vdash \{\mathsf{baz} - 5 + \mathsf{quux} + 5 = 42\}\ c\ \{\mathsf{baz} + \mathsf{quux} = 42\}}}{\vdash \{\mathsf{baz} + \mathsf{quux} = 42\}\ c\ \{\mathsf{baz} + \mathsf{quux} = 42\}}$$

where the elided tree is

$$\textsc{Seq}\ \frac{\textsc{Asg}\ \dfrac{}{\{\mathsf{baz} - 5 + \mathsf{quux} + 5 = 42\}\ \mathsf{baz} := \mathsf{baz} - 5\ \{\mathsf{baz} + \mathsf{quux} - 5 = 42\}} \qquad \textsc{Asg}\ \dfrac{}{\{\mathsf{baz} + \mathsf{quux} - 5 = 42\}\ \mathsf{quux} := \mathsf{quux} + 5\ \{\mathsf{baz} + \mathsf{quux} = 42\}}}{\vdash \{\mathsf{baz} - 5 + \mathsf{quux} + 5 = 42\}\ c\ \{\mathsf{baz} + \mathsf{quux} = 42\}}$$

(iv) $\vdash \{\textbf{true}\}\ \textbf{if}\ \mathsf{y} = 0\ \textbf{then}\ \mathsf{z} := 2\ \textbf{else}\ \mathsf{z} := \mathsf{y} \times \mathsf{y}\ \{\mathsf{z} > 0\}$

Answer: *Let's start the derivation using the rule for conditionals:*

$$\textsc{If}\ \frac{\textsc{Cons}\ \dfrac{\vdots}{\vdash \{\textbf{true} \wedge \mathsf{y} = 0\}\ \mathsf{z} := 2\ \{\mathsf{z} > 0\}} \qquad \textsc{Cons}\ \dfrac{\vdots}{\vdash \{\textbf{true} \wedge \neg(\mathsf{y} = 0)\}\ \mathsf{z} := \mathsf{y} \times \mathsf{y}\ \{\mathsf{z} > 0\}}}{\vdash \{\textbf{true}\}\ \textbf{if}\ \mathsf{y} = 0\ \textbf{then}\ \mathsf{z} := 2\ \textbf{else}\ \mathsf{z} := \mathsf{y} \times \mathsf{y}\ \{\mathsf{z} > 0\}}$$

Let's consider each of the proof subtrees in turn.

$$\textsc{Cons}\ \frac{\vDash \textbf{true} \wedge \mathsf{y} = 0 \Rightarrow 2 > 0 \qquad \textsc{Asg}\ \dfrac{}{\{2 > 0\}\ \mathsf{z} := 2\ \{\mathsf{z} > 0\}} \qquad \vDash \mathsf{z} > 0 \Rightarrow \mathsf{z} > 0}{\vdash \{\textbf{true} \wedge \mathsf{y} = 0\}\ \mathsf{z} := 2\ \{\mathsf{z} > 0\}}$$

Where here the assertion $\textbf{true} \wedge \mathsf{y} = 0 \Rightarrow 2 > 0$ *is always valid because* $\vDash 2 > 0$.

$$\textsc{Cons}\ \frac{\vDash \textbf{true} \wedge \neg(\mathsf{y} = 0) \Rightarrow \mathsf{y} \times \mathsf{y} > 0 \qquad \textsc{Asg}\ \dfrac{}{\{\mathsf{y} \times \mathsf{y} > 0\}\ \mathsf{z} := \mathsf{y} \times \mathsf{y}\ \{\mathsf{z} > 0\}} \qquad \vDash \mathsf{z} > 0 \Rightarrow \mathsf{z} > 0}{\vdash \{\textbf{true} \wedge \neg(\mathsf{y} = 0)\}\ \mathsf{z} := \mathsf{y} \times \mathsf{y}\ \{\mathsf{z} > 0\}}$$

The assertion $\textbf{true} \wedge \neg(\mathsf{y} = 0) \Rightarrow \mathsf{y} \times \mathsf{y} > 0$ *is valid because either* $\vDash \mathsf{y} \times \mathsf{y} > 0$ *or* $\nvDash \textbf{true} \wedge \neg(\mathsf{y} = 0)$*. To see this we can simplify* $\nvDash \textbf{true} \wedge \neg(\mathsf{y} = 0)$ *to* $\nvDash \neg(\mathsf{y} = 0)$*, and then to* $\vDash \mathsf{y} = 0$*. And it is always the case that either* $\vDash \mathsf{y} = 0$ *or* $\vDash \mathsf{y} \times \mathsf{y} > 0$*.*

(v) $\vdash \{\textbf{true}\}\ \mathsf{y} := 10; \mathsf{z} := 0; \textbf{while}\ \mathsf{y} > 0\ \textbf{do}\ \mathsf{z} := \mathsf{z} + \mathsf{y}\ \{\mathsf{z} = 55\}$

Answer: *This is a "trick" question in that the loop never terminates. (This wasn't intentional; Prof Chong made a mistake when writing the question. But luckily the Hoare triple is still valid!)*
Let's consider the while loop. So the loop invariant we will use is $\mathsf{y} > 0$*.*

$$\textsc{While}\ \frac{\dfrac{\vdots}{\vdash \{\mathsf{y} > 0 \wedge \mathsf{y} > 0\}\ \mathsf{z} := \mathsf{z} + \mathsf{y}\ \{\mathsf{y} > 0\}}}{\vdash \{\mathsf{y} > 0\}\ \textbf{while}\ \mathsf{y} > 0\ \textbf{do}\ \mathsf{z} := \mathsf{z} + \mathsf{y}\ \{\mathsf{y} > 0 \wedge \mathsf{y} \leq 0\}}$$

Note that the post condition is $\mathsf{y} > 0 \wedge \mathsf{y} \leq 0$*. This is equivalent to* **false***! And* **false** *implies anything. In particular, we have that* $\vDash \mathsf{y} > 0 \wedge \mathsf{y} \leq 0 \Longrightarrow \mathsf{z} = 55$*.*

(vi) $\vdash \{\textbf{true}\}\ \mathsf{y} := 10; \mathsf{z} := 0; \textbf{while}\ \mathsf{y} > 0\ \textbf{do}\ (\mathsf{z} := \mathsf{z} + \mathsf{y}; \mathsf{y} := \mathsf{y} - 1)\ \{\mathsf{z} = 55\}$

Answer: *This is what the previous question was actually meant to be.... The loop invariant we will use is that* $\mathsf{y} \geq 0 \wedge \mathsf{z} = 10 + 9 + \cdots + (\mathsf{y} + 1)$ *which we can write as* $\mathsf{y} \geq 0 \wedge \mathsf{z} = \sum_{i=\mathsf{y}+1}^{10} i$*.*
Let's first of all prove that the loop invariant is established when the program enters the loop (we leave part of the proof tree elided, as an exercise for the reader):

$$\textsc{Cons.}\ \frac{\vDash \textbf{true} \Longrightarrow 10 = 10 \wedge 0 = 0 \qquad \dfrac{\vdots}{\vdash \{10 = 10 \wedge 0 = 0\}\ \mathsf{y} := 10; \mathsf{z} := 0;\ \{\mathsf{y} = 10 \wedge \mathsf{z} = 0\}} \qquad \vDash (\mathsf{y} = 10 \wedge \mathsf{z} = 0) \Longrightarrow \mathsf{y} \geq 0 \wedge \mathsf{z} = \sum_{i=\mathsf{y}+1}^{10} i}{\vdash \{\textbf{true}\}\ \mathsf{y} := 10; \mathsf{z} := 0;\ \{\mathsf{y} \geq 0 \wedge \mathsf{z} = \sum_{i=\mathsf{y}+1}^{10} i\}}$$

Now let's show that it is in fact a loop invariant. For brevity let $S \equiv \sum_{i=\mathsf{y}+1}^{10} i$ *and* $S' \equiv \sum_{i=\mathsf{y}-1+1}^{10} i$*.*

$$\textsc{While}\ \frac{\dfrac{\vDash \mathsf{y} \geq 0 \wedge \mathsf{z} = S \wedge \mathsf{y} > 0 \Longrightarrow \mathsf{y} - 1 \geq 0 \wedge \mathsf{z} + \mathsf{y} = S' \qquad \overset{1}{\dfrac{\vdots}{D}} \qquad \vDash \mathsf{y} \geq 0 \wedge \mathsf{z} = S \Longrightarrow \mathsf{y} \geq 0 \wedge \mathsf{z} = S}{\vdash \{\mathsf{y} \geq 0 \wedge \mathsf{z} = S \wedge \mathsf{y} > 0\}\ \mathsf{z} := \mathsf{z} + \mathsf{y}; \mathsf{y} := \mathsf{y} - 1\ \{\mathsf{y} \geq 0 \wedge \mathsf{z} = S\}}}{\vdash \{\mathsf{y} \geq 0 \wedge \mathsf{z} = S\}\ \textbf{while}\ \mathsf{y} > 0\ \textbf{do}\ (\mathsf{z} := \mathsf{z} + \mathsf{y}; \mathsf{y} := \mathsf{y} - 1)\ \{\mathsf{y} \geq 0 \wedge \mathsf{z} = S \wedge \mathsf{y} \leq 0\}}$$

where $\overset{1}{\dfrac{\vdots}{D}}$ *is the following derivation (where* $S \equiv \sum_{i=\mathsf{y}+1}^{10} i$ *and* $S' \equiv \sum_{i=\mathsf{y}-1+1}^{10} i$*):*

$$\textsc{Seq}\ \frac{\textsc{Asg}\ \dfrac{}{\vdash \{\mathsf{y} - 1 \geq 0 \wedge \mathsf{z} + \mathsf{y} = S'\}\ \mathsf{z} := \mathsf{z} + \mathsf{y}\ \{\mathsf{y} - 1 \geq 0 \wedge \mathsf{z} = S'\}} \qquad \textsc{Asg.}\ \dfrac{}{\vdash \{\mathsf{y} - 1 \geq 0 \wedge \mathsf{z} = S'\}\ \mathsf{y} := \mathsf{y} - 1\ \{\mathsf{y} \geq 0 \wedge \mathsf{z} = S\}}}{\vdash \{\mathsf{y} - 1 \geq 0 \wedge \mathsf{z} + \mathsf{y} = S'\}\ \mathsf{z} := \mathsf{z} + \mathsf{y}; \mathsf{y} := \mathsf{y} - 1\ \{\mathsf{y} \geq 0 \wedge \mathsf{z} = S\}}$$

Finally, we can use the fact that $\vDash \mathsf{y} \geq 0 \wedge \mathsf{z} = S \wedge \mathsf{y} \leq 0 \Longrightarrow \mathsf{z} = 55$ *to construct a proof of the desired triple (where* $c \equiv \mathsf{y} := 10; \mathsf{z} := 0; \textbf{while}\ \mathsf{y} > 0\ \textbf{do}\ (\mathsf{z} := \mathsf{z} + \mathsf{y}; \mathsf{y} := \mathsf{y} - 1)$*):*

$$\textsc{Cons}\ \frac{\vDash \textbf{true} \Longrightarrow \textbf{true} \qquad \textsc{Seq}\ \dfrac{\vdots}{\vdash \{\textbf{true}\}\ c\ \{\mathsf{y} \geq 0 \wedge \mathsf{z} = S \wedge \mathsf{y} \leq 0\}} \qquad \vDash \mathsf{y} \geq 0 \wedge \mathsf{z} = S \wedge \mathsf{y} \leq 0 \Longrightarrow \mathsf{z} = 55}{\vdash \{\textbf{true}\}\ c\ \{\mathsf{z} = 55\}}$$

3 Dependent Types

(a) Assume that boolvec has kind $(x : \textbf{nat}) \Rightarrow \textbf{Type}$ and init has type $(n : \textbf{nat}) \rightarrow \textbf{bool} \rightarrow \textbf{boolvec}\ n$).
Show that the expression init 5 true has type **boolvec** 5,
That is, prove

$$\Gamma \vdash \mathsf{init}\ 5\ \mathsf{true} : \textbf{boolvec}\ 5$$

where

$$\Gamma = \mathsf{boolvec} :: (x : \textbf{nat}) \Rightarrow \textbf{Type}, \mathsf{init} : (n : \textbf{nat}) \rightarrow \textbf{bool} \rightarrow \textbf{boolvec}\ n.$$

Answer:

$$\dfrac{\dfrac{\overline{\Gamma \vdash \mathit{init} : (n : \textbf{nat}) \rightarrow \textbf{bool} \rightarrow \textbf{boolvec}\ n} \quad \overline{\Gamma \vdash 5 : \textbf{nat}}}{\Gamma \vdash \mathit{init}\ 5 : \textbf{bool} \rightarrow \textbf{boolvec}\ 5} \quad \dfrac{}{\Gamma \vdash \mathit{true} : \textbf{bool}}}{\Gamma \vdash \mathit{init}\ 5\ \mathit{true} : \textbf{boolvec}\ 5}$$

(b) Show that the types **boolvec** $(35 + 7)$ and **boolvec** $((\lambda y : \textbf{nat}.\ y)\ 42)$ are equivalent.
That is, prove that

$$\Gamma \vdash \textbf{boolvec}\ (35 + 7) \equiv \textbf{boolvec}\ ((\lambda y : \textbf{nat}.\ y)\ 42) :: \textbf{Type}$$

where

$$\Gamma = \mathsf{boolvec} :: (x : \textbf{nat}) \Rightarrow \textbf{Type}.$$

Answer: *Let T_1 be defined as*

$$\dfrac{\overline{\Gamma \vdash \textbf{boolvec} \equiv \textbf{boolvec} :: (x : \textbf{nat}) \Rightarrow \textbf{Type}} \quad \overline{\Gamma \vdash 35 + 7 \equiv 42 :: \textbf{nat}}}{\Gamma \vdash \textbf{boolvec}\ (35 + 7) \equiv \textbf{boolvec}\ 42 :: \textbf{Type}}$$

and let T_2 be defined as

$$\dfrac{\dfrac{\overline{\Gamma, y : \textbf{nat} \vdash y : \textbf{nat}} \quad \overline{\Gamma \vdash 42 : \textbf{nat}}}{\Gamma \vdash (\lambda y : \textbf{nat}.\ y)\ 42 \equiv 42 :: \textbf{nat}} \quad \overline{\Gamma \vdash \textbf{boolvec} \equiv \textbf{boolvec} :: (x : \textbf{nat}) \Rightarrow \textbf{Type}}}{\Gamma \vdash \textbf{boolvec}\ 42 \equiv \textbf{boolvec}\ ((\lambda y : \textbf{nat}.\ y)\ 42) :: \textbf{Type}}$$

in

$$\dfrac{T_1 \quad \overline{\Gamma \vdash \textbf{boolvec}\ (35 + 7) \equiv \textbf{boolvec}\ 42 :: \textbf{Type}} \qquad \dfrac{T_2}{\Gamma \vdash \textbf{boolvec}\ 42 \equiv \textbf{boolvec}\ ((\lambda y : \textbf{nat}.\ y)\ 42) :: \textbf{Type}}}{\Gamma \vdash \textbf{boolvec}\ (35 + 7) \equiv \textbf{boolvec}\ ((\lambda y : \textbf{nat}.\ y)\ 42) :: \textbf{Type}}$$

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

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

Answer:

$$(n : \textbf{\textit{nat}}) \to \textbf{\textit{boolvec}}\ n \to \textbf{\textit{boolvec}}\ (n + n)$$

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, ***boolvec*** *n).*

If we wrote ***boolvec*** *$n \to$* ***boolvec*** *$(n+n)$, then n is free and the type isn't well formed. Note that* ***boolvec*** *$\to$* ***boolvec*** *is not well-kinded.*

4 Coq and Dafny (Optional!)

Note that for this course we do not expect you to be deeply familiar with Dafny or Coq. So this section question is for those that are interested in finding out more about these tools.

You can learn more abot Dafny at `https://dafny.org/`. The easiest way to install it is likely as an extension in VS Code. A tutorial is available at `http://dafny.org/dafny/OnlineTutorial/guide.html`.

The Coq website is `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`. The CoqIDE is probably the easiest way to use Coq, but you can also install an Emacs plugin (Proof General, `https://proofgeneral.github.io/`).

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