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

Simply-typed lambda calculus; More types (Lectures 11-12) Section and Practice Problems

Week 7: Tue Mar 12–Fri Mar 15, 2019

---

1 Simply-typed lambda calculus

(a) Add appropriate type annotations to the following expressions, and state the type of the expression.

(i) $\lambda a.\, a + 4$

Answer: *With minimal annotations:*

$$\lambda a : \textbf{\textit{int}}.\, a + 4$$

and the expression has type $\textbf{\textit{int}} \to \textbf{\textit{int}}$.

(ii) $\lambda f.\, 3 + f\ ()$

Answer: *With minimal annotations:*

$$\lambda f : \textbf{\textit{unit}} \to \textbf{\textit{int}}.\, 3 + f\ ()$$

and the expression has type $(\textbf{\textit{unit}} \to \textbf{\textit{int}}) \to \textbf{\textit{int}}$.

(iii) $(\lambda x.\, x)\ (\lambda f.\, f\ (f\ 42))$

Answer: *With minimal annotations:*

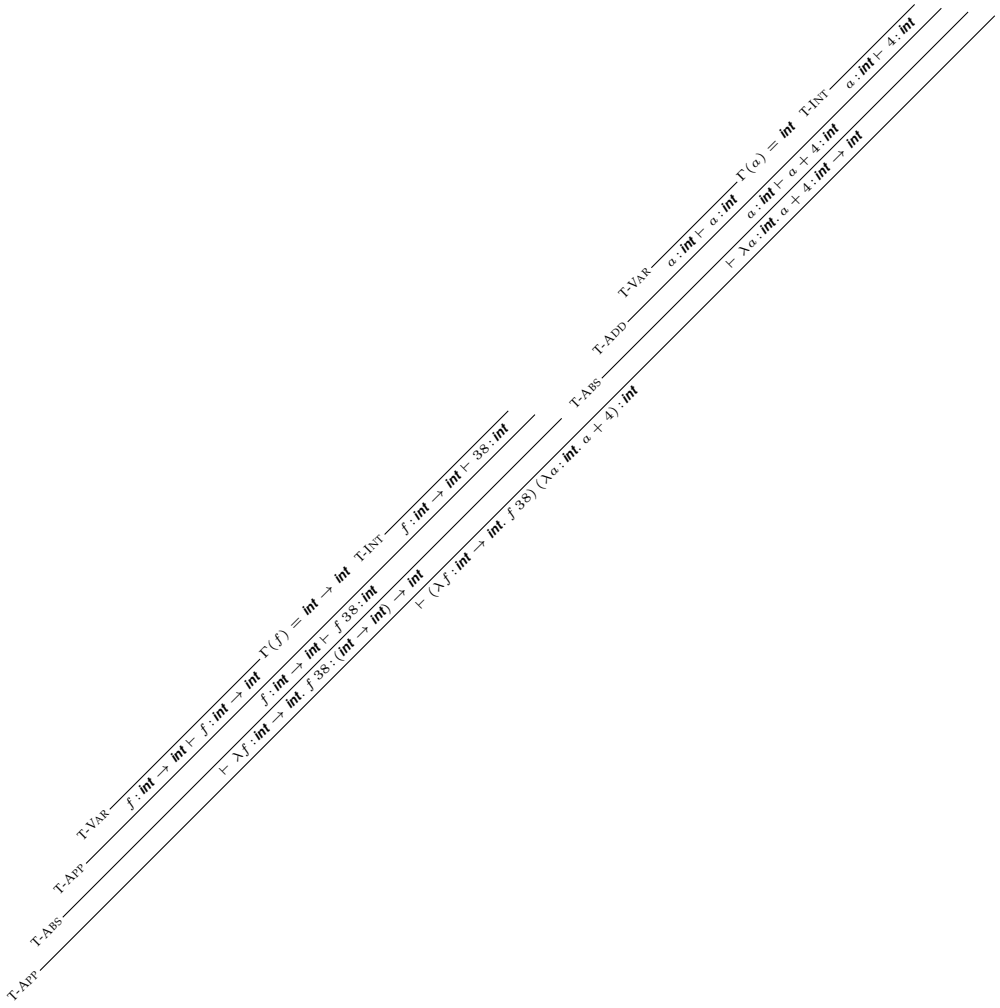
$$(\lambda x : (\textbf{\textit{int}} \to \textbf{\textit{int}}) \to \textbf{\textit{int}}.\, x)\ (\lambda f : \textbf{\textit{int}} \to \textbf{\textit{int}}.\, f\ (f\ 42)).$$

Note that $\lambda x : (\textbf{\textit{int}} \to \textbf{\textit{int}}) \to \textbf{\textit{int}}.\, x$ *has type* $((\textbf{\textit{int}} \to \textbf{\textit{int}}) \to \textbf{\textit{int}}) \to ((\textbf{\textit{int}} \to \textbf{\textit{int}}) \to \textbf{\textit{int}})$.
The expression $\lambda f : \textbf{\textit{int}} \to \textbf{\textit{int}}.\, f\ (f\ 42)$ *has type* $(\textbf{\textit{int}} \to \textbf{\textit{int}}) \to \textbf{\textit{int}}$.
The whole expression has type $(\textbf{\textit{int}} \to \textbf{\textit{int}}) \to \textbf{\textit{int}}$.

(b) For each of the following expressions, give a derivation showing that the expression is well typed.

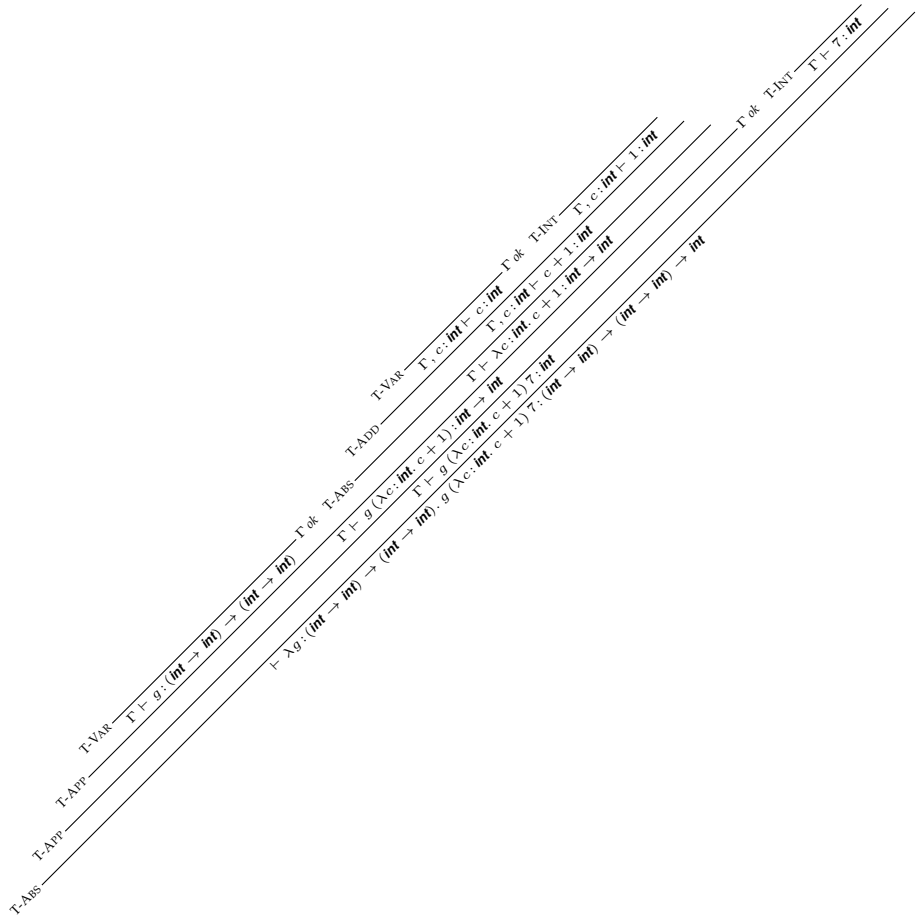
(i) $(\lambda f : \textbf{int} \to \textbf{int}.\, f\ 38)\ (\lambda a : \textbf{int}.\, a + 4)$

Answer:



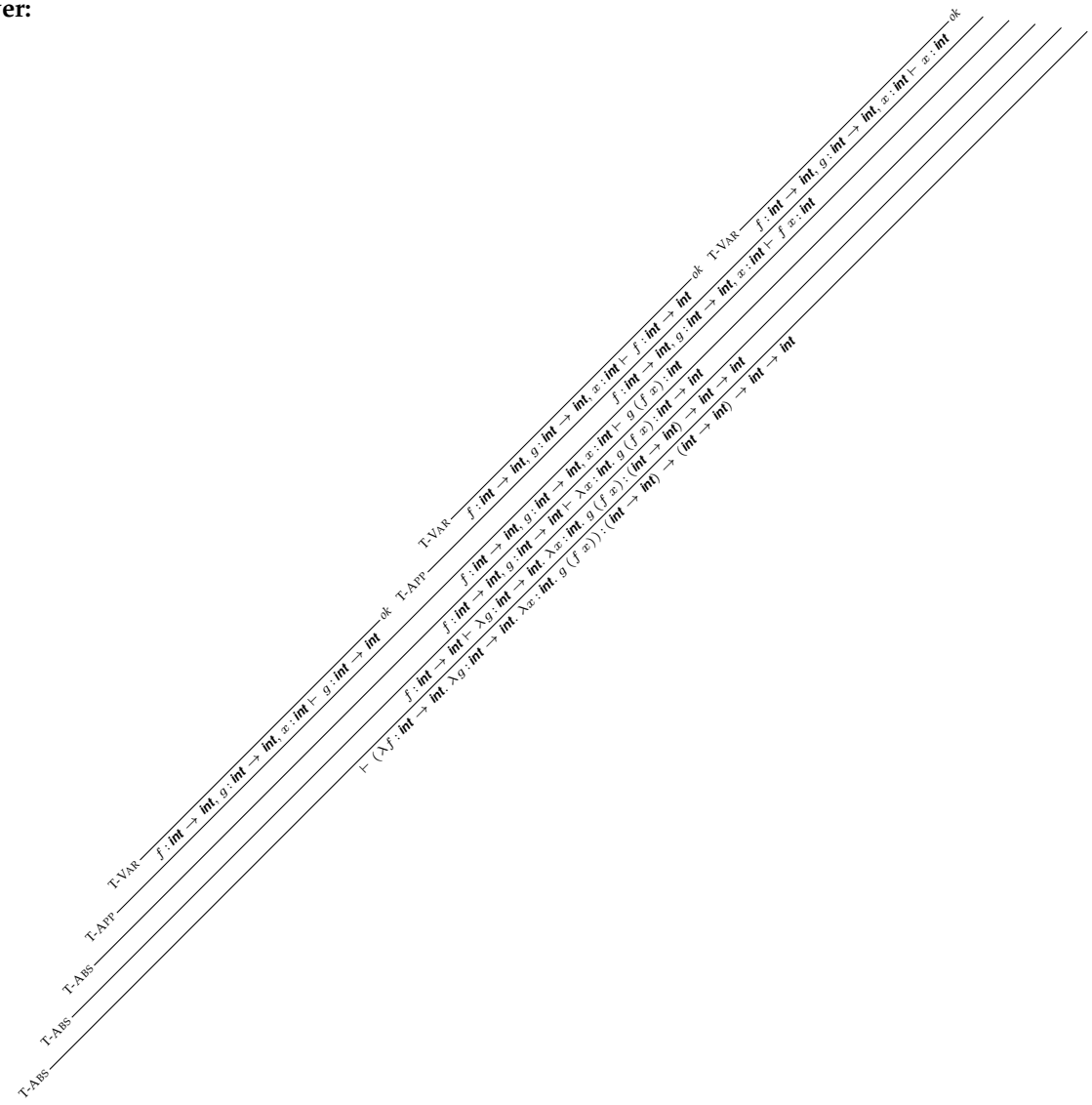
(ii) $\lambda g : (\mathbf{int} \to \mathbf{int}) \to (\mathbf{int} \to \mathbf{int}).\, g\ (\lambda c : \mathbf{int}.\, c + 1)\ 7$

Answer: *We have* $\Gamma = g : (\textbf{\textit{int}} \to \textbf{\textit{int}}) \to (\textbf{\textit{int}} \to \textbf{\textit{int}})$ *for succinctness.*



(iii) $\lambda f : \textbf{int} \to \textbf{int}.\ \lambda g : \textbf{int} \to \textbf{int}.\ \lambda x : \textbf{int}.\ g\ (f\ x)$

Answer:



2 Type soundness

(a) Recall the substitution lemma that we used in the proof of type soundness.

Lemma (Substitution)**.** *If* $x : \tau' \vdash e : \tau$ *and* $\vdash v : \tau'$ *then* $\vdash e\{v/x\} : \tau$.

Using the definition of substitution given in Assignment 2, prove this lemma. You may assume that v does not have any free variables (i.e., $FV(v) = \emptyset$).

Remember to state what set you are performing induction on and what the property is that you are proving for every element in that set. If you are not sure what cases you need to consider, or what you are able to assume in each case of the inductive proof, we strongly suggest that you write down the inductive reasoning principle for the inductively defined set.

Answer: *We recall the definition of substitution, since we will use it in this proof.*

$$y\{e/x\} = \begin{cases} e & \text{if } x = y \\ y & \text{if } x \neq y \end{cases}$$

$$(e_1\ e_2)\{e/x\} = e_1\{e/x\}\ e_2\{e/x\}$$

$$(\lambda y.\, e')\{e/x\} = \begin{cases} \lambda y.\, e' & \text{if } x = y \\ \lambda y.\, (e'\{e/x\}) & \text{if } x \neq y \text{ and } y \notin FV(e) \\ \lambda z.\, ((e'\{z/y\})\{e/x\}) & \text{if } x \neq y \text{ and } y \in FV(e), \text{ where} \\ & \quad z \notin FV(e) \cup FV(e') \cup \{x\} \end{cases}$$

We extend substitution for the new syntactic forms in our language.

$$n\{e/x\} = n$$
$$()\{e/x\} = ()$$
$$e_1 + e_2\{e/x\} = (e_1\{e/x\}) + (e_2\{e/x\})$$

We proceed by structural induction on expressions. That is, we will perform induction on the set of expressions. As an aside, the inductive reasoning principle for the set of expressions for this language is the following:

For any property P,
If

- *$P(n)$ holds*
- *$P(())$ holds*
- *$P(x)$ holds*
- *For all expressions e, if $P(e)$ holds then $P(\lambda x{:}\tau.\, e)$ holds*
- *For all expressions e_1 and e_2, if $P(e_1)$ and $P(e_2)$ holds then $P(e_1\ e_2)$ holds*
- *For all expressions e_1 and e_2, if $P(e_1)$ and $P(e_2)$ holds then $P(e_1 + e_2)$ holds*

then

for all expressions e, $P(e)$ holds.

The property we will prove is actually stronger than the lemma. We will need this stronger property in order to deal with the case for functions. The property is:

$$P(e) = \forall \Gamma, x, \tau, v, \tau'.\ \textit{if } \Gamma[x \mapsto \tau'] \vdash e{:}\tau \textit{ and } \vdash v{:}\tau' \textit{ then } \Gamma \vdash e\{v/x\}{:}\tau$$

We consider the possible cases (which correspond to the 6 bullet points in the inductive reasoning principle above).

- $e = n$

  Assume that $\Gamma[x \mapsto \tau'] \vdash e{:}\tau$ and $\vdash v{:}\tau'$.

  Since $e = n$, we have $e\{v/x\} = e$. Thus, $\Gamma \vdash e\{v/x\}{:}\tau$ holds trivially.

- $e = ()$

  Assume that $\Gamma[x \mapsto \tau'] \vdash e{:}\tau$ and $\vdash v{:}\tau'$.

  Since $e = ()$, we have $e\{v/x\} = e$. Thus, $\Gamma \vdash e\{v/x\}{:}\tau$ holds trivially.

- $e = y$

  Assume that $\Gamma[x \mapsto \tau'] \vdash e{:}\tau$ and $\vdash v{:}\tau'$.

  We consider two subcases, where x and y are the same variable, and where they are different variables.

  - *x and y are the same variable.*
    In this case, we have $\tau = \tau'$ (since $e = x$ and $\Gamma[x \mapsto \tau'] \vdash e : \tau$ means that, by inversion using rule T-VAR*, $\tau = \tau'$). Also, we have $e\{v/x\} = v$. From $\vdash v : \tau'$ we can derive $\Gamma \vdash v : \tau'$, and so $\Gamma \vdash e\{v/x\} : \tau$ holds.*
  - *x and y are different variables.*
    In this case, we have $e\{v/x\} = e$. Thus, $\Gamma \vdash e\{v/x\} : \tau$ holds trivially.

- $e = \lambda y : \tau_y . e'$

  Assume that $\Gamma[x \mapsto \tau'] \vdash e : \tau$ and $\vdash v : \tau'$. Also assume that the property holds for e (i.e., the inductive hypothesis).

  We consider three subcases, corresponding to the three possible cases for substitution of $\lambda y : \tau_y . e'$.

  - *x and y are the same variable.*
    In this case, we have $e\{v/x\} = e$. Thus, $\Gamma \vdash e\{v/x\} : \tau$ holds trivially.
  - *x and y are different variables and $y \notin FV(v)$.*
    In this case, we have $e\{v/x\} = \lambda y : \tau_y . (e'\{v/x\})$.
    By inversion on $\Gamma[x \mapsto \tau'] \vdash e : \tau$, we have $\Gamma[x \mapsto \tau'][y \mapsto \tau_y] \vdash e' : \tau''$ for some τ'' where $\tau = \tau_y \to \tau''$.
    Since x and y are different variables, note that $\Gamma[x \mapsto \tau'][y \mapsto \tau_y]$ is equal to $\Gamma'[x \mapsto \tau']$ where $\Gamma' = \Gamma[y \mapsto \tau_y]$. Because the inductive hypothesis holds for expression e', and $\Gamma'[x \mapsto \tau'] \vdash e' : \tau''$, we have $\Gamma' \vdash (e'\{v/x\}) : \tau''$.
    Using typing rule T-ABS*, we have that $\Gamma \vdash \lambda y : \tau_y . (e'\{v/x\}) : \tau_y \to \tau''$. That is, we have $\Gamma \vdash e\{v/x\} : \tau$, as required.*
  - *x and y are different variables and $y \in FV(v)$.*
    This case is actually impossible. Since v is a value, v can not have any free variables.

- $e = e_1\ e_2$

  Assume that $\Gamma[x \mapsto \tau'] \vdash e : \tau$ and $\vdash v : \tau'$. Also assume that the property holds for e_1 and for e_2 (i.e., the inductive hypothesis).

  From $\Gamma[x \mapsto \tau'] \vdash e : \tau$, by inversion, we have that $\Gamma[x \mapsto \tau'] \vdash e_1 : \tau'' \to \tau$ and $\Gamma[x \mapsto \tau'] \vdash e_2 : \tau''$ for some type τ''. (That is, rule T-APP *is the only typing rule that has a conclusion that matches $\Gamma[x \mapsto \tau'] \vdash e : \tau$, and so it must be the case that the premises of* T-APP *are true.)*

  From the inductive hypothesis, we have that $\Gamma[x \mapsto \tau'] \vdash e_1\{v/x\} : \tau'' \to \tau$ and $\Gamma[x \mapsto \tau'] \vdash e_2\{v/x\} : \tau''$.

  From the definition of substitution, we have that $e\{v/x\} = (e_1\{v/x\})\ (e_2\{v/x\})$.

  Thus, using the typing rule T-APP*, we have that $\Gamma \vdash e\{v/x\} : \tau$, as required.*

- $e = e_1 + e_2$

  Assume that $\Gamma[x \mapsto \tau'] \vdash e : \tau$ and $\vdash v : \tau'$. Also assume that the property holds for e_1 and for e_2 (i.e., the inductive hypothesis).

  *From $\Gamma[x \mapsto \tau'] \vdash e : \tau$, by inversion, we have that $\Gamma[x \mapsto \tau'] \vdash e_1 :$ **int** and $\Gamma[x \mapsto \tau'] \vdash e_2 :$ **int**, and $\tau =$ **int**. (That is, rule* T-ADD *is the only typing rule that has a conclusion that matches $\Gamma[x \mapsto \tau'] \vdash e : \tau$, and so it must be the case that the premises of* T-ADD *are true.)*

  *From the inductive hypothesis, we have that $\Gamma[x \mapsto \tau'] \vdash e_1\{v/x\} :$ **int** and $\Gamma[x \mapsto \tau'] \vdash e_2\{v/x\} :$ **int**.*

  From the definition of substitution, we have that $e\{v/x\} = (e_1\{v/x\}) + (e_2\{v/x\})$.

  Thus, using the typing rule T-ADD*, we have that $\Gamma \vdash e\{v/x\} : \tau$, as required.*

(b) Recall the context lemma that we used in the proof of type soundness.

Lemma (Context). *If* $\vdash E[e_0] : \tau$ *and* $\vdash e_0 : \tau'$ *and* $\vdash e_1 : \tau'$ *then* $\vdash E[e_1] : \tau$.

Prove this lemma.

Remember to state what set you are performing induction on and what the property is that you are proving for every element in that set. If you are not sure what cases you need to consider, or what you are able to assume in each case of the inductive proof, we strongly suggest that you write down the inductive reasoning principle for the inductively defined set.

Answer: *We proceed by structural induction on contexts E. That is, we are doing induction on the set of contexts, which is inductively defined by the grammar:*

$$E ::= [\cdot] \mid E\ e \mid v\ E \mid E + e \mid v + E$$

As an aside, the inductive reasoning principle for the set of contexts is the following:

For any property P,
If

- $P([\cdot])$ *holds*
- *For all contexts E, if $P(E)$ holds then $P(E\ e)$ holds*
- *For all contexts E, if $P(E)$ holds then $P(v\ E)$ holds*
- *For all contexts E, if $P(E)$ holds then $P(E + e)$ holds*
- *For all contexts E, if $P(E)$ holds then $P(v + E)$ holds*

then

for all contexts E, $P(E)$ holds.

So, the property we are proving is:

$$P(E) = \forall e_0, e_1, \tau, \tau'.\ \textit{if}\ \vdash E[e_0] : \tau\ \textit{and}\ \vdash e_0 : \tau'\ \textit{and}\ \vdash e_1 : \tau'\ \textit{then}\ \vdash E[e_1] : \tau$$

We consider the possible cases (which correspond to the 5 bullet points in the inductive reasoning principle above).

- $E = [\cdot]$.

  Assume $\vdash E[e_0] : \tau$ *and* $\vdash e_0 : \tau'$ *and* $\vdash e_1 : \tau'$.

  Since $E[e_0] = e_0$, *we have* $\tau = \tau'$.

  Moreover, since $E[e_1] = e_1$, *from* $\vdash e_1 : \tau'$ *we have* $\vdash E[e_1] : \tau$ *as required.*

- $E = E'\ e$.

  Assume $\vdash E[e_0] : \tau$ *and* $\vdash e_0 : \tau'$ *and* $\vdash e_1 : \tau'$, *and that the property holds for* E'.

  Since $\vdash E'[e_0]\ e : \tau$, *by inversion (i.e., rule* T-APP *is the only rule whose conclusion is an application expression), we must have that* $\vdash E'[e_0] : \tau'' \to \tau$ *for some type* τ'' *and* $\vdash e : \tau''$.

  By the inductive hypothesis (i.e., the property holds of E'), we have that $E'[e_1]$ has type $\tau'' \to \tau$. Using the typing rule T-APP, *we can conclude that* $\vdash E'[e_1]\ e : \tau$. *That is,* $\vdash E[e_1] : \tau$ *as required.*

- $E = v\ E'$.

  Assume $\vdash E[e_0] : \tau$ *and* $\vdash e_0 : \tau'$ *and* $\vdash e_1 : \tau'$, *and that the property holds for* E'.

  Since $\vdash v\ E'[e_0] : \tau$, *by inversion (i.e., rule* T-APP *is the only rule whose conclusion is an application expression), we must have that* $\vdash E'[e_0] : \tau''$ *for some type* τ'', *and* $\vdash v : \tau'' \to \tau$.

  By the inductive hypothesis (i.e., the property holds of E'), we have that $E'[e_1]$ has type τ''. Using the typing rule T-APP, *we can conclude that* $\vdash v\ E'[e_1] : \tau$. *That is,* $\vdash E[e_1] : \tau$ *as required.*

- $E = E' + e$.

  Assume $\vdash E[e_0] : \tau$ *and* $\vdash e_0 : \tau'$ *and* $\vdash e_1 : \tau'$, *and that the property holds for* E'.

  Since $\vdash E'[e_0] + e : \tau$, *by inversion (i.e., rule* T-ADD *is the only rule whose conclusion is an addition expression), we must have that* $\vdash E'[e_0] :$ ***int*** *and* $\vdash e :$ ***int***, *and that* $\tau =$ ***int***.

  By the inductive hypothesis (i.e., the property holds of E'*), we have that* $E'[e_1]$ *has type* ***int***. *Using the typing rule* T-ADD, *we can conclude that* $\vdash E'[e_1] + e : \tau$. *That is,* $\vdash E[e_1] : \tau$ *as required.*

- $E = v + E'$.

  Assume $\vdash E[e_0] : \tau$ *and* $\vdash e_0 : \tau'$ *and* $\vdash e_1 : \tau'$, *and that the property holds for* E'.

  Since $\vdash v + E'[e_0] : \tau$, *by inversion (i.e., rule* T-ADD *is the only rule whose conclusion is an addition expression), we must have that* $\vdash E'[e_0] :$ ***int*** *and* $\vdash v :$ ***int***, *and that* $\tau =$ ***int***.

  By the inductive hypothesis (i.e., the property holds of E'*), we have that* $E'[e_1]$ *has type* ***int***. *Using the typing rule* T-ADD, *we can conclude that* $\vdash v + E'[e_1] : \tau$. *That is,* $\vdash E[e_1] : \tau$ *as required.*

Since all these cases go though, using the inductive reasoning principle, we can conclude that the property holds for all contexts. That is exactly the lemma we were trying to prove.

3 Products and Sums

For these questions, use the lambda calculus with products and sums (Lecture 13§1.1).

(a) Write a program that constructs two values of type $\textbf{int} + (\textbf{int} \to \textbf{int})$, one using left injection, and one using right injection.

Answer:

$$\textit{let}\ a : \textbf{\textit{int}} + (\textbf{\textit{int}} \to \textbf{\textit{int}}) = \textit{inl}_{\textbf{\textit{int}} + (\textbf{\textit{int}} \to \textbf{\textit{int}})}\ 3\ \textit{in}$$
$$\textit{inr}_{\textbf{\textit{int}} + (\textbf{\textit{int}} \to \textbf{\textit{int}})}\ \lambda x : \textbf{\textit{int}}.\ 3$$

(b) Write a function that takes a value of type $\textbf{int} + (\textbf{int} \to \textbf{int})$ and if the value is an integer, it adds 7 to it, and if the value is a function it applies the function to 42.

Answer:

$$\lambda a : \textbf{\textit{int}} + (\textbf{\textit{int}} \to \textbf{\textit{int}}).\ \textit{case}\ a\ \textit{of}\ \lambda y : \textbf{\textit{int}}.\ y + 7 \mid \lambda f : \textbf{\textit{int}} \to \textbf{\textit{int}}.\ f\ 42$$

(c) Give a typing derivation for the following program.

$$\lambda p : (\textbf{unit} \to \textbf{int}) \times (\textbf{int} \to \textbf{int}).\ \lambda x : \textbf{unit} + \textbf{int}.\ \textsf{case}\ x\ \textsf{of}\ \#1\ p \mid \#2\ p$$

Answer: *For brevity, let* $e_1 \equiv \lambda x : \textbf{\textit{unit}} + \textbf{\textit{int}}.\ \textit{case}\ x\ \textit{of}\ \#1\ p \mid \#2\ p$ *and let* $\Gamma = \{p : (\textbf{\textit{unit}} \to \textbf{\textit{int}}) \times (\textbf{\textit{int}} \to \textbf{\textit{int}}), x : \textbf{\textit{unit}} + \textbf{\textit{int}}\}$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash x : \textbf{\textit{unit}} + \textbf{\textit{int}}}\ \text{T-VAR} \quad \dfrac{\dfrac{}{\Gamma \vdash p : (\textbf{\textit{unit}} \to \textbf{\textit{int}} \times \textbf{\textit{int}} \to \textbf{\textit{int}})}\ \text{T-VAR}}{\Gamma \vdash \#1\ p : \textbf{\textit{unit}} \to \textbf{\textit{int}}}\ \text{T-LPROJ} \quad \dfrac{\dfrac{}{\Gamma \vdash p : (\textbf{\textit{unit}} \to \textbf{\textit{int}} \times \textbf{\textit{int}} \to \textbf{\textit{int}})}\ \text{T-VAR}}{\Gamma \vdash \#2\ p : \textbf{\textit{int}} \to \textbf{\textit{int}}}\ \text{T-RPROJ}}{\Gamma \vdash \textit{case}\ x\ \textit{of}\ \#1\ p \mid \#2\ p : \textbf{\textit{int}}}\ \text{T-CASE}}{p : (\textbf{\textit{unit}} \to \textbf{\textit{int}}) \times (\textbf{\textit{int}} \to \textbf{\textit{int}}) \vdash \lambda x : \textbf{\textit{unit}} + \textbf{\textit{int}}.\ \textit{case}\ x\ \textit{of}\ \#1\ p \mid \#2\ p : (\textbf{\textit{unit}} + \textbf{\textit{int}}) \to \textbf{\textit{int}}}\ \text{T-ABS}}{\vdash \lambda p : (\textbf{\textit{unit}} \to \textbf{\textit{int}}) \times (\textbf{\textit{int}} \to \textbf{\textit{int}}).\ e_1 : ((\textbf{\textit{unit}} \to \textbf{\textit{int}}) \times (\textbf{\textit{int}} \to \textbf{\textit{int}})) \to (\textbf{\textit{unit}} + \textbf{\textit{int}}) \to \textbf{\textit{int}}}\ \text{T-ABS}$$

(d) Write a program that uses the term in part (c) above to produce the value 42.

Answer: *We refer to the term in part (c) above as f.*

$$f\ (\lambda x : \textbf{\textit{unit}} \to \textbf{\textit{int}}.\ 42, \lambda x : \textbf{\textit{int}}.\ 41)\ \textit{inl}_{\textbf{\textit{unit}}+\textbf{\textit{int}}}\ ()$$

4 Recursion

(a) Use the $\mu x.\ e$ expression to write a function that takes a natural number n and returns the sum of all even natural numbers less than or equal to n. (You can assume you have appropriate integer comparison operators, and also a modulus operator.)

Answer:

$$\mu f.\ \lambda n.\ \textbf{if}\ n \leq 0\ \textbf{then}\ 0\ \textbf{else if}\ (n\ \textit{mod}\ 2) = 0\ \textbf{then}\ n + f\ (n-2)\ \textbf{else}\ f\ (n-1)$$

(b) Try executing your program by applying it to the number 5.

Answer: *The program executes correctly and returns 6. For brevity, we will refer to the expression from the answer above as F.*

$$\begin{aligned}
&F\ 5\\
\longrightarrow&(\lambda n.\ \textbf{if}\ n \leq 0\ \textbf{then}\ 0\ \textbf{else if}\ (n\ \textit{mod}\ 2) = 0\ \textbf{then}\ n + F\ (n-2)\ \textbf{else}\ F\ (n-1))\ 5\\
\longrightarrow&\textbf{if}\ 5 \leq 0\ \textbf{then}\ 0\ \textbf{else if}\ (5\ \textit{mod}\ 2) = 0\ \textbf{then}\ 5 + F\ (5-2)\ \textbf{else}\ F\ (5-1)\\
\longrightarrow&\textbf{if}\ \textit{false}\ \textbf{then}\ 0\ \textbf{else if}\ (5\ \textit{mod}\ 2) = 0\ \textbf{then}\ 5 + F\ (5-2)\ \textbf{else}\ F\ (5-1)\\
\longrightarrow&\textbf{if}\ (5\ \textit{mod}\ 2) = 0\ \textbf{then}\ 5 + F\ (5-2)\ \textbf{else}\ F\ (5-1)\\
\longrightarrow&\textbf{if}\ 1 = 0\ \textbf{then}\ 5 + F\ (5-2)\ \textbf{else}\ F\ (5-1)\\
\longrightarrow&\textbf{if}\ \textit{false}\ \textbf{then}\ 5 + F\ (5-2)\ \textbf{else}\ F\ (5-1)\\
\longrightarrow&F\ (5-1)\\
\longrightarrow&(\lambda n.\ \textbf{if}\ n \leq 0\ \textbf{then}\ 0\ \textbf{else if}\ (n\ \textit{mod}\ 2) = 0\ \textbf{then}\ n + F\ (n-2)\ \textbf{else}\ F\ (n-1))\ (5-1)\\
\longrightarrow&(\lambda n.\ \textbf{if}\ n \leq 0\ \textbf{then}\ 0\ \textbf{else if}\ (n\ \textit{mod}\ 2) = 0\ \textbf{then}\ n + F\ (n-2)\ \textbf{else}\ F\ (n-1))\ 4\\
\longrightarrow&\textbf{if}\ 4 \leq 0\ \textbf{then}\ 0\ \textbf{else if}\ (4\ \textit{mod}\ 2) = 0\ \textbf{then}\ 4 + F\ (4-2)\ \textbf{else}\ F\ (4-1)\\
\longrightarrow&\textbf{if}\ \textit{false}\ \textbf{then}\ 0\ \textbf{else if}\ (4\ \textit{mod}\ 2) = 0\ \textbf{then}\ 4 + F\ (4-2)\ \textbf{else}\ F\ (4-1)\\
\longrightarrow&\textbf{if}\ (4\ \textit{mod}\ 2) = 0\ \textbf{then}\ 4 + F\ (4-2)\ \textbf{else}\ F\ (4-1)\\
\longrightarrow&\textbf{if}\ 0 = 0\ \textbf{then}\ 4 + F\ (4-2)\ \textbf{else}\ F\ (4-1)\\
\longrightarrow&\textbf{if}\ \textit{true}\ \textbf{then}\ 4 + F\ (4-2)\ \textbf{else}\ F\ (4-1)\\
\longrightarrow&4 + F\ (4-2)\\
\longrightarrow&4 + (\lambda n.\ \textbf{if}\ n \leq 0\ \textbf{then}\ 0\ \textbf{else if}\ (n\ \textit{mod}\ 2) = 0\ \textbf{then}\ n + F\ (n-2)\ \textbf{else}\ F\ (n-1))\ (4-2)\\
\longrightarrow&4 + (\lambda n.\ \textbf{if}\ n \leq 0\ \textbf{then}\ 0\ \textbf{else if}\ (n\ \textit{mod}\ 2) = 0\ \textbf{then}\ n + F\ (n-2)\ \textbf{else}\ F\ (n-1))\ 2\\
\longrightarrow&4 + (\textbf{if}\ 2 \leq 0\ \textbf{then}\ 0\ \textbf{else if}\ (2\ \textit{mod}\ 2) = 0\ \textbf{then}\ 2 + F\ (2-2)\ \textbf{else}\ F\ (2-1))\\
\longrightarrow&4 + (\textbf{if}\ \textit{false}\ \textbf{then}\ 0\ \textbf{else if}\ (2\ \textit{mod}\ 2) = 0\ \textbf{then}\ 2 + F\ (2-2)\ \textbf{else}\ F\ (2-1))\\
\longrightarrow&4 + (\textbf{if}\ (2\ \textit{mod}\ 2) = 0\ \textbf{then}\ 2 + F\ (2-2)\ \textbf{else}\ F\ (2-1))\\
\longrightarrow&4 + (\textbf{if}\ 0 = 0\ \textbf{then}\ 2 + F\ (2-2)\ \textbf{else}\ F\ (2-1))
\end{aligned}$$

$\longrightarrow 4 + (\textbf{if } \textit{true} \textbf{ then } 2 + F\ (2-2) \textbf{ else } F\ (2-1))$
$\longrightarrow 4 + (2 + F\ (2-2))$
$\longrightarrow 4 + (2 + (\lambda n.\, \textbf{if } n \leq 0 \textbf{ then } 0 \textbf{ else if } (n \textit{ mod } 2) = 0 \textbf{ then } n + F\ (n-2) \textbf{ else } F\ (n-1))\ (2-2))$
$\longrightarrow 4 + (2 + (\lambda n.\, \textbf{if } n \leq 0 \textbf{ then } 0 \textbf{ else if } (n \textit{ mod } 2) = 0 \textbf{ then } n + F\ (n-2) \textbf{ else } F\ (n-1))\ 0)$
$\longrightarrow 4 + (2 + (\lambda n.\, \textbf{if } n \leq 0 \textbf{ then } 0 \textbf{ else if } (n \textit{ mod } 2) = 0 \textbf{ then } n + F\ (n-2) \textbf{ else } F\ (n-1))\ 0)$
$\longrightarrow 4 + (2 + (\textbf{if } 0 \leq 0 \textbf{ then } 0 \textbf{ else if } (0 \textit{ mod } 2) = 0 \textbf{ then } 0 + F\ (0-2) \textbf{ else } F\ (0-1)))$
$\longrightarrow 4 + (2 + (\textbf{if } \textit{true} \textbf{ then } 0 \textbf{ else if } (0 \textit{ mod } 2) = 0 \textbf{ then } 0 + F\ (0-2) \textbf{ else } F\ (0-1)))$
$\longrightarrow 4 + (2 + (0))$
$\longrightarrow^* 6$

(c) Give a typing derivation for the following program. What happens if you execute the program?

$$\mu p\colon(\textbf{int} \to \textbf{int}) \times (\textbf{int} \to \textbf{int}).\ (\lambda n\colon\textbf{int}.\, n+1, \#1\ p)$$

Answer: *For brevity, we write τ_p for the type $(\textbf{\textit{int}} \to \textbf{\textit{int}}) \times (\textbf{\textit{int}} \to \textbf{\textit{int}})$.*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{T-VAR}\ \dfrac{}{p\colon\tau_p, n\colon\textbf{\textit{int}} \vdash n\colon\textbf{\textit{int}}} \quad \text{T-INT}\ \dfrac{}{p\colon\tau_p, n\colon\textbf{\textit{int}} \vdash 1\colon\textbf{\textit{int}}}}{\text{T-SUM}\ p\colon\tau_p, n\colon\textbf{\textit{int}} \vdash n+1\colon\textbf{\textit{int}}}}{\text{T-ABS}\ p\colon\tau_p \vdash \lambda n\colon\textbf{\textit{int}}.\, n+1\colon\textbf{\textit{int}} \to \textbf{\textit{int}}} \quad \text{T-PROJ}\ \dfrac{\text{T-VAR}\ \dfrac{}{p\colon\tau_p \vdash p\colon\tau_p}}{p\colon\tau_p \vdash \#1\ p\colon\textbf{\textit{int}} \to \textbf{\textit{int}}}}{\text{T-PAIR}\ p\colon\tau_p \vdash (\lambda n\colon\textbf{\textit{int}}.\, n+1, \#1\ p)\colon\tau_p}}{\text{T-REC}\ \vdash \mu p\colon\tau_p.\, (\lambda n\colon\textbf{\textit{int}}.\, n+1, \#1\ p)\colon\tau_p}$$

Now, if you actually tried to execute this expression under a Call-By-Name semantics, it would unfold the recursive expression to $(\lambda n\colon\textbf{\textit{int}}.\, n+1, \#1\ P)$, where P is the recursive expression $\mu p\colon(\textbf{\textit{int}} \to \textbf{\textit{int}}) \times (\textbf{\textit{int}} \to \textbf{\textit{int}}).\ (\lambda n\colon\textbf{\textit{int}}.\, n+1, \#1\ p)$. While the first element of the pair is a value, the second $\#2\ P$ is not, and so we would attempt to evaluate that expression. However, that requires evaluating the expression $P \equiv \mu p\colon(\textbf{\textit{int}} \to \textbf{\textit{int}}) \times (\textbf{\textit{int}} \to \textbf{\textit{int}}).\ (\lambda n\colon\textbf{\textit{int}}.\, n+1, \#1\ p)$.

So, under Call-by-Name semantics, the program will not terminate.

5 References

(a) Give a typing derivation for the following program.

$$\begin{array}{l}\mathsf{let}\ a\colon\textbf{int ref} = \mathsf{ref}\ 4\ \mathsf{in}\\ \mathsf{let}\ b\colon(\textbf{int} \to \textbf{int})\ \textbf{ref} = \mathsf{ref}\ \lambda x\colon\textbf{int}.\, x + 38\ \mathsf{in}\\ !b\ !a\end{array}$$

Answer: *For brevity, we will write e for the expression above, and e_b for the subexpression $\textbf{\textit{let}}\ b : (\textbf{\textit{int}} \to \textbf{\textit{int}})\ \textbf{\textit{ref}} = \textit{ref}\ \lambda x\colon\textbf{\textit{int}}.\, x + 38\ \textit{in}\ !b\ !a$*

$$\dfrac{\dfrac{\text{T-INT}\ \dfrac{}{\vdash 4\colon\textbf{\textit{int}}}}{\text{T-ALLOC}\ \vdash \textit{ref}\ 4\colon\textbf{\textit{int ref}}} \quad \text{T-LET}\ \dfrac{\text{T-ALLOC}\ \dfrac{\text{T-ABS}\ \dfrac{\vdots\ 1}{a\colon\textbf{\textit{int ref}}, x\colon\textbf{\textit{int}} \vdash x + 38\colon\textbf{\textit{int}}}}{a\colon\textbf{\textit{int ref}} \vdash \lambda x\colon\textbf{\textit{int}}.\, x + 38\colon\textbf{\textit{int}} \to \textbf{\textit{int}}} \quad \dfrac{\vdots\ 2}{a\colon\textbf{\textit{int ref}}, b\colon(\textbf{\textit{int}} \to \textbf{\textit{int}})\ \textbf{\textit{ref}} \vdash !b\ !a\colon\textbf{\textit{int}}}}{a\colon\textbf{\textit{int ref}} \vdash e_b\colon\textbf{\textit{int}}}}{\text{T-LET}\ \vdash e\colon\textbf{\textit{int}}}$$

The subderivation marked $\vdots_1$ is:

$$\text{T-ADD}\ \dfrac{\text{T-VAR}\ \dfrac{}{a : \textbf{\textit{int ref}}, x : \textbf{\textit{int}} \vdash x : \textbf{\textit{int}}} \qquad \text{T-INT}\ \dfrac{}{a : \textbf{\textit{int ref}}, x : \textbf{\textit{int}} \vdash 38 : \textbf{\textit{int}}}}{a : \textbf{\textit{int ref}}, x : \textbf{\textit{int}} \vdash x + 38 : \textbf{\textit{int}}}$$

The subderivation marked $\vdots_2$ is:

$$\text{T-APP}\ \dfrac{\text{T-DEREF}\ \dfrac{\text{T-VAR}\ \dfrac{}{\Gamma_{ab} \vdash b : (\textbf{\textit{int}} \to \textbf{\textit{int}})\ \textbf{\textit{ref}}}}{\Gamma_{ab} \vdash\ !b : \textbf{\textit{int}} \to \textbf{\textit{int}}} \qquad \text{T-DEREF}\ \dfrac{\text{T-VAR}\ \dfrac{}{\Gamma_{ab} \vdash a : \textbf{\textit{int ref}}}}{\Gamma_{ab} \vdash\ !a : \textbf{\textit{int}}}}{\Gamma_{ab} \vdash\ !b\ !a : \textbf{\textit{int}}}$$

where $\Gamma_{ab} = a : \textbf{\textit{int ref}}, b : (\textbf{\textit{int}} \to \textbf{\textit{int}})\ \textbf{\textit{ref}}$.

(b) Execute the program above for 4 small steps, to get configuration $\langle e, \sigma \rangle$. What is an appropriate Σ such that $\emptyset, \Sigma \vdash e : \tau$ and $\Sigma \vdash \sigma$?

Answer:

$$\begin{aligned}
&\langle \textit{let}\ a : \textbf{\textit{int ref}} = \textit{ref}\ 4\ \textit{in let}\ b : (\textbf{\textit{int}} \to \textbf{\textit{int}})\ \textbf{\textit{ref}} = \textit{ref}\ \lambda x : \textbf{\textit{int}}.\ x + 38\ \textit{in}\ !b\ !a, \emptyset \rangle \\
\longrightarrow &\langle \textit{let}\ a : \textbf{\textit{int ref}} = \ell_a\ \textit{in let}\ b : (\textbf{\textit{int}} \to \textbf{\textit{int}})\ \textbf{\textit{ref}} = \textit{ref}\ \lambda x : \textbf{\textit{int}}.\ x + 38\ \textit{in}\ !b\ !a, [\ell_a \mapsto 4] \rangle \\
\longrightarrow &\langle \textit{let}\ b : (\textbf{\textit{int}} \to \textbf{\textit{int}})\ \textbf{\textit{ref}} = \textit{ref}\ \lambda x : \textbf{\textit{int}}.\ x + 38\ \textit{in}\ !b\ !\ell_a, [\ell_a \mapsto 4] \rangle \\
\longrightarrow &\langle \textit{let}\ b : (\textbf{\textit{int}} \to \textbf{\textit{int}})\ \textbf{\textit{ref}} = \ell_b\ \textit{in}\ !b\ !\ell_a, [\ell_a \mapsto 4, \ell_b \mapsto \lambda x : \textbf{\textit{int}}.\ x + 38] \rangle \\
\longrightarrow &\langle !\ell_b\ !\ell_a, [\ell_a \mapsto 4, \ell_b \mapsto \lambda x : \textbf{\textit{int}}.\ x + 38] \rangle
\end{aligned}$$

An appropriate store typing context is $\Sigma = \ell_a \mapsto \textbf{\textit{int}}, \ell_b \mapsto \textbf{\textit{int}} \to \textbf{\textit{int}}$.