CS 599 D1: Mid-Term
Total: 80 pts

1 Type Safety of LL1 [20 pts]

Recall the LL1 language.

$$\begin{array}{rl} \text{Expressions} & e ::= \mathsf{true} \mid \mathsf{false} \mid \mathsf{if}\ e\ \mathsf{then}\ e\ \mathsf{else}\ e \mid \overline{n} \mid e + e \mid \mathsf{let}\ x : \tau = e\ \mathsf{in}\ e \mid x \\ \text{Types} & \tau ::= \mathsf{bool} \mid \mathsf{int} \end{array}$$

Type System

$$\frac{}{\Gamma \vdash \mathsf{true} : \mathsf{bool}}\ \text{TT} \qquad \frac{}{\Gamma \vdash \mathsf{false} : \mathsf{bool}}\ \text{FF} \qquad \frac{\Gamma \vdash e : \mathsf{bool} \quad \Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau}\ \textsc{If}$$

$$\frac{}{\Gamma \vdash \overline{n} : \mathsf{int}}\ \textsc{Num} \qquad \frac{\Gamma \vdash e_1 : \mathsf{int} \quad \Gamma \vdash e_2 : \mathsf{int}}{\Gamma \vdash e_1 + e_2 : \mathsf{int}}\ \textsc{Add} \qquad \frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma, x : \tau_1 \vdash e_2 : \tau}{\Gamma \vdash \mathsf{let}\ x : \tau_1 = e_1\ \mathsf{in}\ e_2 : \tau}\ \textsc{Let}$$

$$\frac{}{\Gamma, x : \tau \vdash x : \tau}\ \textsc{Var}$$

Semantics

$$\frac{}{\mathsf{true\ value}}\ \text{TT-V} \qquad \frac{}{\mathsf{false\ value}}\ \text{FF-V} \qquad \frac{e \mapsto e'}{\mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 \mapsto \mathsf{if}\ e'\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2}\ \textsc{If-S}$$

$$\frac{}{\mathsf{if\ true\ then}\ e_1\ \mathsf{else}\ e_2 \mapsto e_1}\ \textsc{If-T} \qquad \frac{}{\mathsf{if\ false\ then}\ e_1\ \mathsf{else}\ e_2 \mapsto e_2}\ \textsc{If-F} \qquad \frac{}{\overline{n}\ \mathsf{value}}\ \textsc{Num-V}$$

$$\frac{e_1 \mapsto e_1'}{e_1 + e_2 \mapsto e_1' + e_2}\ \textsc{Add-L} \qquad \frac{v_1\ \mathsf{value} \quad e_2 \mapsto e_2'}{v_1 + e_2 \mapsto v_1 + e_2'}\ \textsc{Add-R} \qquad \frac{}{\overline{v_1} + \overline{v_2} \mapsto \overline{v_1 \oplus v_2}}\ \textsc{Add-V}$$

$$\frac{e_1 \mapsto e_1'}{\mathsf{let}\ x : \tau = e_1\ \mathsf{in}\ e_2 \mapsto \mathsf{let}\ x : \tau = e_1'\ \mathsf{in}\ e_2}\ \textsc{Let-S} \qquad \frac{v\ \mathsf{value}}{\mathsf{let}\ x : \tau = v\ \mathsf{in}\ e_2 \mapsto [v/x]e_2}\ \textsc{Let-V}$$

As you have seen in the lectures and assignments, this language is type safe, i.e., it satisfies progress and preservation.

Theorem 1 (Preservation) *If $\Gamma \vdash e : \tau$ and $e \mapsto e'$, then $\Gamma \vdash e' : \tau$.*

Theorem 2 (Progress) *If $\cdot \vdash e : \tau$ then either $e \mapsto e'$ or e value.*

For the next set of problems, we will change some rules of the type system or semantics and you need to determine if the language is still type safe. **If the language is type safe, give an intuitive explanation of why (you don't need to prove the theorems). If the language is not type safe, clearly state whether the language violates progress, preservation, or both. Also, give an example expression for each of the violated theorems and explain how the theorem is violated.**

Problem 1 (5 pts) *What happens if the* If *typing rule is defined as follows instead?*

$$\frac{\Gamma \vdash e : \mathsf{bool} \quad \Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash \mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \tau_1}\ \textsc{If}$$

Solution. Preservation theorem is violated. Consider the expression $e = \textsf{if false then } 5 \textsf{ else true}$. Due to the If rule, $e : \textsf{int}$. But, $e \mapsto \textsf{true}$ and $\textsf{true} : \textsf{bool}$. Progress theorem holds.

Problem 2 (5 pts) *What happens if the* FF *typing rule is defined as follows instead?*

$$\frac{}{\Gamma \vdash \textsf{false} : \textsf{int}}\ \text{FF}$$

Solution. Progress theorem is violated. Consider $e = \textsf{false} + 2$. Due to the modified FF rule, $e : \textsf{int}$. But there are no rules to step e. Hence, progress is violated. Preservation is not violated.

Problem 3 (5 pts) *What happens if the* Num *typing rule is defined as follows instead*

$$\frac{}{\Gamma \vdash \overline{n} : \textsf{bool}}\ \text{Num}$$

and we add one more rule to the semantics?

$$\frac{}{\textsf{if } \overline{n} \textsf{ then } e_1 \textsf{ else } e_2 \mapsto e_1}\ \text{If-N}$$

Solution. Neither progress or preservation is violated. The only expression that depends on booleans is the if expression. Consider $\textsf{if } e \textsf{ then } e_1 \textsf{ else } e_2$. Either e evaluates to true, false, or $\overline{n}$. Since $\textsf{if } \overline{n} \textsf{ then } e_1 \textsf{ else } e_2 \mapsto e_1$, progress holds. Since $\textsf{if } \overline{n} \textsf{ then } e_1 \textsf{ else } e_2 : \tau$ implies that $e_1 : \tau$, preservation holds.

Problem 4 (5 pts) *What happens if the* Var *typing rule is defined as follows instead?*

$$\frac{}{\Gamma, x : \tau_1 \vdash x : \tau_2}\ \text{Var}$$

Solution. Preservation theorem is violated. Consider $e = \textsf{let } x = 5 \textsf{ in } x$. Using the Var rule,

$$\frac{\cdot \vdash 5 : \textsf{int} \qquad \dfrac{}{x : \textsf{int} \vdash x : \textsf{bool}}\ \text{Var}}{\cdot \vdash \textsf{let } x = 5 \textsf{ in } x : \textsf{bool}}\ \text{Let}$$

But $e \mapsto [5/x]x = 5$ and $5 : \textsf{int}$. Progress is not violated.

2 Two Counter Machine via Linear Inference [20 pts]

In this section, we will solve the 'Two Counter Machine' problem using rules of linear inference. We have borrowed the definition of the Counter Machine from Wikipedia and simplified it for convenience. The machine consists of the following instructions:

1. $\textsf{INC}(r)$: INCrement the contents of register r and continue in sequence.
2. $\textsf{DEC}(r)$: DECrement the contents of register r and continue in sequence.
3. $\textsf{JZ}(r, j)$: IF register r contains zero THEN Jump to instruction j ELSE continue in sequence.
4. $\textsf{HALT}$: HALT computation.

A two counter machine program is represented as a sequence of instructions identified by an index. Execution starts with instruction 0 and two registers r_1 and r_2 containing value 0. For e.g.,

```
0: INC(r1)
1: INC(r2)
2: DEC(r1)
3: JZ(r1, 0)
4: HALT
```

We will encode the two counter machine using 3 propositions: $\mathsf{reg1}(m)$, $\mathsf{reg2}(n)$, and $\mathsf{pc}(i)$. The proposition $\mathsf{reg1}(m)$ means that register r_1 contains number m; the proposition $\mathsf{reg2}(n)$ means that register r_2 contains number n, and $\mathsf{pc}(i)$ means that the program is currently executing instruction at index i.

In the following problems, you will describe the rules of linear inference for a generic two counter machine. Each rule will define how the 3 propositions are updated for each instruction.

Problem 5 (3 pts) *What is the initial state for this problem?*

Solution.

$$\mathsf{reg1}(0) \quad \mathsf{reg2}(0) \quad \mathsf{pc}(0)$$

Problem 6 (4 pts) *Consider an arbitrary instruction at index i. What are the possible instructions for this index?*

Solution. The possibilities for i-th instruction are

1. $i : \mathsf{INC}(r_1)$
2. $i : \mathsf{INC}(r_2)$
3. $i : \mathsf{DEC}(r_1)$
4. $i : \mathsf{DEC}(r_2)$
5. $i : \mathsf{JZ}(r_1, j)$
6. $i : \mathsf{JZ}(r_2, j)$
7. $i : \mathsf{HALT}$

Problem 7 (7 pts) *For each of the instructions you defined above, write the corresponding inference rule and provide a unique name to each rule. For the* HALT *instruction, your final state should produce the final values of the registers. Your inference rules should not include the instruction in the premise or the conclusion. They will simply be an external side condition. In other words, you can write inference rules as:*
If the i-th instruction is ..., then the corresponding inference rule is ....

Solution.

1. If $i : \mathsf{INC}(r_1)$,
$$\frac{\mathsf{reg1}(m) \qquad \mathsf{pc}(i)}{\mathsf{reg1}(m+1) \qquad \mathsf{pc}(i+1)}\ \text{INC1}$$
2. If $i : \mathsf{INC}(r_2)$,
$$\frac{\mathsf{reg2}(n) \qquad \mathsf{pc}(i)}{\mathsf{reg2}(n+1) \qquad \mathsf{pc}(i+1)}\ \text{INC2}$$
3. If $i : \mathsf{DEC}(r_1)$,
$$\frac{\mathsf{reg1}(m) \qquad \mathsf{pc}(i)}{\mathsf{reg1}(m-1) \qquad \mathsf{pc}(i+1)}\ \text{DEC1}$$
4. If $i : \mathsf{DEC}(r_2)$,
$$\frac{\mathsf{reg2}(n) \qquad \mathsf{pc}(i)}{\mathsf{reg2}(n-1) \qquad \mathsf{pc}(i+1)}\ \text{DEC2}$$
5. If $i : \mathsf{JZ}(r_1, j)$,
$$\frac{m \neq 0 \qquad \mathsf{reg1}(m) \qquad \mathsf{pc}(i)}{\mathsf{reg1}(m) \qquad \mathsf{pc}(i+1)}\ \text{JZ1-POS} \qquad\qquad \frac{\mathsf{reg1}(0) \qquad \mathsf{pc}(i)}{\mathsf{reg1}(0) \qquad \mathsf{pc}(j)}\ \text{JZ1-ZERO}$$

6. If $i : \mathsf{JZ}(r_2, j)$,

$$\dfrac{n \neq 0 \qquad \mathsf{reg2}(n) \qquad \mathsf{pc}(i)}{\mathsf{reg2}(n) \qquad \mathsf{pc}(i+1)}\ \text{JZ2-POS} \qquad\qquad \dfrac{\mathsf{reg2}(0) \qquad \mathsf{pc}(i)}{\mathsf{reg2}(0) \qquad \mathsf{pc}(j)}\ \text{JZ1-ZERO}$$

7. If $i : \mathsf{HALT}$,

$$\dfrac{\mathsf{reg1}(m) \qquad \mathsf{reg2}(n) \qquad \mathsf{pc}(i)}{\mathsf{reg1}(m) \qquad \mathsf{reg2}(n)}\ \text{HALT}$$

Problem 8 (6 pts) *Now, consider the example program shown earlier in Section 2. Apply the rules of inference (with their names) you defined above to this program. Does the program ever reach the* HALT *instruction? Why or why not?*

Solution.

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{reg1}(0) \qquad \mathsf{reg2}(0) \qquad \mathsf{pc}(0)}{\mathsf{reg1}(1) \qquad \mathsf{reg2}(0) \qquad \mathsf{pc}(1)}\ \text{INC1}}{\mathsf{reg1}(1) \qquad \mathsf{reg2}(1) \qquad \mathsf{pc}(2)}\ \text{INC2}}{\mathsf{reg1}(0) \qquad \mathsf{reg2}(1) \qquad \mathsf{pc}(3)}\ \text{DEC1}}{\mathsf{reg1}(0) \qquad \mathsf{reg2}(1) \qquad \mathsf{pc}(0)}\ \text{JZ1-ZERO}$$

Since we end up with $\mathsf{reg1}(0)$, we will again loop through this sequence and so, will never reach the HALT instruction.

3 Linear Proofs [20 pts]

Problem 9 (20 pts) *For the following problems, determine if the proposition is provable or not. If it is provable, show the derivation and construct the corresponding session-typed proof term by giving appropriate channel names to each proposition. If it is not provable, briefly explain why.*

To provide a session-typed term for types of the form $A \oplus B$ and $A \,\&\, B$, consider adding the following labels: Instead of using $A \oplus B$, use $\oplus\{\mathsf{left} : A, \mathsf{right} : B\}$. Similarly, instead of using $A \,\&\, B$, use $\&\{\mathsf{left} : A, \mathsf{right} : B\}$.

Solution.
(i) $A \multimap (B \,\&\, C) \vdash (A \multimap B) \,\&\, (A \multimap C)$ **[7 pts]**

$$\dfrac{\dfrac{\dfrac{\dfrac{x : B \vdash y \leftrightarrow x :: (y : B)}{\begin{array}{c} x : B \,\&\, C \vdash \\ x.\mathsf{left}\,;\, y \leftrightarrow x \\ :: (y : B)\end{array}}\ \&\text{L}}{\begin{array}{c} z : A, x : A \multimap (B \,\&\, C) \vdash \\ \mathsf{send}\ x\ z\,;\, x.\mathsf{left}\,;\, y \leftrightarrow x \\ :: (y : B)\end{array}}\ \otimes\text{L}}{\begin{array}{c} x : A \multimap (B \,\&\, C) \vdash \\ z \leftarrow \mathsf{recv}\ y\,;\, \mathsf{send}\ x\ z\,;\, x.\mathsf{left}\,;\, y \leftrightarrow x \\ :: (y : A \multimap B)\end{array}}\ \multimap\text{R} \qquad \dfrac{\dfrac{\dfrac{x : B \vdash y \leftrightarrow x :: (y : B)}{\begin{array}{c} x : B \,\&\, C \vdash \\ x.\mathsf{right}\,;\, y \leftrightarrow x \\ :: (y : B)\end{array}}\ \&\text{L}}{\begin{array}{c} z : A, x : A \multimap (B \,\&\, C) \vdash \\ \mathsf{send}\ x\ z\,;\, x.\mathsf{right}\,;\, y \leftrightarrow x \\ :: (y : B)\end{array}}\ \otimes\text{L}}{\begin{array}{c} x : A \multimap (B \,\&\, C) \vdash \\ z \leftarrow \mathsf{recv}\ y\,;\, \mathsf{send}\ x\ z\,;\, x.\mathsf{left}\,;\, y \leftrightarrow x \\ :: (y : A \multimap C)\end{array}}\ \multimap\text{R}}{\begin{array}{c} x : A \multimap (B \,\&\, C) \vdash \\ \mathsf{case}\ y\ (\mathsf{left} \Rightarrow z \leftarrow \mathsf{recv}\ y\,;\, \mathsf{send}\ x\ z\,;\, x.\mathsf{left}\,;\, y \leftrightarrow x \mid \mathsf{right} \Rightarrow z \leftarrow \mathsf{recv}\ y\,;\, \mathsf{send}\ x\ z\,;\, x.\mathsf{left}\,;\, y \leftrightarrow x) \\ :: y : (A \multimap B) \,\&\, (A \multimap C)\end{array}}\ \&\text{L}$$

(ii) $(A \multimap C) \mathbin{\&} (B \multimap C) \vdash (A \oplus B) \multimap C$ **[7 pts]**

$$
\dfrac{
\dfrac{
\dfrac{x : B \vdash y \leftrightarrow x :: (y : B)}{\begin{array}{c} z : A, x : (A \multimap C) \vdash \\ \mathsf{send}\ x\ z\ ;\ y \leftrightarrow x \\ :: (y : B) \end{array}} \multimap\mathrm{L}
}{\begin{array}{c} z : A, x : (A \multimap C) \mathbin{\&} (B \multimap C) \vdash \\ x.\mathsf{left}\ ;\ \mathsf{send}\ x\ z\ ;\ y \leftrightarrow x \\ :: (y : C) \end{array}} \mathbin{\&}\mathrm{L}
\qquad
\dfrac{
\dfrac{x : C \vdash (y \leftrightarrow x) :: (y : C)}{\begin{array}{c} z : B, x : B \multimap C \vdash \\ \mathsf{send}\ x\ z\ ;\ y \leftrightarrow x \\ :: (y : B) \end{array}} \multimap\mathrm{L}
}{\begin{array}{c} z : B, x : (A \multimap C) \mathbin{\&} (B \multimap C) \vdash \\ x.\mathsf{right}\ ;\ \mathsf{send}\ x\ z\ ;\ y \leftrightarrow x \\ :: (y : C) \end{array}} \mathbin{\&}\mathrm{L}
}{
\dfrac{\begin{array}{c} z : A \oplus B, x : (A \multimap C) \mathbin{\&} (B \multimap C) \vdash \\ \mathsf{case}\ z\ (\mathsf{left} \Rightarrow x.\mathsf{left}\ ;\ \mathsf{send}\ x\ z\ ;\ y \leftrightarrow x \mid \mathsf{right} \Rightarrow x.\mathsf{right}\ ;\ \mathsf{send}\ x\ z\ ;\ y \leftrightarrow x) \\ :: (y : C) \end{array}}{\begin{array}{c} x : (A \multimap C) \mathbin{\&} (B \multimap C) \vdash \\ z \leftarrow \mathsf{recv}\ y\ ;\ \mathsf{case}\ z\ (\mathsf{left} \Rightarrow x.\mathsf{left}\ ;\ \mathsf{send}\ x\ z\ ;\ y \leftrightarrow x \mid \mathsf{right} \Rightarrow x.\mathsf{right}\ ;\ \mathsf{send}\ x\ z\ ;\ y \leftrightarrow x) \\ :: (y : (A \oplus B) \multimap C) \end{array}} \multimap\mathrm{R}
}
$$

(iii) $A \multimap (B \multimap C) \vdash (A \otimes B) \multimap C$ **[6 pts]**

$$
\dfrac{
\dfrac{
\dfrac{
\dfrac{x : C \vdash y \leftrightarrow x :: (y : C)}{z : B, x : B \multimap C \vdash \mathsf{send}\ x\ z\ ;\ y \leftrightarrow x :: (y : C)} \multimap\mathrm{L}
}{a : A, z : B, x : A \multimap (B \multimap C) \vdash \mathsf{send}\ x\ a\ ;\ \mathsf{send}\ x\ z\ ;\ y \leftrightarrow x :: (y : C)} \multimap\mathrm{L}
}{z : A \otimes B, x : A \multimap (B \multimap C) \vdash a \leftarrow \mathsf{recv}\ z\ ;\ \mathsf{send}\ x\ a\ ;\ \mathsf{send}\ x\ z\ ;\ y \leftrightarrow x :: (y : C)} \otimes\mathrm{L}
}{x : A \multimap (B \multimap C) \vdash z \leftarrow \mathsf{recv}\ y\ ;\ a \leftarrow \mathsf{recv}\ z\ ;\ \mathsf{send}\ x\ a\ ;\ \mathsf{send}\ x\ z\ ;\ y \leftrightarrow x :: (y : (A \otimes B) \multimap C)} \multimap\mathrm{R}
$$

4 Session-Typed Programming [20 pts]

The session type of list is defined as follows:

$$\mathsf{type}\ \mathtt{list}_A = \oplus\{\mathbf{cons} : A \otimes \mathtt{list}_A, \mathbf{nil} : 1\}$$

Problem 10 (15 pts) *Define a process called* `reverse` *with the following type*

$$\mathsf{decl}\ \mathtt{reverse} : (l : \mathtt{list}_A) \vdash (m : \mathtt{list}_A)$$

The process takes as input a list l and produces the elements at output list m in the reverse order, i.e., elements in m are in the reverse order of elements in l. You are welcome to define and use helper processes and types.

Solution.

```
decl nil : . |- (l : listA)
decl cons : (x : A), (t : listA) |- (l : listA)

proc l <- nil = l.nil ; close l
proc l <- cons x t = l.cons ; send l x ; l <-> t

decl reverseH : (l : listA), (k : listA) |- (m : listA)
proc m <- reverseH l k =
  case l (
    cons => x <- recv l ;
            kn <- cons x k ;
            m <- reverseH l kn
  |  nil => wait l ;
            m <-> k
  )

proc m <- reverse l = k <- nil ; m <- reverseH l k
```

Problem 11 (5 pts) *Give a process definition for* `arbitrary` *with the following type*

$$\mathsf{decl}\ \mathtt{arbitrary} : (x : A) \vdash (y : B)$$

without knowing the definitions of types A and B. In other words, the process should be well-typed for arbitrary types A and B.

Solution.

```
proc y <- arbitrary x =
  y <- arbitrary x
```

Session Type Syntax and Type System

Syntax

Expressions $P ::= x.\mathsf{k}\,;\, P \mid \mathsf{case}\ x\ (\ell \Rightarrow P_\ell)_{\ell \in L} \mid y \leftarrow \mathsf{recv}\ x\,;\, P \mid \mathsf{send}\ x\ y\,;\, P \mid \mathsf{wait}\ x\,;\, P \mid \mathsf{close}\ x$
$\mid\ x \leftrightarrow y \mid x \leftarrow f\ \overline{y}\,;\, P$

Types $A, B ::= \oplus\{\ell : A_\ell\}_{\ell \in L} \mid \&\{\ell : A_\ell\}_{\ell \in L} \mid A \otimes B \mid A \multimap B \mid \mathbf{1}$

Type System

$$\frac{(k \in L) \qquad \Delta \vdash P :: (x : A_k)}{\Delta \vdash (x.k\,;\, P) :: (x : \oplus\{\ell : A_\ell\}_{\ell \in L})}\ \oplus R \qquad \frac{(\forall \ell \in L) \qquad \Delta, x : A_\ell \vdash Q_\ell :: (z : C)}{\Delta, x : \oplus\{\ell : A_\ell\}_{\ell \in L} \vdash (\mathsf{case}\ x\ (\ell \Rightarrow Q_\ell)_{\ell \in L}) :: (z : C)}\ \oplus L$$

$$\frac{(\forall \ell \in L) \qquad \Delta \vdash P_\ell :: (x : A_\ell)}{\Delta \vdash (\mathsf{case}\ x\ (\ell \Rightarrow P_\ell)_{\ell \in L)} :: (x : \&\{\ell : A_\ell\}_{\ell \in L})}\ \& R \qquad \frac{(k \in L) \qquad \Delta, x : A_k \vdash Q :: (z : C)}{\Delta, x : \&\{\ell : A_\ell\}_{\ell \in L} \vdash (x.k\,;\, Q) :: (z : C)}\ \& L$$

$$\frac{\Delta \vdash P :: (x : B)}{\Delta, y : A \vdash (\mathsf{send}\ x\ y\,;\, P) :: (x : A \otimes B)}\ \otimes R \qquad \frac{\Delta, y : A, x : B \vdash Q :: (z : C)}{\Delta, x : A \otimes B \vdash (y \leftarrow \mathsf{recv}\ x\,;\, Q) :: (z : C)}\ \otimes L$$

$$\frac{\Delta, y : A \vdash P :: (x : B)}{\Delta \vdash (y \leftarrow \mathsf{recv}\ x\,;\, P) :: (x : A \multimap B)}\ \multimap R \qquad \frac{\Delta, x : B \vdash Q :: (z : C)}{\Delta, x : A \multimap B, y : A \vdash (\mathsf{send}\ x\ y\,;\, Q) :: (z : C)}\ \multimap L$$

$$\frac{}{\cdot \vdash (\mathsf{close}\ x) :: (x : \mathbf{1})}\ \mathbf{1}R \qquad \frac{\Delta \vdash Q :: (z : C)}{\Delta, x : \mathbf{1} \vdash (\mathsf{wait}\ x\,;\, Q) :: (z : C)}\ \mathbf{1}L \qquad \frac{}{x : A \vdash (y \leftrightarrow x) :: (y : A)}\ \mathsf{id}$$

$$\frac{\mathsf{decl}\ f : \overline{y' : A'} \vdash (x : A) \in \Sigma \qquad \Delta, x : A \vdash Q :: (z : C)}{\Delta, \overline{y : A'} \vdash (x \leftarrow f\ \overline{y}\,;\, Q) :: (z : C)}\ \mathsf{def}$$