1 Parametricity

Recall from class the following logical relation for System F (where we assume a call-by-value operational semantics for System F, as we did in class and in Homework 4).

Below, the metavariable $R$ ranges over relations on values (i.e., $R$ denotes a set of pairs of values $(v, v')$). The metavariable $\eta$ ranges over finite maps from type variables $\alpha$ to closed values $x$ to closed values $v$ (i.e., values with no free type or term variables).

The proof is by induction on the typing derivation $\Delta; \Gamma \vdash e : \tau$.

- Case: $\Delta; \Gamma \vdash x : \tau_1 \vdash e : \tau_2$ (ABS)

**Proof**

We are required to show that $\Delta; \Gamma \vdash \lambda x : \tau_1. e : \tau_1 \rightarrow \tau_2$.

Consider arbitrary $\eta, \gamma$, and $\gamma'$ such that
We are required to show that \( \gamma(\eta^1(\lambda x: \tau_1. e)) \sim \gamma'(\eta^2(\lambda x: \tau_1. e)) : \tau_1 \rightarrow \tau_2 \mid \eta \)
\[ \equiv \lambda x: \eta^1(\tau_1). \gamma(\eta^1(e)) \sim \lambda x: \eta^2(\tau_1). \gamma'(\eta^2(e)) : \tau_1 \rightarrow \tau_2 \mid \eta. \]

Take \( v = \lambda x: \eta^1(\tau_1). \gamma(\eta^1(e)) \).
Take \( v' = \lambda x: \eta^2(\tau_1). \gamma'(\eta^2(e)) \).

We are required to show that
\[- \lambda x: \eta^1(\tau_1). \gamma(\eta^1(e)) \rightarrow^* \lambda x: \eta^1(\tau_1). \gamma(\eta^1(e)), \]
which is immediate since \( \lambda x: \eta^1(\tau_1). \gamma(\eta^1(e)) \) is a value,
\[- \lambda x: \eta^2(\tau_1). \gamma'(\eta^2(e)) \rightarrow^* \lambda x: \eta^2(\tau_1). \gamma'(\eta^2(e)), \]
which is immediate since \( \lambda x: \eta^2(\tau_1). \gamma'(\eta^2(e)) \) is a value, and
\[- \lambda x: \eta^1(\tau_1). \gamma(\eta^1(e)) \equiv \lambda x: \eta^2(\tau_1). \gamma'(\eta^2(e)) : \tau_1 \rightarrow \tau_2 \mid \eta, \]
which we conclude as follows:

Consider arbitrary \( v_1 \) and \( v'_1 \) such that
\[* v_1 \equiv v'_1 : \tau_1 \mid \eta. \]
We are required to show \( \gamma(\eta^1(e))[v_1/x] \sim \gamma'(\eta^2(e))[v'_1/x] : \tau_2 \mid \eta. \)

Applying the induction hypothesis to \( \Delta; \Gamma, x : \tau_1 \vdash e : \tau_2 \), we have that \( \Delta; \Gamma, x : \tau_1 \vdash e \sim e : \tau_2 \).

Instantiate the latter with \( \eta, \gamma[x \mapsto v_1], \) and \( \gamma'[x \mapsto v'_1] \). Note that
\[* \eta \models \Delta, \]
which follows from above, and
\[* \gamma[x \mapsto v_1] \equiv \gamma'[x \mapsto v'_1] : \Gamma, x : \tau_1 \mid \eta, \]
which follows from
\* \( \gamma \approx \gamma' : \Gamma \mid \eta \) (follows from above), and
\* \( v_1 \equiv v'_1 : \tau_1 \mid \eta \) (follows from above).

Hence, \( \gamma[x \mapsto v_1](\eta^1(e)) \sim \gamma'[x \mapsto v'_1](\eta^2(e)) : \tau_2 \mid \eta. \)

Thus, \( \gamma(\eta^1(e))[v_1/x] \sim \gamma'(\eta^2(e))[v'_1/x] : \tau_2 \mid \eta, \) as we were required to show.

\[\square\]

**Case:** \( \Delta; \Gamma \vdash e_1 : \tau_1 \rightarrow \tau_2 \quad \Delta; \Gamma \vdash e_2 : \tau_1 \)

**(APP)**

Proof

We are required to show that \( \Delta; \Gamma \vdash e_1 e_2 \sim e_1 e_2 : \tau_2. \)

Consider arbitrary \( \eta, \gamma, \) and \( \gamma' \) such that
\[- \eta \models \Delta, \]
\[- \gamma \approx \gamma' : \Gamma \mid \eta. \]

We are required to show that \( \gamma(\eta^1(e_1 e_2)) \sim \gamma'(\eta^2(e_1 e_2)) : \tau_2 \mid \eta \)
\[\equiv \gamma(\eta^1(e_1)) \gamma(\eta^2(e_2)) \sim \gamma'(\eta^1(e_1)) \gamma'(\eta^2(e_2)) : \tau_2 \mid \eta. \]

Applying the induction hypothesis to \( \Delta; \Gamma \vdash e_1 : \tau_1 \rightarrow \tau_2, \) we have that \( \Delta; \Gamma \vdash e_1 \sim e_1 : \tau_1 \rightarrow \tau_2. \)

Instantiate the latter with \( \eta, \gamma, \) and \( \gamma' \). Note that
\[- \eta \models \Delta, \]
\[- \gamma \approx \gamma' : \Gamma \mid \eta. \]

Hence, \( \gamma(\eta^1(e_1)) \sim \gamma'(\eta^2(e_1)) : \tau_1 \rightarrow \tau_2 \mid \eta. \)

From the latter, there exist \( v_1 \) and \( v'_1 \) such that
\[- \gamma(\eta^1(e_1)) \rightarrow^* v_1, \]
\[- \gamma'(\eta^2(e_1)) \rightarrow^* v'_1, \] and
\[ v_1 \approx v'_1 : \tau_1 \rightarrow \tau_2 \mid \eta. \]

From the latter it follows that \( v_1 = \lambda x : \eta^1(\tau_1). e_{11} \) and \( v'_1 = \lambda x : \eta^2(\tau_1). e'_{11}. \)

Applying the induction hypothesis to \( \Delta; \Gamma \vdash e_2 : \tau_1 \), we have that \( \Delta; \Gamma \vdash e_2 \sim e_2 : \tau_1. \)

Instantiate the latter with \( \eta, \gamma, \) and \( \gamma'. \) Note that

- \( \eta \models \Delta, \)
- \( \gamma \approx \gamma' : \Gamma \mid \eta. \)

Hence, \( \gamma(\eta^1(e_2)) \sim \gamma'(\eta^2(e_2)) : \tau_1 \mid \eta. \)

From the latter, there exist \( v_2 \) and \( v'_2 \) such that

- \( \gamma(\eta^1(e_2)) \rightarrow^* v_2, \)
- \( \gamma'(\eta^2(e_2)) \rightarrow^* v'_2, \) and
- \( v_2 \equiv v'_2 : \tau_1 \mid \eta. \)

Instantiate \( \lambda x : \eta^1(\tau_1). e_{11} \approx \lambda x : \eta^2(\tau_1). e'_{11} : \tau_1 \rightarrow \tau_2 \mid \eta \) (from above) with \( v_2 \) and \( v'_2. \) Note that

- \( v_2 \equiv v'_2 : \tau_1 \mid \eta. \)

Hence, \( e_{11}[v_2/x] \sim e'_{11}[v'_2/x] : \tau_2 \mid \eta. \)

From the latter, there exist \( v \) and \( v' \) such that

- \( e_{11}[v_2/x] \rightarrow^* v, \)
- \( e'_{11}[v'_2/x] \rightarrow^* v', \) and
- \( v \approx v' : \tau_2 \mid \eta. \)

Thus, there exist \( v \) and \( v' \) such that

- \( \gamma(\eta^1(e_1)) \gamma(\eta^1(e_2)) \rightarrow^* v \) (from above by operational semantics),
- \( \gamma'(\eta^2(e_1)) \gamma'(\eta^2(e_2)) \rightarrow^* v' \) (from above by operational semantics), and
- \( v \approx v' : \tau_2 \mid \eta, \) which follows from above.

\[ \square \]
Exercise 1. Do the remaining cases of the proof. You may make use of the following lemmas:

Lemma 1. If $\eta \models \Delta$ and $\Delta \vdash \sigma$ and $R = \{(v, v') \mid v \approx v' : \tau \mid \eta\}$, then $v_1 \approx v_2 : \tau[\sigma/\alpha] \mid \eta$ iff $v_1 \approx v_2 : \tau \mid \eta[\alpha \mapsto (\eta^1(\sigma), \eta^2(\sigma), R)]$.

Lemma 2. If $v \approx v' : \tau \mid \eta$, then $\vdash v : \eta^1(\tau)$ and $\vdash v' : \eta^2(\tau)$.

(a) Case: $\Delta; \Gamma \vdash \Gamma(x) = \tau$ (VAR)

(b) Case: $\Delta, \alpha; \Gamma \vdash e : \tau$ (TABS)

(c) Case: $\Delta; \Gamma \vdash e : \forall \alpha. \tau \vdash \Delta \vdash \sigma$ (TAPP)

Let us add product types $(\tau_1 \times \tau_2)$ to System F. The logical relation for the extended language is exactly as above, except that we must also define when values of product type are related:

$\langle v_1, v_2 \rangle \approx \langle v'_1, v'_2 \rangle : \tau_1 \times \tau_2 \mid \eta$ iff $v_1 \approx v'_1 : \tau_1 \mid \eta$ and $v_2 \approx v'_2 : \tau_2 \mid \eta$

To complete the parametricity proof for the extended language, we need only show the following 3 cases of the parametricity proof pertaining to the intro and elim forms for products:

- Case: $\Delta; \Gamma \vdash e_1 : \tau_1 \Delta; \Gamma \vdash e_2 : \tau_2 \vdash \Delta; \Gamma \vdash \langle e_1, e_2 \rangle : \tau_1 \times \tau_2$ (PAIR)

- Case: $\Delta; \Gamma \vdash e : \tau_1 \times \tau_2 \vdash \Delta; \Gamma \vdash \text{fst } e : \tau_1$ (FST)

- Case: $\Delta; \Gamma \vdash e : \tau_1 \times \tau_2 \vdash \Delta; \Gamma \vdash \text{snd } e : \tau_2$ (SND)

Exercise 2. For this exercise, we extend System F with sum types $\tau_1 + \tau_2$ and existential types $\exists \alpha. \tau$.

(a) Extend the logical relation defined above to handle sum types $\tau_1 + \tau_2$ and existential types $\exists \alpha. \tau$ (i.e., define when values of type $\tau_1 + \tau_2$ are related, and when values of type $\exists \alpha. \tau$ are related).

Complete the following cases of the parametricity proof:

(b) Case: $\Delta; \Gamma \vdash e : \tau_1 \vdash \Delta; \Gamma \vdash \text{inl } e : \tau_1 + \tau_2$ (INL)

(c) Case: $\Delta; \Gamma \vdash e : \tau_1 + \tau_2 \Delta; \Gamma \vdash x : \tau_1 \vdash e_1 : \tau \Delta; \Gamma \vdash y : \tau_2 \vdash e_2 : \tau \vdash \Delta; \Gamma \vdash \text{case } e \text{ of inl } x \Rightarrow e_1 \mid \text{inr } y \Rightarrow e_2 : \tau$ (CASE)

(d) Case: $\Delta \vdash \sigma \Delta; \Gamma \vdash \text{pack } \sigma, e \text{ as } \exists \alpha. \tau \vdash \exists \alpha. \tau$ (PACK)

(e) Case: $\Delta; \Gamma \vdash e_1 : \exists \alpha. \tau \Delta \vdash \tau_2 \Delta, \alpha; \Gamma \vdash e_2 : \tau_2 \Delta; \Gamma \vdash \text{unpack } e_1 \text{ as } \alpha, x \text{ in } e_2 : \tau_2$ (UNPACK)
2 Representation Independence, Bisimilarity

Parametricity is the essence of representation independence for abstract types.

A relation \( R : \tau_1 \leftrightarrow \tau_2 \) is a bisimulation between two packages of type \( \exists \alpha. \tau \): \( \text{pack } \tau_1, v_1 \) as \( \exists \alpha. \tau \) if and only if:

\[
v_1 \approx v_2 : \tau \mid \emptyset[\alpha \mapsto (\tau_1, \tau_2, R)]
\]

When two packages \( e_1 \) and \( e_2 \) are bisimilar, they are indistinguishable by any client — that is, the execution of the client cannot depend on whether \( e_1 \) or \( e_2 \) is the provided package.

In other words, if \( \alpha; x : \tau \vdash e_c : \tau_c \), then

\[
\text{unpack } e_1 \text{ as } \alpha, x \text{ in } e_c \sim \text{unpack } e_2 \text{ as } \alpha, x \text{ in } e_c : \tau_c \mid \emptyset
\]

Exercise 3. Show that the following two implementations of counters are bisimilar.

\[
\text{Counter } = \exists \alpha. \{ \text{new : } \alpha, \\
\quad \text{inc : } \alpha \rightarrow \alpha, \\
\quad \text{get : } \alpha \rightarrow \text{int} \}
\]

\[
c_1 \quad = \quad \{ \text{new = } 0, \\
\quad \text{inc = } \lambda x : \text{int}. x + 1, \\
\quad \text{get = } \lambda x : \text{int}. x \}\n\]

\[
\text{cntr1 } = \quad \text{pack int, c1 as Counter}
\]

\[
c_2 \quad = \quad \{ \text{new = } 0, \\
\quad \text{inc = } \lambda x : \text{int}. x - 1, \\
\quad \text{get = } \lambda x : \text{int}. 0 - x \}\n\]

\[
\text{cntr2 } = \quad \text{pack int, c2 as Counter}
\]

To show that \( \text{cntr1} \) and \( \text{cntr2} \) are bisimilar, it suffices to give a relation \( R : \text{int} \leftrightarrow \text{int} \) and show that \( R \) is such that the implementations of new, inc and get in both packages are equivalent relative to \( R \). That is, show:

(a) \( \text{c1.new } \sim \text{c2.new : } \alpha \mid \emptyset[\alpha \mapsto (\text{int, int}, R)] \)

(b) \( \text{c1.inc } \sim \text{c2.inc : } \alpha \rightarrow \alpha \mid \emptyset[\alpha \mapsto (\text{int, int}, R)] \)

(c) \( \text{c1.get } \sim \text{c2.get : } \alpha \rightarrow \text{int} \mid \emptyset[\alpha \mapsto (\text{int, int}, R)] \)
Exercise 4. Given the relation \( R = \{(l, (b, f)) \mid l = \text{append} \ b(\text{reverse} \ f)\} \) (where \( R : \text{intlist} \leftrightarrow (\text{intlist} \times \text{intlist}) \)), show that the following two implementations of queues are bisimilar. As for the last exercise, you can show this fact by showing that the implementations of all the operations are equivalent relative to \( R \). Note: two lists are related if all of their elements are related.

signature Queue =
  sig
    type queue
    empty : queue
    insert : int -> queue -> queue
    remove : queue -> (int * queue)
  end

structure Q1
  struct
    type queue = int list
    val empty = nil
    fun insert (x, xs) = x::xs
    fun remove xs =
      let val (x, xs) = rev xs in (x, rev xs) end
  end

structure Q2
  struct
    type queue = int list * int list
    val empty = (nil, nil)
    fun insert (x, (bs, fs)) = (x::bs, fs)
    fun remove (bs, nil) = remove (nil, rev bs)
    | remove (bs, f::fs) = (f, (bs, fs))
  end

Note that the SML signature for queues above may be written as follows:

\[
\text{Queue} = \exists \alpha. \left\{ \begin{array}{l}
\text{empty} : \alpha, \\
\text{insert} : \text{nat} \rightarrow \alpha \rightarrow \alpha, \\
\text{remove} : \alpha \rightarrow (\text{int} \times \alpha) \end{array} \right\}
\]

3 Subtyping

Read TAPL Section 15.1, 15.2, and pages 196–198 (Variants, Lists, References).

Exercise 5. For each of the following questions, answer Yes or No. If the answer is Yes, show the subtyping derivation. If the answer is No, give either a term that demonstrates how type safety breaks if we allow the two types in the subtype relation, or a short explanation of why type safety is preserved even if we allow the two types in the subtype relation.

(a) Is \( \{x : \text{Top} \rightarrow \text{Ref} \text{Top}\} \) a subtype of \( \{x : \text{Top} \rightarrow \text{Top}\} \)?

(b) Is \( \{x : \text{Top} \rightarrow \text{Ref} \text{Top}\} \) a subtype of \( \{x : \text{Ref} \text{Top} \rightarrow \text{Ref} \{y : \text{Top}\}\} \)?

(c) Is \( \{x : \text{Ref} \{y : \text{Top}\}\} \) a subtype of \( \{x : \text{Ref} \text{Top}\} \)?

(d) Is \( \{x : \text{Top}\} \) a subtype of \( \{x : \{\}\}\)?