yz709-Sem-sup3

5 Subtyping

Exercise 5.1

(*a*) Explain the reasoning behind the subtyping rule for function types.

- For sub-typing functions, if $f : T_1 \to T_2$, argument types are contra-variant (sub-types), return types are covariant (super-types), we can give f more arguments that it needs (hence subtype of T_1), only require a subset of the return values from f (hence a supertype of T_2)
- From another perspective, the callee is a subtype of a caller, the caller can pass in more arguments than needed by the callee, and get returned more return values from the callee than required.

```
T1' <: T1     T2 <: T2'
-----------------------
T1 -> T2 <: T1' -> T2'
```

(*b*) For each of the two bogus *T* ref subtype rules on slide 202, give an example program that is typable with that rule but gets stuck at runtime.

- A value of reference type can be used both to store and load, so any subtyping ($T_1\ ref <: T_2\ ref$) would let one store a record with few fields and use it in a context where more fields are expected

```
(1)
    T <: T'
---------------
T ref <: T' ref

assume bool <: int, so bool ref <: int ref
       if b ∈ bool, then b ∈ {true,false}

e.g., <let x = ref true in x := !x + 3; if !x then 1 else 2 end,s>
      -> <x := !x + 3; if !x then 1 else 2 end; kill x, s + {x -> true}>
      -> <x := (int)true + 3; if !x then 1 else 2 end; kill x, s + {x -> true}>
      -> <x := 4; if !x then 1 else 2 end; kill x, s + {x -> true}>
      -> <if !x then 1 else 2 end; kill x, s + {x -> 4}>
      -/-> because Γ |- !x : int, and !x = 4 but we expect true or false

(2)
    T' <: T
--------------
T ref <: T' ref

assume {lab1:p,lab2:q} <: {lab1:p}, so {lab1:p} ref <: {lab1:p,lab2:q} ref

e.g., <let x = ref {lab1:p} in (#lab1 !x) + (#lab2 !x) end,s>
      -> <(#lab1 !x) + (#lab2 !x); kill x, s + {x -> {lab1:p}}>
      -> < p + (#lab2 !x); kill x, s + {x -> {lab1:p}}>
      -/-> because x has only one field, but we expect more fields.
```

💡 Comments:

Exercise 5.2

Exercise 5.2. For each of the following, either give a type derivation or explain why it is untypable:

(*a*) $\{\} \vdash \{\mathrm{p} = \{\mathrm{p} = \{\mathrm{p} = \{\mathrm{p} = 3\}\}\}\} : \{\mathrm{p} : \{\}\}$

(*b*) $\{\} \vdash \mathbf{fn}\ x : \{\mathrm{p} : \mathrm{bool}, \mathrm{q} : \{\mathrm{p} : \mathrm{int}, \mathrm{q} : \mathrm{bool}\}\} \Rightarrow \#q\ \#p\ x : ?$

(*c*) $\{\} \vdash \mathbf{fn}\ x : \{\mathrm{p} : \mathrm{int}\} \rightarrow \mathrm{int} \Rightarrow (f\ \{\mathrm{q} = 3\}) + (f\ \{\mathrm{p} = 4\}) : ?$

(*d*) $\{\} \vdash \mathbf{fn}\ x : \{\mathrm{p} : \mathrm{int}\} \rightarrow \mathrm{int} \Rightarrow (f\ \{\mathrm{q} = 3, \mathrm{p} = 2\}) + (f\ \{\mathrm{p} = 4\}) : ?$

```
(a): I didn't get (a), what is :{p:{}}?
{} |- {p = 3} : {p:int}
----------------------------------
{} |- {p1 = {p = 3}} : {p1:{p:int}
-------------------------------------------------
{} |- {p2 = {p1 = {p = 3}}} : {p2:{p1:{p:int}}}
-------------------------------------------------
{} |- {p3 = {p2 = {p1 = {p = 3}}}} : {p3:{p2:{p1:{p:int}}}}

(b): untypable because #q #p x has an evaluation order #q(#p(x)) from inner to outer, hence once
```

```
 we evaluated Γ |- #p(x) : bool, we can no longer apply #q onto
a boolean, we expect a record type.

(c): untypable because Γ |- {q=3}: {q:int}, Γ |- {p=4}: {p:int}, there is no
subtyping relation between the two, and the function f has a unique type, it
either accept type {q:int} or {p:int} or neither.

(d): T can be any type that addition is defined upon, e.g., int, float, double
{},x:{p:int}->int |- f: {p:int} -> T,
{},x:{p:int}->int |-                    {},x:{p:int}->int |- f: {p:int} -> T,
  {q=3,p=2} <: {p=2} : {p:int},         {},x:{p:int}->int |- {p=4} : {p:int}
-------------------------------------   ---------------------------------
{},x:{p:int}->int |- f{q=3,p=2}:T       {},x:{p:int}->int |- f{p=4}:T
----------------------------------------------------------------------------
{}, x:{p:int}->int |-  (f{q=3,p=2}) + (f{p=4}): T
----------------------------------------------------------------------------
{} |- fn x: {p:int} -> int => (f{q=3,p=2}) + (f{p=4}): ({p:int} -> int) -> T
```

Comments:

Exercise 5.3

Exercise 5.3. State the subtyping rules for sums, **let val** $x : e_1 = T$ **in** e_2 and **let rec** $x : e_1 = T$ **in** e_2.

```
(1) sums:
T1 <: T2  U1 <: U2
------------------
T1 + U1 <: T2 + U2

(2) assume let val x:e1 = T in e2 is written as (λ(x:T).e2)(e1)
T1' <: T1  T2 <: T2'
----------------------------------------------------------
let val x:e1 = T1 in e2:T2 <: let val x:e1 = T1' in e2:T2'

(3) assume let rec x:e1 = T in e2 is written as
           e1 = (fn y:T1 => e3):T1 -> T2
           let rec x:T1 -> T2 = e1 in e2:T3
T1' <: T1  T2 <: T2'  T3 <: T3'
-----------------------------------------------------------------------
let rec x:e1 = T1 -> T2 in e2:T3 <: let rec x:e1 = T1' -> T2' in e2:T3'
```

Comments:

6 Concurrency

Exercise 6.1

Exercise 6.1. Show all possible reduction sequences for $e_1 \parallel e_2$ from the initial store $\{l_1 \mapsto 10, l_2 \mapsto 40\}$ and no locks acquired, where:

$$\begin{array}{rcl} e_1 & = & \textbf{lock}\ m;\ l_1 := !l_1 - 2;\ l_2 := !l_1 + 1;\ \textbf{unlock}\ m \\ e_2 & = & \textbf{lock}\ m;\ l_2 := !l_2 + 3;\ l_1 := !l_1 - 3;\ \textbf{unlock}\ m \end{array}$$

Show the derivations of the possible candidates for the first reduction.

```
(1)
<e1||e2,{l1->10,l2->40},{m->false}>
-><(l1:=!l1 - 2;l2:=!l1+1;unlock m)||e2,{l1->10,l2->40},{m->true}>
-><(l1:=8;l2:=!l1+1;unlock m)||e2,{l1->10,l2->40},{m->true}>
-><(l2:=!l1+1;unlock m)||e2,{l1->8,l2->40},{m->true}>
-><(l2:=9;unlock m)||e2,{l1->8,l2->40},{m->true}>
-><(unlock m)||e2,{l1->8,l2->9},{m->true}>
-><()||(lock m;l2:=!l2+3;l1:=!l1-3;unlock m),{l1->8,l2->9},{m->false}>
-><()||(l2:=!l2+3;l1:=!l1-3;unlock m),{l1->8,l2->9},{m->true}>
-><()||(l2:=12;l1:=!l1-3;unlock m),{l1->8,l2->9},{m->true}>
-><()||(l1:=!l1-3;unlock m),{l1->8,l2->12},{m->true}>
-><()||(l1:=5;unlock m),{l1->8,l2->12},{m->true}>
-><()||(unlock m),{l1->5,l2->12},{m->true}>
-><()||(),{l1->5,l2->12},{m->false}>

(2)
<e1||e2,{l1->10,l2->40},{m->false}>
-><e1||(l2:=!l2+3;l1:=!l1-3;unlock m),{l1->10,l2->40},{m->true}>
-><e1||(l2:=43;l1:=!l1-3;unlock m),{l1->10,l2->40},{m->true}>
-><e1||(l1:=!l1-3;unlock m),{l1->10,l2->43},{m->true}>
-><e1||(l1:=7;unlock m),{l1->10,l2->43},{m->true}>
-><e1||(unlock m),{l1->7,l2->43},{m->true}>
-><(lock m;l1:=!l1 - 2;l2:=!l1+1;unlock m)||(),{l1->7,l2->43},{m->false}>
-><(l1:=!l1 - 2;l2:=!l1+1;unlock m)||(),{l1->7,l2->43},{m->true}>
-><(l1:=5;l2:=!l1+1;unlock m)||(),{l1->7,l2->43},{m->true}>
-><(l2:=!l1+1;unlock m)||(),{l1->5,l2->43},{m->true}>
-><(l2:=6;unlock m)||(),{l1->5,l2->43},{m->true}>
-><(unlock m)||(),{l1->5,l2->6},{m->true}>
-><()||(),{l1->5,l2->6},{m->false}>

(3) the possible candidates for the first reduction
<e1,s,{m->false}> -> <e1',s,{m->true}>
-----------------------------------------------
<e1||e2,s,{m->false}> -> <e1'||e2,s,{m->true}>

<e2,s,{m->false}> -> <e2',s,{m->true}>
-----------------------------------------------
<e1||e2,s,{m->false}> -> <e1||e2',s,{m->true}>
```

💡 Comments:

Exercise 6.2

Exercise 6.2. Can you show all the conditions for O2PL are necessary, by giving for each an example that satisfies all the others, and either is not serialisable or deadlocks?

- Conditions for O2PL include: (1) a fixed lock acquisition order, e.g., based on lock index; (2) we cannot acquire any locks after at least one lock has been released, i.e., there is a lock acquiring phase and a lock releasing phase.
- (1) if we acquire locks in a random order, e.g., `e2` acquires lock `m2` first and then `m1`, then we may encounter a deadlock case where `e1` acquires lock `m1` and spins to wait for lock `m2` and `e2` acquires lock `m2`, and spins to wait for lock `m1`, both threads are waiting for the other thread to release the lock, so none is progressing.

```
e1 = lock m1;lock m2;l1 := !l2 + 1;unlock m2;unlock m1;
e2 = lock m2;lock m1;l2 := !l1 + 3;unlock m1;unlock m2;

<e1||e2,{l1 -> 1, l2 -> 2}, {m1 -> false, m2 -> false}>
-> <(lock m2;l1 := !l2 + 1;unlock m2;unlock m1;)||e2,
      {l1 -> 1, l2 -> 2}, {m1 -> true, m2 -> false}>
-> <(lock m2;l1 := !l2 + 1;unlock m2;unlock m1;)||
    (lock m1;l2 := !l1 + 3;unlock m1;unlock m2;),
      {l1 -> 1, l2 -> 2}, {m1 -> true, m2 -> true}>
-/-> deadlock
```

- (2) if we interleave locking and unlocking, e.g., `e1` acquires lock `m1` then released `m1` before acquiring lock `m2`, this may lead to not serialisable schedule
  - If we execute `e1` then `e2`, the store should be `{l1 -> 4, l2 -> 1}`, if we execute `e2` then `e1`, the store should be `{l1 -> 6, l2 ->5}`, but with an interleaving execution of the two transactions, the dirty read of `l1` and `l2` makes the final schedule not serialisable.

```
e1 = lock m1;l1 := !l1+1;unlock m1;lock m2;l2 := !l1-1;unlock m2;
e2 = lock m1;l1 := !l2+3;unlock m1;

<e1||e2,{l1 -> 1, l2 -> 2}, {m1 -> false, m2 -> false}>
-> <(l1 := !l1+1;unlock m1;lock m2;l2 := !l1-1;unlock m2;)||e2,
      {l1 -> 1, l2 -> 2}, {m1 -> true, m2 -> false}>
-> <(l1 := 2;unlock m1;lock m2;l2 := !l1-1;unlock m2;)||e2,
      {l1 -> 1, l2 -> 2}, {m1 -> true, m2 -> false}>
-> <(unlock m1;lock m2;l2 := !l1-1;unlock m2;)||e2,
      {l1 -> 2, l2 -> 2}, {m1 -> true, m2 -> false}>
-> <(lock m2;l2 := !l1-1;unlock m2;)||e2,
      {l1 -> 2, l2 -> 2}, {m1 -> false, m2 -> false}>
-> <(lock m2;l2 := !l1-1;unlock m2;)||(l1 := !l2+3;unlock m1;),
      {l1 -> 2, l2 -> 2}, {m1 -> true, m2 -> false}>
-> <(lock m2;l2 := !l1-1;unlock m2;)||(l1 := 5;unlock m1;),
      {l1 -> 2, l2 -> 2}, {m1 -> true, m2 -> false}>
-> <(lock m2;l2 := !l1-1;unlock m2;)||(unlock m1;),
      {l1 -> 5, l2 -> 2}, {m1 -> true, m2 -> false}>
-> <(lock m2;l2 := !l1-1;unlock m2;)||(),
```

```
      {l1 -> 5, l2 -> 2}, {m1 -> false, m2 -> false}>
-> <(l2 := !l1-1;unlock m2;)||(),
      {l1 -> 5, l2 -> 2}, {m1 -> false, m2 -> true}>
-> <(l2 := 4;unlock m2;)||(),
      {l1 -> 5, l2 -> 2}, {m1 -> false, m2 -> true}>
-> <(unlock m2;)||(),
      {l1 -> 5, l2 -> 4}, {m1 -> false, m2 -> true}>
-> <()||(),
      {l1 -> 5, l2 -> 4}, {m1 -> false, m2 -> false}>
```

💡 Comments:

7 Semantic equivalence

Exercise 7.1

Exercise 7.1. Let e_1 and e_2 be expressions and Γ_1 and Γ_2 be contexts such that $\Gamma_1 \vdash e_1 : \mathrm{unit}$ and $\Gamma_2 \vdash e_2 : \mathrm{unit}$. Show that, if Γ_1 and Γ_2 are disjoint, then $e_1;\ e_2 \simeq_{\Gamma}^{\mathrm{unit}} e_2;\ e_1$, where $\Gamma = \Gamma_1 \cup \Gamma_2$.

- Prove $e_1; e_2 \simeq_{\Gamma}^{unit} e_2; e_1$ holds where $\Gamma = \Gamma_1 \cup \Gamma_2$, we have either $< e_1; e_2, s > \to^{w}$ and $< e_2; e_1, s > \to^{w}$, i.e., both reduces to an infinite loop, or there exists some v, s' such that $< e_1; e_2, s > \to^{*} < v, s' >$ and $< e_2; e_1, s > \to^{*} < v, s' >$, i.e., results in the same value and the store.
- Γ is the typing environment, $\Gamma \vdash e : T$ means e has type T under the assumptions Γ on the types of locations that may occur in e. Hence $\Gamma_1 \vdash e_1 : \mathrm{unit}$ means e_1 would only influence locations $l \in dom(\Gamma_1)$, and $\Gamma_2 \vdash e_2 : \mathrm{unit}$ means e_2 would only influence locations $l \in dom(\Gamma_2)$
- Since Γ_1 and Γ_2 are disjoint with each other, we have $\Gamma \vdash e_1; e_2 : \mathrm{unit}$ and $\Gamma \vdash e_2; e_1 : \mathrm{unit}$ where $\Gamma = \Gamma_1 \cup \Gamma_2$
- Case when $< e_1; e_2, s > \to^{w}$:
  - We have either $< e_1, s > \to^{w}$ or $< e_1; e_2, s > \to^{*} < skip; e_2, s' >$ and $< e_2, s' > \to^{w}$.
  - If $< e_1, s > \to^{w}$, then $< e_2; e_1, s > \to^{w}$ holds as even e_2 evaluates to a value, e_1 would lead to an infinite loop;
  - if $< e_1; e_2, s > \to^{*} < skip; e_2, s' >$ and $< e_2, s' > \to^{w}$, then $< e_2; e_1, s > \to^{w}$ directly holds.
- Case when there exists some v, s' such that $< e_1; e_2, s > \to^{*} < v, s' >$:

- We reduces the expression by (seq1) $< e_1; e_2, s_1 > \to^* < skip; e_2, s_2 >$ and this reduction sequence only changes the locations $l \in dom(\Gamma_1) \subseteq dom(s)$, where $dom(\Gamma_2) \cap dom(\Gamma_1) = \phi$. Then the reduction sequence reduces further with (seq2) $< skip; e_2, s_2 > \to^* < v, s_3 >$ where we only changes the store $l \in dom(\Gamma_2)$. Since $\Gamma_2 \vdash e_2 : unit$, we have $\Gamma \vdash e_2 : unit$ as Γ_1 is disjoint from Γ_2, therefore $v = skip$ and $< e_2, s_2 > \to^* < skip, s_3 >$, and we have $< e_1; e_2, s_1 > \to^* < skip, s_3 >$.
- Thus for $< e_2; e_1, s_1 >$, we first reduces by (seq1) $< e_2; e_1, s_1 > \to^* < skip; e_1, s_4 >$ which only changes the locations $l \in dom(\Gamma_2) \subseteq dom(s)$, $dom(\Gamma_2) \cap dom(\Gamma_1) = \phi$. Then the reduction sequence reduces further with (seq2) $< skip; e_1, s_4 > \to^* < skip, s_5 >$, hence we have $< e_2; e_1, s_1 > \to < skip, s_5 >$
- To prove $s_3 = s_5$, since each part of the reduction sequence (i.e., the one uses (seq1) and the one uses (seq2)) changes locations in disjoint sets, hence there is no single location that would be changed by both parts of the reduction, hence the order does not influence the final storage as long as we start with the same store. Thus we have $< e_2; e_1, s > \to^* < v, s' >$, i.e., results in the same value and the store.

💡 Comments:

Exercise 7.2

Exercise 7.2. The following L3 judgements hold:

$$l : \text{int ref} \vdash l := 0 : \text{unit}$$
$$l : \text{int ref} \vdash l := 1 : \text{unit}$$

Show that these two assignments are not contextually equivalent.

- Contextual equivalence for L_3 is defined by $C[e_1] \simeq_\Gamma^T C[e_2]$ to hold $\Longleftrightarrow$ Suppose $\Gamma \vdash e_1 : T$ and $\Gamma \vdash e_2 : T$, if for every context C such that $\{\} \vdash C[e_1] : unit$ and $\{\} \vdash C[e_2] : unit$, we have:
  - either both reduces into an infinite loop $< C[e_1], \{\} > \to^w$ and $< C[e_2], \{\} > \to^w$
  - or for some s_1 and s_2, we have $< C[e_1], \{\} > \to^* < skip, s_1 >$ and $< C[e_2], \{\} > \to^* < skip, s_2 >$
- Informally, we can replace the first expression with the second one without affecting the program's observable results (e.g., in L3, whether both produce an infinite reduction, or both terminates).

- When we have $e_1 := (l := 0)$ and $e_2 := (l := 1)$, the only context cases that gives $\{\} \vdash C[e_1] : unit$ and $\{\} \vdash C[e_2] : unit$ are
    - $C ::= \_; e_2 | e_1; \_ | \text{if e then } \_ \text{ else } e_3 | \text{if e then } e_2' \text{ else } \_ \ | \text{while e do } \_$
- All the cases expect the while loop would be contextually equivalent for these two expressions because only the storage differs.
- Case $C ::= \text{while e do } \_$: if $e ::= (!l >= 1)$, and the $l \mapsto 1$ in the initial store, then we would have in the first expression, the program halts after one loop, thus $< C[e_1], \{\} > \to^* < skip, s_1 >$, and in the second expression, the program evaluates infinitely.

💡 Comments: My argument for the while loop is not valid because I have to initialise the storage, which is not allowed. Could you please give me a hint.

Exercise 7.3

Exercise 7.3. Prove the following cases for Congruence for L1:

(a) **if** $\_$ **then** e_2 **else** e_3

(b) $e_1\ op\ \_$

(c) $\_;\ e_2$

- (a): Case $C = (\text{if } \_ \text{ then } e_2 \text{ else } e_3)$:
    - Suppose $e \simeq_\Gamma^T e', \Gamma \vdash \text{if } e \text{ then } e_2 \text{ else } e_3 : T'$ and $\Gamma \vdash \text{if } e' \text{ then } e_2 \text{ else } e_3 : T'$. By examing the typing rules we have $T = \text{bool}$.
    - To show $C[e] \simeq_\Gamma^T C[e']$, we have to show for all s such that $\text{dom}(\Gamma) \subseteq \text{som}(s)$, then $\Gamma \vdash \text{if } e \text{ then } e_2 \text{ else } e_3 : T'$ and $\Gamma \vdash \text{if } e' \text{ then } e_2 \text{ else } e_3 : T'$ and either
        1. $< \text{if } e \text{ then } e_2 \text{ else } e_3, s > \to^w$ and $< \text{if } e' \text{ then } e_2 \text{ else } e_3, s > \to^w$ or
        2. for some v, s', we have $< \text{if } e \text{ then } e_2 \text{ else } e_3, s > \to^* < v, s' >$ and $< \text{if } e' \text{ then } e_2 \text{ else } e_3, s > \to^* < v, s' >$
    - Consider the possible reduction sequences of a state $< \text{if } e \text{ then } e_2 \text{ else } e_3, s >$, then:
        - Case $< \text{if } e \text{ then } e_2 \text{ else } e_3, s > \to^w$: then either
            - there is an infinite reduction sequence for $< e, s >$ so by $e \simeq_\Gamma^T e'$ there would also be an infinite reduction sequence for $< e', s >$, so by (if1), there is an infinite reduction sequence of $< \text{if } e' \text{ then } e_2 \text{ else } e_3, s >$;

- or there is an infinite reduction sequence for e_2 or e_3, since $e \simeq_{\Gamma}^{T} e'$, both uses reduction rule (if1) to evaluate to the same value (i.e., both evaluate to $true$ or both to $false$, hence they will both evaluate $< e_2, s' >$ or $< e_3, s' >$, hence there is an infinite reduction sequence of $< \text{if } e' \text{ then } e_2 \text{ else } e_3, s >$

  - Case for some v, s', we have $< \text{if } e \text{ then } e_2 \text{ else } e_3, s > \to^* < v, s' >$: then all reductions would be instances of (if1) until the last one which would be either (if2) $< \text{if true then } e_2 \text{ else } e_3, s > \to < e_2, s' >$ or (if3) $< \text{if false then } e_2 \text{ else } e_3, s > \to < e_3, s' >$. Since we have assumed $e \simeq_{\Gamma}^{bool} e'$, if $< e, s > \to^* < true, s' >$, then $< e', s > \to^* < true, s' >$, and similarly if $< e, s > \to^* < false, s' >$, then $< e', s > \to^* < false, s' >$. Therefore, all reduction uses (if1) will arrive at the same result, and the final reduction rule would be the same for both sequence (i.e., either both uses (if2) or both uses (if3) based on the value e and e' reduces to). Thus we have $< \text{if } e' \text{ then } e_2 \text{ else } e_3, s > \to^* < v, s' >$ holds

- (b): Case $C = (e_1 \; op \; \_)$:
  - Suppose $e \simeq_{\Gamma}^{T} e', \Gamma \vdash e_1 \; op \; e : T'$ and $\Gamma \vdash e_1 \; op \; e' : T'$. By examing the typing rules we have two cases based on the operator.
    - (1) consider when op is $+$: then $T = \text{int}$ and $T' = \text{int}$
    - (2) consider when op is $\geq$: then $T = \text{int}$ and $T' = \text{bool}$
  - To show $C[e] \simeq_{\Gamma}^{T} C[e']$, we have to show for all s such that $\text{dom}(\Gamma) \subseteq \text{som}(s)$, then $\Gamma \vdash e_1 \; op \; e : T'$ and $\Gamma \vdash e_1 \; op \; e' : T'$ and either
    1. $< e_1 \; op \; e, s > \to^w$ and $< e_1 \; op \; e', s > \to^w$ or
    2. for some v, s', we have $< e_1 \; op \; e, s > \to^* < v, s' >$ and $< e_1 \; op \; e', s > \to^* < v, s' >$
  - Consider the possible reduction sequences of a state $< e_1 \; op \; e, s >$, then:
    - Case $< e_1 \; op \; e, s > \to^w$: then either there is an infinite reducetion sequence for $< e_1, s >$ or for $< e, s >$. If we have an infinite reduction sequence for $< e_1, s >$, then $< e_1 \; op \; s', s >$ would reduce e_1 first and get the same infinte reduction sequence, so we are done. Otherwise if we have an infinite reduction sequence for $< e, s >$, then by $e \simeq_{\Gamma}^{T} e'$ there would also be an infinite reduction sequence for $< e', s >$, so by (op2), there is an infinite reduction sequence of $< e_1 \; op \; e', s >$
    - Case for some v, s', we have $< e_1 \; op \; e, s > \to^* < v, s' >$: then all reductions would be instances of (op1) and (op2) until the last one which would be $< v_1 \; op \; v_2, s > \to < v, s >$, we have assumed $e \simeq_{\Gamma}^{int} e'$, if $< e, s > \to^* < v_2, s' >$, then $< e', s > \to^* < v_2, s' >$. Therefore, all reduction uses (op1) and (op2) will arrive at the same result $< v_1 \; op \; v_2, s >$, and the final reduction rule

would be the same for both sequence (i.e., either both uses (op+) or both uses (op≥) based on the operator op). Thus we have $< e_1 \ op \ e', s > \rightarrow^* < v, s' >$ holds

- (c): Case $C = (\_; e_2)$:
  - Suppose $e \simeq^T_\Gamma e', \Gamma \vdash e; e_2 : T'$ and $\Gamma \vdash e'; e_2 : T'$. By examing the typing rules we have $T = \mathrm{unit}$ and $T' = \mathrm{unit}$
  - To show $C[e] \simeq^T_\Gamma C[e']$, we have to show for all s such that $\mathrm{dom}(\Gamma) \subseteq \mathrm{som}(s)$, then $\Gamma \vdash e; e_2 : T'$ and $\Gamma \vdash e'; e_2 : T'$ and either
    1. $< e; e_2, s > \rightarrow^w$ and $< e'; e_2, s > \rightarrow^w$ or
    2. for some v, s', we have $< e; e_2, s > \rightarrow^* < v, s' >$ and $< e'; e_2, s > \rightarrow^* < v, s' >$
  - Consider the possible reduction sequences of a state $< e; e_2, s >$, then:
    - Case $< e; e_2, s > \rightarrow^w$: then either there is an infinite reduction sequence for $< e, s >$ or we evaluate $< e, s > \rightarrow^* < skip, s' >$and there is an infinite sequence for e_2. If we have an infinite sequence for $< e, s >$ then by $e \simeq^T_\Gamma e'$, there would also be an infinite sequence for $< e', s >$, so we have an inifnite sequence for $< e'; e_2, s >$. Otherwise if we have an infinite sequence for $< e_2, s' >$, then by $e \simeq^T_\Gamma e'$, we will have $< e', s > \rightarrow^* < skip, s' >$, and continues evaluate $< e_2, s' >$, therefore we also have an infinite sequence for $< e'; e_2, s >$
    - Case for some v, s', we have $< e; e_2, s > \rightarrow^* < v, s' >$ and $< e'; e_2, s > \rightarrow^* < v, s' >$: then all reductions would be instances of (seq1), until we arrive at $< skip; e_2, s'' >$, by $e \simeq^T_\Gamma e'$, we have assumed if $< e, s > \rightarrow^* < skip, s'' >$, then $< e', s' > \rightarrow^* < skip, s'' >$, therefore both seuqence would continue evaluate $< skip; e_2, s'' >$ and arrive at the same result $< v, s' >$

💡 Comments:

Past paper question (y2013p6q10)

This question is about a variation on a fragment of the L2 language in which functions take two arguments. The language has the following expressions:

$$e \quad ::= \quad x \ | \ \mathsf{fn}\,(x_1, x_2) \Rightarrow e \ | \ e_0\,(e_1, e_2) \ | \ n$$

where x ranges over variables and n ranges over integers. As usual, $\mathsf{fn}\,(x, y) \Rightarrow e$ is binding: we work up-to α-equivalence and require that x and y are different.

(a) Write down a call-by-name operational semantics for this language. [2 marks]

(a)

(fn) $\langle (\mathsf{fn}\,(x_1, x_2) \Rightarrow e)\,(y_1, y_2), s\rangle \rightarrow \langle \{y_1/x_1, y_2/x_2\} e, s\rangle$

(app)
$$\frac{\langle e_0, s\rangle \rightarrow \langle e_0', s'\rangle}{\langle e_0\,(e_1, e_2), s\rangle \rightarrow \langle e_0'\,(e_1, e_2), s'\rangle}$$

(b) Consider the following type system. The types are

$$T \ ::= \ \mathsf{int} \mid \mathsf{ret} \mid (T_1, T_2) \rightarrow \mathsf{ret}$$

A context Γ is a finite partial function from variables to types. The type system is given by the following rules:

$$\frac{-}{\Gamma, x : T, \Gamma' \vdash x : T} \qquad \frac{\Gamma \vdash e_0 : (T_1, T_2) \rightarrow \mathsf{ret} \quad \Gamma \vdash e_1 : T_1 \quad \Gamma \vdash e_2 : T_2}{\Gamma \vdash e_0\,(e_1, e_2) : \mathsf{ret}}$$

$$\frac{-}{\Gamma \vdash n : \mathsf{int}}\ (n \text{ is an integer}) \qquad \frac{\Gamma, x_1 : T_1, x_2 : T_2 \vdash e : \mathsf{ret}}{\Gamma \vdash \mathsf{fn}\,(x_1, x_2) \Rightarrow e : (T_1, T_2) \rightarrow \mathsf{ret}}$$

(The idea is that $(T_1, T_2) \rightarrow \mathsf{ret}$ is a type of functions taking arguments of type T_1 and T_2. However, there are no expressions of type ret in the empty context, and so rather than returning a result you pass it to a 'continuation'.)

(i) Find a type T for which $\vdash \mathsf{fn}\,(x, k) \Rightarrow k\,(3, x) : T$, giving a derivation. [3 marks]

(ii) Give a derivation of the following judgement: [2 marks]

$$k : (\mathsf{int}, \mathsf{ret}) \rightarrow \mathsf{ret} \vdash \mathsf{fn}\,(x, l) \Rightarrow l\,(7, k) : (\mathsf{int}, (\mathsf{int}, (\mathsf{int}, \mathsf{ret}) \rightarrow \mathsf{ret}) \rightarrow \mathsf{ret}) \rightarrow \mathsf{ret}$$

(b) (i)

$$\frac{\Gamma \vdash x : T_1 \quad \Gamma \vdash k : T_2}{\Gamma \vdash \mathsf{fn}(x,k) \Rightarrow k(3,x) : (T_1, T_2) \to \mathsf{ret}}$$

$$\frac{\Gamma \vdash k : (T_3, T_4) \to \mathsf{ret} \quad \Gamma \vdash 3 : T_3 \quad \Gamma \vdash x : T_4}{\Gamma \vdash k(3,x) : \mathsf{ret}}$$

since $\Gamma \vdash 3 : \mathsf{int}$, we have $T_3 = \mathsf{int}$

hence $\Gamma \vdash k : (\mathsf{int}, T_4) \to \mathsf{ret}$

$T_2 = (\mathsf{int}, T_4) \to \mathsf{ret}$

$T_1 = T_4$

hence $T = (T_1, (\mathsf{int}, T_1) \to \mathsf{ret}) \to \mathsf{ret}$

where T_1 is any valid types in $\mathsf{int} \mid \mathsf{ret} \mid (T_1, T_2) \to \mathsf{ret}$

(ii)

$$\frac{\Gamma \vdash x : T_1,\ l : T_2,\ l(7,k) : \mathsf{ret}}{\Gamma \vdash \mathsf{fn}(x,l) \Rightarrow l(7,k)}$$

$$\frac{\Gamma \vdash l : (T_3, T_4) \to \mathsf{ret} \quad \Gamma \vdash 7 : T_3 \quad \Gamma \vdash k : T_4}{\Gamma \vdash l(7,k) : \mathsf{ret}}$$

since $\Gamma \vdash 7 : \mathsf{int}$

$\Gamma \vdash k : (\mathsf{int}, \mathsf{ret}) \to \mathsf{ret}.$

we have $\Gamma \vdash l : (\mathsf{int}, (\mathsf{int}, \mathsf{ret}) \to \mathsf{ret}) \to \mathsf{ret}$

hence we have

$\Gamma \vdash \mathsf{fn}(x,l) : (T_1, (\mathsf{int}, (\mathsf{int}, \mathsf{ret}) \to \mathsf{ret}) \to \mathsf{ret})) \to \mathsf{ret}$

if $\Gamma \vdash x : \mathsf{int}$,

then the judgement holds

(c) Prove the following 'progress' theorem for this language: [6 marks]

> If $\vdash e : T$ then either $e = (\mathsf{fn}\,(x, y) \Rightarrow e')$, or e is an integer, or there is e' such that $e \longrightarrow e'$.

(c) case n : $\overline{\Gamma \vdash \bar{n} : \text{int}}$

case x : $\Gamma \vdash x : \text{int}$ or $\Gamma \vdash x : (\text{fn}(x, y) \Rightarrow e')$

hence progress theorem holds for n and x

case fn : $\dfrac{\Gamma, x_1 : T_1, x_2 : T_2 \vdash e : \text{ret}}{\Gamma \vdash \text{fn}(x_1, x_2) \Rightarrow e : (T_1, T_2) \to \text{ret}}$

function is a value, hence progress theorem holds

case app :

$$\frac{\Gamma \vdash e_0 : (T_1, T_2) \to \text{ret} \quad \Gamma \vdash e_1 : T_1 \quad \Gamma \vdash e_2 : T_2}{\Gamma \vdash e_0(e_1, e_2)}$$

case $\exists e_0', s'$, $\langle e_0, s \rangle \to \langle e_0', s' \rangle$, then by

$$(\text{app}) \ \frac{\langle e_0, s \rangle \to \langle e_0', s' \rangle}{\langle e_0 \ e, s \rangle \to \langle e_0' e, s' \rangle},$$

we have $\langle e_0 (e_1, e_2), s \rangle \to \langle e_0' (e_1, e_2), s' \rangle$

case e_0 is a value, it must be $e_0 = \text{fn}(x, y) \Rightarrow e'$, then by

$(\text{fn}) \ \langle (\text{fn}(x_1, x_2) \Rightarrow e)(y_1, y_2), s \rangle \to \langle \{y_1/x_1, y_2/x_2\} e, s \rangle$

we have $\langle \text{fn}(x, y) \Rightarrow e' (e_1, e_2), s \rangle \to \langle \{e_1/x, e_2/y\} e', s \rangle$

(*d*) We now consider the situation where there is a type **posint** of positive integers which is a subtype of **int**.

Define a subtyping relation <: and extend the type system to accommodate it. Demonstrate it by giving a derivation of the following judgement:

$$k : (\mathsf{int}, \mathsf{ret}) \to \mathsf{ret} \vdash \mathsf{fn}\,(x, l) \Rightarrow l\,(7, k) : (\mathsf{int}, (\mathsf{int}, (\mathsf{posint}, \mathsf{ret}) \to \mathsf{ret}) \to \mathsf{ret}) \to \mathsf{ret}$$

[7 marks]

(d) (base) $\dfrac{}{\text{posint} <: \text{int}}$ $\quad T ::= \text{int} \mid \text{posint} \mid \text{ret} \mid (T_1, T_2) \to \text{ret}$

(reflexive) $\dfrac{}{T <: T}$

(transitive) $\dfrac{T_1 <: T_2 \quad T_2 <: T_3}{T_1 <: T_3}$

(subsumption to permit up-casting) $\dfrac{\Gamma \vdash e : T \quad T <: T'}{\Gamma \vdash e : T'}$

(subtyping for function) $\dfrac{T_1' <: T_1 \quad T_2 <: T_2'}{T_1 \to T_2 <: T_1' \to T_2'}$

(subtyping for products) $\dfrac{T_1 <: T_2 \quad U_1 <: U}{(T_1, U_1) <: (T_2, U_2)}$

Derivation

$$\frac{\Gamma \vdash x : T_1,\ l : T_2,\ l(7, k) : \text{ret}}{\Gamma \vdash fn(x, l) \Rightarrow l(7, k)}$$

$$\frac{\Gamma \vdash l : (T_3, T_4) \to ret \quad \Gamma \vdash 7 : T_3 \quad \Gamma \vdash k : T_4}{\Gamma \vdash l(7, k) : ret}$$

since
$$\frac{posint <: int \quad ret <: ret}{(posint, ret) <: (int, ret)} \text{ (reflexive, products)}$$

$$\frac{(posint, ret) <: (int, ret) \qquad ret <: ret}{(int, ret) \to ret <: (posint, ret) \to ret} \text{ (s-function)}$$

$$\frac{\Gamma \vdash k : (int, ret) \to ret \quad (int, ret) \to ret <: (posint, ret) \to ret}{\Gamma \vdash k : (posint, ret) \to ret} \text{ (subsumption)}$$

since $\Gamma \vdash 7 : int$

$\Gamma \vdash k : (posint, ret) \to ret$

we have $\Gamma \vdash l : (int, (posint, ret) \to ret) \to ret$

hence we have

$\Gamma \vdash fn(x, l) : (T_1, (int, (posint, ret) \to ret) \to ret)) \to ret$

if $\Gamma \vdash x : int$,

then the judgement holds

Past paper question (y2010p6q9)

(a) if $\Gamma \vdash e : T$ and $dom(\Gamma) \subseteq dom(s)$, then either e is a value or there exist e', s' such that $\langle e, s\rangle \rightarrow \langle e', s'\rangle$

(b) Define $\Phi(\Gamma, e, T) \stackrel{\text{def}}{=} (\Gamma \neq \emptyset) \vee (e$ is a value $\vee\ \exists e', s'$ such that $\langle e, s\rangle \rightarrow \langle e', s'\rangle)$

(case if)

$$\frac{\Gamma \vdash e : bool \ , \ \Gamma \vdash e_1 : T \ , \ \Gamma \vdash e_2 : T}{\Gamma \vdash \text{if } e \text{ then } e_1 \text{ else } e_2 : T}$$

Assume $\Phi(\Gamma, e, bool)$, $\Phi(\Gamma, e_1, T)$, $\Phi(\Gamma, e_2, T)$, $\Gamma \vdash e : bool$, $\Gamma \vdash e_1 : T$, $\Gamma \vdash e_2 : T$

Consider an arbitrary store s, assume $dom(\Gamma) \subseteq dom(s)$

write $e' = \text{if } e \text{ then } e_1 \text{ else } e_2$

case e is a value:

① if $e = \text{true}$, then we have progress using (r-if1)

$\langle e', s\rangle \rightarrow \langle e_1, s\rangle$

② if $e = \text{false}$, then we have progress using (r-if2)

$\langle e', s\rangle \rightarrow \langle e_2, s\rangle$

case $\exists e'', s''. \langle e, s\rangle \rightarrow \langle e'', s''\rangle$, then by (r-if3), we have

$\langle e', s\rangle \rightarrow \langle \text{if } e'' \text{ then } e_1 \text{ else } e_2, s''\rangle$

(case deref)

$$\Gamma \vdash !l : bool \quad \text{if } l \in \Gamma$$

consider an arbitrary s with $dom(\Gamma) \subseteq dom(s)$

By the condition $l \in \Gamma$, so $l \in dom(s)$, there is some n with $s(l) = n$.

so $\langle !l, s\rangle \to \langle n, s\rangle$ and $s(l) = n$

(case assign)

$$\frac{\Gamma \vdash e : bool}{\Gamma \vdash l := e : bool} \text{ if } l \in \Gamma$$

assume $\Phi(\Gamma, e, bool)$, $\Gamma \vdash e : bool$

consider an arbitrary s with $dom(\Gamma) \subseteq dom(s)$

case e is a value ($e = b$), then we have progress by (r-assign1)

$$\langle l := b, s\rangle \to \langle b, s\{l \mapsto b\}\rangle \quad b \in \{true, false\}$$

case $\exists e', s'$. such that $\langle e, s\rangle \to \langle e', s'\rangle$, then we have progress by (r-assign2)

$$\langle l := e, s\rangle \to \langle l := e', s'\rangle$$

(case bool)

$$\Gamma \vdash True : bool$$
$$\Gamma \vdash false : bool$$

For all instances Γ, e, T of the conclusion,

$e \in \{true, false\}$, $T = bool$

since True, false are values, e is a value, so $\Phi(\Gamma, e, bool)$ holds

(c)

Semantic equivalence $e_1 \simeq^T_\Gamma e_2 \iff \forall s,\ dom(\Gamma) \subseteq dom(s)$,

we have $\Gamma \vdash e_1 : T$, $\Gamma \vdash e_2 : T$ either $\langle e_1, s\rangle \rightarrow^\omega$ and $\langle e_2, s\rangle \rightarrow^\omega$, i.e., both lead to infinite reduction, or for som v, s', we have $\langle e_1, s\rangle \rightarrow^* \langle v, s'\rangle$ and $\langle e_2, s\rangle \rightarrow^* \langle v, s'\rangle$, i.e., result in same value and store.

Constraint on e: $\exists v$, such that $\langle e, s\rangle \rightarrow^* \langle v, s\rangle$, i.e., e can not lead to infinite reduction but evaluate to $v \in \{true, false\}$, and does not change the store s

hence regardless of the value of e,

$\langle \text{if } e \text{ then } e_1 \text{ else } e_1, s\rangle \rightarrow^* \langle e_1, s\rangle$

(d)

if $\exists v$, such that $\langle e_1, s\rangle \rightarrow^* \langle v, s\rangle$

then $\langle (e_1; e), s\rangle \rightarrow^* \langle e, s\rangle$

if $\exists v'$, such that $\langle e, s\rangle \rightarrow^* \langle v', s\rangle$

then $\langle (e; e_2), s\rangle \rightarrow^* \langle e_2, s\rangle$

if $\langle e_2, s\rangle \rightarrow^* \langle true, s\rangle$, then we have $((e_1; e); e_2)$ semantically equivalent to e_2