Solution by Xiao Jia

Homework 3

DO NOT COPY FROM OTHER STUDENTS!

Problem 1. (10 points) Consider the following program which is written in C syntax.

```
int x = 1;
int f() { return x; }
int main() {
  int x = 2;
  return f();
}
```

(a) What is the return value of `main()` when it uses static scoping? dynamic scoping?

(b) Try to write another program whose execution result will be affected by static/dynamic scoping in a way that is essentially different from the one illustrated above, or explain why you cannot write such a program.

Solution.

(a) static scoping: 1, dynamic scoping: 2

(b) This is an open-ended question and does not have a standard answer. As long as you write something that makes sense, you will get 4-5 points. However, do consider the following C program which uses macros.

```
#define ADD_A(x) x + a

void add_one(int *x) { const int a = 1;
                       *x = ADD_A(*x);  }

void add_two(int *x) { const int a = 2;
                       *x = ADD_A(*x);  }
```

The macro expansion in C compilers uses dynamic scoping. If it uses static scoping, the second program (the one in this solution) will not get compiled. This example is essentially different because the first program (the one in the question) shows a difference in run time while the second program shows a difference in compile time.

■

Problem 2. (30 points) When answering questions, refer to the following definition of the lambda calculus with booleans and recursive functions (LBR from now on):

Syntax:

(types) $t ::= \texttt{bool} \mid t_1 \to t_2$

(expressions) $e ::= x \mid \texttt{true} \mid \texttt{false} \mid \texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 \mid \texttt{fun } f\ (x : t_1) : t_2 = e \mid e_1\ e_2$

(values) $v ::= \texttt{true} \mid \texttt{false} \mid \texttt{fun } f\ (x : t_1) : t_2 = e$

(contexts) $\Gamma ::= \epsilon \mid \Gamma, x : t$

Call-by-value Evaluation:

$$\frac{e \to e'}{\texttt{if } e \texttt{ then } e_1 \texttt{ else } e_2 \to \texttt{if } e' \texttt{ then } e_1 \texttt{ else } e_2} \quad \text{(E-if0)}$$

$$\frac{}{\texttt{if true then } e_1 \texttt{ else } e_2 \to e_1} \quad \text{(E-if1)}$$

$$\frac{}{\texttt{if false then } e_1 \texttt{ else } e_2 \to e_2} \quad \text{(E-if2)}$$

$$\frac{e_1 \to e_1'}{e_1\ e_2 \to e_1'\ e_2} \quad \text{(E-app1)}$$

$$\frac{e_2 \to e_2'}{v\ e_2 \to v\ e_2'} \quad \text{(E-app2)}$$

$$\frac{}{(\texttt{fun } f(x : t_1) : t_2 = e)\ v \to e[v/x][(\texttt{fun } f(x : t_1) : t_2 = e)/f]} \quad \text{(E-app3)}$$

Typing:

$$\frac{}{\Gamma \vdash x : \Gamma(x)} \quad \text{(T-var)}$$

$$\frac{}{\Gamma \vdash \texttt{true} : \texttt{bool}} \quad \text{(T-true)}$$

$$\frac{}{\Gamma \vdash \texttt{false} : \texttt{bool}} \quad \text{(T-false)}$$

$$\frac{\Gamma \vdash e : \texttt{bool} \quad \Gamma \vdash e_1 : t \quad \Gamma \vdash e_2 : t}{\Gamma \vdash \texttt{if } e \texttt{ then } e_1 \texttt{ else } e_2 : t} \quad \text{(T-if)}$$

$$\frac{\Gamma, f : t_1 \to t_2, x : t_1 \vdash e : t_2}{\Gamma \vdash \texttt{fun } f(x : t_1) : t_2 = e : t_1 \to t_2} \quad \text{(T-fun)}$$

$$\frac{\Gamma \vdash e_1 : t_1 \to t_2 \quad \Gamma \vdash e_2 : t_1}{\Gamma \vdash e_1\ e_2 : t_2} \quad \text{(T-app)}$$

(a) Let the expression e be the following:

$$(\texttt{fun foo } (x : \texttt{bool} \to \texttt{bool}) =$$
$$(\texttt{fun bar } (y : \texttt{bool}) = \texttt{if } y \texttt{ then } (x\ \texttt{false}) \texttt{ else } (\texttt{foo } x\ \texttt{true})))$$

Give a complete typing derivation for the judgement below. Mark each level of the derivation with the name of the rule (T-var, T-if, etc.) that you used to make the corresponding inference.

$$\epsilon \vdash e : (\texttt{bool} \to \texttt{bool}) \to \texttt{bool} \to \texttt{bool}$$

(b) Conjecture: In LBR, if $\Gamma \vdash e : t$ and $e' \to e$ then $\Gamma \vdash e' : t$. Prove it or give a counterexample.

(c) Conjecture: For any LBR expression e and any type t_1 there is at most one type t_2 such that $(\texttt{fun}\ f(x : t_1) : t_2 = e)$ is well typed. Prove it or give a counterexample.

Solution.

(a) First we define some symbols:

$$\begin{aligned}
\mathbb{B} &\equiv \texttt{bool} \\
e &\equiv \texttt{fun foo}\ (x : \mathbb{B} \to \mathbb{B}) = e' \\
e' &\equiv \texttt{fun bar}\ (y : \mathbb{B}) = e'' \\
e'' &\equiv \texttt{if}\ y\ \texttt{then}\ (x\ \texttt{false})\ \texttt{else}\ (\texttt{foo}\ x\ \texttt{true}) \\
\Gamma &\equiv \epsilon, \texttt{foo} : (\mathbb{B} \to \mathbb{B}) \to \mathbb{B} \to \mathbb{B}, x : \mathbb{B} \to \mathbb{B}, \texttt{bar} : \mathbb{B} \to \mathbb{B}, y : \mathbb{B}
\end{aligned}$$

Use the symbol $\equiv$ to avoid conflicting usage of $=$. Always define new symbols to denote frequently used objects for better readability. Concise solutions score higher.

Then two auxiliary lemmas:

$$\frac{\dfrac{}{\Gamma \vdash x : \mathbb{B} \to \mathbb{B}}\ \text{T-var} \quad \dfrac{}{\Gamma \vdash \texttt{false} : \mathbb{B}}\ \text{T-false}}{\Gamma \vdash (x\ \texttt{false}) : \mathbb{B}}\ \text{T-app} \tag{I}$$

and

$$\frac{\dfrac{\dfrac{}{\Gamma \vdash \texttt{foo} : (\mathbb{B} \to \mathbb{B}) \to \mathbb{B} \to \mathbb{B}}\ \text{T-var} \quad \dfrac{}{\Gamma \vdash x : \mathbb{B} \to \mathbb{B}}\ \text{T-var}}{\Gamma \vdash (\texttt{foo}\ x) : \mathbb{B} \to \mathbb{B}}\ \text{T-app} \quad \dfrac{}{\Gamma \vdash \texttt{true} : \mathbb{B}}\ \text{T-true}}{\Gamma \vdash (\texttt{foo}\ x\ \texttt{true}) : \mathbb{B}}\ \text{T-app} \tag{II}$$

Finally the original judgement:

$$\frac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash y : \mathbb{B}}\ \text{T-var} \quad \dfrac{}{\Gamma \vdash (x\ \texttt{false}) : \mathbb{B}}\ \text{(I)} \quad \dfrac{}{\Gamma \vdash (\texttt{foo}\ x\ \texttt{true}) : \mathbb{B}}\ \text{(II)}}{\Gamma \vdash e'' : \mathbb{B}}\ \text{T-if}}{\epsilon, \texttt{foo} : (\mathbb{B} \to \mathbb{B}) \to \mathbb{B} \to \mathbb{B}, x : \mathbb{B} \to \mathbb{B} \vdash e' : \mathbb{B} \to \mathbb{B}}\ \text{T-fun}}{\epsilon \vdash e : (\mathbb{B} \to \mathbb{B}) \to \mathbb{B} \to \mathbb{B}}\ \text{T-fun}$$

Keep in mind that the judgement form is $\Gamma \vdash e : t$, not $e : t$ or anything else. Also a derivation has a shape of a tree where premises are on the top. Though not recommended, you can also write the derivation in a linear form like the proofs, but make sure you write enough premises in the "by" clauses, and state everything with the right judgement form.

(b) Hint: Easier to first prove the preservation theorem and then prove by contradiction.

(c) Counterexample: Both

$$\texttt{fun}\ f(x : \texttt{bool}) : \texttt{bool} = (f\ x)$$

and

$$\texttt{fun}\ f(x : \texttt{bool}) : \texttt{bool} \to \texttt{bool} = (f\ x)$$

are well-typed. We prove that this is a valid counterexample by showing that for any types t_1 and t_2, $\texttt{fun}\ f(x : t_1) : t_2 = (f\ x)$ is well-typed and has type $t_1 \to t_2$:

$$\dfrac{\dfrac{\dfrac{}{\Gamma_2 \vdash f : t_1 \to t_2}\ \text{T-var} \quad \dfrac{}{\Gamma_2 \vdash x : t_1}\ \text{T-var}}{\Gamma_2 \vdash (f\ x) : t_2}\ \text{T-app}}{\Gamma_1 \vdash \texttt{fun}\ f(x : t_1) : t_2 = (f\ x) : t_1 \to t_2}\ \text{T-fun}$$

where Γ_1 is an arbitrary context, and $\Gamma_2 \equiv \Gamma_1, f : t_1 \to t_2, x : t_1$.

I believe this is the most interesting question in this assignment. However, no one gives a correct answer to it. All of you believe that the conjecture is true, because you have learnt the lemma *uniqueness of typing* of simply typed lambda calculus (ST for short). But LBR is *not* ST. In ST function abstractions are anonymous, and we have to extend ST with "fix" to implement recursion. In LBR, functions have names, so we can have recursion directly. This leads to a completely different set of semantic and typing rules, and language properties.

■

Problem 3. (30 points) Consider adding trees with leaf node elements of type t (a "t tree") to the simple language above. The additional types, expressions, and values are as follows:

$$\begin{aligned}
t &::= \ldots \text{ (as above) } \ldots \mid t\ \texttt{tree} \\
e &::= \ldots \text{ (as above) } \ldots \mid \texttt{Lf}(e) \mid \texttt{Br}(e_1, e_2) \mid \texttt{case}\ e\ \texttt{of}\ (\texttt{Lf}(x) \Rightarrow e_1 \mid \texttt{Br}(x, y) \Rightarrow e_2) \\
v &::= \ldots \text{ (as above) } \ldots \mid \texttt{Lf}(v) \mid \texttt{Br}(v_1, v_2)
\end{aligned}$$

Here are the operational rules:

$$\frac{e \to e'}{\texttt{Lf}(e) \to \texttt{Lf}(e')} \quad \text{(L)}$$

$$\frac{e_1 \to e_1'}{\texttt{Br}(e_1, e_2) \to \texttt{Br}(e_1', e_2)} \quad \text{(B1)}$$

$$\frac{e_2 \to e_2'}{\texttt{Br}(v_1, e_2) \to \texttt{Br}(v_1, e_2')} \quad \text{(B2)}$$

$$\frac{e \to e'}{\texttt{case}\ e\ \texttt{of}\ (\texttt{Lf}(x) \Rightarrow e_1 \mid \texttt{Br}(x, y) \Rightarrow e_2) \to \texttt{case}\ e'\ \texttt{of}\ (\texttt{Lf}(x) \Rightarrow e_1 \mid \texttt{Br}(x, y) \Rightarrow e_2)} \quad \text{(C1)}$$

$$\frac{}{\texttt{case}\ \texttt{Lf}(v)\ \texttt{of}\ (\texttt{Lf}(x) \Rightarrow e_1 \mid \texttt{Br}(x, y) \Rightarrow e_2) \to e_1[v/x]} \quad \text{(C2)}$$

$$\frac{}{\texttt{case}\ \texttt{Br}(v_1, v_2)\ \texttt{of}\ (\texttt{Lf}(x) \Rightarrow e_1 \mid \texttt{Br}(x, y) \Rightarrow e_2) \to e_2[v_1/x][v_2/y]} \quad \text{(C3)}$$

(a) Give typing rules for the `Lf` (leaf), `Br` (branch), and `case` expressions.

(b) Prove the cases of the Preservation Theorem that pertain to the `Lf`, `Br` and `case` expressions. You may assume that the Exchange, Weakening and Substitution Lemmas have been proven for you.

Exchange: If $\Gamma_1, x_1 : t_1, x_2 : t_2, \Gamma_2 \vdash e : t$ then $\Gamma_1, x_2 : t_2, x_1 : t_1, \Gamma_2 \vdash e : t$.

Weakening: If $\Gamma \vdash e : t$ and $x \notin \mathrm{Dom}(\Gamma)$ then $\Gamma, x_1 : t_1 \vdash e : t$.

Substitution: If $\Gamma, x_1 : t_1 \vdash e : t$ and $\vdash v_1 : t_1$ then $\Gamma \vdash e[v_1/x_1] : t$.

Solution.

(a)

$$\frac{\Gamma \vdash e : t}{\Gamma \vdash \texttt{Lf}(e) : t\ \texttt{tree}} \quad \text{(T-Lf)}$$

$$\frac{\Gamma \vdash e_1 : t \qquad \Gamma \vdash e_2 : t}{\Gamma \vdash \texttt{Br}(e_1, e_2) : t\ \texttt{tree}} \quad \text{(T-Br)}$$

$$\frac{\Gamma \vdash e : t'\ \texttt{tree} \qquad \Gamma, x : t' \vdash e_1 : t \qquad \Gamma, x : t', y : t' \vdash e_2 : t}{\Gamma \vdash \texttt{case}\ e\ \texttt{of}\ (\texttt{Lf}(x) \Rightarrow e_1 \mid \texttt{Br}(x, y) \Rightarrow e_2) : t} \quad \text{(T-case)}$$

$\Gamma \vdash \texttt{Lf}(x) : t'\ \texttt{tree}$ and $\Gamma \vdash \texttt{Br}(x, y) : t'\ \texttt{tree}$ are unnecessary and ill-defined because x, y are unbound. Don't use them as premises. Also e and e_1, e_2 can have different types.

(b) Note that we only need to prove the cases related to `Lf`, `Br` and `case`.

Theorem 1 (Preservation Theorem)**.** *If* $\Gamma \vdash e : t$ *and* $e \to e'$*, then* $\Gamma \vdash e' : t$.

Proof. By induction on the derivation of $\Gamma \vdash e : t$.

Case T-Lf: $e \equiv \texttt{Lf}(e_1)$.

If $e \to e'$ then $\exists e_1'$ such that $e_1 \to e_1'$ and $\texttt{Lf}(e_1) \to \texttt{Lf}(e_1')$.

(1) $\Gamma \vdash e_1 : t$ (by inversion of T-Lf)

(2) $\Gamma \vdash e_1' : t$ (by I.H.)

(3) $\Gamma \vdash \texttt{Lf}(e_1') : t\ \texttt{tree}$ (by T-Lf)

Case proved.

Case T-Br: $e \equiv \texttt{Br}(e_1, e_2)$.

If $e \to e'$ there are two subcases:

Subcase 1: e_1 is not a value (B1), $\exists e_1'$ such that $e_1 \to e_1'$ and $\texttt{Br}(e_1, e_2) \to \texttt{Br}(e_1', e_2)$.

(1) $\Gamma \vdash e_1 : t$ and $\Gamma \vdash e_2 : t$ (by inversion of T-Br)

(2) $\Gamma \vdash e_1' : t$ (by I.H.)

(3) $\Gamma \vdash \texttt{Br}(e_1', e_2) : t\ \texttt{tree}$ (by T-Br)

Subcase 2: e_1 is a value (B2), $\exists e_2'$ such that $e_2 \to e_2'$ and $\texttt{Br}(e_1, e_2) \to \texttt{Br}(e_1, e_2')$.

(1) $\Gamma \vdash e_1 : t$ and $\Gamma \vdash e_2 : t$ (by inversion of T-Br)

(2) $\Gamma \vdash e_2' : t$ (by I.H.)

(3) $\Gamma \vdash \texttt{Br}(e_1, e_2') : t\ \texttt{tree}$ (by T-Br)

Case proved.

Case T-case: $e \equiv \texttt{case}\ e_0\ \texttt{of}\ (\texttt{Lf}(x) \Rightarrow e_1 \mid \texttt{Br}(x, y) \Rightarrow e_2)$.

If $e \to e'$ there are three subcases:

Subcase 1: e_0 is not a value (C1), $\exists e_0'$ such that $e_0 \to e_0'$

(1) $\Gamma \vdash e_0 : t'\ \texttt{tree}$, $\Gamma, x : t' \vdash e_1 : t$ and $\Gamma, x : t', y : t' \vdash e_2 : t$ (by inversion of T-case)

(2) $\Gamma \vdash e_0' : t'\ \texttt{tree}$ (by I.H.)

(3) $\Gamma \vdash \texttt{case}\ e_0'\ \texttt{of}\ (\texttt{Lf}(x) \Rightarrow e_1 \mid \texttt{Br}(x, y) \Rightarrow e_2) : t$ (by T-case)

Subcase 2: $e_0 = \texttt{Lf}(v)$ is a leaf (C2)

(1) $\Gamma \vdash e_0 : t'\ \texttt{tree}$, $\Gamma, x : t' \vdash e_1 : t$ and $\Gamma, x : t', y : t' \vdash e_2 : t$ (by inversion of T-case)

(2) $\Gamma \vdash v : t'$ (by inversion of T-Lf)

(3) $\Gamma \vdash e_1[v/x] : t$ (by Substitution Lemma)

Subcase 3: $e_0 = \texttt{Br}(v_1, v_2)$ is a branch (C3)

(1) $\Gamma \vdash e_0 : t'\ \texttt{tree}$, $\Gamma, x : t' \vdash e_1 : t$ and $\Gamma, x : t', y : t' \vdash e_2 : t$ (by inversion of T-case)

(2) $\Gamma \vdash v_1 : t'$ and $\Gamma \vdash v_2 : t'$ (by inversion of T-Br)

(3) $\Gamma, y : t', x : t' \vdash e_2 : t$ (by Exchange Lemma)

(4) $\Gamma, y : t' \vdash e_2[v_1/x] : t$ (by Substitution Lemma)

(5) $\Gamma \vdash e_2[v_1/x][v_2/y] : t$ (by Substitution Lemma)

Case proved. □

Some obvious premises such as (1), (2), etc. are omitted in the "by" clauses for brevity.

■

Problem 4. Write an interpreter for the core λ calculus in Java.

Solution.

Listing 1: `lambda.Main`

```
package lambda;

import java.util.Scanner;

public class Main {

  public static void main(String[] args) {
    StringBuffer term = new StringBuffer();

    try (Scanner input = new Scanner(System.in)) {
      while (input.hasNextLine()) {
        term.append(input.nextLine());
        term.append(' ');
      }
    }

    Exp exp = parse(term.toString());
    System.err.println(exp);
    System.out.println(exp.reduce());
  }

  private static Exp parse(String s) {
    int leftParen = 0;
    while (leftParen < s.length() && s.charAt(leftParen) != '(')
      leftParen += 1;
    if (leftParen >= s.length())
      // Form x
      return new Var(s.trim());

    int rightParen = s.length() - 1;
    while (rightParen > leftParen && s.charAt(rightParen) != ')')
      rightParen -= 1;
    if (rightParen <= leftParen)
      throw new Error("Cannot find right parenthesis in: " + s);

    s = s.substring(leftParen + 1, rightParen).trim();

    if (s.startsWith("lambda") && !Character.isLetterOrDigit(s.charAt(6)))
    {
      leftParen = 0;
      while (leftParen < s.length() && s.charAt(leftParen) != '(')
        leftParen += 1;
      if (leftParen >= s.length())
        throw new Error("Cannot find left parenthesis in: " + s);

      rightParen = leftParen + 1;
      while (rightParen < s.length() && s.charAt(rightParen) != ')')
        rightParen += 1;
      if (rightParen >= s.length())
        throw new Error("Cannot find right parenthesis in: " + s);
```

```
50
51        // Form (lambda (x) e)
52        String x = s.substring(leftParen + 1, rightParen).trim();
53        String e = s.substring(rightParen + 1);
54        return new Abs(new Var(x), parse(e));
55      }
56
57      // Form (e1 e2)
58      int k = -1; // separate e1 and e2
59
60      if (s.charAt(0) == '(') {
61        int balance = 0;
62
63        for (int i = 0; i < s.length(); i++) {
64          if (s.charAt(i) == '('
65            balance += 1;
66          else if (s.charAt(i) == ')')
67            balance -= 1;
68
69          if (balance == 0) {
70            k = i + 1;
71            break;
72          }
73        }
74
75        if (k == -1)
76          throw new Error("Parentheses unbalanced in: " + s);
77      } else { // must be a name
78        k = 0;
79        while (k < s.length() && Character.isLetterOrDigit(s.charAt(k)))
80          k += 1;
81        if (k >= s.length())
82          throw new Error("Second expression expected in: " + s);
83      }
84
85      String e1 = s.substring(0, k);
86      String e2 = s.substring(k);
87      return new App(parse(e1), parse(e2));
88    }
89  }
```

Listing 2: `lambda.Exp`

```
1  package lambda;
2
3  public abstract class Exp {
4
5    public abstract Exp reduce();
6
7    public abstract Exp substitute(Var x, Exp v);
8  }
```

Listing 3: `lambda.Var`

```
package lambda;

public class Var extends Exp {

  private String s;

  public Var(String s) {
    this.s = s;
  }

  public Exp reduce() {
    return this;
  }

  public String toString() {
    return s;
  }

  public boolean equals(Object other) {
    if (other instanceof Var)
      return s.equals(((Var) other).s);
    return false;
  }

  public Exp substitute(Var x, Exp v) {
    if (equals(x))
      return v;
    return this;
  }
}
```

Listing 4: `lambda.Abs`

```
package lambda;

public class Abs extends Exp {

  private Var x;
  private Exp e;

  public Abs(Var x, Exp e) {
    this.x = x;
    this.e = e;
  }

  public Exp reduce() {
    return this;
  }

  public String toString() {
```

```
    return "(lambda (" + x + ") " + e + ")";
  }

  public Exp apply(Exp v) {
    return e.substitute(x, v);
  }

  public Exp substitute(Var y, Exp v) {
    if (x.equals(y))
      return this;
    return new Abs(x, e.substitute(y, v));
  }
}
```

Listing 5: `lambda.App`

```
package lambda;

public class App extends Exp {

  private Exp e1, e2;

  public App(Exp e1, Exp e2) {
    this.e1 = e1;
    this.e2 = e2;
  }

  public Exp reduce() {
    Exp lam = e1.reduce();
    Exp val = e2.reduce();
    // They must both reduce to values, and only functions can be values.
    if (lam instanceof Abs && val instanceof Abs)
      return ((Abs) lam).apply(val).reduce();
    return new App(lam, val);
  }

  public String toString() {
    return "(" + e1 + " " + e2 + ")";
  }

  public Exp substitute(Var x, Exp v) {
    return new App(e1.substitute(x, v), e2.substitute(x, v));
  }
}
```

Only some of you finished this question and submitted to me, and you are still allowed and encouraged to submit before the next homework deadline. This question is not included in the scoring of this assignment, but may be used as part of the next assignment if necessary. ■