Homework 3
AAA616, Fall 2022

Hakjoo Oh

Due: 11/30, 23:59

Problem 1 The goal of this assignment is to implement the 0-CFA analysis for the following language:

$$
\begin{array}{rcll}
e & \to & t^l & \text{expressions (labelled terms)} \\
t & \to & n & \text{terms (unlabelled expressions)} \\
 & | & x & \\
 & | & \texttt{fn}\ x\ \texttt{=>}\ e_0 & \\
 & | & \texttt{fun}\ f\ x\ \texttt{=>}\ e_0 & \\
 & | & e_1\ e_2 & \\
 & | & \texttt{if}\ e_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2 & \\
 & | & \texttt{let}\ x\ \texttt{=}\ e_1\ \texttt{in}\ e_2 & \\
 & | & e_1\ op\ e_2 &
\end{array}
$$

The 0-CFA constraints are generated as follows:

$$
\begin{array}{rcl}
\mathcal{C}(n^l) & = & \emptyset \\
\mathcal{C}(x^l) & = & \{S(x) \subseteq S(l)\} \\
\mathcal{C}((\texttt{fn}\ x\ \texttt{=>}\ e_0)^l) & = & \{\{\texttt{fn}\ x\ \texttt{=>}\ e_0\} \subseteq S(l)\} \cup \mathcal{C}(e_0) \\
\mathcal{C}((\texttt{fun}\ f\ x\ \texttt{=>}\ e_0)^l) & = & \{\{\texttt{fun}\ f\ x\ \texttt{=>}\ e_0\} \subseteq S(l)\} \cup \mathcal{C}(e_0) \\
 & & \cup \{\{\texttt{fun}\ f\ x\ \texttt{=>}\ e_0\} \subseteq S(f)\} \\
\mathcal{C}((t_1^{l_1}\ t_2^{l_2})^l) & = & \mathcal{C}(t_1^{l_1}) \cup \mathcal{C}(t_2^{l_2}) \\
 & & \cup \{\{t\} \subseteq S(l_1) \implies S(l_2) \subseteq S(x) \\
 & & \quad \mid t = (\texttt{fn}\ x\ \texttt{=>}\ t_0^{l_0}) \in \mathsf{Term}\} \\
 & & \cup \{\{t\} \subseteq S(l_1) \implies S(l_0) \subseteq S(l) \\
 & & \quad \mid t = (\texttt{fn}\ x\ \texttt{=>}\ t_0^{l_0}) \in \mathsf{Term}\} \\
 & & \cup \{\{t\} \subseteq S(l_1) \implies S(l_2) \subseteq S(x) \\
 & & \quad \mid t = (\texttt{fun}\ f\ x\ \texttt{=>}\ t_0^{l_0}) \in \mathsf{Term}\} \\
 & & \cup \{\{t\} \subseteq S(l_1) \implies S(l_0) \subseteq S(l) \\
 & & \quad \mid t = (\texttt{fun}\ f\ x\ \texttt{=>}\ t_0^{l_0}) \in \mathsf{Term}\} \\
\mathcal{C}((\texttt{if}\ t_0^{l_0}\ \texttt{then}\ t_1^{l_1}\ \texttt{else}\ t_2^{l_2})^l) & = & \\
\mathcal{C}((\texttt{let}\ x\texttt{=}t_1^{l_1}\ \texttt{in}\ t_2^{l_2})^l) & = & \\
\mathcal{C}((t_1^{l_1}\ op\ t_2^{l_2})^l) & = &
\end{array}
$$

and the constraints can be solved by the fixed point algorithm:

$$
\begin{aligned}
&\mathsf{solve}(C,S) = \\
&\quad \text{let } S' = \mathsf{update}(C,S) \\
&\quad \text{if } \forall a.S'(a) \subseteq S(a) \text{ then } S \\
&\quad \text{else } \mathsf{solve}(C,S')
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{update}(C,S) = \\
&\quad \text{for } c \text{ in } C: \\
&\quad\quad \text{if } c = (\{t\} \subseteq S(a)): \\
&\quad\quad\quad S(a) := S(a) \cup \{t\} \\
&\quad\quad \text{if } c = (S(a_1) \subseteq S(a_2)): \\
&\quad\quad\quad S(a_2) := S(a_2) \cup S(a_1) \\
&\quad\quad \text{if } c = (\{t\} \subseteq S(a_1) \implies S(a_2) \subseteq S(a_3)): \\
&\quad\quad\quad \text{if } t \in S(a_1) \text{ then } S(a_3) := S(a_3) \cup S(a_2) \\
&\quad \text{return } S
\end{aligned}
$$

The template code in OCaml is given as follows:

```
type exp = term * label
and term =
    | CONST of int
    | VAR of string
    | FN of string * exp
    | RECFN of string * string * exp
    | APP of exp * exp
    | IF of exp * exp * exp
    | LET of string * exp * exp
    | BOP of op * exp * exp
and label = int
and op = PLUS | MINUS | MULT | DIV

let string_of_exp (_,l) = string_of_int l
let string_of_term term =
    match term with
    | CONST n -> string_of_int n
    | VAR x -> x
    | FN (x, e) -> "FN " ^ x ^ " -> " ^ string_of_exp e
    | RECFN (f, x, e) -> "RecFN " ^ f ^ " " ^ x ^ " " ^ string_of_exp e
    | APP (e1, e2) -> string_of_exp e1 ^ " " ^ string_of_exp e2
    | IF (e1,e2,e3) -> "IF " ^ string_of_exp e1 ^ " " ^ string_of_exp e2 ^ " " ^ string_of_exp e3
    | LET (x,e1,e2) -> "LET " ^ string_of_exp e1 ^ " " ^ string_of_exp e2
    | BOP (_,e1,e2) -> "BOP " ^ string_of_exp e1 ^ " " ^ string_of_exp e2

let ex1 = (APP (
                    (FN ("x", (VAR "x", 1)) , 2),
                    (FN("y", (VAR "y", 3)), 4)), 5)
let ex2 = (LET ("g", (RECFN("f", "x", ((APP ((VAR "f", 1), ((FN ("y", (VAR "y", 2)), 3))), 4))), 5),
                         (APP((VAR "g", 6), (FN("z", (VAR "z", 7)), 8)), 9)), 10)
let ex3 = (LET ("f", (FN ("x", (VAR "x", 1)), 2),
                         (APP (
                            (APP (
                                (VAR "f", 3),
                                (VAR "f", 4)), 5),
                            (FN ("y", (VAR "y", 6)), 7))
```

```
                        , 8)
                      )
                  , 9)

type eqn = SUBSET of data * data | COND of data * data * data * data
and data = T of term | C of label | V of string

type constraints = eqn list

module Term = struct
   type t = term
   let compare = compare
 end
module Terms = Set.Make(Term)

let string_of_terms terms =
    Terms.fold (fun t s -> s ^ string_of_term t ^ ", ") terms ""

module Label = struct
   type t = label
   let compare = compare
 end
module AbsCache = struct
    module Map = Map.Make(Label)
    type t = Terms.t Map.t
    let empty = Map.empty
    let find l m = try Map.find l m with _ -> Terms.empty
    let add l t m = Map.add l (Terms.union t (find l m)) m
    let order m1 m2 = Map.for_all (fun l set -> Terms.subset set (find l m2)) m1
    let print m =
        Map.iter (fun l terms ->
           print_endline (string_of_int l ^ " |-> " ^ string_of_terms terms)
        ) m
end
module Var = struct
    type t = string
    let compare = compare
end
module AbsEnv = struct
    module Map = Map.Make(Var)
    type t = Terms.t Map.t
    let empty = Map.empty
    let find l m = try Map.find l m with _ -> Terms.empty
    let add l t m = Map.add l (Terms.union t (find l m)) m
    let order m1 m2 = Map.for_all (fun l set -> Terms.subset set (find l m2)) m1
    let print m =
        Map.iter (fun x terms ->
           print_endline (x ^ " |-> " ^ string_of_terms terms)
        ) m
end

let cfa : exp -> AbsCache.t * AbsEnv.t
=fun exp -> (AbsCache.empty, AbsEnv.empty) (* TODO *)
```

Implement function `cfa`:

```
cfa : exp -> AbsCache.t * AbsEnv.t
```

For example, `cfa ex1` produces

```
1 |-> FN y -> 3,
2 |-> FN x -> 1,
3 |->
4 |-> FN y -> 3,
5 |-> FN y -> 3,
x |-> FN y -> 3,
```

`cfa ex2` produces

```
1 |-> RecFN f x 4,
2 |->
3 |-> FN y -> 2,
4 |->
5 |-> RecFN f x 4,
6 |-> RecFN f x 4,
7 |->
8 |-> FN z -> 7,
9 |->
10 |->
f |-> RecFN f x 4,
g |-> RecFN f x 4,
x |-> FN y -> 2, FN z -> 7,
```

and `cfa ex3` produces

```
1 |-> FN x -> 1, FN y -> 6,
2 |-> FN x -> 1,
3 |-> FN x -> 1,
4 |-> FN x -> 1,
5 |-> FN x -> 1, FN y -> 6,
6 |-> FN y -> 6,
7 |-> FN y -> 6,
8 |-> FN x -> 1, FN y -> 6,
9 |-> FN x -> 1, FN y -> 6,
f |-> FN x -> 1,
x |-> FN x -> 1, FN y -> 6,
y |-> FN y -> 6,
```