Assignment 5
COMP 302 Programming Languages and Paradigms

Andrew Cave and Francisco Ferreira
MCGILL UNIVERSITY: School of Computer Science

Due Date: 5 Apr 2013

In this homework, we will explore more concepts related to evaluation, type checking and type inference for MiniML. The language we are considering in this homework is essentially the same as we saw in class; it includes tuples (pairs written as $(e_1,\ e_2)$), functions, function application, and recursion and is hence a subset of SML.

You will find several files in the directory `mini-ml` which allow you to parse input files written in MinML, and print back programs in MinML, and the implementation of the interpreter has been described in class.

Q1 (25 points) In the first part of the homework, we will use bidirection typechecking and we also allow type variables to model polymorphism.

IMPORTANT: To get started, read the `README` file, which describes how to run the existing code, evaluate MinML code, type-check MinML code, etc. To get a feel for the source-level syntax, check out some of the example programs in the directory `examples`. At this point we do not have a type checker, so the typing annotations in the program are useless and they will not be checked. We will implement a type checker in the last homework building on the same framework.

Your task: First, read the notes `bitype.pdf` listed on the schedule webpage. Then, go into `bitype.sml` and write the cases of the functions `synth`, `check` and `bind` for if-then-else expressions, tuples, functions, function applications, etc. described below. To translate between the rules and the code, note that

$\Gamma \vdash e \Rightarrow \tau$	corresponds to	`synth` Γ `e` (returning τ)
$\Gamma \vdash e \Leftarrow \tau$	corresponds to	`check` Γ `e` τ (returning `()`)
$\Gamma \vdash decs \Rightarrow \Gamma'$	corresponds to	`bind` Γ `decs` (returning Γ, Γ')

(In `bind`, the result is Γ, Γ', which is the standard notation in typing rules for the concatenation of contexts; viewing Γ and Γ' as lists, it just means Γ appended with Γ'.)

In the directory `mini-ml`, complete the implementation of the bi-directional type checker by adding the cases in 1 inside the `bitype.sml`.

See the directory `examples` for some examples. To test type checker, you can use the top-level function `Top.file_type`. For example:

```
- Top.file_type "examples/if.mml";
```

Checking rules

$$\frac{\Gamma \vdash e \Leftarrow \mathsf{bool} \qquad \Gamma \vdash e_1 \Leftarrow \tau \qquad \Gamma \vdash e_2 \Leftarrow \tau}{\Gamma \vdash \mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 \Leftarrow \tau}\ \text{T-IF} \qquad \frac{\Gamma, x{:}\tau \vdash e \Leftarrow \tau'}{\Gamma \vdash (\mathsf{fn}\ x \Rightarrow e\ ) \Leftarrow (\tau \texttt{ -> } \tau')}\ \text{T-FN}$$

$$\frac{\Gamma \vdash e_1 \Leftarrow \tau_1 \qquad \cdots \qquad \Gamma \vdash e_n \Leftarrow \tau_n}{\Gamma \vdash (e_1, \ldots, e_n) \Leftarrow (\tau_1 * \cdots * \tau_n)}\ \text{T-TUPLE}$$

Synthesizing rules

$$\frac{\Gamma, f{:}\tau \vdash e \Leftarrow \tau}{\Gamma \vdash (\mathsf{rec}\ f : \tau \Rightarrow e\ ) \Rightarrow \tau}\ \text{T-REC} \qquad \frac{\Gamma \vdash e_1 \Rightarrow \tau \texttt{ -> } \tau' \qquad \Gamma \vdash e_2 \Leftarrow \tau}{\Gamma \vdash e_1\, e_2 \Rightarrow \tau'}\ \text{T-APP}$$

$$\frac{\Gamma \vdash \mathit{decs} \Rightarrow \Gamma' \qquad \Gamma, \Gamma' \vdash e \Rightarrow \tau}{\Gamma \vdash \mathsf{let}\ \mathit{decs}\ \mathsf{in}\ e\ \ \mathsf{end} \Rightarrow \tau}\ \text{T-LET}$$

Rules for declarations

$$\frac{\Gamma \vdash \mathit{dec}_1 \Rightarrow \Gamma_1 \qquad \Gamma, \Gamma_1 \vdash \mathit{decs} \Rightarrow \Gamma_2}{\Gamma \vdash \mathit{dec}_1\ \mathit{decs} \Rightarrow \Gamma_1, \Gamma_2}\ \text{T-DECS}$$

$$\frac{\Gamma \vdash e \Rightarrow \tau}{\Gamma \vdash (\mathsf{val}\ x = e) \Rightarrow (x : \tau)}\ \text{T-BY-VAL}$$

$$\frac{\Gamma \vdash e \Rightarrow (\tau_1 * \cdots * \tau_n)}{\Gamma \vdash (\mathsf{val}\ (x_1, \ldots, x_n) = e) \Rightarrow (x_1 : \tau_1), \ldots, (x_n : \tau_n)}\ \text{T-BY-VAL-TUPLE}$$

Figure 1: Typing rules for Mini-ML

Q2 15 points Extend the function `subst:(exp*string) -> exp -> exp` for substitution in the file `eval.sml` to handle tuples, functions, function application.

The first argument to the function `subst` contains a tuple of an expression `e'` and a variable name `x` denoting the substitution `[e'/x]`. The second argument is an expression `e` to which we apply the substitution. In other words,

```
subst (e',x) e = [e'/x]e
```

i.e. the function `subst` will replace any free occurrence of the variable `x` in the expression e' by the expression e

You can then for example test your substitution function as follows:

```
- E.subst (M.Int 5, "x") (M.If(M.Bool(true), M.Var "x", M.Var "y"));
val it = If (Bool true,Int 5,Var "y") : MinML.exp
```

Q3 (25 points) Extend the function `eval:exp -> exp` to evaluate let, tuples, functions, function application and recursion following the operational semantics given below. Recall that the judgement for evaluation:

$$e \Downarrow v \quad \text{expression } e \text{ evaluates to value } v$$

$$\frac{e \Downarrow v}{\mathsf{let}\ \cdot\ \mathsf{in}\ e\ \ \mathsf{end} \Downarrow v}\ \text{B-NO-DEC} \qquad \frac{e_1 \Downarrow v_1 \quad [v_1/x](\mathsf{let\ decs\ in}\ e\ \ \mathsf{end}) \Downarrow v}{\mathsf{let\ val}\ x_1 = e_1\ \mathsf{decs\ in}\ e\ \ \mathsf{end} \Downarrow v}\ \text{B-LET-VAL}$$

$$\frac{e_1 \Downarrow (v_1, \ldots, v_n) \quad [v_1/x_1, \ldots, v_n/x_n](\mathsf{let\ decs\ in}\ e\ \ \mathsf{end}) \Downarrow v}{\mathsf{let\ val}\ (x_1, \ldots, x_n) = e_1\ \mathsf{decs\ in}\ e\ \ \mathsf{end} \Downarrow v}\ \text{B-LET-TUPLE}$$

$$\frac{\text{for all } i \ \ e_i \Downarrow v_i}{(e_1,\ \ldots,\ e_n) \Downarrow (v_1, \ldots, v_m)}\ \text{B-TUPLE} \qquad \frac{[\mathsf{rec}\ f : \tau \Rightarrow e\ /f]e \Downarrow v}{\mathsf{rec}\ f : \tau \Rightarrow e\ \ \Downarrow v}\ \text{B-REC}$$

$$\frac{}{\mathsf{fn}\ x \Rightarrow e\ \ \Downarrow \mathsf{fn}\ x \Rightarrow e}\ \text{B-FN} \qquad \frac{e_1 \Downarrow \mathsf{fn}\ x \Rightarrow e \quad e_2 \Downarrow v_2 \quad [v_2/x]e \Downarrow v}{e_1\ e_2 \Downarrow v}\ \text{B-APP}$$

To test your evaluator, you can use the top-level function `Top.file_eval`. For example:

```
- Top.file_eval "examples/if.mml";
 3 ;
val it = () : unit
```

Q4 (35 points) For this question, go to the directory `mini-ml-inf`. Your task is to implement the function `unify:T.typ * T.typ -> unit` in the file `typing.sml`, which checks whether two types are unifiable, i.e. if we can make them syntactically equal. In this question we modified the datatype for MiniML types as follows:

```
datatypetp =                    (* MiniML types  *)
   Arrow of tp * tp             (* T := T1 -> T2 *)
 | Product of tp list           (*    | T1 * T2  *)
 | Int                          (*    | INT      *)
 | Bool                         (*    | BOOL     *)
 | TVar of (tp option) ref      (*    | a        *)
```

You should follow the description of unification in the class notes. Type variables are modeled via references. An uninstantiated type variable is modeled as a pointer to a cell with content `NONE`. In other words to create a new type variable we can simply use a function

```
fun freshVAR () = VAR (ref NONE)
```

Once we know what a type variable should be instantiated with we simply assign it the correct type. For example, if we have a type variable `VAR x`, then `x` has type `(typ option) ref` and we can replace every occurrence of `x` by the type `INT` using assignment `x := SOME(INT)`.

This will destructively update the type variable `x` and directly propagate the instantiation for it. No extra implementation of a substitution function is necessary to propagate instantiations.

Your task is to implement the function `unify:T.typ * T.type -> unit` which tests whether two types are unifiable. If two types are unifiable, they will be denoting the same type after unification succeeds. If unification fails, raise an exception.Follow the algorithm described in the notes to unify two types, and fill in the implementation for `unify` in file `typing.sml`.

We have provided a file `unify-test.sml` which shows you how to test the `unify` and verify it is working correctly.

Q5 (15 points) **EXTRA CREDIT** In this question, implement the function `typeOf:MiniML.exp -> T.typ` in the file `typing.sml` in the directory `mini-ml-inf`. Instead of annotating the input argument to functions and recursion with types to resolve ambiguity, we would like to infer the type without type annotations in our expressions. Copy and modify your implementation from Q1 in such a way that the function `typeOf` will infer the most genera type for an expression using unification. This is quite easy: if you don't know the type, you generate a new type variable, and instead of checking for equality, you call unification.

Fill in the code for `typeOf:MinML.exp -> T.typ` in file `mini-ml-inf/typing.sml`.