1 Instructions

Begin by downloading the file hw5-base.ml from the course website and renaming it to hw5.ml. This file contains the functions that you will use and modify in the homework. You will need to define new functions and modify step_cmd, but you should not need to modify any of the other predefined functions. Submit your completed hw5.ml via Gradescope. As always, please don’t hesitate to ask for help on Piazza (https://piazza.com/class/jushbmjmi1l3yb).

2 Typechecking Object-Oriented Programs

In the second half of HW2, you wrote a function that took an arithmetic or boolean expression and returned its type. In this homework, you’ll fill in pieces of a similar typechecker for a simple object-oriented language.

The file hw5-base.ml defines the types exp of expressions and cmd of commands for a simple Java-like language. It also defines two core functions: type_of, which takes an expression and returns its type, and typecheck_cmd, which takes a command and checks whether it is well-typed (returning true if it is and false otherwise). Mathematically, type_of ct Γ e should return Some t exactly when Γ ⊢ e : t, and typecheck_cmd ct Γ c should return true exactly when Γ ⊢ c : ok. The following problems will ask you to extend these two functions to handle the object-oriented features of the language.

Both type_of and typecheck_cmd take two additional arguments: a class table ct, which maps class names to their definitions, and a type context gamma, which holds the types of variables. A class definition (type cdecl) takes the form Class (c, d, fields, methods), where c is the name of the class being defined, d is its superclass, fields is a list of field declarations (each consisting of a type and a field name), and methods is a list of method declarations. As an example, here is a class declaration in concrete syntax:
class Square extends Shape {
    int side;

    int area() {
        return this.side * this.side;
    }
}

and here is its corresponding OCaml term:

Class ("Square", "Shape", [(Tint, "side")],
    [MDecl (Tint, "area", [],
        Return (Mul (GetField (Var "this", "side"),
            GetField (Var "this", "side")))))])

The file also includes the following helper functions:

- **fields**, which takes a class table and a class, and returns the list of fields of that class (including those defined in superclasses)

- **methods**, which takes a class table and a class, and returns the list of methods of that class (including those defined in superclasses)

- **types_of_params**, which takes a list of parameters/field definitions and returns just their types

- **field_type**, which takes a class table, class, and field name, and returns the type of that field of the class, if it exists

- **lookup_method**, which takes a class table, class, and method name, and returns the declaration of that method of the class, if it exists

- **subtype**, which takes a class table and two types t1 and t2, and returns true if t1 is a subtype of t2 (i.e., t1 <: t2)

- **typecheck_list**, which takes a class table, a type context, a list of expressions, and a list of types, and returns true if each expression in the list has the corresponding type according to **type_of**.

### 3 Problems

1. (5 points) Extend the provided **type_of** function with a case for **GetField**, the field access expression, according to the following rule:
\[
\Gamma \vdash e : C \quad \text{field_type } C \ f = \tau
\]
\[
\Gamma \vdash e.f : \tau
\]

Note that the class table \(\text{ct}\) is implicit in this and the following rules; it does not appear in the rules, but it still needs to be passed as an argument to any relevant functions.

Once you have completed this problem, \texttt{type_of ct0 gamma0 test0} should return \texttt{Some Tint}.

2. (5 points) In class, our typing rules included a \textit{subsumption} rule, saying that we can always treat an object of type \(\tau_1\) as also being of type \(\tau_2\) as long as \(\tau_1 <: \tau_2\). However, the subsumption rule is difficult to translate into code, since it could be applied in every expression case. An alternative approach is to introduce subtyping checks in each of the commands. For instance, the modified assignment rule is:

\[
\begin{align*}
\Gamma(x) &= \tau_1 \\
\Gamma \vdash e : \tau_2 \\
\tau_2 &<: \tau_1 \\
\Gamma \vdash x = e : \text{ok}
\end{align*}
\]

In other words, we can assign \(e\) to \(x\) as long as the type of \(e\) is a subtype of the type of \(x\). Extend the provided \texttt{typecheck_cmd} function with a case for \texttt{Assign}, using this rule. Once you have completed this problem, \texttt{typecheck_cmd ct0 gamma2 test1} should return \texttt{true}.

3. (5 points) Extend the provided \texttt{typecheck_cmd} function with a case for \texttt{SetField}, the field assignment command, according to the following rule:

\[
\begin{align*}
\Gamma \vdash e : C \\
\text{field_type } C \ f = \tau_1 \\
\Gamma \vdash e_1 : \tau_2 \\
\tau_2 &<: \tau_1
\end{align*}
\]
\[
\Gamma \vdash e.f = e_1 : \text{ok}
\]

Once you have completed this problem, \texttt{typecheck_cmd ct0 gamma0 test2} should return \texttt{true}.

4. (4 points) Extend the provided \texttt{typecheck_cmd} function with a case for \texttt{New}, the object creation command, according to the following rule:

\[
\begin{align*}
\Gamma(x) &= \tau_0 \\
\text{fields } C &= \tau_1 \ f_1, \ldots, \tau_n \ f_n \\
\Gamma \vdash e_1 : \tau_1 \\
\ldots \\
\Gamma \vdash e_n : \tau_n \\
C &<: \tau_0
\end{align*}
\]
\[
\Gamma \vdash x = \text{new } C(e_1, \ldots, e_n) : \text{ok}
\]

The function \texttt{types_of_params} can be used to extract the types from the list of fields of a class, and \texttt{typecheck_list} can be used to check whether a list of expressions matches a list of types.

Once you have completed this problem, \texttt{typecheck_cmd ct0 gamma0 test3} should return \texttt{true}.

3
5. (6 points) Extend the provided `typecheck_cmd` function with a case for `Invoke`, the method invocation command, according to the following rule:

\[
\begin{align*}
\Gamma(x) &= \tau_0 \\
\Gamma \vdash e : C \\
\text{lookup_method } C \quad m &= \tau \quad m(\tau_1 \; x_1, \ldots, \tau_n \; x_n) \\
\Gamma \vdash e_1 : \tau_1 \\ & \quad \vdots \\ \Gamma \vdash e_n : \tau_n \\
\tau <: \tau_0 \\
\Gamma \vdash x = e.m(e_1, \ldots, e_n) : \text{ok}
\end{align*}
\]

Once you have completed this problem, `typecheck_cmd ct0 gamma1 test4` should return `true`.