HW5 – Object-Oriented Languages

CS 476, Fall 2023

1 Instructions

Begin by downloading the file `hw5-base.ml` from the course website and renaming it to `hw5.ml`. Then fill in your answers to the problems, adding or modifying definitions as you see fit. Submit your completed `hw5.ml` via Gradescope. As always, please don't hesitate to ask for help on Piazza (`https://piazza.com/class/lkwp62qwo734i9/`).

2 Typechecking Object-Oriented Programs

In the second half of HW2, you wrote parts of a typechecking function for arithmetic and boolean expressions. In this homework, you'll write parts 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). Both `type_of` and `typecheck_cmd` also take a *type context* `gamma`, which holds the types of variables and the definitions of classes. Mathematically, `type_of` Γ e should return `Some t` exactly when $\Gamma \vdash e : t$, and `typecheck_cmd` Γ c should return `true` exactly when $\Gamma \vdash c : \text{ok}$. The following problems will ask you to complete the implementation of these two functions.

The file also includes the following helper functions:

- `fields`, which takes a type context and a class, and returns the list of fields of that class (including those defined in superclasses)
- `methods`, which takes a type context 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 type context, class, and field name, and returns the type of that field of the class, if it exists
- `lookup_method`, which takes a type context, class, and method name, and returns the declaration of that method of the class, if it exists

- `typecheck_list`, which takes 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`.
- `subtype`, which takes a type context and two types `t1` and `t2`, and returns `true` if `t1` <: `t2`.

1. (5 points) The `subtype` function depends on an unimplemented helper function `supers`, which takes a type context and a class name `c`, and should return the list of all superclasses of `c` in the context (i.e., `c`'s superclass, `c`'s superclass's superclass, etc.). Implement the `supers` function. Remember that the built-in class `Object` has no superclass.

   Once you have completed this problem,
   `subtype ct0 (ClassTy "Square") (ClassTy "Object")` should return `true`.

2. (5 points) Extend the provided `type_of` function with a case for `GetField`, the field access expression, according to the following rule:

$$\frac{\Gamma \vdash e : C \qquad (\mathsf{field\_type}\ C\ f = \tau)}{\Gamma \vdash e.f : \tau}$$

   Once you have completed this problem, `type_of gamma0 exp2` should return `Some IntTy`.

3. (5 points) Extend the provided `typecheck_cmd` function with a case for `New`, the object creation command, according to the following rule:

$$\frac{(\Gamma(x) = \tau_0) \qquad (\mathsf{fields}\ C = \tau_1\ f_1, ..., \tau_n\ f_n) \qquad \Gamma \vdash e_1 : \tau_1\ \ ...\ \ \Gamma \vdash e_n : \tau_n \qquad C <: \tau_0}{\Gamma \vdash x\ \texttt{:=}\ \texttt{new}\ C(e_1, ..., e_n) : \mathrm{ok}}$$

   Note that we can assign the new object of class C to x as long as C is a subtype of the type of x (written $C <: \tau_0$ in the rule). The function `types_of_params` can be used to extract the types from the list of fields of a class, and `typecheck_list` can be used to check whether a list of expressions matches a list of types.

   Once you have completed this problem, `typecheck_cmd gamma0 cmd3` should return `true`.

4. (for graduate students) Extend the provided `typecheck_cmd` function with a case for `Invoke`, the method invocation command, according to the following rule:

$$\frac{\begin{array}{c}(\Gamma(x) = \tau_0) \qquad \Gamma \vdash e : C \qquad (\mathsf{lookup\_method}\ C\ m = \tau\ m(\tau_1\ x_1, ..., \tau_n\ x_n)) \\ \Gamma \vdash e_1 : \tau_1\ \ ...\ \ \Gamma \vdash e_n : \tau_n \qquad\qquad\qquad\qquad \tau <: \tau_0\end{array}}{\Gamma \vdash x\ \texttt{:=}\ e.m(e_1, ..., e_n) : \mathrm{ok}}$$

   Once you have completed this problem, `typecheck_cmd gamma1 cmd4` should return `true`.