CSE P505, Spring 2006, Assignment 4
Due: Tuesday 23 May 2006, 5:00PM

Last updated: May 9. Problem 3 is independent of problems 1 and 2. Consult the posted ML files and `lang.pdf`.

1. Complete the functions `subtype` and `typecheck` to provide a type-checker for the language defined in `hw4Prob1.ml` and described in `lang.pdf`. For `subtype t1 t2`, return true if `t1` is a subtype of `t2`, else return false. For `typecheck`, raise the `DoesNotTypecheck` exception or return the type of the expression. The exception carries a string: you may find it useful to use different strings for different errors, but the strings will not be graded.
   For subtyping:

   - The only subtype of `IntT` is itself.
   - Subtyping for arrow types is as usual.
   - Subtyping for records includes width and permutation as in class. For depth, the subtyping depends on the field's access modifier. You must be sound but as expressive as possible. (The correct rules are for you to figure out.)

   For typechecking:

   - The "natural" typechecking rules are mostly left to you. See `lang.pdf`.
   - `If(e1,e2,e3)` typechecks if `e1` has a subtype of `IntT`[1] and one of `e2` and `e3` has a subtype of the other. The supertype of these two types is the type of the whole expression. (This is the rule in languages like Java; see the extra credit.)
   - `RecordV e` should not typecheck because it should not appear in source programs.
   - For `Get` and `Set`, be sure to consult the access modifier.
   - Call `checkType` on every explicit type in the program (the result is `()`, but it may raise an exception). For `RecordE`, be sure the field names are unique. Together, these two checks ensure no record type will ever have repeated fields.

   Begin Hints:

   - For subtyping, do not include code for reflexivity and transitivity. In other words, create a straightforward algorithm that *is* reflexive and transitive, though that may not be obvious.
   - The sample solution for `subtype` is 13 lines (including the 2 lines given to you) and uses `List.for_all`.
   - The typechecker does not need a subsumption rule. Instead you just use `subtype` whereever subsumption may be needed. For example, in an applicaton, see if the argument has a subtype of the type the function expects.
   - The sample solution for `typecheck` is 61 lines, (including the 10 lines given to you) and uses `List.map`.

   End Hints

2. Consider these two functions in our language (only their types differ):

```
Lam("x",RecordT[("l",IntT,Read)],
    RecordE[("l1",Get(Var("x"),"l")); ("l2",Var("x"))])
Lam("x",RecordT[("l",IntT,Both)],
    RecordE[("l1",Get(Var("x"),"l")); ("l2",Var("x"))])
```

   In English, describe:

[1] It happens that means `e1` has exactly type `IntT`, but in general we always allow subtypes.

- A situation where using the first function would typecheck but the second would not
- A situation where using the second function would typecheck but the first would not
- How to use *bounded polymorphism* to overcome this code duplication (a few sentences is plenty, we do not need a full language design and description of the type system)

Begin Hints:

- Bounded polymorphism for *types* is one topic in Lecture 8, but here think about access modifiers.

End Hints

3. `hw4clients.mli` describes 3 clients that are passed 3 different interfaces to a "counter" where an interface has the counter itself and functions for incrementing the counter, permitting decrements on the counter, decrementing the counter, and printing the counter's value. `hw4clients.ml` provides (trivial) client implementations. You need to change `hw4prob3.ml`. A Makefile is provided to check that everything compiles (`make`) and that it has the right types (`make interfaces`).

   The goal in all cases is to enforce this *protocol*: A client may not decrement the counter unless it has previously passed the counter to the function that allows decrements on it. (Once it has been passed once, any number of decrements are allowed.) **Hint: 5–6 lines is enough in each case.**

   (a) The type of `client1` does not enforce the protocol. Write `dynamicCheck` to have type `'a t1 -> int`. It takes a counter interface and produces a new interface also of type `'a t1` that checks the protocol dynamically by wrapping a couple functions in the old interface and raising `ProtocolViolation` as soon as the protocol is violated. It calls `client1` with this new interface.
   Hint: Use mutation (yes, really)

   (b) The type of `client2` enforces the protocol. Write `withholdDecr` to have type `'a t1 -> int`. It takes a counter interface, produces a new interface (of type `'a t2`) by wrapping the functions in the old interface, and calls `client2` with this new interface.
   Hint: The allow-decrement function returns the decrement function.

   (c) The type of `client3` enforces the protocol. Write `useTypes` to have type `'a t1 -> int`. It takes a counter interface and produces a new interface (of type `'a t3`) by wrapping the functions in the old interface, and calls `client3` with this new interface.
   Hint: There is a very easy solution; the type `'a t3` is doing all the hard stuff.

   (d) In a few (or less) English sentences, explain how each of the three approaches enforces the protocol.

4. **(Extra Credit)**

   (a) Show that our language does not have *least supertypes*. That is, demonstrate four types $\tau_1, \tau_2, \tau_3, \tau_4$, such that $\tau_1 <: \tau_3$, $\tau_1 <: \tau_4$, $\tau_2 <: \tau_3$, and $\tau_2 <: \tau_4$, but $\tau_3 \not<: \tau_4$ and $\tau_4 \not<: \tau_3$. Hint: you may want to define additional types, which help show your claims.

   (b) First, for each of the three approaches in problem (3), explain how to change the client's type and the wrapper you wrote so you can pass the client *two* counters, with the protocol that you cannot call decrement with a counter until *that* counter has been passed to the allow-decrements function. Second, explain why approaches (b) and (c) cannot be extended to a *list* of counters and what addition to ML's type system would make this possible.

Turn in:

- Email your solutions to Ben. Include `hw4Prob1.ml` and `hw4Prob3.ml` as attachments. Put your solutions to problem 2 and 3d in a text, pdf, or Word document.
- **If you are using Seminal, please include your backup files.**