CSE 505, Fall 2006, Assignment 4
Due: Tuesday 28 November 2006, 5:00PM

Last updated: November 18

You also need `hw4.ml`, available on the course website.

1. (Curry-Howard Isomorphism)

   (a) Consider the logical formula $((p + q) \to r) \to ((q \to r) + (p \to r))$ (where $+$ is disjunction and $\to$ is implication). Explain in English what this formula says and argue informally that it is true.

   (b) Consider the type $((p + q) \to r) \to ((q \to r) + (p \to r))$ (where p, q, and r are base types for which there are no constants) in simply-typed lambda-calculus with sums (as in lecture). Give *two* programs with this type, where one uses an expression of type $q \to r$ to build the result and the other uses an expression of type $p \to r$.

   (c) Consider the logical formula $((q \to r) + (p \to r)) \to ((p + q) \to r)$. Explain in English what this formula says and argue informally that it is false.

   (d) Consider the type $((q \to r) + (p \to r)) \to ((p + q) \to r)$. Argue informally why it is impossible to write a function that has this type. (Do not say because of the Curry-Howard Isomorphism; explain in terms of what values of various types might be.)

2. (References and Subtyping) Consider a simply-typed lambda-calculus including mutation (as defined in homework 3), records, and subtyping (as defined in lecture 11). In other words, it has mutable references and immutable records, plus all the subtyping rules considered in lecture. This "combined language" has no subtyping rule for reference types yet (see below).

   (a) **Extra Credit:** Extend your type-safety proof from homework 3 to encompass this new language (i.e., with the expression forms for records, subsumption, and the subtyping rules from lecture).

   (b) Write down an inference rule allowing *covariant* subtyping for reference types. Show this rule is *unsound*. (To show a rule is unsound, assume the language without the rule is sound (as proven in the previous problem). Then give an example program, show that your example type-checks using the rule, and that evaluating the program can get to a stuck state.)

   (c) Write down an inference rule allowing *contravariant* subtyping for reference types. Show this rule is *unsound*.

   (d) Write down an inference rule allowing *invariant* subtyping for reference types. Invariant subtyping means it must be covariant and contravariant. This rule is sound, but you do not have to show it. However, show that this rule is not *admissable* (i.e., it allows programs to type-check that could not type-check before). Keep in mind our language already has reflexive subtyping (so we can already derive $\tau \leq \tau$ for all τ).

3. (System F and parametricity)

   (a) Suppose in System F we have $\cdot; \cdot \vdash e : \forall\alpha.(\alpha * \alpha) \to (\alpha * \alpha)$. Describe all the possible values that e could be equivalent to.

   (b) Suppose in Caml we have a value v of type `'a * 'a -> 'a * 'a`. Describe *all* the ways v could behave.

   (c) Unsurprisingly, a Caml function of type `'a -> int -> unit` can never "tell" if its first argument is an `int`. Surprisingly, a Caml function of type `'a ref -> int ref -> unit` can *sometimes* "tell" if its first argument is an `int ref`, *without* using pointer-equality on references and *without* using any kind of equality on expressions of type `'a`. Write such a function. (This is tricky; feel free to ask for hints. You can, and in fact need to, use comparison on integers.)

4. (Implementing subtyping and strong interfaces) See `hw4.ml` for a simple definition for types; you will add subtype checking. Much of the file is duplicated for parts (b) and (c), so we explain the intended use by a client in terms of the first version:

   - The client can build whatever types they want using the constructors in `typ`.
   - To see if `t1` is a subtype of `t2`, the client calls `make_subsumption1 t1 t2`, which raises an exception if they are not subtypes, else it returns a "subsumption."
   - The function `check` takes a "subsumption." As you will see in parts (d)–(g), this function does not really check anything correctly, and after fixing the interface the function is essentially unnecessary.

   (a) Implement `is_subtype1`, which should return true if `t1` is a subtype of `t2` according to the rules from lecture and problem 2. Note you will need an algorithm, so there will be no use of transitivity. The sample solution is only about 12 lines long. When writing `is_subtype1`, do not use `cache1`, but do understand how `make_subsumption1` uses it.

   (b) Implement `is_subtype2`, which should return true if `t1` is a subtype of `t2` according to the rules from lecture and problem 1. The algorithm should be largely the same as with `is_subtype1`, but (1) any pair of types encountered during the algorithm that have a subtype relationship should be added to `cache2`, and (2) `cache2` should be consulted at each step in the recursion. The sample solution has a helper function mutually recursive with `is_subtype2`.

   (c) Describe a sequence of n calls to `make_subsumption1` that take a total of $O(n^2)$ time but the same calls to `make_subsumption2` take a total of $O(n)$ time (i.e, $O(1)$ per call).

   (d) Create the "most permissive" interface for `hw4.ml` by using `ocamlc -i` and put it in `hw4.mli`.

   (e) Make the minimal changes to `hw4.mli` such that no client call to `check` can ever cause an exception to be raised. Explain in a comment in `hw4.mli` what changes you made and why.

   (f) Make the minimal changes to `hw4.mli` such that no client call to `check` can ever return false. Explain in a comment in `hw4.mli` what changes you made and why.

   (g) Having made these interface changes, modify the implementation of `check` to be much more efficient.

What to turn in:

- Hard-copy (written or typed) answers to problems 1–3 and 4c.
- Caml source code in files `hw4.ml` and `hw4.mli` for problem 4.

Email your source code to Anna as `firstname-lastname-hw4.tgz` or `firstname-lastname-hw4.zip`. The code should untar/unzip into a directory called `firstname-lastname-hw4`. Hard copy solutions should be put in Anna's grad student mailbox or given to her directly.