Implementing uML: Hindley-Milner Type Inference

Formerly due *Thursday, March 31, at 11:59PM*
Deadline extended! to *Sunday, April 3 at 11:59PM*

In this assignment you will implement Hindley-Milner type inference, which represents the current ``best practice" for flexible static typing. The assignment has two purposes:

- To help you develop a deep understanding of type inference
- To help you continue to build your ML programming skills

Complete problems U and T below, and from Chapter 7 in Ramsey and Kamin, complete Exercises 1, 2, 5, 6, 8 and 9.

Two problems to do by yourself

Complete exercises 1 and 2 on page 286 of Ramsey and Kamin.
These exercises explore some implications of type inference. The answers to both questions should go into file `1-2.uml`; your answer to question 2 should appear in a comment.
In your answer to question 1, **do not use `letrec`**.

Six problems to do with a partner

Complete exercises 5, 6, 8 and 9 from pages 287–288 of Ramsey and Kamin, and the two problems U and T below.
For the coding problems you'll be modifying the interpreter in `book-code/bare/uml/ml.sml`.

U. Test cases for unification.
Submit three test cases for unification. At least two of these test cases should be for types that have no unifier. Assuming that *we provide a function* `unifyTest : ty * ty -> answer`, put your test cases in file `utest.sml` as three successive calls to `unifyTest`. Do *not* define `unifyTest` yourself.

Here is a sample `utest.sml` file:

```
val _ = unifyTest (TYVAR "a", TYVAR "b")
val _ = unifyTest (CONAPP (TYCON "list", [TYVAR "a"]), TYCON "int")
val _ = unifyTest (TYCON "bool", TYCON "int")
```

Naturally, you will supply your own test cases.

T. Test cases for type inference.
Submit three test cases for type inference. At least two of these test cases should be for terms that fail to type check. Each test case should be a definition written in uML. Put your test cases in a file `ttest.uml`. Here is a sample `ttest.uml` file:

```
(val weird (lambda (x y z) (cons x y z)))
(+ 1 #t)
(lambda (x) (cons x x))
```

Naturally, you will supply your own test cases.

For the remaining problems, here are some additional remarks and suggestions.

- Implement unification
  Complete Exercise 6 on page 287 of Ramsey and Kamin. Be sure your unifier produces the correct result on our three test cases and also on your three test cases.
  This problem is probably the most difficult part of the assignment, and you are well advised to show your unification code to the course staff before proceeding with type inference.

Hints: Read the **Unification** section on pages 272–273. Be careful when you unify a list that you use all the information you compute, and that you use it as soon as possible. You'll be passing subsitutions like mad. It may be easiest to unify the tails first, then the heads. For ideas, you might want to look at the `typesof` function on page 276 of Ramsey and Kamin.

- Plan for type inference
  Complete Exercise 5 on page 287 of Ramsey and Kamin. Put your answer in file `rules.pdf`.
- Implement type inference
  Complete Exercise 8 in Ramsey and Kamin.
- New primitives
  Complete Exercise 9 in Ramsey and Kamin.

This is one assignment where it pays to run a lot of tests, of both good and bad definitions. *The most effective test of your algorithm is not that it properly assign types to correct terms, but that it reject ill-typed terms.* This assignment is your best chance to earn the large bonuses available by finding bugs in the instructor's code. I have posted a functional topological sort that makes an interesting test case.

Incidentally, if you call your interpreter `ml.sml`, you can build a standalone version in `a.out` by running `mosmlc ml.sml` or a faster version in `ml` by running `mlton -output ml ml.sml`.

Extra Credit

For extra credit, you may complete any of the following:

- Mutation, as in Exercise 13(a)(b) and possibly (c)
- Explicit types, as in Exercise 14
- Better error messages, as in Exercise 10(a)(b) and possibly (c)
- Tuples, as in Exercise 11
- Generative types, as in Exercise 12
- A proof of correctness of unification, i.e., Exercise 7, possibly omitting (d).

Of these problems the most interesting are probably Mutation (easy) and Explicit types (not easy).

Type soundness (very difficult).

Prove that the uML interpreter never raises `BugInTypeInference`. That is, prove that well-typed uML programs don't go wrong.

I'll accept such a proof at any time during the term, not just in time for this homework. Doing this extra credit correctly will almost certainly make a difference to your final course grade (unless you're already on track for an A).

Testing

The course interpreter is located in `/comp/105/bin/uml`. If your interpreter can process the initial basis and infer correct types, you are doing OK.

The *real* test of your interpreter is that it should reject incorrect definitions. You should prepare a dozen or so definitions that should not type check, and make sure they don't. For example:

```
(val bad (lambda (x) (cons x x)))
(val bad (lambda (x) (cdr (pair x x))))
```

Pick your toughest three test cases to submit for problem T.

Avoid common mistakes

Here are couple of common mistakes:

- A surprisingly common mistake is to **fail to apply a substitution**, as shown in the textbook (the example in the section on unification). Depending on how you organize your unifier, you will have more then on opportunity to make this mistake. And you can make the same mistake in type inference itself.
- Another common mistake is to create **too many fresh variables**.
- Another surprisingly common mistake is to include redundant cases in the code for inferring the type of a list literal. As is almost always true with functions that consume lists, it's sufficient to write one case for `NIL` and one case for `PAIR`.

There are also some common assumptions which are mistaken:

- It is a mistake to assume that an element of a literal list always has a monomorphic type.
- It is a mistake to assume that `begin` is never empty.

What to submit

For your solo work, run `submit105-ml-inf-solo` to submit file `1-2.uml`.

For your work with a partner, run `submit105-ml-inf-pair` to submit these files:

- `README`, telling us with whom you collaborated, how long you worked, what parts you finished (including any extra credit), and so on
- `utest.sml`, containing your answer to problem U
- `ttest.uml`, containing your answer to problem T
- `rules.pdf`, containing your answer to problem 5
- `ml.sml`, containing your answers to problems 6, 8, and 9
- optional file `transcript`

In the `README`, please tell us what parts of the assignment you have completed, including any extra-credit parts. If you want to show us additional evidence that your code works, put it in file `transcript`.

Your solutions are going to be evaluated automatically. We must be able to compile your solution in Moscow ML by typing, e.g.,

```
mosmlc ml.sml
```

If there are errors in this step, we will not grade your solution. Also, if you have defined any new exceptions, make sure they are handled. It's not acceptable for your interpreter to crash with an unhandled exception just because some code didn't type-check.