
HW 2 – Unification, Type Derivation and Lambda Calculus

CS 421 – Fall 2006

Revision 1.2

Assigned Tuesday, October 3, 2006

Due Wednesday October 18, 2006, 9:00 AM - Note the unusual time!

To be turned in at 2106 Siebel Center, office of Andrea Whitesell

Extension 48 hours (20% penalty)

1 Change Log

1.0 Initial Release.

1.1 Updated the title, corrected a few typos, added a clarification to problem 3.

1.2 Added “in f” to problem 2.

2 Turn-In Procedure

Your answers to the following questions are to be hand-written neatly or printed on one or more sheets of paper, each with your name in the upper right corner. The homework is to be turned in to my administrative assistant, Andrea Whitesell in 2106 Siebel Center by 9:00am of Wednesday 18 October 2006. Alternately, you may hand it to Prof. Elsa Gunter in person before the deadline.

3 Objectives and Background

The purpose of this HW is to test your understanding of

- How to unify a system of equations
- How to perform type derivations in simplified OCaml
- Alpha and beta conversion in the lambda calculus
- The consequences of different evaluation schemes
- How to represent datatypes in the lambda calculus

Another purpose of HW2 is to provide you with experience answering non-programming written questions of the kind you may experience on the midterm and final.

Caution: It is strongly advised that you know how to do these problems before the midterm.

4 Problems

1. Give a most general unifier for the following set of equations (unification problem). Capital letters (A, B, C, D) denote variables of unification. The lower-case letters (f, l, n, p) are constants or term constructors. (f and p have arity 2 - i.e., take 2 arguments, l has arity 1, and n has arity 0 - i.e. it is a constant.) Show your work by listing the operations performed in each step of the unification and the result of that step.

$$\{p(B, C) = p(l(n), l(p(B, A))), f(A, D) = f(A, p(l(n), C))\}$$

2. Give a complete type derivation for the following typing judgment's:

```
{ } ⊢ (let f =
      let rec f = fun a -> fun b ->
                    if a = 0 then b else f (a - 1) (b + 1)
      in fun x -> f x x in f) : int -> int
```

3. Given the following term:

$$(\lambda x. \lambda z. x(\lambda u. \lambda y. y)(xzz))((\lambda u. \lambda v. u(uv))(\lambda w. \lambda z. wz))(\lambda x. \lambda y. yx)$$

reduce this term as much as possible using each of

1. eager evaluation
2. lazy evaluation
3. unrestricted $\alpha\beta$ -reduction (i.e. by $\alpha\beta$ conversion that can be applied anywhere)

Label each step of reduction with the rule justifying it. You do not need to label uses of congruence.

4. (Extra Credit) Using the methodology discussed in class, give the lambda terms that encode the constructors, and the function `fold_left` for the `list` startup given as follows:

```
type 'a list = Nil | Cons of ('a * ('a list))
```

When translating to a lambda term, `Cons` should be treated as curried.