Computational Logic (320441) Fall 2015
Assignment 4: λ-Calculus
– Given Oct. 20, Due Oct. 29 –

In this assignment, we will implement the λ-calculus in `ProLog` or `Scala`. We will build on our work from the assignment on first-order tableaux, and we will extend the formulae by types and λ-expressions.

Problem 4.1 (Types)

Represent types as `ProLog` terms or `Scala` classes. 10pt

- For `ProLog`, use constants `e` and `t` for the base types, and the infix operator `->` (use the appropriate `op` declaration). Write a predicate `wft/1` that succeeds if its argument is a well-formed type.
- For `Scala` define classes `E`, `T` (for base types) and `Arrow` for composite ones.

Problem 4.2 (λ-terms)

Represent function application and lambda abstraction in `ProLog` or `Scala`. 5pt

- For `ProLog`, the types of constants will be given by a functional predicate `tconst/2`, which maps every constant to a type, e.g. we represent the fact that the `love` is a binary predicate by `tconst(love, e -> e -> t)`. Function application is represented by the infix operator `@`, so that we would represent "*Peter loves Mary*" as `love @ peter @ mary`. λ-abstractions will be represented as triples of the form `lambda(x,e,B)`, where the first argument is the bound variable – we use a `ProLog` constant for it, the second is its type, and the third the body (another formula).

Hint: Note that application is left-associative in contrast to the type constructor `->` above, which is right-associative, use the right operator declaration, so that you can save brackets.

- For `Scala`, define case classes `Cons(name, type)` and `Var(name)` for constants and variables where `name` is a `string` and `type` and λ-type from the previous problem. Moreover, declare `Apply(f,x)` and `Lambda(x, e, B)` where the arguments are the same as for the `ProLog` description.

Problem 4.3 (Type-Checking)

10pt

- For `ProLog`, define a type checking predicate `tc/2`, where `tc(F,T)` checks the whether the type of the formula `F` is `T`.

Hint: As the λ-binder introduces type assumptions for bound variables, you will need an internal predicate `tcaux/3`, which takes a list of type assumptions for the bound variables as an argument to make the recursion go through.

Note that the `tc` can also compute the type of course.

- For `Scala` define a function `tc(f)` that returns the type of the formula `f`. Raise an exception if the input is ill-typed.

Problem 4.4 (Free in)
Find out whether a variable is free in a formula. 10pt

- For `ProLog`, we have represented variables in the λ-calculus by `ProLog` variables, so we will have to determine whether some variable is free in a formula. Write a predicate `freein/2` that does that.
- For `Scala`, write a function `freein(f,x)` that checks if `x` is free in `f`.

Problem 4.5 (Free/Bound Variables)

15pt

- For `ProLog`, we have represented variables in the λ-calculus by `ProLog` variables, so we will need to have functions (functional predicates) that give us the free and bound variables of a λ-term.

Hint: For the predicate `free` interpret any atom (`ProLog` constant) that is is not a constant as a variable.

- For `Scala`, define two functions `free(a)` and `bound(a)` that return the free and, respectively, bound variables from `a`.

Hint: Totally disregard types in these functions.

Problem 4.6 (Alphabetic Variants)
Check whether two λ-terms can be obtained from each other by renaming bound variables. 10pt
Write a `ProLog` predicate or a `Scala` function `alphavariants/2` that checks whether two λ-terms can be obtained from each other by renaming bound variables

---

Hint: The best way to do this is to recurse down the two formulae in parallel, keeping a table of variable equivalences.

Problem 4.7 (Substitution)

25pt

We will need a notion of substitution in our representation of the λ-calculus.

- For `ProLog`, write a predicate `subst/4`, such that the query `subst(a,x,b,R)` binds `R` to the result of substituting `a` for every free occurrence of `x` in `b`.

Hint: Remember that $[\mathbf{B}/X](\lambda X.\mathbf{A}) = \mathbf{A}$ and that for computing $[\mathbf{B}/X](\lambda Y.\mathbf{A})$, where $Y \in \text{free}(\mathbf{B})$ we need to rename the variable Y in $\lambda Y.\mathbf{A}$ to avoid variable capture.

- For `Scala`, write a function `subst(a,x,b)` that returns result of substituting `a` for every free occurrence of `x` in `b`.

Problem 4.8 (β-Normalization)
Implement β-normalization in your λ-calculus. 15pt

- For `ProLog`, write a predicate `betanf/2`, so that the query `betanf(X,Y)` binds `Y` to the β-normal form of `X`.
- For `Scala`, write a function `betanf(x)` that returns the β-normal form of `x`.