Swinburne University Of Technology

Faculty of Science, Engineering and Technology

ASSIGNMENT COVER SHEET

Subject Code:	COS30023
Subject Title:	Languages in Software Development
Assignment number and title:	8X, Typed Lambda Calculus
Due date:	**optional, October 27, 2014, 10:30, on paper**
Lecturer:	Dr. Markus Lumpe

Your name:_____

Marker's comments:

Problem	Marks	Obtained
1	58	
Total	58	

Extension certification:

This assignment has been given an extension and is now due on _____

Signature of Convener:_____

Problem Set 8: Typed Lambda Calculus

A Type System for the Lambda Calculus with Constants:

- Number: `n` : **Int**

  A number `n` has type **Int**.

- Variable: $\Gamma \vdash$ `x` : Γ(`x`)

  If `x` $\in$ *dom*(Γ), then variable `x` has type Γ(x).

- Abstraction:
$$\frac{\Gamma;\ \texttt{x : t} \vdash \texttt{e : t'}}{\Gamma \vdash \texttt{(lambda x t . e) : (t -> t')}}$$

  If under the extended type environment Γ; `x : t` expression `e` has type `t'`, then `(lambda x t . e)` has type `(t -> t')` under the type environment Γ

- Application:
$$\frac{\Gamma \vdash e_1 \texttt{ : (t -> t')} \qquad \Gamma \vdash e_2 \texttt{ : t}}{\Gamma \vdash \texttt{(}e_1\ e_2\texttt{): t'}}$$

  If under the same type environment Γ e_1 has type `(t -> t')` and e_2 has type `t`, then `(`e_1 e_2`)` has type `t'` under type environment Γ.

Problem 1

Consider the following BNF specification:

```
CompilationUnit     ::=  (LambdaExp)+ <EOF>

LambdaExp           ::=  <Number>
                      |  <Variable>
                      |  LOOKAHEAD(2)
                         ( lambda <Variable> Type . LambdaExp )
                      |  ( <LambdaExp> <LambdaExp> )

Type                ::=  Int
                      |  ( Type -> Type )
```

- Define a front-end in JavaCC for this grammar using `TypedLambda` as parser name.
- Define the corresponding abstract syntax tree classes that extend the following top-level classes for lambda expressions and types:

```
public abstract class TypedLambdaExpression
{
 public abstract LambdaType typeCheck( Hashtable<String,LambdaType> aGamma );
 public abstract String toString();
}

public abstract class LambdaType
{
 public abstract boolean match( LambdaType aOtherType );
 public abstract String toString();
}
```

  Remember, in order to assign a term a type we need to use a type environment Γ that provides a mapping from all free variables in the term to proper types. The parameter `aGamma` for method `typeCheck` serves this purpose. The method `typeCheck` either returns the type of the expression, or throws a `RuntimeException` if we cannot deduce a proper type according to the typing rules defined above.
- Incorporate the abstract syntax into the specification.
- In this assignment, the abstract syntax tree nodes `IntegerType` and `FunctionType` represent types that both are subclasses of `LambdaType`. All type classes have to implement the `match` method. Two `IntegerType` objects match immediately. To match two `FunctionType` objects, you need to match both their argument types and their result types. So, if given two function types t → t' and s → s', then the method `match` will return true only, when t matches s and t' matches s'.

The main method:

```
public static void main( String[] Args )
{
  try
  {
    TypedLambda lp = new TypedLambda( new FileInputStream( Args[0] ) );
    ArrayList< TypedLambdaExpression> exprs = lp.CompilationUnit();

    for ( TypedLambdaExpression exp : exprs )
    {
      try
      {
        System.out.println( "Checking: " + exp );

        LambdaType type = exp.typeCheck( new Hashtable<String,LambdaType>() );

        System.out.println( "SUCCESS: " + exp + " has type " + type );
      }
      catch (RuntimeException e)
      {
        System.out.println( "Oops, type error encounted: " + e.getMessage() );
      }
    }
  }
  catch (ParseException e)
  {
    System.out.println( "Syntax Error : \n" + e.toString() );
  }
  catch (FileNotFoundException e)
  {
    System.out.println( e.toString() );
  }
  catch (RuntimeException e)
  {
    System.out.println( "Oops, type error encountered: " + e.getMessage() );
  }
}
```

Tests:

```
// type error encountered: Function type expected for '(x z)'.
(lambda x (Int -> Int). (lambda y (Int -> Int). (lambda z Int. ((x z) (y z)))))

// Success (add one one)
(((lambda m ((Int->Int) -> (Int -> Int)) .
  (lambda n ((Int -> Int) -> (Int -> Int)) .
    (lambda s (Int -> Int) .
      (lambda z Int .
        ((m s) ((n s) z))))))
  ((lambda n ((Int->Int) -> (Int -> Int)) .
      (lambda s (Int -> Int) . (lambda z Int . (s ((n s) z)))))
   (lambda s (Int -> Int) . (lambda z Int . z))))
    ((lambda n ((Int->Int) -> (Int -> Int)) .
       (lambda s (Int -> Int) . (lambda z Int . (s ((n s) z)))))
     (lambda s (Int -> Int) . (lambda z Int . z))))
```

Submission deadline: optional, Monday, October 27, 2014, 10:30.

Submission procedure: on paper.