Assignment 3 – COMP 527: Computation and Logic

Winter 2016
Due Feb 25, 2016

Exercise 1 (45 pts) Defining theories in LF.

In class, we showed how to encode logical formulas and natural deduction rules for them in LF concentrating on $\top$, conjunction, and implication. Your task here is to encode also disjunction and negation.

5 pts Define an extension to the data type `o` which describes formulas to support encoding of disjunction and negation using `or` for disjunction and `bot` for negation.

20 pts Define constants that correspond to introduction and elimination rules of disjunction and negation, i.e. extend the type family `nd`.

5 pts Describe new term constants that correspond to introduction and elimination rules of disjunction and negation, i.e. extend the type family `tm`.

15 pts Describe proof terms for the following formulas (which you proved in HW1):

- $\neg A \vee \neg B \supset (\neg(A \wedge B))$.
- $\neg A \wedge \neg B \supset \neg(A \vee B)$.
- $((A \wedge B) \vee C) \supset (A \vee C) \wedge (B \vee C)$

Choose what is easier for you: either define

```
rec q0 : [⊢ nd (imp (or (imp A bot) (imp B bot))  (imp (&  A  B) bot))
     ] = ? ;
```

and fill in the question mark; or define

```
rec q0 : [⊢ tm (imp (or (imp A bot) (imp B bot))  (imp (&  A  B) bot))
     ] = ? ;
```

and fill in the question mark. Proceed for all the above formulas

Exercise 2 (30 pts) Defining theoris in LF.

We define here lists in LF as follows:

```
LF list:type =
| nil : list
| cons: nat → list → list;
```

10 pts Define a type family `sorted` in LF together with constants describing that a list is sorted if its elements are in increasing order.

10 pts Define a type family `member` which takes in two arguments, X and a list L and it defines when X is a member of L and implement it in LF.

10 pts e a type family `insert` that takes in three arguments, X, an input list L and a list L' which describes the list L where X has been inserted. Insert should preserve the order, i.e. if L was sorted, then L' should be sorted as well. Implement the predicate `insert` in Beluga.

Remember that you can test your implementations of `sorted`, `member`, and `insert` by using the logic programming engine and asking queries. Give 3 test cases for `member`, `sorted`, and `insert`.

Exercise 3 (25 pts) Inductive proof as recursive program.

In class, we gave two definitions of even numbers. The first definition `even` defined even numbers starting from zero and adding two. The second definition `ev` defined even numbers mutual recursively with a number being odd.

```
LF even: nat → type =
| ev_z : even z
| ev_s : even N → even (s (s N));

LF ev: nat → type =
| e_z : ev z
| e_s : odd N → ev (s N)

and odd: nat → type =
| o_s: ev N → odd (s N);
```

We now want to prove that the two definitions are equivalent.

Theorem (Completeness): If ⊢ `even N` then ⊢ `ev N` and ⊢ `odd s N`.

Theorem (Soundness)

1. If ⊢ `odd N` then ⊢ `even (s N)`.
2. If ⊢ `ev N` then ⊢ `even N`.

Your task is to write the inductive proofs of these theorems as recursive functions in Beluga.