CSL765: Introduction to (Logic and Functional) Programming
I semester 2019-20

The language $\Lambda_{FL}(X)$ extends **FL(X)** to allow lambda-terms. All the well-typed terms in the language $\Lambda_{FL}(X)$ have a β-normal form. In this assignment, you have to check if a term is well-typed as per the types given in Typing FL Expressions. You can use the inference rules given in Typing axioms. If the term is well-typed, convert it to its β-normal form.

Problem Statement

Your task is to generate a β-normal form for any given input in language $\Lambda_{FL}(X)$. For this assignment you don't have to consider the language with recursion i.e. $\Lambda_{RecFL}(X)$. You can use the code of Assignment-3 to reduce terms of **FL(X)** to normal forms[1]. We have attached a signature file with this document. Your task is to create a module and implement the signature file named `signatureLAMBDAFLX.sml`.

What you have to do

1. Create a file `structureLAMBDAFLX.sml` implementing the signature provided in `signatureLAMBDAFLX.sml` file.
2. Name your structure that implements the given signature `LambdaFlx`. We will use `open LambdaFlx` in our script to use your structure.
3. Write a parser and tokenizer to process the input considering -
   - The name of the constructors are same as they are provided in language signature viz. Z, P, T, F, ITE, IZ, GTZ and $LAMBDA$, where $LAMBDA$ stands for λ.
   - Every constructor will be followed by at lease one space character.
   - The form of constructor P, S, IZ, and GTZ will be `‘(’ <cons name> ‘ ’ <lterm> ‘)’`
   - The form of constructor ITE will be `‘(’ ITE ‘<’ <lterm> ‘,’ <lterm1> ‘,’ <lterm0> ‘>’ ‘)’`.
   - The form of λ-term constructor will be `LAMBDA ‘ ’ <lterm0> ‘[’ <lterm> ‘]’`, where `<lterm0>` is the name of a variable in the language $\Lambda_{FL}(X)$.
   - The format of function application is `‘(’ <lterm> ‘ ’ <lterm> ‘)’`
   - You can safely assume any token apart from constructors, parenthesis, comma, square brackets and angular brackets to be the name of a variable i.e, a string $\in X$.
4. Check if the given input is a well-formed expression in the language $\Lambda_{FL}(X)$.
5. The function `fromString: string -> lterm` should take a string as input and return the lterm created after parsing. The function should raise `exception Not_wellformed` if the input string is not well formed in the language **FL0**.
6. Implement the function `toString: lterm -> string` which returns a string form of input lterm. The format of the output string should match with the format given above.
7. Function `fromInt: int -> lterm` takes an integer and converts it to an $SS \ldots SZ$ or $PP \ldots PZ$ form in the datatype lterm. For example `fromInt ~2` should return $(P\ (P\ Z))$.

[1]Since there is a clash of constructor names between the two data types `term` and `lterm`, disambiguation by prefixing each use of a constructor with the structure name may be required

8. Function `toInt: lterm -> int` should take a lterm with a sequence of constructors S and P applied over Z and return the corresponding integer number e.g, `toInt (S (S Z))` should return 2. If the lterm consists of a mix of S and P raise the exception `not_nf`. If the input lterm consists any other constructor, raise `not_int` exception.

9. Function `isWellTyped: lterm -> bool` uses type inference rules over the input lterm and returns `true` if the input lterm is well-typed, `false` otherwise. The function should raise exception `not_wellformed` if the input term is not a valid term in the language.

10. Function `betanf: lterm -> lterm` takes a lterm and returns its β-normal form. If the input lterm is not well-typed or well-formed, the function should raise `not_welltyped or not_wellformed` exceptions, respectively.

11. You are **not supposed to handle the exception in your program**. Our script depends on you raising the exception from your code.

12. Do not write any code in the signature file `signatureLAMBDAFLX.sml` attached with the assignment.

Submission Instruction

Submit a `structureLAMBDAFLX.sml` file on Moodle `https://moodle.iitd.ac.in`

Important Notes

1. Do not change any of the names given in the signature. You are not even allowed to change upper-case letters to lower-case letters or vice-versa.

2. You may define any new functions you like besides those mentioned in the signature.

3. Follow the input output specification as given. We will be using automated scripts to execute the code for evaluation. In case of mismatch, you will be awarded zero marks.

4. All of your code should be in file `structureLAMBDAFLX.sml`

5. Make sure your structure matches with the signature provided. We will not entertain any requests regarding minor change in the signature.

6. You can use the code form Assignment 3.