Computer Science 3MI3 – 2020 homework 7

Pretty printing a representation of the untyped -calculus.

Mark Armstrong

November 1st, 2020

Contents

Introduction

For this homework, you will extend the implementations of the `UL` untyped -calculus in Scala and Ruby, which are provided in assignment 2, with a *pretty printing* method.

This language implementation makes use of unnamed variables through *de Bruijn* indices. This simplifies the implementation; specifically,

- we eliminate the need to replace names of variables with "fresh" names during substitution; instead,
  - (a relatively tedious problem; how do we keep track of fresh names?)
- we must only shift variable indexes according to the number of enclosing 's (variable binders.)
  - (a relatively trivial problem.)

The one major downside to a representation using unnamed variables is that the terms are far less human readable. Instead of terms such as

```
 x →  y →  z → x y z
```

we now have terms such as

```
   2 1 0
```

Hence, the task of this homework to implement a *pretty printer*, that chooses variable names for a term and "prints" the term (in fact, it should convert terms to strings, not print them directly.)

Updates

November 5th

- The suggested output has been adjusted slightly to account for parentheses being added around terms being applied to each other.
- Sample code for how to interact with the `ULTerm` type was added to the assignment; you may want to try it out.
- The `ULTerm` code provided for the assignment has been updated,
  - first to correct a typo with a variable name, and
  - second to add `toString` methods which result in better output of the Scala `ULTerm` type.

  You may wish to apply those updates to your code as well.

Boilerplate

Submission procedures

Submission method

Homework should be submitted to your McMaster CAS Gitlab respository in the `cs3mi3-fall2020` project.

Ensure that you have **pushed** the commits to the remote repository in time for the deadline, and not just committed to your local copy.

Naming requirements

Place all files for the homework inside a folder titled `hn`, where `n` is the number of the homework. So, for homework 1, use the folder `h1`, for homework 2 the folder `h2`, etc. Ensure you do not capitalise the `h`.

Unless otherwise instructed in the homework questions, place all of your code for the homework in a single file in the `hn` folder named `hn.ext`, where `ext` is the appropriate extension for the language used according to this list:

- For Scala, `ext` is `sc`.
- For Prolog, `ext` is `pl`.
- For Ruby, `ext` is `rb`.

- For Clojure, `ext` is `clg`.

If multiple languages are used in the homework, submit a `hn.ext` file for each language.

If the language supports multiple different file extensions, you must still follow the extension conventions above.

Incorrect naming of files may result in up to a 10% deduction in your grade.

Do not submit testing or diagnostic code

Unless you are instructed to do so in the homework questions, **you should not submit testing code with your homework submission**.

This includes

- any `main` function,
- any `print` statements which output information **that is not directly requested as console output in the homework questions**.

If you do not wish to remove diagnostic print statements manually, you will have to find a way to ensure that they disabled in your final submission.

For instance, by using a wrapper on the print function or macros.

Due date and allowance for technical difficulties

Homework is due on the second Sunday following its release, by the end of the day (midnight). Submissions past 00:00 may not be considered.

If you experience technical difficulties leading up to the submission time, please contact Mark **ASAP** with the details of the problem and, if possible, attach the current state of your homework to the communication. This information will help ensure we are able to accept your submission once the technical difficulties are resolved.

Proper conduct for coursework

Individual work

Unless explicitely stated in the homework questions, all homework in this course is intended to be *individually completed.*

You are welcome to discuss the content of the homework in the public forum of the class Microsoft Teams team homework channel, though obviously solutions or partial solutions should not be posted or described.

Private discussions about the homework cannot reasonably be forbidden, but such discussions should follow the same guidelines as public discussions.

Inappopriate collaboration via private discussions which is later discovered by course staff may be considered academic dishonesty.

When in doubt, make the discussion private, or report its contents to the course staff by making a note of it in your homework.

To clarify what is considered appropriate discussions of homework content, here are some examples:

1. Discussing the language features introduced or needed for the homework.
    - Such as relevant builtin datatypes and datatype definition methods and their general use.
    - Code snippets that are not partial solutions to the homework are welcome and encouraged.
2. Questions of the form "What is meant by `x`?", "Does `x` really mean `y`?" or "Is there a mistake with `x`?"
    - Of course, questions of those form which would be answered by partial solutions are not considered appropriate.
3. Questions or advice about errors that may be encountered.
    - Such as "If you see a `scala.MatchError` you should probably add a catch-all `_` case to your `match` expressions."

Language library resources

Unless explicitely stated in the questions, it is not expected that you will use any language library resources in the homeworks.

Possible exceptions to this rule include implementations of datatypes we discuss in this course, such as lists or options/maybes, if they are included in a standard library instead of being builtin.

Basic operations on such types would also be allowed.

- For instance, `head`, `tail`, `append`, etc. on lists would not require explicit permission to be used.
- More complex operations such as sorting procedures would require permission before you used them.

Additionally, the standard *higher-order* operations including `map`, `reduce`, `flatten`, and `filter` are permitted generally, unless the task is to implement such a higher-order operator.

Part 1: The "pretty printer" `prettify` method [30 marks]

Implement a method `prettify` on the `UTTerm` type provided in assignment. The implementation of the `UTTerm` type can be found

- here in Scala, and
- here in Ruby.

Your method should take a `UTTerm` are return a string. As an example, given the `UTTerm` representing the -term (with unnamed variables)

```
  2 (1 0)
```

should produce a string such as

```
lambda x . lambda y . lambda z . (x) ((y) (z))
```

The exact choices of variable names do not matter; however, you must not change the meaning of the term. For instance, considering the above term, output

```
lambda x . lambda y . lambda x . (x) ((y) (x))
```

would be *incorrect*.

(You may want to use a helper method which has an argument to keep track of the variable names already in use. But as usual, the implementation details are within your control.)

Testing

:TODO: