Lab 2: Substitution Evaluator, applicative order

January 10, 2024

The goal of this lab is to implement our second evaluator for the *kl* interpreter, using the substition model, in applicative order. The goal is not to produce a fully functional interpreter, you are not expected to cover all the possible use cases for the language, but a restricted set which is delimited by the unit test.

1 Get Started

The base code used in the labs can be found in the directory `/courses/TDDE55/Labs/Lab2/`. In your repository for lab submission:

```
mkdir Lab2
cd Lab2
cp /courses/TDDE55/Labs/Lab2/normal_order.py .
```

2 Instructions

The base code is similar to the normal order, and in `applicative_order.py`, you will find the following classes: `Value`, `Environment`, `Function` and `Evaluator`.

In the `Evaluator` class:

- In the constructor (`__init__`), you will find the initialisation of the parser, you should use the same parser as in *Exercise 1*.
- The `eval` function is the entry point for evaluating a *kl* expression. It is devided in three steps:
  1. *parsing* using the Lark parser
  2. *eval_ast* which is a recursive function that evaluate a tree, and you need to implement that function. It is recommended you set the `debug` argument to `True` during testing to visualise the AST.

On a side note, normal and applicative order are very similar, feel free to re-use the code from Lab1, from either `reduce` or `expand` functions (copying is fine, you should not try to build a library).

3 Test

You can test your evaluator with:

```
tdde55_lab2_tests dir_to_lab2
```

The test suite can be found in `/courses/TDDE55/Labs/Lab2/Tests/evaluator.py`. The following test cases have been defined:

- `test_00_literal` test an expression with just a number
- `test_01_arithmetic` test arithmetic
- `test_02_cond` test conditional expression
- `test_03_assignment` test assigning a literal to a named value and computing expression
- `test_04_custom_function` test defining and calling a custom function
- `test_05_custom_function_div_0` test passing an invalid expression as an argument, notice that this test is different from *Lab 1*
- `test_06_custom_high_order_function` test defining a high order function

The first three tests should pass without changes to the evaluator, as they require no expansion.

4 Demonstration

- Make sure the unit test works on the thinlinc or lab computer
- Make sure your `eval_ast` is easy to understand and well commented