Bonus Assignment 5: GUI programming

Deadline: December 30th, 2022, 23:59

1 Introduction

In this assignment, you will adapt Assignment 3, to create a simple GUI for displaying abstract syntax trees. The program you write for this assignment is obtained by reusing and integrating your programs submitted for the previous assignments, and specified below.

This assignment is a bonus assignment, and as such does not have the same strict requirements as before. Your submission will be judged either as passing or as failing: a bonus point is awarded for submissions that are in line with the spirit of this assignment.

2 Interface

The program is compilable and works as a stand-alone program. For Javascript programmers, one can use a toolkit such as Electron (bundling a Web browser with your program) or NodeGui (Qt for Javascript). For C/C++ programmers, one can use a GUI toolkit such as GTK or Qt.

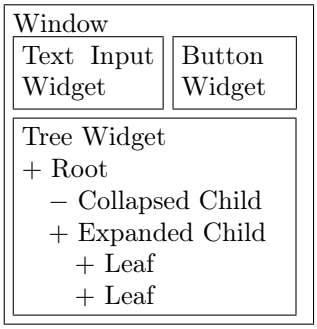
Make sure your program has clear instructions in the README how to compile and run it: when in doubt, ask the student assistant before submission whether any problems occur with compiling and running your program.

Input. The program shows a window with a text input widget in which the user can type a string of characters. Next to the input box is a button widget. If the user presses the button, the text in the input box is parsed as a judgement consisting of an expression and a type (see Assignment 3).

Process. After parsing the input into an abstract syntax tree, the program performs type checking (see Assignment 3 for type checking). The type checking process halts, and if type checking was succesful, the abstract syntax tree is displayed in a tree widget. If type checking was unsuccesful, a message dialog is shown that prompts the user with a message that the input was invalid (and you may show a partially constructed abstract syntax tree).

3 Mock-up

The following shows a mock-up of the visible widgets.



For example, the input $\lambda x^A\ x : A \to A$ shows the following tree:

+ Judgement
  + LambdaExpression (infer: $(A \to A)$)
    + Variable x
    + VarType A
    + VarExpression x (infer: A)
  + ArrowType
    + VarType A
    + VarType A

Another example, the input $\lambda x^A \lambda y^{A \to B}(y\ x) : A \to (A \to B) \to B$ shows:

+ Judgement
  + LambdaExpression (infer: $(A \to ((A \to B) \to B))$)
    + Variable x
    + VarType A
    + LambdaExpression (infer: $((A \to B) \to B)$)
      + Variable y
      + ArrowType
        + VarType A
        + VarType B
      + AppExpression (infer: B)
        + VarExpression y (infer: $(A \to B)$)
        + VarExpression x (infer: A)
  + ArrowType
    + VarType A
    + ArrowType
      + ArrowType
        + VarType A
        + VarType B
      + VarType B

(You may show the inferred type within brackets after each expression.)