



Proost 0.1.0

User manual

ENS Paris-Saclay

Arthur ADJEDJ
Vincent LAFEYCHINE

Augustin ALBERT
Lucas TABARY-MAUJEAN

4th December 2022

1 Introduction

This program provides the user a set of tools to work with λ -terms as described in the Calculus of Construction. Through the Curry-Howard correspondence; users may associate a property or theorem they are willing to prove to a type which can be expressed in CoC. Proving it then corresponds to constructing a term of that type.

Release 0.1.0 of the project includes a toplevel interface also called `proost`, which is the main way users can interact with this piece of software. For a use directly through the crate APIs, please refer to their respective documentations.

2 Toplevel session

In the toplevel, users are greeted with a prompt. There, they may enter the following commands:

- `import file1 file2` typechecks and loads the file in the current environment;
- `search v` looks for the definition of variable `v`;
- `def a := t` defines an alias `a` that can be used in any following command;
- `def a: ty := t` defines an alias `a` that is checked to be of type `ty`;
- `check u: t` verifies `u` has type `t`;
- `check u` provides the type of `u`;
- `eval u` provides the definition of `u`.

If the command succeeds, the toplevel returns a green check mark, with an associated result if there is any. Otherwise, a red cross indicates an error occurred, next to some details about it. The command is discarded and the user may enter another command.

The toplevel provides to a certain extent history browsing, either *via* the up and down arrow keys or some autocompletion from previous commands.

3 Language

This release includes no standard library, which means users have to build their theories from scratch. Besides, this version includes no notion of axioms, which means terms have to be built from encodings within CoC.

Language syntax is as such:

- Functions (λ -abstractions) are defined with the keyword `fun`: `fun x: A => u`, `fun x y: A => u` (both `x` and `y` are of type `A`), `fun x: A, y: B => u` (multiple arguments, where `B` may depend on `x`);
- Dependent function types (Π -types) are defined with a pair of parentheses before an arrow, as in: `(x: A) -> B`, `(x y: A) -> B` or `(x: A, y: B) -> C` (where the distinctions are similar to the previous item. Additionally, there is some usual syntactic sugar when the output type does not mention the input argument, which corresponds to usual function types: `A -> B`, `A -> B -> C` (right-associativity));
- Function application of two terms `u` and `v` is simply written `u v`, and is left-associative when there are multiple arguments;

- Variables are regular strings, which may only correspond to bound variables or previously defined terms;
- The type of propositions and higher-order types are written `Prop` and `Type i`, as usual.

This release includes a single example file `contraposition.mdln` which defines elementary propositional calculus operators and their constructors.