A theorem prover for scientific and educational purposes

M. Frank, C. Kreitz

{mafrank,kreitz}@uni-potsdam.de

Potsdam University
Institute for Computer Science

ThEdu’17, Gothenburg, August 6, 2017
Outline

1 Motivation and Context
2 Preliminaries
3 The IDE
4 Demonstration
5 Supported Platforms and Future Work
This work is a part of a (PhD) project with the aim to provide a framework:

- for rapidly prototyping calculi
- for the easy construction of ATP systems
- for (almost) lossless communication with ITP systems
- with a shallow learning curve for students
Motivation

Theorem provers are extremely valuable. We can prove

- Correctness of algorithms
- Termination of algorithms

We can even synthesise software from proofs.

But the way to a successful and happy user of theorem provers is a path of trial and tribulation for students.
In software construction courses, students are accustomed to IDEs with
- a shallow learning curve
- support with code completion, syntax highlighting, outlines, ...
- many hints by the IDE (tooltips)
- rather good usability

Theorem provers usually have
- a rather steep learning curve
- a less sophisticated usability
Motivation (3)

Many students have problems understanding functional programming, which is necessary for understanding theorem proving. Thus:

- improve the support for students in understanding the basics of theorem proving, like functional programming
- give them a tool with functionalities they are used to
- there is a need for training modes for theorem provers

While our main goal is a general theorem proving framework, this talk focuses on using it as a tool for $\lambda$-evaluation in the class room.
We analysed some $\lambda$-evaluation tools like

- the Penn Lambda Calculator
- the lambda calculus tracer TILC
- and many more online- and offline-tools

The benefit for students is low for most of the analysed tools as

- some of them do only evaluate terms (without interaction)
- some have no or no good visualisation (like binding scopes)
Focus and Related Work (2)

We provide an IDE based on our theorem proving framework

- with a mode for lambda term evaluation, manipulation and visualisation
- with some state of the art functionality of IDEs
- that is specifically adopted to wishes of students
- is being used regular in class since mid 2017
Untyped lambda calculus

I assume we all know the definition of untyped lambda calculus:

- $x$ is a variable,
- $\lambda x.t$ is an abstraction, binding occurrences of variable $x$ in term $t$
- $st$ is an application, i.e. $s$ is applied to $t$

Also, we know

- $\alpha$-conversion: $\lambda x.t \xrightarrow{\alpha} \lambda x'.t[x/x']$
- $\beta$-reduction: $(\lambda x.t)s \xrightarrow{\beta} t[x/s]$
Untyped lambda calculus

We introduce additionally

- The named term reference, i.e. abbreviations for term definitions
- $\equiv$-expansion: $t \xrightarrow{\equiv} def_t$

For the term $True =: \lambda x.\lambda y.x$, $True \xrightarrow{\equiv} \lambda x.\lambda y.x$ holds.
Features (1)

The IDE

- works on plain text files
- Encodes special characters automatically into UTF-8
- has an outline of defined terms
- has a manipulation view for deeper inspection and manipulation of terms.

Application of rules is possible in multiple ways

- $\alpha$-conversion via double-click on the bound variable
- $\beta$-reduction via drag and drop or shortcut keys
- $\equiv$-expansion via double-click on the named term reference
The tool currently supports highlighting:
- of corresponding parentheses
- of the variables bound by an abstraction
- of the abstraction binding a variable

Also, it supports
- code completion for named terms
- infos about named term (name, arity, definition) as tooltip
Demo

DEMO
Supported Platforms

The tool is implemented in C++ in a platform independent way. Current support:

- Linux binary and AppImage (distribution-independent)
- first successful tests on Windows

   No installation - just download\(^1\) and execute

\(^1\) [http://www.cs.uni-potsdam.de/~mafrank/](http://www.cs.uni-potsdam.de/~mafrank/)
Future Work

1. Improvement of performance
2. Extension to typed lambda calculus
3. Stable Windows (and potentially Mac) version
4. Transformation of the source code to JavaScript via Emscripten → platform independent online version
5. Extension of the tool to further parts of the framework.
References


