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

Context

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 (1)

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

Motivation (2)

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 λ-evaluation in the class room.

Focus and Related Work (1)

We analysed some λ-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
- $s\,t$ is an application, i.e. s is applied to t

Also, we know

- α-conversion: $\lambda x.t \stackrel{\alpha}{\to} \lambda x'.t[x/x']$
- β-reduction: $(\lambda x.t)s \stackrel{\beta}{\to} t[x/s]$

Untyped lambda calculus

We introduce additionally

- The named term reference, i.e. abbreviations for term definitions
- $\equiv$-expansion: $t \stackrel{\equiv}{\rightarrow} \mathit{def}_t$

For the term *True* $=: \lambda x.\lambda y.x$, *True* $\stackrel{\equiv}{\rightarrow} \lambda x.\lambda y.x$ 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

- α-conversion via double-click on the bound variable
- β-reduction via drag and drop or shortcut keys
- $\equiv$-expansion via double-click on the named term reference

Features (2)

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/

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 $\rightarrow$ platform independent online version
5. Extension of the tool to further parts of the framework.

References

- H. P. Barendregt. The Lambda Calculus Its Syntax and Semantics, volume 103. North Holland, revised edition, 1984. http://www.cs.ru.nl/ henk/Personal Webpage.
- EDWIN BRADY. Idris, a general-purpose dependently typed programming language: Design and implementation. Journal of Functional Programming, 23:552–593, 9 2013.
- Robert L. Constable, Stuart F. Allen, H. M. Bromley, W. R. Cleaveland, J. F. Cremer, R. W. Harper, Douglas J. Howe, T. B. Knoblock, N. P. Mendler, P. Panangaden, James T.
- Sasaki, and Scott F. Smith. Implementing Mathematics with the Nuprl Proof Development System. Prentice-Hall, NJ, 1986.
- Lucas Champollion, Joshua Tauberer, and Maribel Romero. The Penn Lambda Calculator: Pedagogical Software for Natural Language Semantics. Universität Konstanz, 2007.
- The Coq development team. The Coq proof assistant reference manual. LogiCal Project, 2004. Version 8.0.
- Ulf Norell. Dependently typed programming in agda. In Proceedings of the 4th International Workshop on Types in Language Design and Implementation, TLDI '09, pages 1–2, New York, NY, USA, 2009. ACM.
- Tobias Nipkow, Markus Wenzel, and Lawrence C. Paulson. Isabelle/HOL: A Proof Assistant for Higher-order Logic. Springer-Verlag, Berlin, Heidelberg, 2002.
- David Ruiz and Mateu Villaret. Tilc: The interactive lambda-calculus tracer. Electron. Notes Theor. Comput. Sci., 248:173–183, August 2009.
- Peter Sestoft. Demonstrating lambda calculus reduction. Electr. Notes Theor. Comput. Sci., 45:424–432, 2001.