Formal proofs, variable binding, and program extraction from proofs

Colloquium Logicum 2016
(10 - 12 September 2016, Hamburg)

Gyesik Lee
Hankyong National University

Overview

1. Verification of proofs
2. Hales' proof of the Kepler conjecture
3. Computerization of mathematical proofs
4. Issues in computerizing proofs
5. Extraction of programs from proofs

Verification of proofs

How do we come to see that a mathematical argument is correct?

- Prove it, then
- check whether the proof provided uses only given assumptions, already known facts, admitted axioms and inference rules.

Verification of proofs

- However, many officially published work contains (*un*)detected errors.
- Still this process is considered generally reliable.

Verification of proofs

There are however cases where this seemingly obvious process has difficulties.

Hales' proof of the Kepler conjecture

- The Kepler conjecture
  - No arrangement of equally sized spheres filling space has a greater average density than that of the cubic close packing and hexagonal close packing arrangements.
  - The density of these arrangements is around $\pi/3\sqrt{2} \simeq 0.7404$.

Hales' proof of the Kepler conjecture

- Hales' proof in August 1998 consisted of
  - 300 pages of texts and
  - 3 Gigabytes of computer programs and data.
- Submitted to Ann. Math.
  - after 5 years of refereeing process
  - the panel of 12 referees was 99% certain of the correctness of the proof.
  - Ann. Math. published the text proofs (121 pages long) only.

Hales' proof of the Kepler conjecture

What does "99% certainty" mean in mathematics?

Hales' proof of the Kepler conjecture

What was the problem?

Geuvers' comments

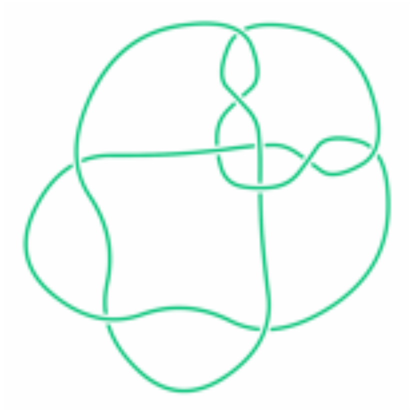
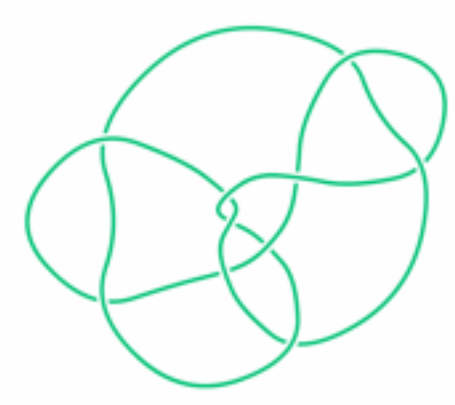
- Hales needed to prove that 1039 complicated inequalities hold.
- He used computer programs to verify the inequalities.
- The referees had problems with his approach:
  - verifying the inequalities themselves by hand would be impossible
  - one week per inequality is still 25 man years of work.
- They did not considered to verify the computer programs Hales used.
- To say the truth, *they could not*.

Worse cases

There are even cases in which some wrong statements were considered to be proved for a long period of time.

Worse case 1

- In the mathematical theory of knots, the *Perko pair*, named after Kenneth Perko, found in *1973*, is a pair of entries in classical knot tables that actually represent the same knot.
- The Perko pair gives a counterexample to a theorem claimed by Little in *1885* that they are separate knots.

Worse case 2

- Gödel claimed in the last sentence of the paper "*On the decision problem for the functional calculus of logic*" (1933):

  "In conclusion, I would still like to remark that Theorem I can also be proved, by the same method, for formulas that contain the identity sign."

- Theorem 1 concerns the decidability of the class called $[\exists^*\forall^2\exists^*, \mathit{all}, (0)]$.
- This claim was believed to be true for more than thirty years.
- But Stål Aanderaa showed in the mid-1960s that Gödel's proof would not actually work if the formulas contained equality.
- Finally, in 1983 Warren D. Goldfarb proved that the class mentioned by Gödel was not decidable.

Response

Mathematicians seem to have recognized the unreliability of checking process.

Response

- In 2000 the Clay Mathematics Institute (CMI) announced million dollar prizes for the solution of seven *Millennium Problems*.
- But there are conditions according to which the prize would be awarded:
  - two years after the appearance of the solution in a *refereed mathematics publication of worldwide repute;*
  - and after *general acceptance in the mathematics community*.
- But why wait two years?
- What does the ``*general acceptance in the mathematics community*" mean?
- Still these two conditions prove against the reliability of the traditional proof checking process.

Suggested solutions

- People like Doron Zeilberger suggest two ways to improve the process.
  - *In his blog post "If You Want Mathematical Truth, You Better Pay For It!"*
  - or *Computerization*

Computerization of mathematical proofs

- Back to Hales' proof of the Kepler conjecture
- In 2004, Hales himself announced his intention to have *formal* version of his original proof.
- His aim was to remove any remaining uncertainty about the validity of his proof by creating a *formal proof* that can be verified by some automated proof checking software, that is by some computer programs.
- His intention was then realized through a project called *Flyspeck* on 10th August 2014, 10 years after his announcement.
- *A formal proof of the Kepler conjecture* (Arxiv, 01.2015) with 22 authors.
- He used the two proof assistants, *HOL Light* and *Isabelle*.

Computerization of mathematical proofs

What does it mean to have a *formal version of proofs*?

Understanding proof assistants

- Geuvers' paper gives a detailed and kind explanation of the basic ideas of proof assistants, targeting mathematicians without any background in computer science:

  H. Geuvers, *Proof assistants: History, ideas and future, 2009*.

- With some interest, it would not be so difficult to read the paper.

Understanding proof assistants

- In order to understand how proof assistants like HOL Light and Isabelle work, it is necessary to understand
  - how mathematicians set up a theory and
  - how they define and prove mathematical properties.

Understanding proof assistants

- A proof assistant
  - is a computer software to assist with the development of proofs by human-machine interaction
  - and contains some sort of interactive proof editor with which a human can guide the search for proofs, *the details of which are stored in a computer.*

Foundation for proof assistants

- Mizar
  - Tarski–Grothendieck set theory with classical logic
- PVS
  - A classical, typed higher-order logic
- HOL family (HOL4, HOL Light, ProofPower)
  - A classical higher-order logic
- Isabelle
  - Zermelo-Fraenkel set theory (ZFC), higher-order logic
- Coq
  - Calculus of Inductive Constructions (CIC)
- Agda
  - Unified Theory of Dependent Types (UTT)
- Lean
  - Homotopy Type Theory (?)

Curry-Howard-de Bruijn correspondence

- A proof assistant provides a meta-theory where one can develop concrete mathematical theories using the idea of Curry-Howard-de Bruin correspondence:
  - Curry(1958): Hilber-style propositional logic corresponds to simply-typed combinatory logic.
  - Howard(1969): Gentzen's natural deduction corresponds to some simply-typed lambda-calculus.
  - de Bruijn's Automath(1967): the first practical system that exploited the Curry-Howard correspondence.
  - Martin-Löf's type theory with W-type(1980): corresponding to an intuitionistic logic with the strength of $\Pi^1_1\text{-CA}_0$.
  - Griffin(1990): The idea of Curry-Howard-de Bruin correspondence can be extended to classical logic.

Curry-Howard-de Bruijn correspondence

- The base idea of the *Curry-Howard-de Bruijn correspondence*:

$$\Gamma \vdash_{\text{logic}} \varphi \qquad \text{iff} \qquad \bar{\Gamma} \vdash_{\text{type theory}} M : \varphi$$

- The term M codes the proof of φ .
- Proving becomes constructing proof terms.
- Checking correctness of a proof corresponds to type checking.
- Type checking is decidable in many theories.

Curry-Howard-de Bruijn correspondence

- In case of the Coq proof assistant:

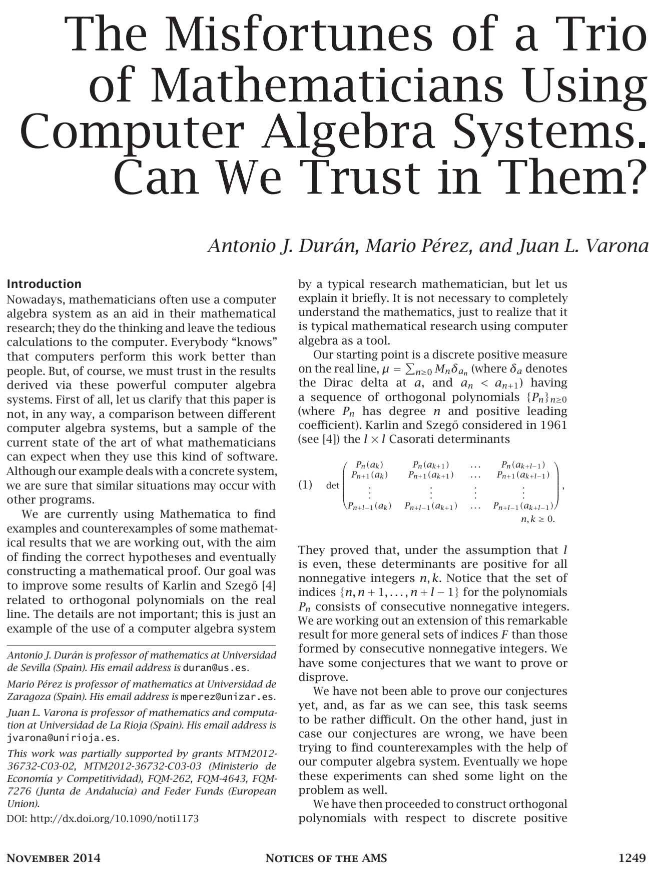
$$\begin{tikzcd}
\boxed{\text{Tactics / Syntax}} \arrow[r, Rightarrow] & \boxed{t : A\ ?} \arrow[r, Rightarrow] & \boxed{\text{Library}} \\
& \boxed{\text{Kernel}} \arrow[u, Rightarrow] &
\end{tikzcd}$$

State of affairs

- Proof assistants are already successfully adopted by programming language groups.
- On the other hand, many mathematicians use computer algebra systems and Latex, but not that much of proof assistants.

A side-trip

- But, don’t trust too much in computer algebra systems.
  - See Notices of AMS, Oct. 2014.



The Misfortunes of a Trio of Mathematicians Using Computer Algebra Systems. Can We Trust in Them?

Antonio J. Durán, Mario Pérez, and Juan L. Varona

Introduction

Nowadays, mathematicians often use a computer algebra system as an aid in their mathematical research; they do the thinking and leave the tedious calculations to the computer. Everybody “knows” that computers perform this work better than people. But, of course, we must trust in the results derived via these powerful computer algebra systems. First of all, let us clarify that this paper is not, in any way, a comparison between different computer algebra systems, but a sample of the current state of the art of what mathematicians can expect when they use this kind of software. Although our example deals with a concrete system, we are sure that similar situations may occur with other programs.

We are currently using Mathematica to find examples and counterexamples of some mathematical results that we are working out, with the aim of finding the correct hypotheses and eventually constructing a mathematical proof. Our goal was to improve some results of Karlin and Szegő [4] related to orthogonal polynomials on the real line. The details are not important; this is just an example of the use of a computer algebra system by a typical research mathematician, but let us explain it briefly. It is not necessary to completely understand the mathematics, just to realize that it is typical mathematical research using computer algebra as a tool.

Our starting point is a discrete positive measure on the real line, $\mu = \sum_{n\geq 0} M_n \delta_{a_n}$ (where δ_a denotes the Dirac delta at a, and $a_n < a_{n+1}$) having a sequence of orthogonal polynomials $\{P_n\}_{n\geq 0}$ (where P_n has degree n and positive leading coefficient). Karlin and Szegő considered in 1961 (see [4]) the $l \times l$ Casorati determinants

$$\det \begin{pmatrix} P_n(a_k) & P_n(a_{k+1}) & \dots & P_n(a_{k+l-1}) \\ P_{n+1}(a_k) & P_{n+1}(a_{k+1}) & \dots & P_{n+1}(a_{k+l-1}) \\ \vdots & \vdots & \vdots & \vdots \\ P_{n+l-1}(a_k) & P_{n+l-1}(a_{k+1}) & \dots & P_{n+l-1}(a_{k+l-1}) \end{pmatrix}, \quad n, k \geq 0. \tag{1}$$

They proved that, under the assumption that l is even, these determinants are positive for all nonnegative integers n, k. Notice that the set of indices $\{n, n+1, \dots, n+l-1\}$ for the polynomials P_n consists of consecutive nonnegative integers. We are working out an extension of this remarkable result for more general sets of indices F than those formed by consecutive nonnegative integers. We have some conjectures that we want to prove or disprove.

We have not been able to prove our conjectures yet, and, as far as we can see, this task seems to be rather difficult. On the other hand, just in case our conjectures are wrong, we have been trying to find counterexamples with the help of our computer algebra system. Eventually we hope these experiments can shed some light on the problem as well.

We have then proceeded to construct orthogonal polynomials with respect to discrete positive

Antonio J. Durán is professor of mathematics at Universidad de Sevilla (Spain). His email address is `duran@us.es`.

Mario Pérez is professor of mathematics at Universidad de Zaragoza (Spain). His email address is `mperez@unizar.es`.

Juan L. Varona is professor of mathematics and computation at Universidad de La Rioja (Spain). His email address is `jvarona@unirioja.es`.

This work was partially supported by grants MTM2012-36732-C03-02, MTM2012-36732-C03-03 (Ministerio de Economía y Competitividad), FQM-262, FQM-4643, FQM-7276 (Junta de Andalucía) and Feder Funds (European Union).

DOI: http://dx.doi.org/10.1090/noti1173

State of affairs in Math

- Proof of the 4 color theorem
- Proof of the Odd Order theorem
- Proof of the Kepler conjecture
- Some projects formalizing mathematics using a proof assistant
  - group theory
  - algebra
  - analysis
  - real number computation
  - algebraic topology
  - category theory
  - homotopy type theory

Issues in computerizing proofs

How close are we to a world where every paper on programming languages is accompanied by an electronic appendix with machine-checked proofs?

-POPLmark Challenge

Issues in computerizing proofs

To gauge progress in this area, we issue here a set of challenge problems, dubbed the POPLmark Challenge, chosen to exercise many aspects of programming languages that are known to be difficult to formalize.

- variable binding
- alpha-conversion
- substitution
- quantification

Issues about variable binding

A main issue in formal developments of meta-theory concerns the representation and manipulation of terms with variable binding.

Issues about variable binding

Close to pen-and-paper developments

and easy to handle?

Variable convention

- Barendregt's Variable Convention:

  If M1, M2, …, Mk occur in a certain mathematical context (e.g. definition, proof), then in these terms all bound variables are chosen to be different from the free variables.

- This convention implicitly requires alpha-conversion:

  Two expressions are identified when they differ only in the names of used bound variables.

- Alpha-equivalence:

$$\forall x\,(x < x + 1) \quad \equiv \quad \forall y\,(y < y + 1)$$

Variable convention

But dealing with alpha-conversion in a formal way is a nightmare, in particular when substitution and reduction are involved.

Representation of variables

- There have been many suggestions to overcome the problems related to variable binding.

- de Bruijn indices
  - $\lambda.(0\,1)$

- locally nameless
  - $\lambda.(0\,a)$ (well-formed)
  - $\lambda.(1\,a)$ (not well-formed)

- locally-named
  - $\lambda x.(x\,a)$ (well-formed)
  - $\lambda x.(y\,a)$ (not well-formed)

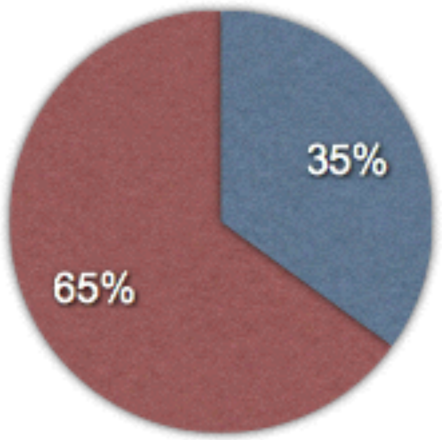
- traditional (nominal)
  - $\lambda x.(x\,y)$

Representation of variables

- de Bruijn style
  - no alpha-conversion necessary
  - less infrastructure and popular in implementation
  - but not human readable
- locally nameless style
  - compromise between de Bruijn style and traditional style
  - very popular in formalization work
- locally named style
  - corresponds to usual usage of variables in math and logic (Frege, Gentzen, ...)
  - human readable
  - avoids much of alpha-conversion, but not 100%
- traditional style
  - not practical in many proof assistants
  - but some like Isabelle/HOL provide nominal techniques.

Safety of System F< using locally names

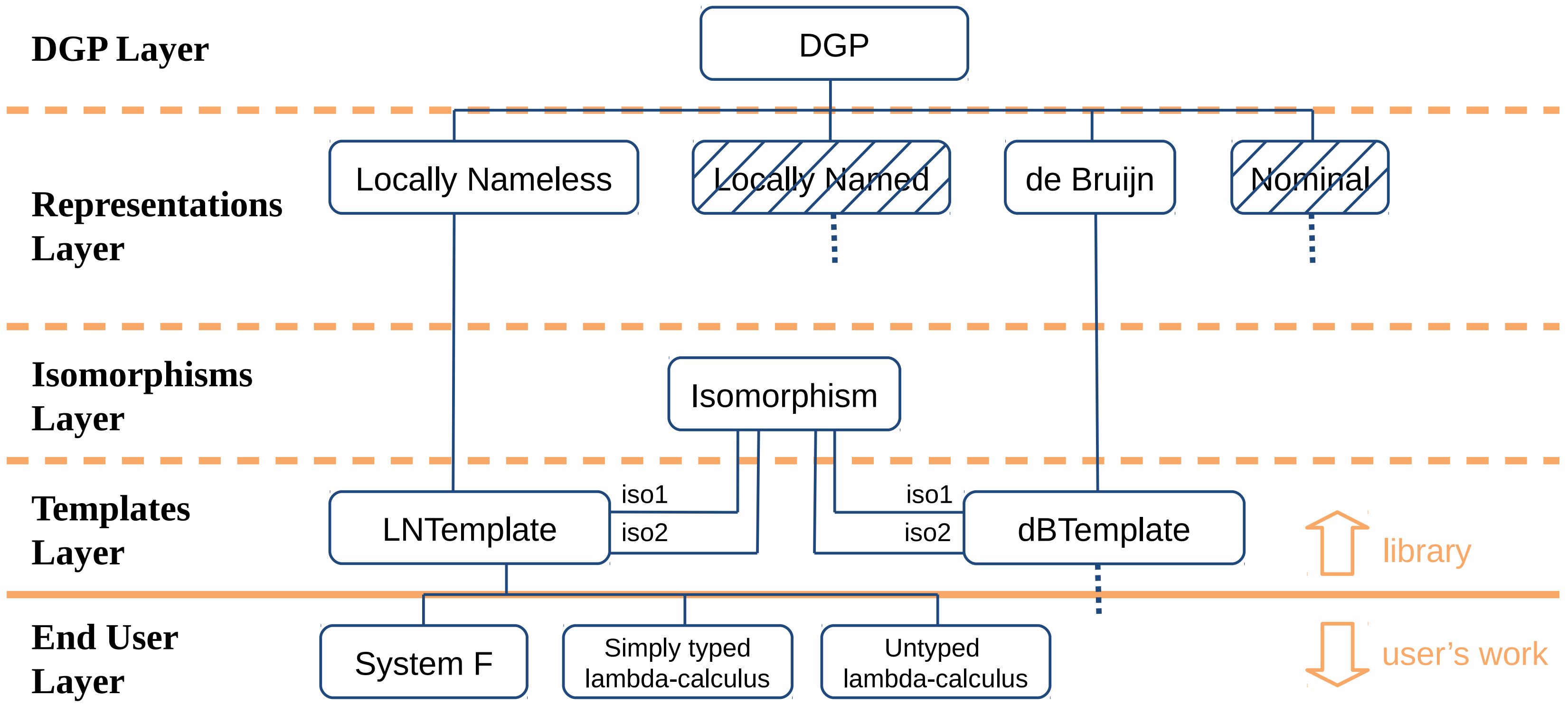
- 65% of Aydemir et al.(2008)’s work: infrastructure operations and lemmas



GMeta: Reducing infrastructure

- *A Generic Formal Metatheory Framework for First-order Representations* (G.L., Oliveira, Cho, and Yi, ESOP 2012)
- A DGP(data type generic programming) universe is used to represent a large family of object languages.
- The DGP universe is independent of the particular choice of first-order representations.
- GMeta library
  - Sound, generic, reusable and extensible infrastructure for first-order representations

GMeta: Reducing infrastructure



GMeta: Reducing infrastructure

		Infrastructure overhead	Total	Ratio
STLC (locally nameless)	Aydemir et al.[1]	17	31	55%
	GMETA	1	15	7%
System $F_{<:}$ (de Bruijn)	Vouillon[2]	41	101	41%
	GMETA	3	65	5%
System $F_{<:}$ (locally nameless)	Aydemir et al.[1]	60	93	65%
	GMETA	25	58	43%
	GMETA advanced	11	45	24%

(number of definitions and lemmas)

Extraction of programs from proofs

- Many proof assistants like Coq provide the facility to extract programs from proofs.
- Coq e.g. can be used to build certified and relatively efficient functional programs, extracting them from either Coq functions or Coq proofs of specifications.
- The functional languages available in Coq are OCaml, Haskell, and Scheme.

Extraction of programs from proofs

- Example: Cut elimination process
- A first-order intuitionistic logic with cut and Kripke style semantics
  - sound and complete w.r.t. a Kripke style semantics
  - in particular, the completeness part produces always cut-free proofs
  - then the combination of soundness and completeness becomes a process which produces a cut-free proof given a proof with cuts.

Thank you for your attention.