A typed language for tactics, what for?

Lourdes del Carmen González-Huesca

lgonzale@pps.univ-paris-diderot.fr

Yann Régis-Gianas

yrg@pps.univ-paris-diderot.fr

INRIA Paris-Rocquencourt

ANR Paral-ITP Project

September 2012

Project

Paral-ITP

Pervasive Parallelism in Highly-Trustable
Interactive Theorem Proving Systems
(Parallélisation des systèmes de preuve interactifs de haute fiabilité)

Document Model:

Formal Repository
Managing formal content
Formal Analysis of versions

} Get a fine grained repository of changes

Which language can represent changes, dependencies and resource reuse in a proof development?

Project
Paral-ITP

$\Longrightarrow$ A language for manage repository changes:

- for a structured depict of reusable terms
- based in logical frameworks to represent inference rules
- the types will give confidence in terms and may let to follow dependencies

Reuse proof terms, for that:

↳ study the changes on a syntax with binders.
↳ develop a safe logical framework for program transformations of proof derivations.
↳ **build typed tactics.**

What could be a "nice" typed tactic language for Coq?

Proof assistants and tactics

LCF style[1]

1. Proof development system:
   - interactive construction of goal directed proofs
   - the user takes control of proofs with *tactics*
   - proofs are represented as proof-terms or proof-objects
2. Proof-checker (kernel):
   - gives reliability to proof-terms

Tactic - partial sub-goaling method with a validation

$$\begin{aligned} \textit{type tactic} &= \textit{goal} \rightarrow (\textit{goal list} * \textit{proof}) \\ \textit{type proof} &= \textit{thm list} \rightarrow \textit{thm} \end{aligned}$$

it can be seen as a program which manipulates proofs

[1] Barras (2010) Proof assistants; Gordon et al. (1979) *Edinburgh LCF*

Coq[2]

- a formal proof management system
- kernel based on CIC (a small and safe checker)
- interactive development of machine-checked proofs
- the tactics are built from atomic and expressions of $\mathcal{L}_{tac}$
- a proof generates a term t which is verified by the kernel

The *proof term* t has some deficiencies:

- is a "big" term for manipulate, in contrast with the size of the proof
- when altering definitions happens some of the following:
  - no change in proof, but t changes
  - automation failure and update in proofs
- t is far from a source code

t is not reusable, not even modifiable

[2] Coq (2012) Manual; Bertot and Casteran (2004) *Coq'Art*.

A solution...

For the repository management we propose to store scripts in the kernel to reuse them.

The scripts have to maintain the safety of the kernel, therefore we also propose improve confidence of scripts.

- ↳ adding types in scripts
- ↳ work out scripts in a programming language

Develop a (programming) language for build typed tactics

- ↳ change the architecture of the kernel to empower the representation of typed proof scripts

State of the art

What has been done for tactic improvement and for reasoning about properties of programming languages?

- analysis the origins of tactics: LCF
- design frameworks for both: representation of languages and express-study their properties
  - ↳ a usual way is to use logical frameworks for metatheory[3]:

object language		meta language
logic	+	computation
representation language		computational language

[3] Pfenning (1997) Computation and deduction.

LF[4]

A language for defining logical systems

(a calculus with dependent types)

- a signature containing the constants assigned to types and kinds and contexts for free variables of types:

$$\Gamma \vdash_\Sigma K \qquad \Gamma \vdash_\Sigma A : K \qquad \Gamma \vdash_\Sigma M : A$$

- definitional equality between objects on the same level (β conversion)

object logic	LF
syntactic category	type
proof checking	type checking

[4] Harper et al. (1993) A framework for defining logics; Pfenning (2001) Logical frameworks.

Contextual Type Theory

Contextual LF

... the truth of a proposition in logic is not absolute,
but depends on the context we consider it in.[5]

- extends LF with contexts and context objects
- uses higher order abstract syntax (HOAS) for represent object expressions
- $\varphi.M : A[\varphi]$ is a *closed* term with a contextual type
- normal objects: λ-abstraction and neutral terms
- bidirectional typing in object level: check normal terms and synthesize neutral terms
- contextual objects and contexts are included in both levels

[5] Nanevski et al. (2007) Contextual Modal Type Theory.

Beluga$^{\mu 6}$

logical framework (contextual LF) + computational language (Mini ML)

- a *pure* functional language, for dependently typed programming
- designed to investigate programming and reasoning with structures that provide support for binders
  - uses HOAS for mechanizing meta-theory
  - has *contextual objects and contexts* since implements LF
- is also a framework for proof development

[6] http://www.cs.mcgill.ca/~complogic/beluga/

VeriML[7]

$$\underset{(\lambda \mathrm{HOL}^{\mathsf{ind}})}{\text{logical framework}} \quad + \quad \underset{\text{(ML fragment)}}{\text{computational language}}$$

A language design which integrates a logical framework inside a computational language:

- a type-safe language, generates valid proof-terms
- support effects and first class objects
- uses contextual type theory for variable binding in the computational language

[7] Stampoulis and Shao (2010) VeriML.

Example

Encoding a tactic

Tactic for proving if a proposition is a tautology [8]

$$tauto : \prod P : Prop.\ option\ LT\ (P)$$

$$tauto : \prod \phi : ctx.\ \prod P : [\phi]Prop.\ option\ LT([\phi]P)$$

$$tauto : \prod \phi : ctx.\ \prod P : [\phi]Prop.\ hyplist\phi \rightarrow option\ LT([\phi]P)$$

free vars. packed ctx hyp. packages lift

$[\varphi]t$ describes the terms of type T living in context φ
Lifting: $LT(\cdot)$ for types $\langle\cdot\rangle$ for terms

[8] Not coded in Coq.

Example

continued

$$tauto : \prod \phi : ctx. \prod P : [\phi]Prop.\ hyplist\phi \rightarrow\ option\ LT([\phi]P)$$

$$\begin{aligned} tauto\ \phi\ P\ \ell = holcase\ P\ of \\ \quad |P_1 \rightarrow P_2 \ \longmapsto\ let\ \ell_1 = hyplistWeaken\ \phi\ P_1\ \ell\ \ in \\ \quad\quad let\ \ell_2 = cons\ \langle P_1, \langle [\phi, pf_1 : P_1]\ pf_1 \rangle\rangle\ \ell_1\ in \\ \quad\quad\quad do\ \ x \leftarrow tauto(\phi, pf_1 : P_1) P_2 \ell_2; \\ \quad\quad\quad [\phi]\ \lambda y : P_1.(X/id_\phi, y) \\ \quad |\_ \ \longmapsto hyplistFind\ \phi\ P\ \ell \end{aligned}$$

$$hyplist = \lambda\phi : ctx.\ list\ (\Sigma H : [\phi]Prop.LT([\phi]H))$$

`it carries the proof object of each hypothesis`

$$hyplistWeaken\ \phi\ P_1\ \ell\ :\ hyplist(\phi, x : P_1)$$ `extend the context`

$$hyplistFind\ \phi\ P\ \ell\ :\ option\ LT([\phi]P)$$ `find a variable in the context`

Languages of VeriML

λHOL$^{\text{ind}}$ and ML fragment

- A logical framework with:
  - higher order logic with support of inductive definitions
  - explicit proof objects (witnesses of derivations)
  - use dependent types for logical terms and integrate them in the computational language
- The computational language which:
  - is an extension of a ML fragment (*nat*, *bool*, higher order function types, mutable arrays)
  - is (clearly) separated from the λHOL$^{\text{ind}}$ and are associated by the `lift` function

Focus on writing tactics for improve scalability and modularity

Summary

- HOAS add an extra complexity when encoding, it does not allow to scale proofs
- VeriML seems to be a well approach but not at all for the final user:

  ↳ encoding is observably complicated since there are interactions between logical and computational languages
  ↳ is a heavy and powerful new language to learn
  ↳ there is more work on an extension[9]
    - extensible conversion rule for proof checking
    - static proof scripts for static checking of tactics
    - reuse of proof scripts and enables modular proof checking
    - logical environments as current proof contexts or current proof states

[9] Stampoulis and Shao (2012) Static and user-extensible proof checking.

Summary

continued

	Beluga$^{\mu}$	VeriML
mutable references	✗	✓
effects	✗	✓
dependent types	✓	✓
logic & comp. languages	no separated	separated
manage (object) contexts	✓	✓
designed for tactics	✗	✓

they construct valid terms

Both are logical frameworks

but Coq can be used as a logical framework too

Conclusions

- Typed tactics inside a logical framework for:
  - have safe proof terms
  - keep trace of changes in proofs
- Incorporate information (context) to tactics for:
  - ↳ modular and reusable tactics
  - ↳ small safe proof objects that can be added to the kernel
- Consider Coq as a logical framework for specify and verify tactics with contexts, that is:

Coq as a language for tactics

References I

A. Asperti, W. Ricciotti, C. Coen, and E. Tassi. A new type for tactics. *Proceedings of ACM SIGSAM PLMMS*, 2009.

B. Aydemir, A. Charguéraud, B. C. Pierce, R. Pollack, and S. Weirich. Engineering formal metatheory. In *ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), San Francisco, California*, pages 3–15. ACM, 2008.

H. Barendregt and H. Geuvers. Proof-assistants using dependent type systems. In A. Robinson and A. Voronkov, editors, *Handbook of automated reasoning*, pages 1149–1238. Elsevier Science Publishers B. V., Amsterdam, The Netherlands, 2001.

B. Barras. Proof assistants. http://www.lix.polytechnique.fr/~barras/mpri/notes/index-2-7-2.html, 2010. Master Parisien de Recherche en Informatique, notes of the course.

Y. Bertot and P. Casteran. *Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions*. Springer-Verlag, Berlin, Germany, 2004.

A. Cave and B. Pientka. Programming with binders and indexed data-types. In *Proceedings of the 39th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL'12, pages 413–424, New York, USA, 2012. ACM.

The Coq Proof Assistant Reference Manual Version 8.4. The Coq Development Team, 2012. August 12, 2012.

References II

D. Delahaye. A tactic language for the system coq. In *Proceedings of Logic for Programming and Automated Reasoning (LPAR), Reunion Island*, volume 1955 of *Lecture Notes in Computer Science*, pages 85–95. Springer, 2000.

M. J. C. Gordon, R. Milner, and C. P. Wadsworth. *Edinburgh LCF*, volume 78 of *Lecture Notes in Computer Science*. Springer, 1979.

R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *J. ACM*, 40(1): 143–184, 1993.

A. Nanevski, F. Pfenning, and B. Pientka. Contextual modal type theory. *ACM Trans. Comput. Logic*, 9(3):23:1–23:49, 2008.

F. Pfenning. Computation and deduction, 1997. Draft (April 2, 1997).

F. Pfenning. Logical frameworks. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, pages 1063–1147. Elsevier Science Publishers B. V., Amsterdam, The Netherlands, 2001.

B. C. Pierce. *Types and programming languages*. MIT Press, Cambridge, MA, USA, 2002.

A. Stampoulis and Z. Shao. Veriml: typed computation of logical terms inside a language with effects. In *Proceedings of the 15th ACM SIGPLAN international conference on Functional programming*, ICFP'10, pages 333–344, New York, NY, USA, 2010. ACM. Extended version: http://flint.cs.yale.edu/flint/publications/verimltr.pdf.

References III

A. Stampoulis and Z. Shao. Static and user-extensible proof checking. In *Proceedings of the 39th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '12, pages 273–284, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1083-3. Extended version: http://flint.cs.yale.edu/flint/publications/sv11exprf.pdf.