Towards a Logic-Independent Proof Language

Florian Rabe

Computer Science, University Erlangen-Nürnberg, Germany
LRI, University Paris-Sud, France

Upscale Meeting, Paris, March 19 2018

Goals

- Library integration
  - merge libraries
  - move libraries to other systems
  - dynamically use other systems in a proof
- Heterogeneous development
  - global library independent of tools — like specification
  - move to concrete tools for proving — like implementation
  - different tools for different parts
- Verification
  - independently check proofs
  - redundancy by using multiple tools
  - guard against mis-formalization

Challenges

- Incompatible foundations
  - type system
  - logic
  - proof system
- Incompatible tools
  - extra-logical features
    e.g., module system, unification hints, reflection
  - tactic system
  - auxiliary systems e.g., decision procedures, automated provers
- Incompatible libraries
  - choice of definitions
  - module structure e.g., flat vs. parametric vs. packaged
  - coding of inexpressible features
    e.g., partial functions, subtyping

Modular Foundation

- Define foundational features in logical framework
  e.g., dependent function types, type-indexed universal quantifier
- Aim for coherence
  - features should be combinable
    e.g., any type system with any logic
  - features should be translatable
    e.g., simply-typed into dependently-typed functions
  - compatible notations across all features
- Aim for tool-independence
  - features should be naturally embeddable into tool foundations
  - avoid features that are biased towards one tool
- Focus on specification
  - domain objects, propositions, truth
  - type- and proof-checking
  - no support for finding proofs

Modular Foundation: Usage

- Specify a problem
  - import minimal set of needed foundational features
  - axiomatize assumptions
  - state conjecture
- Generate stubs for different proof tools
- Proof tools find, export proofs
- Independent proof checker(s) for reference foundation

Framework: MMT

Design principle

- few orthogonal concepts
- uniform representations of diverse languages

sweet spot in the expressivity-simplicity trade off

MMT Concepts

- theory = named set of declarations
  - foundations, logics, type theories, classes, specifications, ...
- theory morphism = compositional translation
  - inclusions, translations, models, katamorphisms, ...
- constant = named atomic declaration
  - function symbols, theorems, rules, ...
  - may have type, definition, notation
- term = unnamed complex entity, formed from constants
  - expressions, types, formulas, proofs, ...
- typing $\vdash_T s : t$ between terms relative to a theory
  - well-formedness, truth, consequence ...

Small Scale Example (1)

Logical frameworks in MMT

```
theory LF {
   type
   Pi        # Π V1 . 2                    name[ : type][#notation]
   arrow     # 1 → 2
   lambda    # λ V1 . 2
   apply     # 1 2
}
```

Logics in MMT/LF

```
theory Logic: LF {
   prop : type
   ded  : prop → type # ⊢ 1                judgments-as-types
}
theory FOL: LF {
  include Logic
  term        : type                       higher-order abstract syntax
  forall      : (term → prop) → prop #  ∀ V1 . 2
}
```

Small Scale Example (2)

FOL from previous slide:

```
theory FOL: LF {
  include Logic
  term        : type
  forall      : (term → prop) → prop  #  ∀ V1 . 2
}
```

Proof-theoretical semantics of FOL

```
theory FOLPF: LF {
  include FOL
                                                    rules are constants
  forallIntro  : ΠF:term→prop.
                    (Πx:term.⊢ (F x)) → ⊢ ∀(λx:term.F x)
  forallElim   : ΠF:term→prop.
                    ⊢ ∀(λx:term.F x) → Πx:term.⊢ (F x)
}
```

Small Scale Example (3)

FOL from previous slide:

```
theory FOL: LF {
  include Logic
  term        : type
  forall      : (term → prop) → prop #  ∀ V1 . 2
}
```

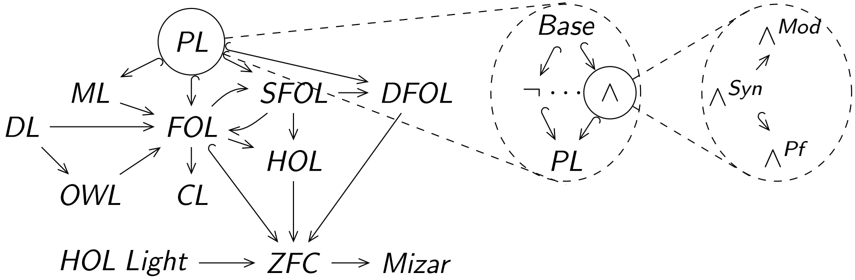
Algebraic theories in MMT/LF/FOL:

```
theory Magma : FOL {
  comp : term → term → term # 1 ∘ 2
}
theory SemiGroup          : FOL {include Magma, ...}
theory CommutativeGroup : FOL {include SemiGroup, ...}
theory Ring : FOL {
  additive: CommutativeGroup
  multiplicative: Semigroup
  ...
}
```

Logic Diagrams in LATIN

An example fragment of the LATIN logic diagram

- nodes: MMT/LF theories
- edges: MMT/LF theory morphisms



- each node is root for library of that logic
- each edge yields library translation functor

library *integration* very difficult though

Domain Formalization

- Foundational features must be coupled with reference formalizations of domain knowledge
  - base values — various number types, strings, etc.
  - standard datatypes — lists, sets, finite maps, etc.
  - mathematical structures — algebraic hierarchy, polynomials, etc.
- Relatively easy because only axiomatizations needed
- Must be aligned with implementations in tool libraries
  - initially collect from tool libraries
  - gradually drive tool communities to add compatibility layer

Proof Levels

- low level: proof terms
  - foundation-specific but not tool-specific
  - independently checkable
  - possibly large, typically unstructured
- mid level: sequence of steps to recreate proof

  e.g., LCF inference rules, imperative tactic invocations
  - foundation-specific, often tool-specific
  - can be a good trade-off between size and reproducibility
  - but recreation may fail even if same tool is used

    e.g., different versions, timeouts, etc.
- high level: focus on insight-relevant structure

  e.g., intermediate assertions, induction hypothesis, case splits
  - very robust in the long run
  - may be irreproducible in the short run
  - commonly foundation- and tool-specific
  - foundation-independent solution possible

Theorem Statements

Based on reference library:

- theory identifiers T — foundational or domain feature
- contexts C
- expressions E — terms, types, formulas, proof terms, etc.

$$Theorem \quad ::= \quad x \ \textbf{assert} \ C \vdash_{T^*} E \ \textbf{proof} \ P$$

where

- a: identifier
- T^*: list of needed foundation/domain features
- C: assumptions — e.g., type variables
- E: theorem statement
- P: high-level proof (see sequel)

High-Level Proofs

The non-controversial cases

P	$::=$	E	low-level proof term
	$\mid$	**use** a^*	partial proof using theorems/tactics a_i
	$\mid$	**let** $x : E = E\ ;\ P$	local definition
	$\mid$	**hence** $x : E$ **by** $P;P$	forward step

More difficult:

- tactic invocation (including most primitive rules)
- local assumptions (implies/forall introduction)
- case distinction (or elimination, induction)
- backward steps (change the current goal(s))

Tactic Invocation

P	$::=$	E	low-level proof term
	$\mid$	**use** a^*	partial proof using theorems/tactics a_i
	$\mid$	**let** $x : E = E$; P	local definition
	$\mid$	**hence** $x : E$ **by** $P;P$	forward step
	$\mid$	$a(P^*)$	proof rule/tactic a applied to arguments

Indispensable: needed for foundation/tool-specific extensibility
But:

- also need reference library for tactics
  - declare names a with type/notation information
  - no definition/implementation needed
- $a(P^*)$ expressive enough for some high-level structure, e.g.,
  - $a =$ **use**
  - $a =$ **cases**

  Which of these should be distinguished production?

Local Assumptions

P	$::=$	E	low-level proof term
	$\mid$	**use** a^*	partial proof using theorems/tactics a_i
	$\mid$	**let** $x : E = E\ ;\ P$	local definition
	$\mid$	**hence** $x : E$ **by** $P;P$	forward step
	$\mid$	$a(P^*)$	proof rule/tactic a applied to arguments
	$\mid$	**assume** $x : E;P$	local parameter/assumption

Meaning:

- **assume** $x : E;P$ corresponds to $a(\lambda x : E.P)$
- implicit application of a based on type of P

Questions:

- Redundant?
- How to choose a?

Case Distinction

P	$::=$	E	low-level proof term
	$\mid$	**use** a^*	partial proof
	$\mid$	**let** $x : E = E\ ;\ P$	local definition
	$\mid$	**hence** $x : E$ **by** $P;P$	forward step
	$\mid$	$a(P^*)$	proof rule/tactic application
	$\mid$	**assume** $x : E;P$	local parameter/assumption
	$\mid$	**case** $P\,\{(x : E \to P)^*\}$	case distinction

Meaning:

- **case** $P\,\{x : E_1 \to P_1, \ldots, x : E_n \to P_n\}$ corresponds to $a(P, \lambda x : E_1.P_1, \ldots, \lambda x : E_n.P_n)$
- implicit application of a based on type of P

Questions:

- Redundant?
- How to choose a?

Backward Steps

P	$::=$	E	low-level proof term
	$\mid$	**use** a^*	partial proof using theorems/tactics a_i
	$\mid$	**let** $x : E = E \; ; P$	local definition
	$\mid$	**hence** $x : E$ **by** $P;P$	forward step
	$\mid$	**goals** $(x : E)^*$ **by** $P;P$	backward step
	$\mid$	**close** x **by** $P;P$	close specific subgoal

Purpose:

- Not needed for verification
- Needed to capture high-level proof structure
  - human readability
  - reproduce original proof script

Questions:

- Special case for reduction to a single goal?

  no goal name needed
- Should original goal have a name?
- Allow operations on the entire set of open goals?