

Towards an implementation in LambdaProlog of the two level Minimalist Foundation

A. Fiori

C. Sacerdoti Coen

University of Padova

University of Bologna

Padova, 27/09/2017

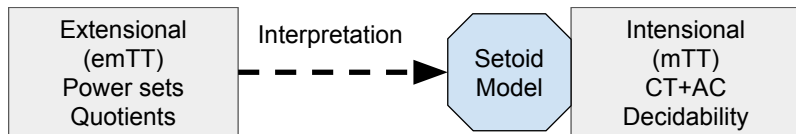
- 1 Introduction
- 2 Type checkers for the two levels of the Minimalist Foundation
- 3 Interpreting the extensional level in the intensional level
- 4 Conclusions and Future Works

Outline

- 1 Introduction
- 2 Type checkers for the two levels of the Minimalist Foundation
- 3 Interpreting the extensional level in the intensional level
- 4 Conclusions and Future Works

The Minimalist Foundation

- The Minimalist Foundation [MaiettiSambin 2005, Maietti 2009] is a two-level formal system.
- The extensional level allows for quotients.
- The intensional satisfies the proof-as-programs paradigm.
- The Minimalist Foundation is compatible with all major constructive foundations for mathematics



Outline of Our Work

Work in Progress

- Type checkers for the two levels of the Minimalist Foundation (implemented in λ Prolog).
- Implementation (in λ Prolog) of the interpretation from the extensional level to the intensional level.

Future Works

- Formal validation of the interpretation (in Abella).
- Proof assistant over the extensional level (in $ELPI = \lambda$ Prolog + Constraint Programming)
- Code extraction at the intensional level.

We implemented the two level system completed in [Maietti 2009]

What Programming Language to Formalize a Theory?

Characteristics of λ -Prolog

- 1 very high level language, usable by a logician/mathematician
- 2 easy definition of structures with binders
- 3 α -equality and capture-avoiding substitution for free
- 4 simple encoding of inference rules
- 5 automatic management of non-determinism/backtracking
- 6 simple reasoning on the programs (simple semantics)

λ Prolog is the smallest extension to Prolog able to treat syntaxes with binders

Higher Order Logic Programming (HOLP)

λ Prolog = Prolog $\cup \{\Rightarrow, \forall\}$ in queries

$$\frac{\begin{array}{c} [c] \\ \vdots \\ q \end{array}}{c \Rightarrow q}$$

Locally scoped,
hypothetical reasoning

$$\frac{c\{y/x\} \quad y \text{ fresh}}{\pi x \setminus c}$$

Generation of
fresh names

HOAS + $\{\Rightarrow, \forall\}$ for entering binders in recursive definition

The Hello-World of λ Prolog

Type-Checking for Simply Typed λ -calculus

$$\frac{\Gamma \vdash M : A \rightarrow B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B} \quad \frac{\Gamma, x : A \vdash F x : B}{\Gamma \vdash \lambda x. F x : A \rightarrow B} \quad \frac{(x : A) \in \Gamma}{\Gamma \vdash x : A}$$

Representation of Simply Typed λ -calculus

```
type app      term -> term -> term.
type lam      (term -> term) -> term.
```

Type-Checking/Inference in λ Prolog

```
of (app M N) B :- of M (arr A B), of N A.
of (lam F) (arr A B) :- pi x\ of x A => of (F x) B.
```


Outline

- 1 Introduction
- 2 Type checkers for the two levels of the Minimalist Foundation**
- 3 Interpreting the extensional level in the intensional level
- 4 Conclusions and Future Works

Preliminary Work: Minor Changes to the Calculus

Syntax directed version of the rules

From

$$\frac{x \in A \quad A = B}{x \in B} \quad \frac{f \in \prod_{x \in B} C(x) \quad t \in B}{f t \in C(t)}$$

to

$$\frac{f \in \prod_{x \in B} C(x) \quad t \in B}{f t \in C(t)}$$

Deterministic equality check

From $(\lambda_{x \in B} C(x)) t = C(t)$

to $(\lambda_{x \in B} C(x)) t \triangleright C(t)$ and $(s = t) := s \triangleright^{**} \triangleleft t$

Preliminary Work: Major Changes to the Calculus

Problem: proofs are not recorded at the extensional level

$$\frac{\text{true} \in \text{Eq}(C, c, d)}{c = d \in C} \quad \frac{\text{true} \in B \quad \text{true} \in C \quad B \text{ props} \quad C \text{ props}}{\text{true} \in B \wedge C}$$

Discarded solution

The typechecker takes the whole **derivation** in input.
The datatype for the derivation is yet another typed λ -calculus.

Partial solution

Keep proof terms as in the intensional level.
To a user we can still show *true* because of proof irrelevance.
It does not solve the problem of the *conv* rule.

Preliminary Work: Major Changes to the Calculus

Full solution: deterministic equality check in the ext. level

From arbitrary conversion proofs

$$\frac{true \in Eq(C, c, d)}{c = d \in C}$$

to contextual closure + context lookup rule

$$\frac{(x \in Eq(C, c, d)) \in \Gamma}{c = d \in C}$$

and new LetIn term constructor

$$\frac{p \in P \quad t \in T[x \in P]}{let\ x := p \in P\ in\ t \in T}$$

Preliminary Work: Changes for Code Reuse

Π Introduction rule

$$\frac{B \text{ set} \quad c(x) \in C(x) [x \in B] \quad C(x) \text{ set} [x \in B]}{\lambda x^B. c(x) \in \Pi_{x \in B} C(x)}$$

```
of (lambda B F) (setPi B C) IE :-
  isType B _ IE,
  (pi x\ locDecl x B => isType (C x) _ IE)
  pi x\ locDecl x B => of (F x) (C x) IE.
```

Π Formation rule

$$\frac{B \text{ set} \quad C(x) \text{ set} [x \in B]}{\Pi_{x \in B} C(x) \text{ set}} \qquad \frac{B \text{ col} \quad C(x) \text{ col} [x \in B]}{\Pi_{x \in B} C(x) \text{ col}}$$

```
isType (setPi B C) KIND3 IE :-
  isType B KIND1 IE,
  pi x locDecl x B => isType (C x) KIND2 IE,
  pts_pi KIND1 KIND2 KIND3.
```

Typechecking and future works

Typechecking

- Code reuse between levels.
- Code reduction via PTS-style.
- Extremely modular code.

Future works

- Complete and debug all the rules.
- The changes to the calculi have to be validated
- The ξ -rule at the intentional level must be removed.
Requires a syntax directed version of explicit substitutions.

Outline

- 1 Introduction
- 2 Type checkers for the two levels of the Minimalist Foundation
- 3 Interpreting the extensional level in the intensional level**
- 4 Conclusions and Future Works

Design of the Interpretation

The Interpretation in a Nutshell

- In the Minimalist Foundation types are interpreted in dependent setoids.
- The interpretation on types is defined by structural recursion.
- For simple types (the singleton, the empty set, naturals) the setoid equality is the intensional propositional equality
- The equality of functions imposes the ξ rule
- Proof irrelevance is imposed by the interpretation
- Lack of impredicative quantifications avoids user-defined type equalities: this is NOT homotopy type theory

Design of the Interpretation

The Interpretation is Rich and Complex

- Requires lots of (proof) terms to be defined by meta-level recursion on types, terms and derivations of equalities
 - proofs of reflexivity, symmetry, transitivity
 - proofs that equivalences behave as congruences for every user defined function
 - canonical isomorphisms between interpretation of extensionally equal types
 - proofs that they are indeed isomorphisms
 - ...
- We are unable to directly use the proof of the paper as they are often given in categorical terms.

The Main Issue

Subsumption becomes coercions

- Equality used to fix mismatching (extensionally convertible) types must become term translation.

$$\frac{x \in A \quad A = B}{x \in B} \text{ becomes } \frac{x \in A \quad ARB}{\sigma x \in B}$$

- σ is defined by recursion also over the proof of $A = B$ (comprising the missing derivations of $Eq(T, c, d)$)
Luckily we made proof search deterministic via let-ins and restricting to congruence rules and context lookup
- An example of an extensionally well typed term with mismatching types

$$\forall_{x \in \mathbb{1}} \forall_{f \in (x =_{\mathbb{1}} \star) \rightarrow \mathbb{1}} (\star =_{\mathbb{1}} x) \Rightarrow f(\text{rfl}(\star)) =_{\mathbb{1}} f(\text{rfl}(\star))$$

Interpretation of Types

```
forall singleton x0 \
  forall (colSigma (fun (propId singleton x0 star) singl
    forall (propId singleton x0 star) x2 \
      forall (propId singleton x0 star) x3 \
        forall (propId singleton star star) x4 \
          propId singleton (fun_app x1 x2) (fun_app x1 x3)) x1 \
forall (propId singleton star x0) x2 \ propId singleton
(fun_app (elim_colSigma x1 (x3 \
  fun (propId singleton x0 star) singleton) x3 \ x4 \
  (impl_app (impl_app (forall_app (forall_app (impl_ap
  (forall_app (forall_app (k_propId singleton) star) x
  x2) star) star) (id singleton star)) (id singleton s
(fun_app (elim_colSigma x1 (x3 \
  fun (propId singleton x0 star) singleton) x3 \ x4 \
  (impl_app (impl_app (forall_app (forall_app (impl_ap
  (forall_app (forall_app (k_propId singleton) star) x
  x2) star) star) (id singleton star)) (id singleton s
```

Auxiliary Predicates for the Interpretation

```

tau (propEq T T1 T2) (propEq T T1' T2') (SIGMA) :-
  (proof_eq T1 T1' F1),
  (proof_eq T2 T2' F2),
  (trad T1 T1i),
  (trad T2 T2i),
  (trad T1' T1i'),
  (trad T2' T2i'),
  (trad T Ti),
  SIGMA = x\ impl_app (
    impl_app ( forall_app ( forall_app ( impl_app ( forall_app
      forall_app (k_propId Ti) T1i) T1i') F1) T2i) T2i')
proof_eq (fun_app F X1) (fun_app F X2) H :-
  proof_eq X1 X2 G,
  trad F F',
  P2F' = elim_colSigma F' _ (x\ y\ y),
  trad X1 X1',
  trad X2 X2',
  H = forall_app (forall_app (forall_app P2F' X1') X2')

```

Outline

- 1 Introduction
- 2 Type checkers for the two levels of the Minimalist Foundation
- 3 Interpreting the extensional level in the intensional level
- 4 Conclusions and Future Works**

Conclusions

Implementing the Minimalist Foundation [Maietti 2009] is non-trivial

- Many different type constructors and rules.
- Many terms need to be provided during the interpretation.
- Extensional type theories pose issues to the implementors.
- Implementation choices impact the calculus.
- The good properties must be preserved.

But the constrained nature of the theory helps

- Structural recursion on types is facilitated by their very rigid structure.
- The propositional equality (int./ext.) is the only type constructor that directly takes terms as arguments.

Conclusions and Future Works

λ Prolog was a great choice

- Takes away the pain due to binders, α -conversion, capture avoiding substitution, etc.
- The code is in 1-1 relation with the new syntax oriented version of the formal inference rules.
- Joint Bologna/INRIA effort to combine λ Prolog with Constraint Programming to smoothly transition to proof assistant implementation.

In the future we wish to extend our work

- Complete and validate (in Abella) the type checkers and interpretation.
- Implement code extraction for the intensional level.
- Implement a proof assistant for the extensional level.
- Validate the proof assistant formalizing Sambin's Basic Picture book (porting proofs from Matita).