Global, Regional, and Local Contexts

Florian Rabe

University Erlangen-Nuremberg

October 2025

Motivation

Representing Polynomials

How would you represent this formula in your favorite system?

$$X^2 + 1 \in \mathbb{Q}[X]$$

Representing Polynomials

How would you represent this formula in your favorite system?

$$X^2 + 1 \in \mathbb{Q}[X]$$

How about these?

$$X^2 + 1 \in \mathbb{Q}[Y]$$

$$Y^2 + 1 \in \mathbb{Q}[X]$$

Representing Polynomials

How would you represent this formula in your favorite system?

$$X^2 + 1 \in \mathbb{Q}[X]$$

How about these?

$$X^2 + 1 \in \mathbb{Q}[Y]$$

$$Y^2 + 1 \in \mathbb{Q}[X]$$

And are these true?

$$\mathbb{Q}[X, Y] = \mathbb{Q}[Y, X]$$

$$\mathbb{Q}[X] \cap \mathbb{Q}[Y] = \mathbb{Q}$$

Are Polynomial Variables Local or Global?

Option 1a: Bound local variables

like OpenMath OMV

- natural first choice
- α-equality has unexpected consequences
- requires implicit binders to interpret standard math

Option 1b: Nameless local binding

- 1a but with de-Bruijn indices
- $\mathbb{Q}[-]$ takes number of variables, not their names

Option 2: Global names

like OpenMath OMS

- avoids renaming/scoping issues
- no insertion of implicit binders everywhere
- but what would the content dictionary be?

local vs. global dichotomy ubiquitous in formal systems

Many Names are Neither Local nor Global

Any sets built inductively, e.g.,

- generated groups as in

  Let $G = \langle x, y | x^2 = e, y^5 = e \rangle$ and consider $xy^2 \in G$ ...

- μ-types with labeled sums as in

  The natural numbers are given by $\mu X.\, \mathit{zero}() \,|\, \mathit{succ}(X)$.
  We write 1 for $\mathit{succ}(\mathit{zero})$.

Dual case: records

- field names are α-renamable inside the record type

  $$\textbf{theory}\,\texttt{Monoid} = \{\textbf{type}\,u,\ \textbf{term}\,\texttt{oper} : (u, u) \to u,\ \ldots\}$$

- but have global scope as selector functions

  $$m.\texttt{oper} : (m.u, m.u) \to m.u \quad \text{given} \quad m : \texttt{Monoid}$$

α-equality not feasible — but semantics depends on context

Local vs. Global Names in MMT

Long-standing issue in MMT

- binary OMS vs. OMV system inherited from OpenMath/MathML and OMDoc
- problematic when representing certain language features
  record types, theory/diagram expressions, ...
- multiple failed attempts to evolve MMT's binary identifier system
  increasingly blocking development in recent years

Since 2024: new language UniFormal

- originally a programming language experiment
- now evolving into MMT2-like MKM kernel language
- includes from scratch redesign of MMT module system
  enabling going around MMT's implicit assumptions

this talk: UniFormal's 3-tiered identifier system

UniFormal Syntax

Basic Syntax

As usual: terms t and types A

- numbers, functions, products, ...
- subtyping, structure types, inductive types
- declarations
  - typed term symbol: **term** $c : A[= t]$ axioms via special types
  - bounded type symbol: **type** $a[= A]$

Basic Syntax

As usual: terms t and types A

- numbers, functions, products, ...
- subtyping, structure types, inductive types
- declarations
  - typed term symbol: **term** $c : A[= t]$ axioms via special types
  - bounded type symbol: **type** $a[= A]$

Special: theory expressions T

- declared/referenced like terms/types: **theory** $\tau = T$
- theory values: list of term/type/theory declarations
  theory $\texttt{Monoid} = \{$**type** $u,$ **term** $\texttt{oper} : (u, u) \to u, \ldots\}$
- union/inheritance via include declarations
  theory $\texttt{Group} = \{$**include** $\texttt{Monoid}, \ldots\}$
- instantiation/sharing by declaration merging (next slide)

Example: Algebraic Hierarchy

"..." indicates formulas/proofs that are omitted for brevity

theory `Monoid` $= \{$**type** u, **term** `oper` $: (u, u) \to u$, **term** $e : u$,
term `assoc` $: \blacktriangleright \forall x, y, z : u.\ \ldots$,
term `neutL` $: \blacktriangleright \forall x : u.\ \ldots$, **term** `neutR` $: \blacktriangleright \forall x : u.\ \ldots\}$

theory `IntAdd` $= \{$**total include** `Monoid`,
type $u =$ `Int`, **term** `oper` $= (x, y) \to x + y$, **term** $e = 0$,
term `assoc` $:= \ldots$, **term** `neutL` $:= \ldots$, **term** `neutL` $:= \ldots$
$\}$

total forces defining all included symbols

theory `Group` $= \{$**include** `Monoid`, $\ldots$, **term** `neutR` $:= \ldots\}$

declaration merging: `neutR` is defined in normal form of `Group`

Example: Models as Types

Theories as types and models:

- $Mod(T)$: theory seen as a type of models
- $mod(T)$: theory (with all fields defined) seen as an instance/model

Product monoid as a function:

$$\begin{array}{l}\textbf{term}\ \texttt{prod} : (Mod(\texttt{Monoid}), Mod(\texttt{Monoid})) \to Mod(\texttt{Monoid}) = \\ \quad (m, n) \to mod(\\ \quad\quad \textbf{type}\ u = (m.u, n.u), \\ \quad\quad \textbf{term}\ \texttt{oper} = (p, q) \to (m.\texttt{oper}(p_1, q_1), n.\texttt{oper}(p_2, q_2)), \\ \quad\quad \ldots \\ \quad)\end{array}$$

$$\textbf{term}\ \mathbb{Z}^2 = \texttt{prod}(mod(\texttt{IntAdd}), mod(\texttt{IntAdd}))$$

One More Novelty: Open vs. Closed Theories

Ubiquitous dichotomy of structuring mechanisms:

system	open	closed
OpenMath/MathML	content dictionary	—
MMT	—	theory
Isabelle	theory	locale, type class
Rocq	package	module, record
sTeX	module	mathstruct
Java	package	class
C++	namespace	class, struct

Key difference

- open: namespaces, cataloging of objects
  different theory = different qualified identifier
- closed: named set of assumptions
  different theory = different context

Open vs. Closed Theories in MMT

OMDoc

- Michael built OMDoc merging influences from
  - OpenMath — only open theories
  - development graphs (e.g., Isabelle locales) — only closed theories
- design flaw: no open/closed distinction

MMT

- inherited design flaw from OMDoc
- major source of friction ever since

in UniFormal: theories declared as either open or closed

Analogy to Open/Closed Worlds

Analogy

- worlds = theories
- inhabitant = named declarations

Open world

- inhabitants can be added/deleted at will
- at worst, downstream references must be adapted
- all language features must anticipate future changes to the world

no relying on the current set of inhabitants

Closed world

- world "fenced off"
- changing the inhabitants creates a different world
- world itself has meaning by fixing a set of inhabitants/assumptions

language features may enumerate inhabitants

Language Features that Enumerate Inhabitants

Theories as Interfaces, record types

- closed world declares signature, axioms — e.g., `Monoid`
- implementation must define every abstract field of C — e.g., `IntAdd`

Grammars, inductive types, term languages

- closed world declares constructors

  e.g., **type** n, **term** $z : n$, **term** $s : n \to n$
- induction on terms must have case for every constructor

Herbrand models, free generation

- closed world fixes true atomic statements
- negation-as-failure: atom is false if all proof paths exhausted

UniFormal goal: represent all such features through closed theories

Contexts

Context Design is Half the Work

Formal systems use judgments like $\Gamma \vdash t : A$ relative to context Γ

Function of context Γ

- provide identifiers currently in scope — related to scoping rules
- retrieve their types, definitions, etc.

may be intertwined with elaboration

Typical textbook/paper

- context = list of bound variables with their types
- practical systems need more complex contexts

Practical formal system

- imported packages, nested scopes, long-range dependencies, ...
- context semantics interacts with the entire language semantics

complex languages require complex data structures for the context

UniFormal Contexts

3 Context levels

- global: forest of open theories
  toplevel names, imports, the whole library, ...
- regional: the current closed theory
- local: list of bound variables blocks, binders, meta-variables, ...

$$\textbf{open theory}\ \tau_1 = \{ \vec{D},\ \textbf{open theory}\ \tau_2 = \{ \vec{E},\ \textbf{theory}\ \tau_3 = \{ \vec{F},\ \textbf{theory}\ \tau_4 = \{\vec{G},\ \textbf{term}\ c = \lambda x : A.t\}\}\}\}$$

Example: context for t consists of

- global: $\vec{D}$, $\vec{E}$ global access to, e.g., $\tau_1.\tau_2.\tau_3$
- regional: $\vec{F}$, $\vec{G}$ access to $\vec{F}$ and τ_4 requires instance of τ_3
- local: x no access to x from outside t

Regional vs. Local/Global

Scoping

- global: visible everywhere — resolve name clashes by qualified names
- regional:
- local: narrow syntactic scope — resolve name clashes by shadowing, α-equality

Regional vs. Local/Global

Scoping

- global: visible everywhere — resolve name clashes by qualified names
- regional: globally visible but controlled access — resolve name clashes by declaration merging
- local: narrow syntactic scope — resolve name clashes by shadowing, α-equality

Regional vs. Local/Global

Scoping

- global: visible everywhere — resolve name clashes by qualified names
- regional: globally visible but controlled access
  resolve name clashes by declaration merging
- local: narrow syntactic scope
  resolve name clashes by shadowing, α-equality

Behavior during traversal

- global: fixed, never changes
- regional:
- local: grows as we traverse binders in expressions

Regional vs. Local/Global

Scoping

- global: visible everywhere — resolve name clashes by qualified names
- regional: globally visible but controlled access
  resolve name clashes by declaration merging
- local: narrow syntactic scope
  resolve name clashes by shadowing, α-equality

Behavior during traversal

- global: fixed, never changes
- regional: switch to a different closed theory
- local: grows as we traverse binders in expressions

Regional vs. Local/Global

Scoping

- global: visible everywhere — resolve name clashes by qualified names
- regional: globally visible but controlled access — resolve name clashes by declaration merging
- local: narrow syntactic scope — resolve name clashes by shadowing, α-equality

Behavior during traversal

- global: fixed, never changes
- regional: switch to a different closed theory
- local: grows as we traverse binders in expressions

Semantics

- global: fixed, tied to qualified identifier
- regional:
- local: none, just α-renamable placeholder

Regional vs. Local/Global

Scoping

- global: visible everywhere — resolve name clashes by qualified names
- regional: globally visible but controlled access — resolve name clashes by declaration merging
- local: narrow syntactic scope — resolve name clashes by shadowing, α-equality

Behavior during traversal

- global: fixed, never changes
- regional: switch to a different closed theory
- local: grows as we traverse binders in expressions

Semantics

- global: fixed, tied to qualified identifier
- regional: depends on how the containing theory is used
- local: none, just α-renamable placeholder

Switching Regions: Examples

Model types: type-check m.`oper` for m : `Monoid`

- infer m : `Monoid`
- switch to region `Monoid` and infer type `oper` : A
- return and translate A to current region

Switching Regions: Examples

Model types: type-check m.`oper` for m : `Monoid`

- infer m : `Monoid`
- switch to region `Monoid` and infer type `oper` : A
- return and translate A to current region

Explicit theory morphisms application

- given theory morphism $\vartheta : C \to D$ and C-term t: use $\vartheta(t)$ in D
- switch the current region from D to C to check term t

Switching Regions: Examples

Model types: type-check m.`oper` for m : `Monoid`

- infer m : `Monoid`
- switch to region `Monoid` and infer type `oper` : A
- return and translate A to current region

Explicit theory morphisms application

- given theory morphism $\vartheta : C \to D$ and C-term t: use $\vartheta(t)$ in D
- switch the current region from D to C to check term t

Nesting of Theories

- a closed anonymous theory inside a closed one inside an open one:

$$\textbf{open theory}\ \texttt{Algebra} = \{\textbf{theory}\ \texttt{Monoid} = \{\ldots,\ \textbf{term}\ \texttt{unitGroup} = \texttt{Group}\{\textbf{type}\ u = \textbf{up}.u | \texttt{hasInverse}, \ldots\}\}\}$$

- **up** refers to the enclosing region, i.e., the monoid

Final Definition

A UniFormal context consists of

- global part: an open theory
- stack of pairs of
  - regional part: closed theory
  - local part: list of bound variables

roughly corresponds to call stack in programming languages

Controlled visibility of lower stack frames

- nested theories: lower regions visible via **up** qualifier
- most other features: no visibility
- some features: lower frames may escape back into higher ones

e.g., quotation and evaluation

Advanced Detail: Mixing Open and Closed Declarations

Open symbols in a closed theory

- corresponds to Java-style static declarations
- allows including constructors/factory methods within a theory

$$\textbf{closed theory}\ \texttt{Monoid} = \{\ldots,\ \textbf{open term}\ \texttt{make} : \ldots \to \mathit{Mod}(\texttt{Monoid}) = \ldots\}$$

Closed symbols in an open theory

- conservative extensions problematic with closed theories
  - extensions typically distributed over many source files
  - combining theories may require expensive renaming
- very neat solution that yields seamless interface extension

$$\textbf{open theory}\ \texttt{Squaring} = \{\textbf{include}\ \texttt{Monoid},\ \textbf{term}\ \texttt{square} = x \to \texttt{oper}(x, x)\}$$

$m.(\mathit{Squaring.square})$ well-typed for $m : \mathit{Mod}(\texttt{Monoid})$

Conclusion

Summary

In the paper

- core grammar for UniFormal
- example feature: record types/classes via closed theories
- rigorous definition of context
- sketch of context-sensitive type-checker
- motivating examples and side remarks

Not in the paper

but implementation of UniFormal PL exists

- rich type system: dependent function types, number types, ...
- additional features: term languages, inductive data types, quotation
- details of type checker
- operational semantics

https://github.com/UniFormal/UPL/

Evaluation and Outlook

Knowledge gain

- open vs. closed theories now canon for me
- theory expressions superior to MMT's 2-layer design
  need to watch for scalability though
- 3-partite context with stack of regions feels right
  smooth implementation for both logics and programming languages
  but details still up for discussion

Open questions and plans

- connect MMT and UniFormal
  - MMT to use open/closed theories and declaration-merging
  - maybe MMT as logical framework instance of UniFormal
- formalize significant case study
  - target: algorithms in AI course
  - parallel design of math definitions (notes) and executable algorithms (exercises)