ESSLLI 2019 - Formalizing the Zoo of Logical Systems

Logic in Computer Science

- $\sim$ 1930: computer science — vision of mechanizing logic
- Competition between multiple logics
  - axiomatic set theory: ZF(C), GBvN, ...
  - λ-calculus:
    - typed or untyped
    - Church-style or Curry-style
  - new types of logic modal, intuitionistic, paraconsistent ,...
- Diversification into many different logics
  - fine-tuned for diverse problem domains
    far beyond predicate calculus
  - deep automation support
    decision problems, model finding, proof search, ...
  - extensions towards programming languages

History of Formal Systems

- late 19th century: formal axiomatizations
- $\sim$ 1900: paradoxa in logic, mathematics
- $\sim$ 1920*s*: vision of mechanizing logic
- $\sim$ 1930*s*: birth of computer science

Desire for automating

- formal representation
- computation
- logical proof

Universal approach to intertwined problems

Competition between multiple languages

- axiomatic set theory: ZF, GBvN, ...
- type theory: Principia Mathematica, λ-calculus
- new logics: modal, intuitionistic, advanced type systems...

Diversification into many different languages

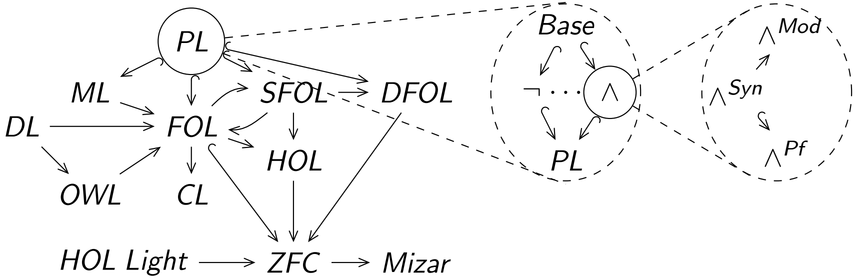
The LATIN Atlas of Logical Systems

The LATIN Atlas is huge: That's me pointing at the theory for first-order logic

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

Vision

UniFormal

a universal framework for the
formal representation of all knowledge and its semantics
in math, logic, and computer science

- Avoid fixing languages wherever possible ...
- ... and instantiate them for different languages
- Use formal meta-languages in which to define languages ...
- ... and avoid fixing even the meta-language
- Obtain foundation-independent results
  - Representation languages
  - Soundness-critical algorithms
  - Knowledge management services
  - User-facing applications

MMT = Meta-Meta-Theory/Tool

Problem:

- logical frameworks not expressive for practical logics
- more system experimentation needed
- trend towards fine-grained user control

Foundation-independence: use logical frameworks without committing to a specific one

Mathematics	Logic	Logical Frameworks	Foundation-Independence
			MMT
		logical frameworks	
	logic, programming language, . . .		
domain knowledge			

Logical Frameworks

= meta-logic in which syntax and semantics of object logics are defined — Automath, LF, Isabelle

Advantages

- Universal concepts — expressions, substitution, typing, equality, ...
- Meta-reasoning — consistency, logic translations, ...
- Rapid prototyping — type reconstruction, theorem proving, ...
- Generic tools — theorem prover, module system, IDE, ...

Simplicity vs. expressivity

- Meta-logic must be simple to be scalable, trustworthy
- Object logic must be expressive to be practical
- Big challenge for frameworks

Designing Logical Frameworks

Typical approach:

- choose a λ-calculus
- add other features
  - meta-logic for logic programming (Twelf)
  - meta-logic with induction on syntax (Abella)
  - proof assistant for object logic (Isabelle)
  - concurrency (CLF)
  - reasoning about contexts (Beluga)
  - rewriting (Dedukti)
  - external side conditions (LLFP)
  - coupling with proof-assistant support (Hybrid)
  - ...

Problems

- Divergence due to choice of other features
- Even hypothetical union not expressive enough for real-life logics

no way to define, e.g., HOL Light, Mizar, PVS

Experimentation with Formal Systems

Customize the system fundamentals

- increasingly complex problem domains

  e.g., mathematics, programming languages
- plain formalization introduces too many artifacts to be human-readable
- therefore: allow users to define how to interpret human input

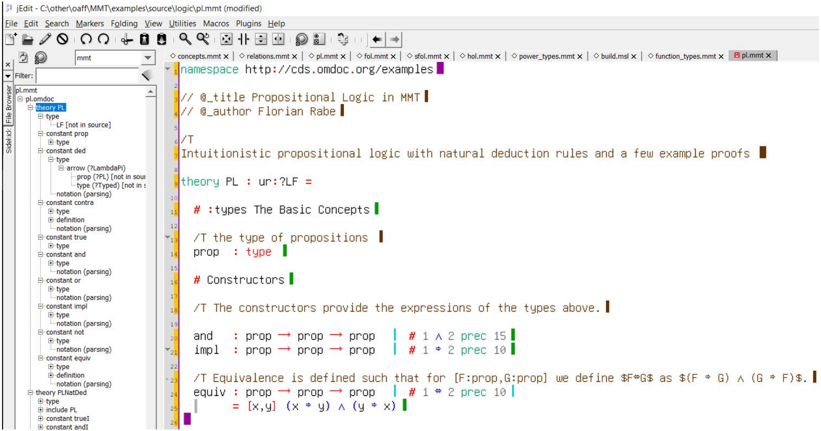
  e.g., custom parsing, type reconstruction

Examples:

- unification hints (Coq, Matita)
  - extra-logical declarations
  - allow users to guide incomplete algorithms (e.g., unification)
- meta-programming (Idris, Lean)
  - expose internal datatypes to user
  - allow users to program extensions in the language itself

Example

Example: Propositional Logic in the MMT IDE



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
  ...
}
```

The UniFormal Library

Large Scale Example: The LATIN Atlas

- DFG project 2009-2012 (with DFKI Bremen and Jacobs Univ.)
- Highly modular network of little logic formalizations
  - separate theory for each
    - connective/quantifier
    - type operator
    - controversial axioms e.g., excluded middle, choice, ...
    - base type
  - reference catalog of standardized logics
  - documentation platform
- Written in MMT/LF
- 4 years, with $\sim$ 10 students, $\sim$ 1000 modules

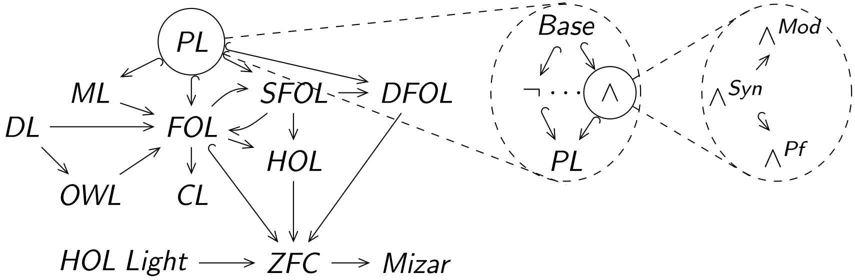
The LATIN Atlas of Logical Systems

The LATIN Atlas is huge: That's me pointing at the theory for first-order logic

Logic Diagrams in LATIN

An example fragment of the LATIN logic diagram

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

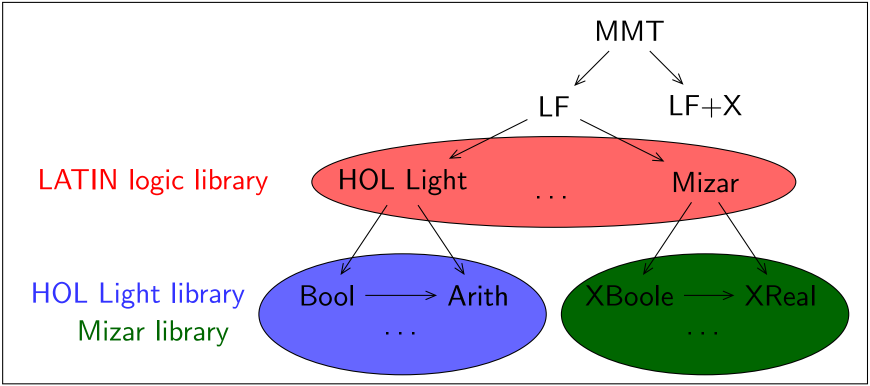


- each node is root for library of that logic
- each edge yields library translation functor

library *integration* very difficult though

OAF: Integration of Proof Assistant Libraries

- DFG project, 2014–2020, 15 contributors
- Big, overlapping libraries joined in MMT as the uniform representation language $>$ 100 GB XML in total
  Mizar, HOL systems, IMPS, Coq, PVS, Isabelle...
- enables archival, comparison, integration



OpenDreamKit: Virtual Math Research Environments

- EU project, 2015-2019, 15 sites, 25 partners
  http://opendreamkit.org/
- MMT as mediator system
  - system-independent formalization of math >200 theories
    no proofs, no algorithms
  - integration of math computation systems
    SageMath, GAP, Singular: services interfaces defined in MMT
  - ...and math databases
    LMFDB, OEIS: database schemas defined in MMT

Example: dynamic retrieval

- SageMath user needs 13th transitive group with conductor 5
- SageMath queries MMT
- MMT retrieves it from LMFDB, translates it to SageMath syntax

MathHub

GitHub-like but for MMT projects https://gl.mathhub.info

- 251 Repositories
- 187 Users
- 28.5 GB in March, probably doubled by now

For example:

Language	Library	Modules	Declarations
MMT	Math-in-the-Middle	220	826
LF	LATIN	529	2,824
PVS	Prelude+NASA	974	24,084
Isabelle	Distribution+AFP	9553	1,472,280
HOL Light	Basic	189	22,830
Coq	$>$ 50 in total	1,979	167,797
Mizar	MML	1,194	69,710
SageMath	Distribution	1,399	
GAP	Library		9,050

MMT as a UniFormal Framework

Design Principles

few primitives . . . that unify different domain concepts

- tiny but universal grammar for expressions
  syntax trees with binding
- standardized semantics of identifiers crucial for interoperability
- high-level definitions derivation, model, soundness, . . .
- foundation-independent: no built-in logic, type system, set theory etc. all algorithms parametric
- theories and theory morphisms for large scale structure
  translation, interpretation, semantics, . . .

Foundation-Independent Development

Typical workflow

1. choose foundation
   type theories, set theories, first-order logics, higher-order logics, ...
2. implement kernel
3. build support algorithms reconstruction, proving, editor, ...
4. build library

Foundation-independent workflow in MMT

1. MMT provides generic kernel
   no built-in bias towards any foundation
2. build support algorithms on top of MMT
3. choose foundation(s)
4. customize MMT kernel for foundation(s)
5. build foundation-spanning universal library

Advantages

- Avoids segregation into mutually incompatible systems
- Formulate maximally general results

  meta-theorems, algorithms, formalizations
- Rapid prototyping for logic systems

  customize MMT as needed, reuse everyting else
- Separation of concerns between
  - foundation developers
  - support service developers: search, axiom selection, ...
  - application developers: IDE, proof assistant, wiki, ...
- Allows evolving foundation along the way

  design flaws often apparent only much later
- Migrate formalizations when systems die
- Archive formalizations for future rediscovery

Modular Framework Definitions in MMT

Individual features given by set of symbols, notations, rules

- $\lambda\Pi$
- Rewriting
- Polymorphism
- Subtyping (ongoing)
- ...

Language definitions are modular themselves

e.g., Dedukti = LF + rewriting

MMT Tool

Mature implementation

- API for representation language — foundation-independent
- Collection of reusable algorithms — no commitment to particular application
- Extensible wherever reasonable
  - storage backends, file formats, user interfaces, . . .
  - operators and rules, language features, checkers, . . .

Separation of concerns between

- Foundation developers — e.g., language primitives, rules
- Service developers — e.g., search, theorem prover
- Application developers — e.g., IDE, proof assistant

Yields rapid prototyping for logic systems

MMT Tool

Mature implementation

- API for representation language foundation-independent
- Collection of reusable algorithms

  no commitment to particular application
- Extensible wherever reasonable

  storage backends, file formats, user interfaces, ...

  operators and rules, language features, checkers, ...

Separation of concerns between

- Foundation developers e.g., language primitives, rules
- Service developers e.g., search, theorem prover
- Application developers e.g., IDE, proof assistant

Yields rapid prototyping for logic systems

But how much can really be done foundation-independently?

MMT shows: not everything, but a lot

MMT-Based Foundation-Independent Results

Logical Result: Representation Language

- MMT theories uniformly represent
  - logics, set theories, type theories, algebraic theories, ontologies, ...
  - module system: state every result in smallest possible theory

    Bourbaki style applied to logic
- MMT theory morphisms uniformly represent
  - extension and inheritance
  - semantics and models
  - logic translations
- MMT objects uniformly represent
  - functions/predicates, axioms/theorems, inference rules, ...
  - expressions, types, formulas, proofs, ...
- Reuse principle: theorems preserved along morphisms

Logical Result: Concepts

MMT allows coherent formal definitions of essential concepts

- Logics are MMT theories
- Foundations are MMT theories — e.g., ZFC set theory
- Semantics is an MMT theory morphism — e.g., from FOL to ZFC
- Logic translations are MMT theory morphisms
- Logic combinations are MMT colimits

Logical Results: Algorithms

- Module system
  modularity transparent to foundation developer
- Concrete/abstract syntax
  notation-based parsing/presentation
- Interpreted symbols, literals
  external model/implementation reflected into MMT
- Type reconstruction
  foundation plugin supplies only core rules
- Simplification
  rule-based, integrated with type reconstruction

- Theorem proving?
- Code generation? Computation?

Knowledge Management Results

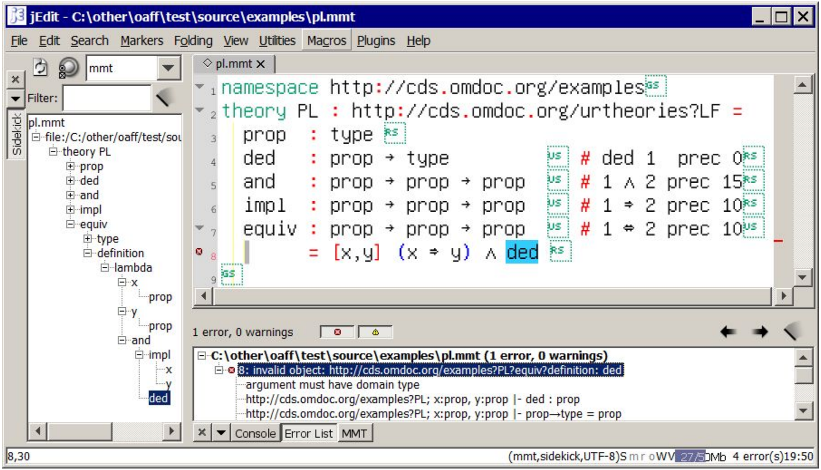
- Change management — recheck only if affected
- Project management — indexing, building
- Extensible export infrastructure — Scala, SVG graphs, LaTeX, HTML, ...
- Search, querying — substitution-tree and relational index
- Browser — interactive web browser, 2D/3D theory graph viewer
- Editing — IDE-like graphical interface, LaTeX integration

IDE

- Inspired by programming language IDEs
- Components
  - jEdit text editor (in Java): graphical interface
  - MMT API (in Scala)
  - jEdit plugin to tie them together

    only $\sim$ 1000 lines of glue code
- Features
  - outline view
  - error list
  - display of inferred information
  - type inference of subterms
  - hyperlinks: jump to definition
  - search interface
  - context-sensitive auto-completion: show identifiers that

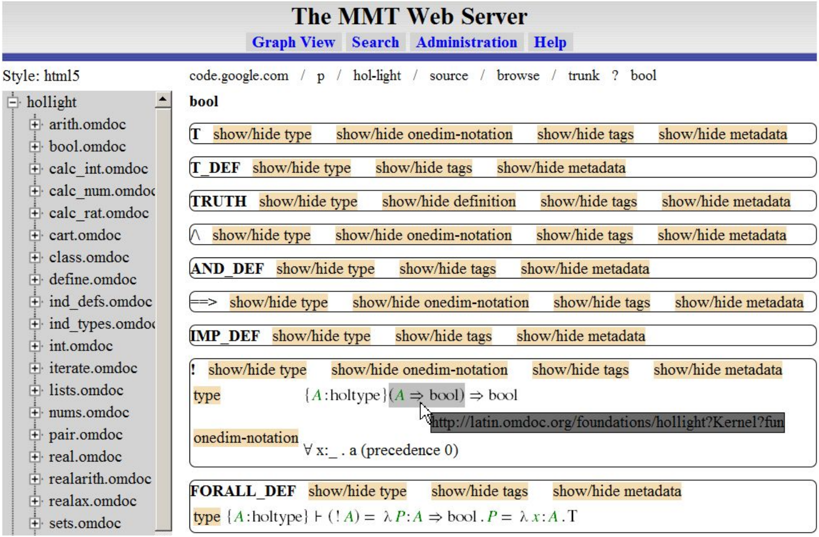
IDE: Example View



An Interactive Library Browser

- MMT content presented as HTML5+MathML pages
- Dynamic page updates via Ajax
- MMT used through HTTP interface with JavaScript wrapper
- Features
  - interactive display e.g., inferred types, redundant brackets
  - smart navigation via MMT ontology
    can be synchronized with jEdit
  - dynamic computation of content
    e.g., definition lookup, type inference
  - graph view: theory diagram as SVG

Browser: Example View

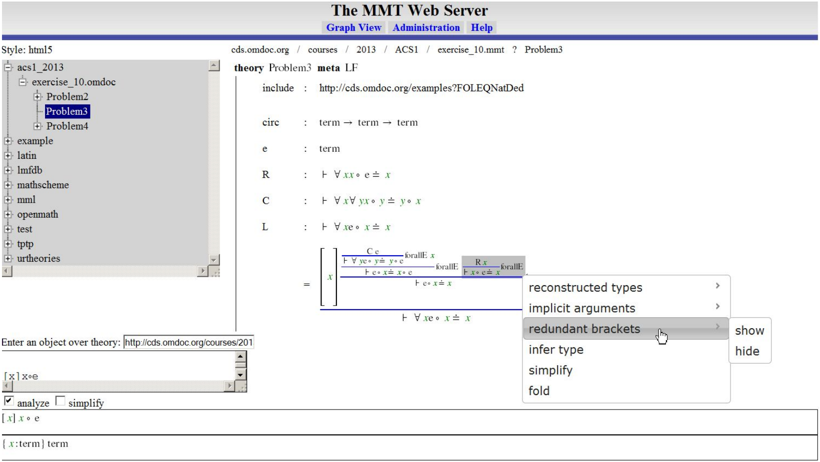


Browser Features: 2-dimensional Notations

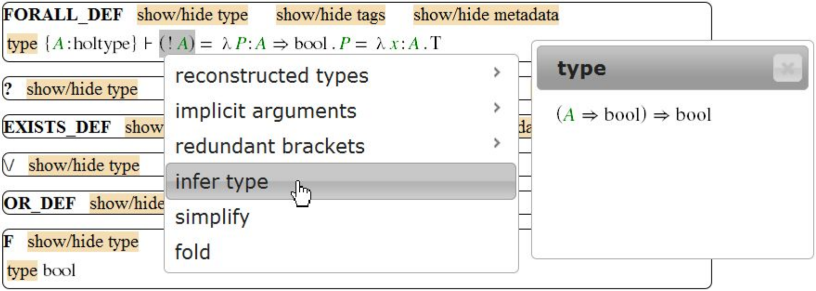
REAL_POW_DIV show/hide type show/hide definition

type $\vdash \forall x\!:\!\mathrm{real}\,.\,\forall y\!:\!\mathrm{real}\,.\,\forall n\!:\!\mathrm{num}\,.\left(\frac{x}{y}\right)^n = \frac{x^n}{y^n}$

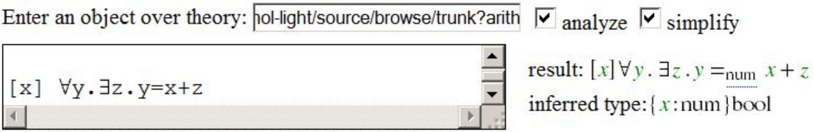
Browser Features: Proof Trees



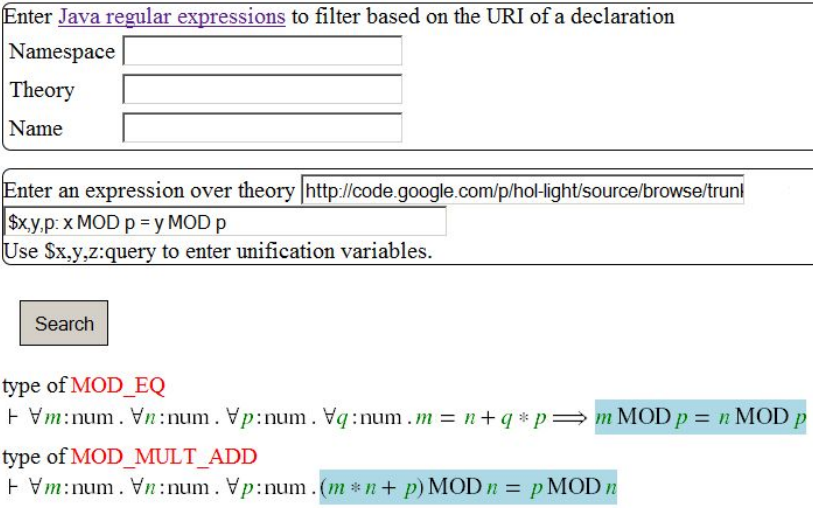
Browser Features: Type Inferece



Browser Features: Parsing



Example Service: Search



Example Service: Theory Graph Viewer

Theory graphs with 1000s of nodes
$\rightarrow$ special visualization tools needed recently even in 3D

demo at `https://www.youtube.com/watch?v=Mx7HSWD5dwg`

LaTeX Integration

- upper part: LaTeX source for the item on associativity
- lower part: pdf after compiling with LaTeX-MMT
- enriched with type inference, cross references, tooltips

e.g., type argument M of equality symbol

```
\begin{mmtscope}
 For all \mmtvar{x}{in M},\mmtvar{y}{in M},\mmtvar{z}{in M}
 it holds that !(x * y) * z = x * (y * z)!
\end{mmtscope}
```

A *monoid* is a tuple $(M, \circ, e)$ where

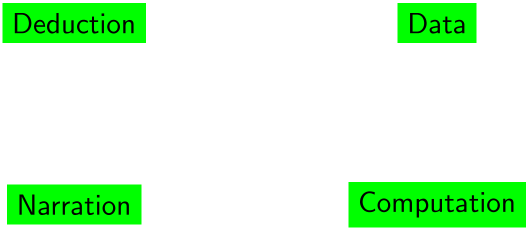
- M is a sort, called the universe.
- $\circ$ is a binary function on M.
- e is a distinguished element of M, the unit.

such that the following axioms hold:

- For all x,y,z it holds that $(x{\circ}y){\circ}z{=}_M x{\circ}(y{\circ}z)$
- For all x it holds that $x{\circ}e{=}_M x$ and $e{\circ}x{=}_M x$.

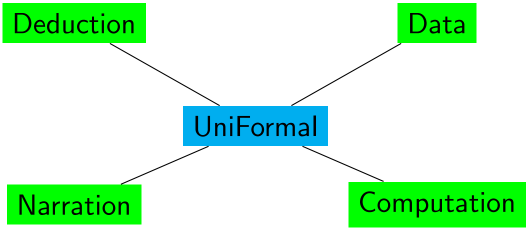
Subsume All Aspects of Knowledge

- Narration: informal-but-rigorous math
  needed for human consumption
- Deduction: logic and type systems
  needed for machine understanding
- Computation: data structures and algorithms
  needed for practical applications
- Data: tabulate large sets and functions
  needed for examples, exploration and efficiency



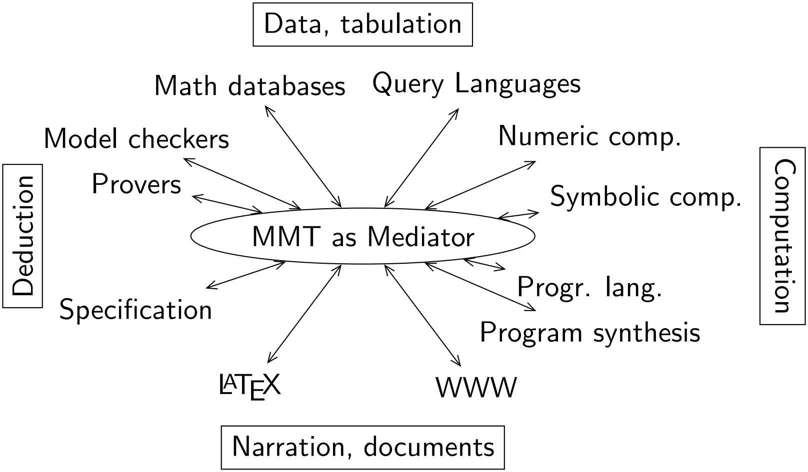
Subsume All Aspects of Knowledge

- Narration: informal-but-rigorous math
  needed for human consumption
- Deduction: logic and type systems
  needed for machine understanding
- Computation: data structures and algorithms
  needed for practical applications
- Data: tabulate large sets and functions
  needed for examples, exploration and efficiency
- Universal representation language
  key to universality, inter-operability



MMT as System Integration Platform

All system interfaces formalized in MMT
$\rightarrow$ semantics-aware tool integration while maintaining existing work flows



Envisioned: A Generic Theorem Prover

- Theorem proving currently highly logic-specific
- But many successful designs actually logic-independent, e.g.,
  - Declarative proof languages
  - Tactic languages
  - Integration of decision procedures
  - Axiom selection
  - Modularity

Claim

- Logic-specific implementations a historical accident
- Possible to build MMT level proving technology

Envisioned: Computational Knowledge

- So far: MMT focuses on declarative formal languages
- Goal: extend to programming languages
  - understand key concepts foundation-independently
    state/effects, recursion, inheritance, ...
  - represent features modularly
  - freely mix logic and computation share declarative aspects
- Syntax is (kind of) easy:
  - programming languages are MMT theories
  - classes/modules are MMT theories
  - programs are expressions
- Semantics is open question: What is the general judgment?

Conclusion

Summary

- MMT: foundation-independent framework for declarative languages
  - representation language
  - implementation
- Easy to instantiate with specific foundations

  rapid prototyping logic systems
- Deep foundation-independent results
  - logical: parsing, type reconstruction, module system, ...
  - knowledge management: search, browsers, IDE, ...
- Serious contender for
  - experimenting with new system ideas
  - generic applications/services
  - universal library
  - system integration platform

Further Resources

Websites

- MMT http://uniformal.github.io
- all my papers https://kwarc.info/people/frabe

Selected publications

- the primary paper on the MMT language (I&C 2013, with M. Kohlhase): *A Scalable Module System*
- a more recent paper on the MMT approach to logic (JLC 2014): *How to Identify, Translate, and Combine Logics?*
- Foundations in LATIN: (MSCS 2011, with M. Iancu) *Formalizing Foundations of Mathematics*
- Modular logics in LATIN (TCS 2011, with F. Horozal): *Representing Model Theory in a Type-Theoretical Logical Framework*
- the primary paper on OAF (JFR 2015, with M. Kohlhase) *QED Reloaded*
- a tutorial/example style paper (LFMTP 2019, with D. Müller) *Rapid Prototyping Formal Systems in MMT: 5 Case Studies*
- example papers available for integrations of the libraries of Mizar, HOL Light, PVS, IMPS, Coq, and (forthcoming) Isabelle

Details: Foundations

Foundations

- Foundation = the most primitive formalism on which everything else is built

  set theories, type theories, logics, category theory, ...
- We can fix the foundation once and for all — but which one?
- In math: usually implicit and arbitrary foundation
  - can be seen as avoiding subtle questions
  - but also as a strength: it's more general
- In CS: each system fixes its own foundational language

  e.g., a variant of type theory or HOL
- Programming languages foundations as well

  but representation of state in MMT still open problem

Fixed Foundations

- Fixing foundation the first step of most implementations
  often foundation and implementation have the same name
- No two implementations for the exact same foundation
  even reimplementations diverge quickly
- Negative effects
  - isolated, mutually incompatible systems
    no sharing of results, e.g., between proof assistants
  - no large scale libraries
    each system's library starts from scratch
  - no library archival
    libraries die with the system
  - comparison of systems difficult
    no common problem set
  - slow evolution
    evaluating a new idea can take years

Details: MMT Syntax

Basic Concepts

Design principle

- few orthogonal concepts
- uniform representations of diverse languages

sweet spot in the expressivity-simplicity trade off

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 ...

Theory Morphisms

Theories

- uniform representation of
  - foundations e.g., logical frameworks, set theories, ...
  - logics, type theories
  - domain theories e.g., algebra, arithmetic, ...
- little theories: state every result in smallest possible theory
  maximizes reuse

Theory morphisms

- uniform representation of
  - extension e.g., Monoid $\rightarrow$ Group
  - inheritance e.g., superclass $\rightarrow$ subclass
  - semantics e.g., FOL $\rightarrow$ ZFC
  - models e.g., Nat: Monoid $\rightarrow$ ZFC
  - translation e.g., typed to untyped FOL
- homomorphic translation of expressions
- preserve typing (and thus truth)

Paradigms

- judgments as types, proofs as terms
  unifies expressions and derivations
- higher-order abstract syntax unifies operators and binders
- category of theories and theory morphisms
  - languages as theories
    unifies logical theories, logics, foundations
  - relations as theory morphisms
    unifies modularity, intepretations, representation theorems
- institution-style abstract model theory
  uniform abstract concepts
- models as morphisms (categorical logic)
  unifies models and translations and semantic interpretations

Abstract Syntax of Terms

Key ideas

- no predefined constants
- single general syntax tree constructor $c(\Gamma; \vec{E})$
- $c(\Gamma; \vec{E})$ binds variables and takes arguments
  - non-binding operators: Γ empty e.g., $\texttt{apply}(\cdot; f, a)$ for $(f\ a)$
  - typical binders: Γ and $\vec{E}$ have length 1

    e.g., $\texttt{lambda}(x:A; t)$ for $\lambda x:A.t$

contexts	Γ	$::=$	$(x[: E][= E])^*$
terms	E	$::=$	
constants		$\vert$	c
variables		$\vert$	x
complex terms		$\vert$	$c(\Gamma; E^*)$

Terms are relative to theory T that declares the constants c

Concrete Syntax of Terms

- Theories may attach notation(s) to each constant declaration
- Notations of c introduce concrete syntax for $c(\Gamma; \vec{E})$

e.g., for type theory

concrete syntax	constant declaration	abstract syntax
$E ::=$		
`type`	`type` #	`type`
$\Pi x : E_1.E_2$	`Pi` # Π V1 . 2	`Pi`$(x : E_1; E_2)$
$E_1 \to E_2$	`arrow` # 1 → 2	`arrow`$(\cdot; E_1, E_2)$
$\lambda x : E_1.E_2$	`lambda` # λ V1 . 2	`lambda`$(x : E_1; E_2)$
$E_1\ E_2$	`apply` # 1 2	`apply`$(\cdot; E_1, E_2)$

Notations

MMT implements parsing and rendering foundation-independently
relative to notations declared in current theory

Notations			$(ARG \mid VAR \mid DELIM)^*[PREC]$
Bound variable	VAR	$::=$	Vn for $n \in \mathbb{N}$
Argument	ARG	$::=$	n for $n \in \mathbb{N}$
Delimiter	$DELIM$	$::=$	Unicode string
Precedence	$PREC$	$::=$	integer

Bound variabes	`forall`	#	$\forall V1.2$
Mixfix	`setComprehension`	#	$\{V1 \in 2 \mid 3\}$
Argument sequences	`plus`	#	$1 + \ldots$
Variable sequences	`forall`	#	$\forall V1, \ldots 2$
Implicit arguments	`functionComposition`	#	$4 \circ 5$

Abstract Syntax of Theories

- Theories are named lists of declarations
- Theory names yield globally unique identifiers for all constants
- Module system: Previously defined theories can be included/instantiated

theory declaration			$T = \{Dec^*\}$
	Dec	$::=$	
constant declaration		$\mid$	$c[: E][= E][\#Notation]$
theory inclusion		$\mid$	$\texttt{include}\ T$
theory instantiation		$\mid$	$\texttt{structure}\ c : T\ \texttt{where}\{Dec^*\}$

Flattening: Every theory is semantically equivalent to one without inclusions/instantiations   intuition: theories are named contexts

Details: Typing

Judgments

- MMT terms subsume terms of specific languages
- Type systems singles out the well-typed terms

For any theory Σ:

$\vdash \Sigma$	$T = \{\Sigma\}$ is a valid theory definition
$\vdash_T \Gamma$	Γ is a valid context
$\Gamma \vdash_T t : A$	t has type A
$\Gamma \vdash_T E = E'$	E and E' are equal
$\Gamma \vdash_T \_ : A$	A is inhabitable

- MMT defines some rules once and for all

  foundation-independent rules

- Foundation-independent declared in theories

  rule for c defines when $c(\Gamma; E^*)$ well-typed

Foundation-Independent Rules

- Lookup rules for atomic terms over a theory $T = \{\Sigma\}$

$$\frac{c : A \text{ in } \Sigma}{\vdash_T c : A} \qquad \frac{c = t \text{ in } \Sigma}{\vdash_T c = t}$$

- Equivalence and congruence rules for equality
- Rules for well-formed theories $T = \{\Sigma\}$

$$\frac{}{\vdash \cdot} \qquad \frac{\vdash \Sigma \quad [\vdash_\Sigma \_ : A] \quad [\vdash_T t : A]}{\vdash \Sigma,\ c[: A][= t]}$$

- Rules for well-formed contexts similar to theories

Foundation-Specific Rules

- Declared in theories as constants
- Module system allows composing foundations

Two options to give rules

1. For a few dedicated meta-logics
   - rules are constants without type, definiens, or notation
   - meaning provided externally on paper or as code snippet
   - necessary to get off the ground
   - Examples:
     - logical framework LF: $\sim$ 10 rules
     - shallow polymorphism: 1 rule
     - modulo rewriting: 1 rule family
2. For any other language
   - include a meta-logic
   - use meta-logic to give rules as typed constants
   - example: modus ponens using meta-logic LF

$$
\begin{array}{lcl}
o & : & \mathtt{type} \\
\Rightarrow & : & o \to o \to o \\
\mathtt{ded} & : & o \to \mathtt{type} \\
\mathtt{mp} & : & \Pi_{A,B}\mathtt{ded}\,(A \Rightarrow B) \to \mathtt{ded}\,A \to \mathtt{ded}\,B
\end{array}
$$

Type Reconstruction

Type checking:

- input: judgement, e.g., $\Gamma \vdash_T t : A$
- output: true/false, error information

Type reconstruction

- input judgment with unknown meta-variables
  - implicit arguments, type parameters
  - omitted types of bound variables
- output: unique solution of meta-variables that makes judgement true
- much harder than type checking

MMT implements foundation-independent type reconstruction

- transparent to foundations
- no extra cost for foundation developer

Implementation

MMT implements foundation-independent parts of type checker

- foundation-independent rules
- simplification, definition expansion
- error reporting
- abstract interfaces for foundation-specific rules, e.g.,
  - infer type
  - check term at given type
  - check equality of two given terms
  - simplify a term

Foundation-specific plugins add concrete rules

- each rule can recurse into other judgements
- example LF: $\sim$ 10 rules for LF, $\sim$ 10 lines of code each