

Formal Logic Definitions for Interchange Languages

Fulya Horozal and Florian Rabe

Jacobs University Bremen, Computer Science

MKM track at CICM 2015

System Integration

- ▶ Formal systems notoriously badly integrated
 - deduction, computation, knowledge management, . . .
 - ▶ different logical foundations
 - ▶ no high-level APIs usually just text files
 - ▶ not designed with integration in mind hard to retro-fit
 - ▶ little reward for integrating systems let alone maintaining the integration
- ▶ To some degree unavoidable
 - ▶ each research group needs unique project
 - ▶ different systems have different strengths
 - ▶ experimentation requires new systems
 - ▶ many systems short-lived anyway diversity can be a good thing

Need for Interchange Languages

Various research efforts towards **tighter** system integration

not this talk

Interchange languages allow **loose** integration

this talk

- ▶ intermediate format to exchange knowledge between systems
- ▶ less efficient send around text files
- ▶ much cheaper allows separation of concerns
- ▶ helps (requires) standardizing/documenting system interfaces beneficial in any case

Two major interchange languages

MathML/OpenMath/OMDoc

- ▶ rooted in CICM community
- ▶ designed as standardized interchange languages
- ▶ (mostly) XML-based, schema validation
- ▶ focus on covering all mathematical knowledge

TPTP

- ▶ rooted in deduction community (mostly CADE)
- ▶ grew out of Geoff Sutcliffe's tool suite (Univ. of Miami)
- ▶ text syntax, good parser support
- ▶ focus on capabilities of theorem provers

Advantages

MathML

- ▶ very simple abstract syntax
- ▶ good documentation
- ▶ wide logical coverage

TPTP

- ▶ both human- and machine-readable/writable
- ▶ widely adopted by deduction systems
- ▶ large centralized collection of challenge problems
- ▶ reference implementation and tool suite

Disadvantages

MathML

- ▶ unwieldy XML syntax intended for machines
- ▶ small collection of decentralized content dictionaries
- ▶ no reference implementation several tools with varying degrees of coverage
- ▶ no formal semantics relegated to documentation of content dictionaries

TPTP

- ▶ some syntax idiosyncrasies
- ▶ narrow focus: FOL, HOL, arithmetic, ...
- ▶ no formal semantics defined in a few papers about individual fragments

Similarities

Pros and cons mostly disjoint

But 2 important similarities

1. Abstract logical properties
 - ▶ not that surprising if we think about it
 - ▶ but not that obvious either
2. Neither has formal semantics.
 - ▶ specifying semantics is hard
 - ▶ doing it formally is even harder

details on next 2 slides

Similarity: Syntax

Languages are quite similar if we abstract from

- ▶ concrete syntax
- ▶ user community
- ▶ intended purpose
- ▶ tool support

MathML Objects

- ▶ constants, variables, application, **arbitrary binding**
- ▶ **all** constants introduced/specified in **content dictionaries**

TPTP Formulas

- ▶ constant, variables, application, **built-in binders** $\forall \exists \lambda \Pi \Sigma \epsilon$
- ▶ **most** constants introduced/specified in **TPTP files**
- ▶ **built-in logic-related operators** but no fixed type system or calculus

Similarity: No Formal Semantics

Standardizing **syntax** is easy

- ▶ sufficient for formal systems to **talk** to each other
- ▶ both get the job done

Standardizing **semantics** is much harder

- ▶ requires type system+calculus
- ▶ necessary for formal systems to **understand** each other
syntax-only standard hides disagreements
- ▶ both allow variants with different semantics

Specifying the type system/calculus of a variant

- ▶ MathML: give content dictionary with logical operators, rules
logic1, quant1, ...
- ▶ TPTP: write paper
fof, tff, thf, thf1, ...

Specifying Semantics Formally

- ▶ Problem: How to give **formal** semantics?
machine-understandable
- ▶ Solution: Use logical framework!

Logical framework = meta-logic for specifying logics

- ▶ well-known examples: LF, Isabelle, λ Prolog
- ▶ specify logics, type theories, ...
- ▶ fully formal, machine-readable

Example: first-order logic in LF

$$\begin{array}{ll}
 \textit{form} & : \textit{type} & \textit{term} & : \textit{type} \\
 \vdash & : \textit{form} \rightarrow \textit{type} & \wedge & : \textit{form} \rightarrow \textit{form} \rightarrow \textit{form} \\
 \wedge I & : \prod_{A:\textit{form}, B:\textit{form}} \vdash A \rightarrow \vdash B \rightarrow \vdash (A \wedge B)
 \end{array}$$

Practical Interchange?

Problem: logical frameworks not practical

- ▶ not good at handling concrete syntax
separate tool must handle the actual interchange
- ▶ specifications often out of sync with actual interchange language
syntax-only standard hides disagreements
- ▶ additional overhead

Solutions: use MMT

details on following slides

What's MMT?

- ▶ Foundation-independent framework
 - ▶ avoid fixing logic/type theory wherever possible
 - ▶ generic theory and implementation
 - ▶ easy to instantiate with specific foundations
 - ▶ continued development since 2006
- ▶ MMT language
 - ▶ generic concepts: theory, morphism, declaration, object
 - ▶ > 200 pages of publications
- ▶ MMT system
 - ▶ > 30,000 lines of Scala code
 - ▶ ~ 10 CICM papers on individual aspects

<https://svn.kwarc.info/repos/MMT/>

MMT and MathML

- ▶ MMT \approx formal-ready version of MathML
 - ▶ MMT theories \approx MathML CDs
 - but with formal types, notations, axioms, theorems
 - ▶ MMT objects \approx MathML objects
 - but with formal typing relation, provability judgment
- ▶ OMDoc/OpenMath/MathML retained as concrete syntax
- ▶ MMT adds
 - ▶ human-friendly text syntax
 - ▶ parser, type-checker
 - ▶ module system
 - ▶ scalable implementation
 - ▶ integration with knowledge management services

MMT and Logical Frameworks

- ▶ MMT type system parametric in set of rules
- ▶ Supply a set rules = implement a new logical framework
rapid prototyping
- ▶ Each rule provided as code snippet
for LF: ~ 10 rules, ~ 10 lines of code each
- ▶ MMT handles
 - ▶ bureaucracy
 - ▶ error reporting
 - ▶ module system
 - ▶ knowledge management
- ▶ Rules provide logical core
good division of labor

MMT and TPTP

- ▶ Collaboration with Geoff Sutcliffe
 - ▶ TPTP type systems specified in LF
 - ▶ TPTP tools translate TPTP problems to LF
- ▶ Effectively: LF specifications official part of TPTP standard
if it doesn't type-check, Geoff complains

Formal logic definitions for interchange languages

Problem summary

- ▶ System integration needs interchange languages
- ▶ MathML/TPTP standardize syntax but semantics is informal
- ▶ Logical frameworks formalize semantics but are not practical

Solution

1. implement logical framework in MMT e.g., LF
2. specify interchange language in MMT/LF e.g., FOL
3. MMT induces
 - ▶ MathML content dictionary
 - ▶ TPTP type checker
 - ▶ MKM services

So we're done? — No: That's when this work started!

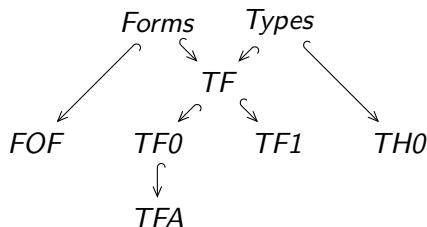
Little-known problem

- ▶ Common logical frameworks can't actually specify logics
not even the syntax of FOL
- ▶ Problem: Can't specify shape of **declarations**
will give examples on next slides

Solved in Fulya Horozal's PhD thesis (2014)

- ▶ Added **declaration patterns** to MMT
MKM 2012 (Horozal, Kohlhase, Rabe)
- ▶ Introduced new logical framework: LFS = LF with sequences
MKM 2014 (Horozal, Rabe, Kohlhase)
- ▶ Formally specified TPTP logics in MMT/LFS
this paper

Modular specifications



- ▶ *Forms*: formulas, propositional logic
- ▶ *Types*: types, typed terms
- ▶ *FOF*: untyped first-order logic
- ▶ *TF*: typed first-order logic
 - ▶ *TF0*: plain
 - ▶ *TF1*: with toplevel polymorphism
 - ▶ *TFA*: with arithmetic
- ▶ *TH0*: higher-order logic

Example: Untyped first-order logic

```

theory FOF = {
  o  : type                formulas
  &  : o → o → o         connectives
  i  : type                terms
  !  : (i → o) → o       quantifiers
  :
  New: pattern for  $n$ -ary function symbols
  pattern fun = [n : nat] {
    f : in → i
  }
  New: pattern for  $n$ -ary predicate symbols
  pattern pred = [n : nat] {
    p : in → o
  }
}

```

Effect: Reject Ill-Patterned Declaration

```

pattern fun = [n : nat] {
  f : in → i
}
pattern pred = [n : nat] {
  p : in → o
}

```

Plain LF allows inadequate declarations in *FOF*-theories

```

foo  : (i → i) → i      higher-order
bar  : o → i             formulas in terms

```

LF with declaration patterns allows only

```

foo  : i → ... → i → i  function symbols
bar  : i → ... → i → o  predicate symbols

```

Example: Typed first-order logic

```

theory TF = {
  tType  : type                types
  tm     : tType → type      terms of a given type
  New: pattern for base types
  pattern baseType = {
    t : tType
  }
  New: pattern for typed  $n$ -ary function symbols
  pattern typedFun = [n : nat][A : tTypen][B : tType] {
    f : [tm Ai]i=1n → tm B
  }
  New: pattern for typed  $n$ -ary predicate symbols
  pattern typedPred = [n : nat][A : tTypen] {
    p : [tm Ai]i=1n → o
  }
}

```

Effect: Distinguish Languages

Plain and polymorphic logics only differ in declaration patterns

Plain *TF0*

```

pattern baseType = {
  t : tType
}
pattern typedFun = [n : nat] [A : tType^n] [B : tType] {
  f : [tm Ai]i=1n → tm B
}

```

Polymorphic *TF1*

```

pattern typeOp = [n : nat] {
  t : tType^n → tType
}
pattern polyFun = [m : nat] [n : nat]
  [A : (tType^m → tType)^n] [B : tType^m → tType] {
  f : Πa:tType^m [tm (Ai a)]i=1n → tm (B a)
}

```

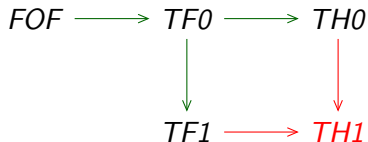
Translating and Combining Logics

Theory morphisms **translate**

untyped FOL to typed FOL $FOF \rightarrow TF0$

typed FOL to polymorphic FOL $TF0 \rightarrow TF1$

typed FOL to higher-order logic $TF0 \rightarrow TH0$



Combination yields: polymorphic higher-order logic $TH1$

- ▶ no informal specification yet syntax already used anyway
- ▶ formal specification obtained out of the box easier and more precise

Conclusion

- ▶ MathML/TPTP standardize syntax but semantics is informal
- ▶ Logical frameworks formalize semantics but are not practical
- ▶ Our approach
 1. use MMT with declaration patterns
 2. instantiate with LFS
 3. specify semantics of interchange languages
- ▶ Obtained specifications of all TPTP variants
 - concise, modular, fully formal
 - new variants by combining features

