# Diagram Operators in a Logical Framework

Navid Roux[1] and Florian Rabe[2]

Computer Science, FAU Erlangen-Nürnberg

LFMTP 2020: Session 1, Formalizing Logics



[1] https://orcid.org/0000-0002-8348-2441

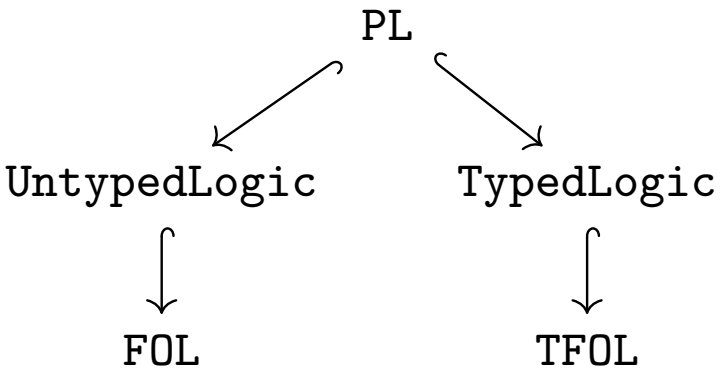
[2] https://orcid.org/0000-0003-3040-3655

# Motivation: Logics in Theory Graphs

Goal: Formalize logics in a reusable and modular way

- Use logical frameworks to formalize logics into theories e.g. LF
- Use inclusions to structure theories into a theory graph

Example:

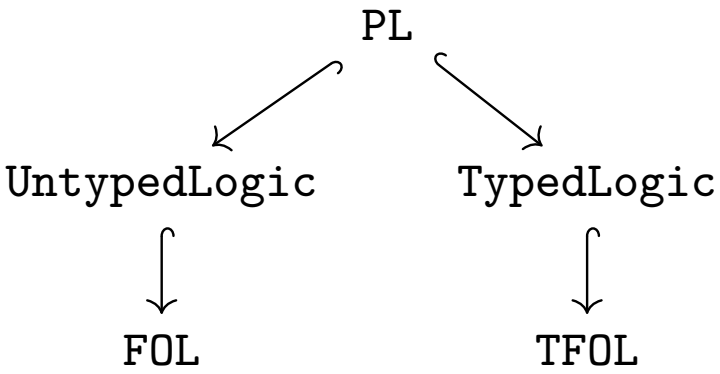


# Motivation: Logics in Theory Graphs

Goal: Formalize logics in a reusable and modular way

- Use logical frameworks to formalize logics into theories e.g. LF
- Use inclusions to structure theories into a theory graph

Example:



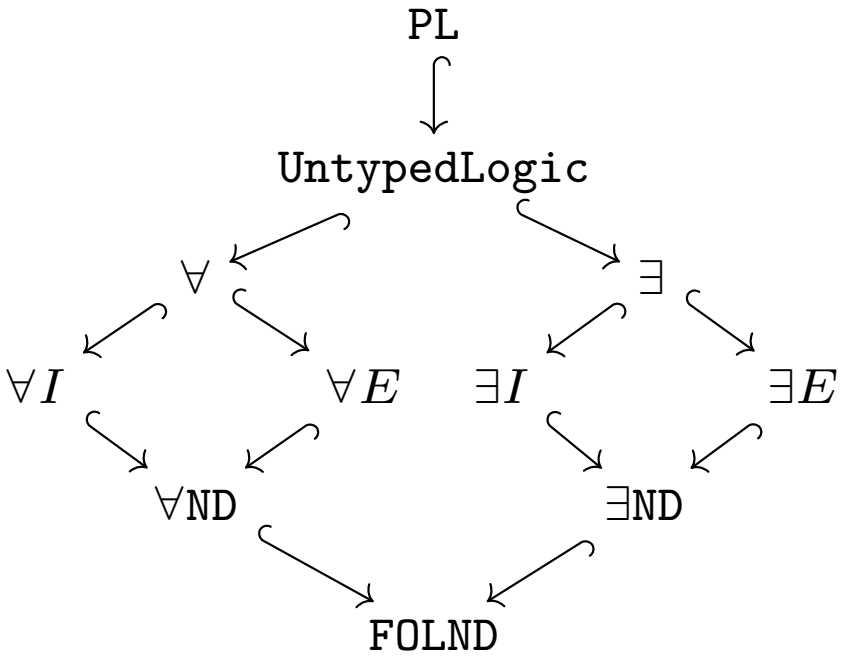

$$\texttt{PL} = \left\{ \begin{array}{ll} \texttt{prop} & : \texttt{type} \\ \vdash & : \texttt{prop} \to \texttt{type} \\ \neg & : \texttt{prop} \to \texttt{prop} \\ \wedge, \vee, \Rightarrow & : \texttt{prop} \to \texttt{prop} \to \texttt{prop} \end{array} \right\}$$

$$\texttt{UntypedLogic} = \left\{ \begin{array}{ll} \textbf{include PL} & \\ \texttt{tm} & : \texttt{type} \end{array} \right\}$$

$$\texttt{TypedLogic} = \left\{ \begin{array}{ll} \textbf{include PL} & \\ \texttt{tp} & : \texttt{type} \\ \texttt{tm} & : \texttt{tp} \to \texttt{type} \end{array} \right\}$$

# Example Problem

## Untyped variant



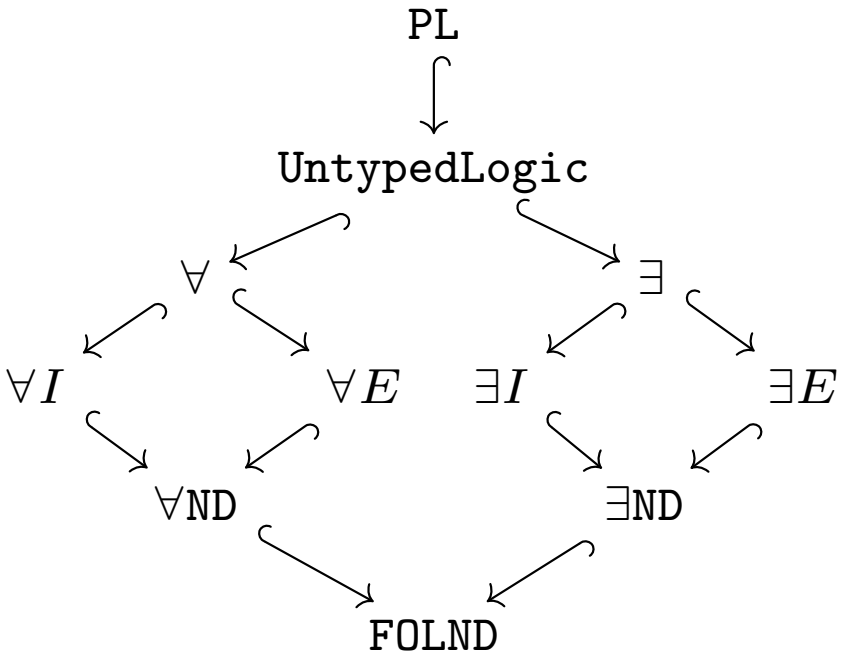

$\forall \quad : (\texttt{tm} \to \texttt{prop}) \to \texttt{prop}$

$\forall I \quad : \Pi P : \texttt{tm} \to \texttt{prop}.$

$(\Pi x : \texttt{tm}.\ \vdash P\, x) \to \vdash \forall P$

# Example Problem

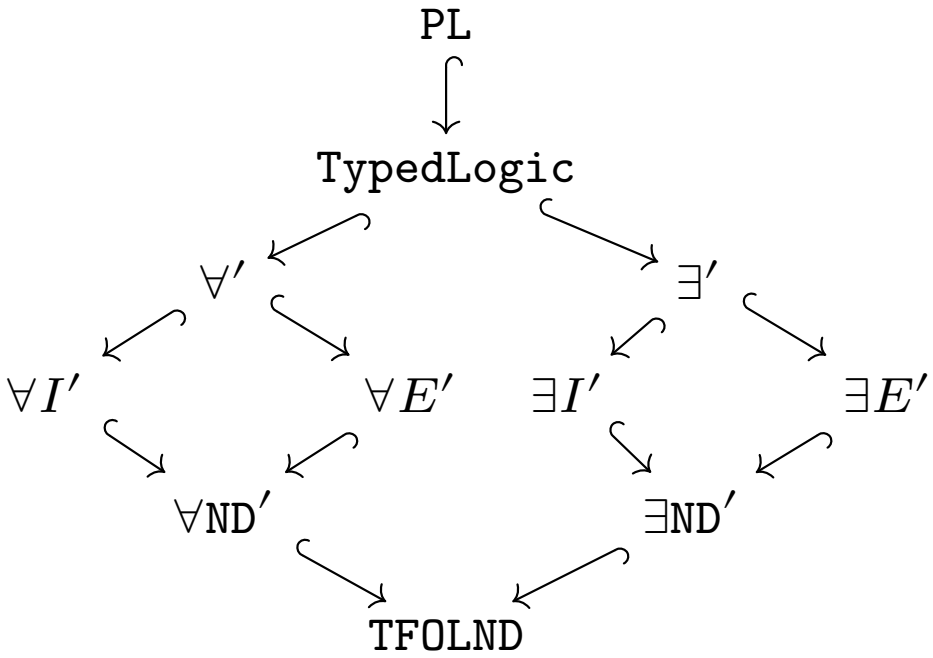
## Untyped variant




$\forall \quad : (\mathtt{tm} \to \mathtt{prop}) \to \mathtt{prop}$

$\forall I \quad : \Pi P \colon \mathtt{tm} \to \mathtt{prop}.$
$(\Pi x \colon \mathtt{tm}.\ \vdash P\, x) \to \vdash \forall P$

## Typed Variant

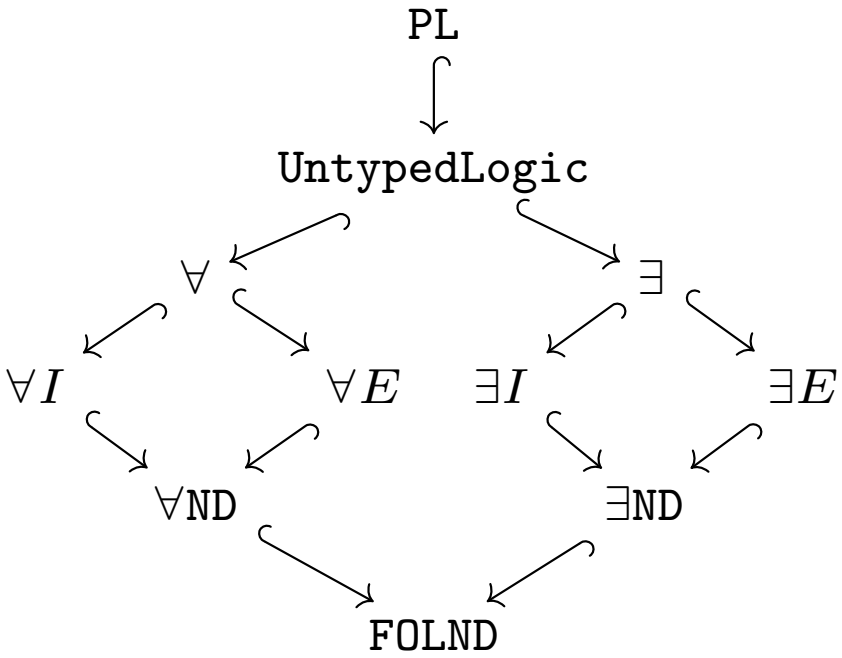



$\forall' \quad : \Pi\, S \colon \mathtt{tp}.\ (\mathtt{tm}\, S \to \mathtt{prop}) \to \mathtt{prop}$

$\forall I' \quad : \Pi\, S \colon \mathtt{tp}.\ \Pi P \colon \mathtt{tm}\, S \to \mathtt{prop}.$
$(\Pi x \colon \mathtt{tm}\, S.\ \vdash P\, x) \to \vdash \forall'\, S\, P$

# Example Problem

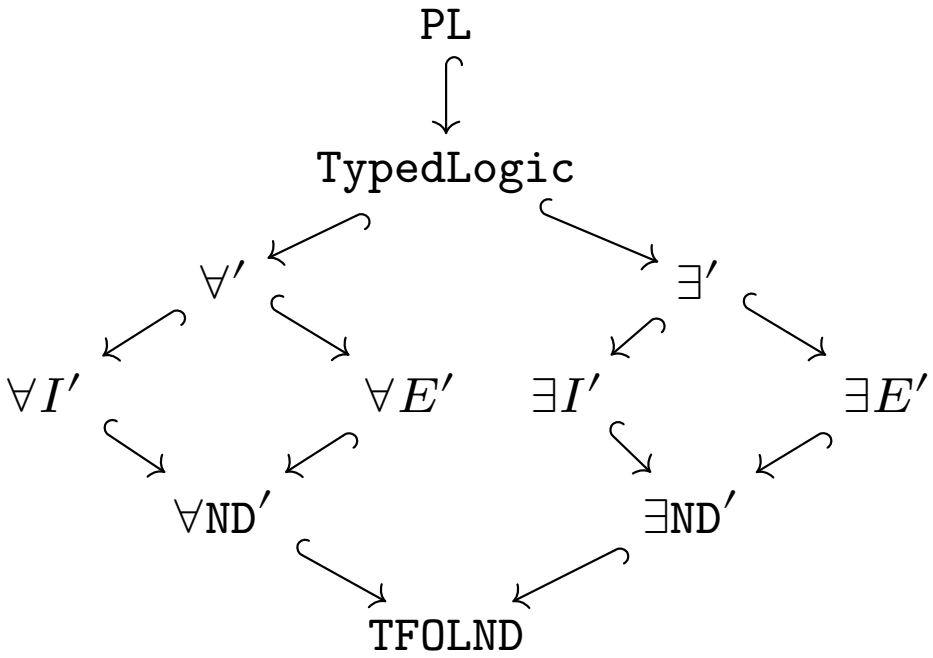
## Untyped variant




$\forall \quad : (\mathtt{tm} \to \mathtt{prop}) \to \mathtt{prop}$

$\forall I \quad : \Pi P \colon \mathtt{tm} \to \mathtt{prop}.$
$(\Pi x \colon \mathtt{tm}.\ \vdash P\, x) \to \vdash \forall P$

## Typed Variant




$\forall' \quad : \Pi\, S \colon \mathtt{tp}.\ (\mathtt{tm}\, S \to \mathtt{prop}) \to \mathtt{prop}$

$\forall I' \quad : \Pi\, S \colon \mathtt{tp}.\ \Pi P \colon \mathtt{tm}\, S \to \mathtt{prop}.$
$(\Pi x \colon \mathtt{tm}\, S.\ \vdash P\, x) \to \vdash \forall'\, S\, P$

Problem: syntactic redundancy that is not expressible in logical framework

# Inevitable Redundancies

Usually module systems good at

- reusing expressions

  definitions

- reusing theories

  inclusions

- transporting across compositional translations

  Coq functors, MMT morphisms

However no elegant methods for anything non-compositional

even if just minor syntactic changes

# Computing Theories

Goal: preserve diagrams, includes, well-typedness, names

Solution Space:

- Where/how does the computation happen?
- How much computation do we allow?

# Computing Theories

Goal: preserve diagrams, includes, well-typedness, names

Solution Space:

- Where/how does the computation happen?
  - in framework via novel abstraction mechanism

    ideal, but don't know how
  - in framework via reflection

    possible, but requires programming extension
  - outside via preprocessor/plugin

    our approach
- How much computation do we allow?

# Computing Theories

Goal: preserve diagrams, includes, well-typedness, names

Solution Space:

- Where/how does the computation happen?
  - in framework via novel abstraction mechanism

    ideal, but don't know how
  - in framework via reflection

    possible, but requires programming extension
  - outside via preprocessor/plugin

    our approach
- How much computation do we allow?
  - Turing-complete, arbitrary AST manipulation

    makes meta-theory difficult
  - **sweet spot?**

    **our goal**
  - purely compositional, e.g. pushouts

    not expressive enough

# MMT/LF Syntax

A module system over LF:

| | | | |
|---|---|---|---|
| $Diag$ | $::=$ | $Thy^*$ | structured diagrams |
| $Thy$ | $::=$ | $T = \{Decl^*\}$ | theories |
| $Decl$ | $::=$ | $c\colon A\,[= A] \mid \textbf{include}\ T$ | declarations |

# MMT/LF Syntax

A module system over LF:

| | | | |
|---|---|---|---|
| $Diag$ | $::=$ | $Thy^*$ | structured diagrams |
| $Thy$ | $::=$ | $T = \{Decl^*\}$ | theories |
| $Decl$ | $::=$ | $c\colon A\,[= A] \mid \textbf{include}\ T$ | declarations |
| $A$ | $::=$ | $\texttt{type} \mid c \mid x \mid A\ A \mid A \to A \mid$ $\lambda x\!:\!A.\ A \mid \Pi x\!:\!A.\ A$ | terms |

# MMT/LF Syntax

A module system over LF:

| | | | |
|---|---|---|---|
| $Diag$ | $::=$ | $(Thy \mid O(T^*))^*$ | structured diagrams |
| $Thy$ | $::=$ | $T = \{Decl^*\}$ | theories |
| $Decl$ | $::=$ | $c\colon A\,[= A] \mid \textbf{include}\ T$ | declarations |
| $A$ | $::=$ | $\texttt{type} \mid c \mid x \mid A\ A \mid A \to A \mid$ $\lambda x\!:\!A.\ A \mid \Pi x\!:\!A.\ A$ | terms |

where $O$ names a diagram operator

# MMT/LF Syntax

A module system over LF:

| | | | |
|---|---|---|---|
| $Diag$ | $::=$ | $(Thy \mid O(T^*))^*$ | structured diagrams |
| $Thy$ | $::=$ | $T = \{Decl^*\}$ | theories |
| $Decl$ | $::=$ | $c\colon A\,[= A] \mid \textbf{include}\ T$ | declarations |
| $A$ | $::=$ | $\texttt{type} \mid c \mid x \mid A\ A \mid A \to A \mid$ $\lambda x\!:\!A.\ A \mid \Pi x\!:\!A.\ A$ | terms |

where $O$ names a diagram operator

# Mmt/LF Syntax

A module system over LF:

| | | | |
|---|---|---|---|
| $Diag$ | $::=$ | $(Thy \mid O(T^*))^*$ | structured diagrams |
| $Thy$ | $::=$ | $T = \{Decl^*\}$ | theories |
| $Decl$ | $::=$ | $c\colon A\,[= A] \mid \textbf{include}\ T$ | declarations |
| $A$ | $::=$ | $\texttt{type} \mid c \mid x \mid A\ A \mid A \to A \mid$ $\lambda x\!:\!A.\ A \mid \Pi x\!:\!A.\ A$ | terms |

where $O$ names a diagram operator

End goal: specify class of operators on flat theories and lift

# Mmt/LF Semantics

Semantics given by flat theories

Interpretation given by flattening $-^\flat$ eliminating includes

- on diagrams:

$$\begin{aligned} .^\flat &:= \varnothing \\ (T = \{\Sigma\}, D)^\flat &:= (T = \{\Sigma^\flat\}, D^\flat) \end{aligned}$$

- on theories:

$$\begin{aligned} .^\flat &:= \varnothing \\ (c\colon A\,[= A], \Sigma)^\flat &:= \{c\colon A\,[= A]\} \cup \Sigma^\flat \\ (\mathbf{include}\ S, \Sigma)^\flat &:= S^\flat \cup \Sigma^\flat \end{aligned}$$

$\Rightarrow$ Results in plain LF theories

# Main Definition

## Definition (Linear Diagram Operator)

For theories $S$, $T$, we call a partial function

$$O\colon \text{flat theories} \to \text{flat theories}$$

**linear from $S$ to $T$** if

- it maps $S$-extensions to $T$-extensions, $O(S) = T$,
- declaration-wise,

$$O(S, \Sigma, c\colon A\,[= a]) = O(S, \Sigma), \Delta_{\Sigma}(c\colon A\,[= a])$$

for some function $\Delta\_(-)$.

# Main Definition

## Definition (Linear Diagram Operator)

For theories $S$, $T$, we call a partial function

$$O\colon \text{flat theories} \to \text{flat theories}$$

**linear from $S$ to $T$** if

- it maps $S$-extensions to $T$-extensions, $O(S) = T$,
- declaration-wise,

$$O(S, \Sigma, c\colon A\,[= a]) = O(S, \Sigma), \Delta_{\Sigma}(c\colon A\,[= a])$$

for some function $\Delta\_(-)$.

- $\Delta\_(-)$ may map a single declaration to multiple
- $\Delta\_(-)$ uniquely determined by $O$ upon existence

# Our Example as a Linear Operator

Situation: we identified the following redundancies:

$$\forall \quad : (\texttt{tm} \to \texttt{prop}) \to \texttt{prop}$$
$$\forall I \quad : \Pi P \colon \texttt{tm} \to \texttt{prop}.\ (\Pi x \colon \texttt{tm}.\ \vdash P\, x) \to \vdash \forall P$$

$$\forall' \quad : \Pi\, S \colon \texttt{tp}.\ (\texttt{tm}\, S \to \texttt{prop}) \to \texttt{prop}$$
$$\forall I' \quad : \Pi\, S \colon \texttt{tp}.\ \Pi P \colon \texttt{tm}\, S \to \texttt{prop}.\ (\Pi x \colon \texttt{tm}\, S.\ \vdash P\, x) \to \vdash \forall'\, S\, P$$

# Our Example as a Linear Operator

Situation: we identified the following redundancies:

$$\begin{array}{lll} \forall & : (\mathtt{tm} \to \mathtt{prop}) \to \mathtt{prop} & \\ \forall I & : \Pi P \colon \mathtt{tm} \to \mathtt{prop}. & \\ & \quad (\Pi x \colon \mathtt{tm}.\ \vdash P\, x) \to \vdash \forall P & \end{array} \xrightarrow{\Delta\_(-)} \begin{array}{lll} \forall' & : \Pi\, S \colon \mathtt{tp}.\ (\mathtt{tm}\, S \to \mathtt{prop}) \to \mathtt{prop} & \\ \forall I' & : \Pi\, S \colon \mathtt{tp}.\ \Pi P \colon \mathtt{tm}\, S \to \mathtt{prop}. & \\ & \quad (\Pi x \colon \mathtt{tm}\, S.\ \vdash P\, x) \to \vdash \forall'\, S\, P & \end{array}$$

# Our Example as a Linear Operator

Situation: we identified the following redundancies:

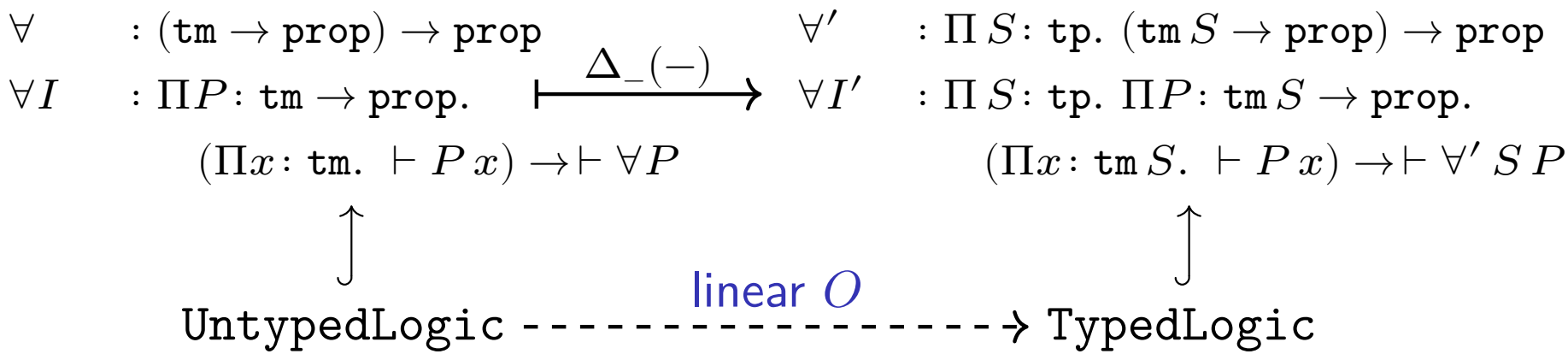
$$\begin{array}{lll} \forall & : (\mathtt{tm} \to \mathtt{prop}) \to \mathtt{prop} & \\ \forall I & : \Pi P \colon \mathtt{tm} \to \mathtt{prop}. & \\ & \quad (\Pi x \colon \mathtt{tm}.\ \vdash P\, x) \to \vdash \forall P & \end{array} \xmapsto{\Delta\_(-)} \begin{array}{lll} \forall' & : \Pi\, S \colon \mathtt{tp}.\ (\mathtt{tm}\, S \to \mathtt{prop}) \to \mathtt{prop} & \\ \forall I' & : \Pi\, S \colon \mathtt{tp}.\ \Pi P \colon \mathtt{tm}\, S \to \mathtt{prop}. & \\ & \quad (\Pi x \colon \mathtt{tm}\, S.\ \vdash P\, x) \to \vdash \forall'\, S\, P & \end{array}$$

Idea: $\Delta_\Sigma(c \colon A\,[= a])$ obtained by

1. adding $\Pi\, S \colon \mathtt{tp}$ to type, $\lambda\, S \colon \mathtt{tp}$ to definiens
2. replacing $\mathtt{tm}$ by $\mathtt{tm}\, S$
3. replacing any $d \in \Sigma$ by $d\, S$

# Our Example as a Linear Operator

Situation: we identified the following redundancies:




Idea: $\Delta_\Sigma(c : A\,[= a])$ obtained by

1. adding $\Pi\, S : \mathtt{tp}$ to type, $\lambda\, S : \mathtt{tp}$ to definiens
2. replacing $\mathtt{tm}$ by $\mathtt{tm}\, S$
3. replacing any $d \in \Sigma$ by $d\, S$

# Lifting

## Definition (Lifting)

Given a linear operator $O$ and a renaming function $O_r$, define its lifting $O(-)$ to diagrams...

- ...on diagrams $D$ element-wise
- ...on theories $T = \{\Sigma\}$ in $D$ by transforming them to $O_r(T) = \{O(\Sigma)\}$ where

$$
\begin{aligned}
O(\cdot) &= \cdot \\
O(\Sigma,\ c\colon A\,[= a]) &= O(\Sigma),\ \Delta_\Sigma(c\colon A\,[= a]) \\
O(\Sigma,\ \textbf{include}\ S) &= O(\Sigma),\ \textbf{include}\ O_r(S).
\end{aligned}
$$

# Semantics Preservation

## Theorem (Semantics Preservation)

*Let $D$ be a well-formed diagram and $O$ the lifting of a linear operator. Then*

1. *$O(D)$ is a well-formed diagram,*
2. *and lifting commutes with flattening:*

$$O(D^\flat) = O(D)^\flat.$$

# Implementation in the MMT System

- Ideally specify operators within LF, benefit from typechecking

  impossible in pure LF

- Practically use MMT's computation rules
  1. Load external computation rule...

```
theory Operators = {
    constant TYPIFY
    rule code.scala.typify
}
```

  2. ...that was given in underlying programming language...

```
object Typify extends LinearOperator {
    def rename(name: String): String = "Typed" + name

    def apply(sigma: List[Declaration], decl: Declaration): List[Declaration] = {
        // add "S: tp" binding
        // replace "tm" by "tm S"
        // replace "d" in sigma by "d S"
    }
}
```

  3. ...and use as macro.

```
fol := Diagram(∀,∀I,∀E,∀ND,∃,∃I,∃E,∃ND,FOLND)
diagram tfol := TYPIFY fol
```

# Our Running Example in Practice

```
namespace latin:/
fixmeta ur:?LF
theory UniversalQuantification =
  include ?UntypedLogic
  forall : (term → prop) → prop | # ∀ 1 | ## ∀ V1
↪ 2 |

theory UniversalQuantificationNDI =
  include ?UniversalQuantification
  forallI  : {P} ({x} ⊦ (P x)) → ⊦ ∀ P |

theory UniversalQuantificationNDE =
  include ?UniversalQuantification
  forallE  : {P,X} ⊦ ∀ P → ⊦ (P X) | # 2 forallE
↪ 3 | role ForwardRule |

theory UniversalQuantificationND =
  include ?UniversalQuantificationNDE
  include ?UniversalQuantificationNDI

theory ExistentialQuantification =
  include ?UntypedLogic
  exists : (term → prop) → prop | # ∃ 1 | ## ∃ V1
↪ 2 |

theory ExistentialQuantificationNDI =
  include ?ExistentialQuantification
  existsI  : {P,X} ⊦ P X → ⊦ ∃ P | # existsI 2 3 |

theory ExistentialQuantificationNDE =
  include ?ExistentialQuantification
  existsE  : {P,C} ⊦ ∃ P → ({x} ⊦ (P x) → ⊦ C)
↪ → ⊦ C | # 3 existsE 4 | role
↪ EliminationRule |

theory ExistentialQuantificationND =
  include ?ExistentialQuantificationNDI
  include ?ExistentialQuantificationNDE

theory IFOL =
  include ?IPL
  include ?UniversalQuantification
  include ?ExistentialQuantification

theory IFOLND =
  include ?IFOL
  include ?IPLND
  include ?UniversalQuantificationND
  include ?ExistentialQuantificationND

theory IFOLEQ =
  include ?IPL
  include ?Equality
  include ?UniversalQuantification
  include ?ExistentialQuantification

theory IFOLEQND =
  include ?IFOLEQ
  include ?IPLND
  include ?EqualityND
  include ?UniversalQuantificationND
  include ?ExistentialQuantificationND

theory FOL =
  include ?PL
  include ?IFOL

theory FOLND =
  include ?FOL
  include ?PLND
  include ?IFOLND

theory FOLEQ =
  include ?FOL
  include ?Equality

theory FOLEQND =
  include ?FOLEQ
  include ?FOLND
  include ?EqualityND

theory RelativizedUniversalQuantification =
  include ?SoftTypedTerms
  include ?FOLEQ
  rforall : tp → (term → prop) → prop | =
↪ [A,P] ∀[x] x:A ⇒ P x | # ∀' 1 2 |

theory RelativizedUniversalQuantificationND =
  include ?RelativizedUniversalQuantification
  include ?FOLEQND
  rforallI : {A,F} ({x} ⊦x:A → ⊦ F x) → ⊦ ∀' A
↪ F | = [A,F,P] forallI [x] impI [xa] P x xa |
  rforallE : {A,F} ⊦ ∀' A F → {x} ⊦x:A → ⊦ F
↪ x | = [A,F,p,x,xa] p forallE x impE xa |

theory RelativizedExistentialQuantification =
  include ?SoftTypedTerms
  include ?FOLEQ
  rexists : tp → (term → prop) → prop | =
↪ [A,P] ∃[x] x:A ∧ P x | # ∃' 1 2 |

theory RelativizedExistentialQuantificationND =
  include
↪ ?RelativizedExistentialQuantification |
  include ?FOLEQND |
  rexistsI : {A,F} {x} ⊦x:A → ⊦ F x → ⊦ ∃' A
↪ F | = [A,F,x,xa,p] existsI x andI xa p |
  rexistsE : {A,F,C} ⊦ ∃' A F → ({x} ⊦x:A → ⊦ F
↪ x → ⊦ C) → ⊦ C | = [A,F,C,p,P] p existsE
↪ [x,q] P x (q andEl) (q andEr) |

theory UniqueExistentialQuantification =
  include ?ExistentialQuantification
  include ?Equality
  existsUnique  : (term → prop) → prop | # ∃¹
↪ 1 |

theory UniqueExistentialQuantificationND =
  include ?UniqueExistentialQuantification
  existsUniqueI : {P} {x} ⊦ P x → ({y} ⊦ P y →
↪ ⊦ x ≐ y) → ⊦ ∃¹ P | # existsUniqueI 3 4 5 |
  existsUniqueE : {P,C} ⊦ ∃¹ P → ({x} ⊦ P x →
↪ ({y} ⊦ P y → ⊦ x ≐ y) → ⊦ C) → ⊦ C | # 4
↪ existsUniqueE 5 | |
```

```
namespace latin:/
fixmeta ur:?LF
theory UniversalQuantification =
  include ?UntypedLogic
  forall : (term → prop) → prop | # ∀ 1 | ## ∀ V1
↪ 2 |

theory UniversalQuantificationNDI =
  include ?UniversalQuantification
  forallI  : {P} ({x} ⊦ (P x)) → ⊦ ∀ P |

theory UniversalQuantificationNDE =
  include ?UniversalQuantification
  forallE  : {P,X} ⊦ ∀ P → ⊦ (P X) | # 2 forallE
↪ 3 | role ForwardRule |

theory UniversalQuantificationND =
  include ?UniversalQuantificationNDE
  include ?UniversalQuantificationNDI

theory ExistentialQuantification =
  include ?UntypedLogic
  exists : (term → prop) → prop | # ∃ 1 | ## ∃ V1
↪ 2 |
```

```
theory ExistentialQuantificationNDI =
  include ?ExistentialQuantification
  existsI  : {P,X} ⊦ P X → ⊦ ∃ P | # existsI 2 3 |

theory ExistentialQuantificationNDE =
  include ?ExistentialQuantification
  existsE  : {P,C} ⊦ ∃ P → ({x} ⊦ (P x) → ⊦ C)
↪ → ⊦ C | # 3 existsE 4 | role
↪ EliminationRule |

theory ExistentialQuantificationND =
  include ?ExistentialQuantificationNDI
  include ?ExistentialQuantificationNDE

theory IFOL =
  include ?IPL
  include ?UniversalQuantification
  include ?ExistentialQuantification

theory IFOLND =
  include ?IFOL
  include ?IPLND
  include ?UniversalQuantificationND
  include ?ExistentialQuantificationND

theory IFOLEQ =
  include ?IPL
  include ?Equality
  include ?UniversalQuantification
  include ?ExistentialQuantification

theory IFOLEQND =
  include ?IFOLEQ
  include ?IPLND
  include ?EqualityND
  include ?UniversalQuantificationND
  include ?ExistentialQuantificationND

theory FOL =
  include ?PL
  include ?IFOL

theory FOLND =
  include ?FOL
  include ?PLND
  include ?IFOLND

theory FOLEQ =
  include ?FOL
  include ?Equality
```

```
theory FOLEQND =
  include ?FOLEQ
  include ?FOLND
  include ?EqualityND

theory RelativizedUniversalQuantification =
  include ?SoftTypedTerms
  include ?FOLEQ
  rforall : tp → (term → prop) → prop | =
↪ [A,P] ∀[x] x:A ⇒ P x | # ∀' 1 2 |

theory RelativizedUniversalQuantificationND =
  include ?RelativizedUniversalQuantification
  include ?FOLEQND
  rforallI : {A,F} ({x} ⊦x:A → ⊦ F x) → ⊦ ∀' A
↪ F | = [A,F,P] forallI [x] impI [xa] P x xa |
  rforallE : {A,F} ⊦ ∀' A F → {x} ⊦x:A → ⊦ F
↪ x | = [A,F,p,x,xa] p forallE x impE xa |

theory RelativizedExistentialQuantification =
  include ?SoftTypedTerms
  include ?FOLEQ
  rexists : tp → (term → prop) → prop | =
↪ [A,P] ∃[x] x:A ∧ P x | # ∃' 1 2 |

theory RelativizedExistentialQuantificationND =
  include
↪ ?RelativizedExistentialQuantification |
  include ?FOLEQND |
  rexistsI : {A,F} {x} ⊦x:A → ⊦ F x → ⊦ ∃' A
↪ F | = [A,F,x,xa,p] existsI x andI xa p |
  rexistsE : {A,F,C} ⊦ ∃' A F → {x} ⊦x:A → ⊦ F
↪ x → ⊦ C) → ⊦ C | = [A,F,C,p,P] p existsE
↪ [x,q] P x (q andEl) (q andEr) |

theory UniqueExistentialQuantification =
  include ?ExistentialQuantification
  include ?Equality
  existsUnique  : (term → prop) → prop | # ∃¹
↪ 1 |

theory UniqueExistentialQuantificationND =
  include ?UniqueExistentialQuantification
  existsUniqueI : {P} {x} ⊦ P x → ({y} ⊦ P y →
↪ ⊦ x ≐ y) → ⊦ ∃¹ P | # existsUniqueI 3 4 5 |
  existsUniqueE : {P,C} ⊦ ∃¹ P → ({x} ⊦ P x →
↪ ({y} ⊦ P y → ⊦ x ≐ y) → ⊦ C) → ⊦ C | # 4
↪ existsUniqueE 5 | |
```

Before: manually kept in sync, in fact `TFOL` was out-of-sync

# Our Running Example in Practice

```
namespace latin:/
fixmeta ur:?LF
theory UniversalQuantification =
  include ?UntypedLogic
  forall : (term → prop) → prop | # ∀ 1 | ## ∀ V1
↪ 2
theory UniversalQuantificationNDI =
  include ?UniversalQuantification
  forallI  : {P} ({x} ⊢ (P x)) → ⊢ ∀ P
theory UniversalQuantificationNDE =
  include ?UniversalQuantification
  forallE  : {P,X} ⊢ ∀ P → ⊢ (P X) | # 2 forallE
↪ 3 | role ForwardRule
theory UniversalQuantificationND =
  include ?UniversalQuantificationNDE
  include ?UniversalQuantificationNDI
theory ExistentialQuantification =
  include ?UntypedLogic
  exists : (term → prop) → prop | # ∃ 1 | ## ∃ V1
↪ 2
theory ExistentialQuantificationNDI =
  include ?ExistentialQuantification
  existsI  : {P,X} ⊢ P X → ⊢ ∃ P | # existsI 2 3
theory ExistentialQuantificationNDE =
  include ?ExistentialQuantification
  existsE  : {P,C} ⊢ ∃ P → ({x} ⊢ (P x) → ⊢ C)
↪ → ⊢ C | # 3 existsE 4 | role
↪ EliminationRule
theory ExistentialQuantificationND =
  include ?ExistentialQuantificationNDI
  include ?ExistentialQuantificationNDE
theory IFOL =
  include ?IPL
```

```
  include ?UniversalQuantification
  include ?ExistentialQuantification
theory IFOLND =
  include ?IFOL
  include ?IPLND
  include ?UniversalQuantificationND
  include ?ExistentialQuantificationND
theory IFOLEQ =
  include ?IPL
  include ?Equality
  include ?UniversalQuantification
  include ?ExistentialQuantification
theory IFOLEQND =
  include ?IFOLEQ
  include ?IPLND
  include ?EqualityND
  include ?UniversalQuantificationND
  include ?ExistentialQuantificationND
theory FOL =
  include ?PL
  include ?IFOL
theory FOLND =
  include ?FOL
  include ?PLND
  include ?IFOLND
theory FOLEQ =
  include ?FOL
  include ?Equality
theory FOLEQND =
  include ?FOLEQ
  include ?FOLND
  include ?EqualityND
theory RelativizedUniversalQuantification =
  include ?SoftTypedTerms
```

```
  include ?FOLEQ
  rforall : tp → (term → prop) → prop | =
↪ [A,P] ∀[x] x:$A ⇒ P x | # ∀' 1 2
theory RelativizedUniversalQuantificationND =
  include ?RelativizedUniversalQuantification
  include ?FOLEQND
  rforallI : {A,F} ({x} ⊢x:$A → ⊢ F x) → ⊢ ∀' A
↪ F | = [A,F,P] forallI [x] impI [xa] P x xa
  rforallE : {A,F} ⊢ ∀' A F → {x} ⊢x:$A → ⊢ F
↪ x | = [A,F,p,x,xa] p forallE x impE xa
theory RelativizedExistentialQuantification =
  include ?SoftTypedTerms
  include ?FOLEQ
  rexists : tp → (term → prop) → prop | =
↪ [A,P] ∃[x] x:$A ∧ P x | # ∃' 1 2
theory RelativizedExistentialQuantificationND =
  include
↪ ?RelativizedExistentialQuantification
  include ?FOLEQND
  rexistsI : {A,F} {x} ⊢x:$A → ⊢ F x → ⊢ ∃' A
↪ F | = [A,F,x,xa,p] existsI x andI xa p
rexistsE : {A,F,C} ⊢ ∃' A F → ({x} ⊢x:$A → ⊢ F
↪ x → ⊢ C) → ⊢ C | = [A,F,C,p,P] p existsE
↪ [x,q] P x (q andEl) (q andEr)
theory UniqueExistentialQuantification =
  include ?ExistentialQuantification
  include ?Equality
  existsUnique  : (term → prop) → prop | # ∃¹
↪ 1
theory UniqueExistentialQuantificationND =
  include ?UniqueExistentialQuantification
  existsUniqueI : {P} {x} ⊢ P x → ({y} ⊢ P y →
↪ ⊢ x ≐ y) → ⊢ ∃¹ P | # existsUniqueI 3 4 5
```

```
  existsUniqueE : {P,C} ⊢ ∃¹ P → ({x} ⊢ P x →
↪ ({y} ⊢ P y → ⊢ x ≐ y) → ⊢ C) → ⊢ C | # 4
↪ existsUniqueE 5 |
```

```
namespace latin:/
fixmeta ur:?LF
theory TypedUniversalQuantification =
  include ?TypedLogic
  forall : {A} (tm A → prop) → prop | # ∀ 2
theory TypedUniversalQuantificationND =
  include ?TypedUniversalQuantification
  forallI  : {A,P} ({x: tm A} ⊢ P x) → ⊢ ∀ [x] P
↪ x
  forallE  : {A,P} ⊢ ∀ P → {x: tm A} ⊢ P x | # 3
↪ forallE 4 | role ForwardRule
theory TypedExistentialQuantification =
  include ?TypedLogic
  exists : {A} (tm A → prop) → prop | # ∃ 2
theory TypedExistentialQuantificationND =
  include ?TypedExistentialQuantification
  existsI  : {A,P} {x: tm A} ⊢ P x → ⊢ ∃ P | #
↪ existsI 3 4
  existsE  : {A,P,C} ⊢ ∃ P → ({x: tm A} ⊢ P x →
↪ ⊢ C) → ⊢ C | # 4 existsE 5 |
theory ISFOL =
  include ?TypedTerms
  include ?IPL
  include ?TypedUniversalQuantification
  include ?TypedExistentialQuantification
theory ISFOLND =
  include ?ISFOL
  include ?IPLND
```

```
  include ?TypedUniversalQuantificationND
  include ?TypedExistentialQuantificationND
theory SFOL =
  include ?PL
  include ?ISFOL
theory SFOLND =
  include ?SFOL
  include ?PLND
  include ?ISFOLND
theory SFOLEQ =
  include ?SFOL
  include ?TypedEquality
  notequal : {A} tm A → tm A → prop | =
↪ [A,x,y] ¬x≐y | # 2 ≠ 3 prec 30
theory SFOLEQND =
  include ?SFOLEQ
  include ?SFOLND
  include ?TypedEqualityND
theory TypedUniqueExistentialQuantification =
  include ?TypedExistentialQuantification
  include ?TypedEquality
  existsUnique  : {A} (tm A → prop) → prop | #
↪ ∃¹ 2
theory TypedUniqueExistentialQuantificationND =
  include
↪ ?TypedUniqueExistentialQuantification
  existsUniqueI : {A,P} {x: tm A} ⊢ P x → ({y} ⊢
↪ P y → ⊢ x ≐ y) → ⊢ ∃¹ P | # existsUniqueI
↪ 3 4 5 |
  existsUniqueE : {A,P,C} ⊢ ∃¹ P → ({x: tm A} ⊢
↪ P x → ({y} ⊢ P y → ⊢ x ≐ y) → ⊢ C) → ⊢
↪ C | # 4 existsUniqueE 5 |
```

Before: manually kept in sync, in fact `TFOL` was out-of-sync

# Our Running Example in Practice

```
namespace latin:/
fixmeta ur:?LF
theory UniversalQuantification =
  include ?UntypedLogic
  forall : (term → prop) → prop | # ∀ 1 | ## ∀ V1
↪ 2
theory UniversalQuantificationNDI =
  include ?UniversalQuantification
  forallI  : {P} ({x} ⊢ (P x)) → ⊢ ∀ P
theory UniversalQuantificationNDE =
  include ?UniversalQuantification
  forallE  : {P,X} ⊢ ∀ P → ⊢ (P X) | # 2 forallE
↪ 3 | role ForwardRule
theory UniversalQuantificationND =
  include ?UniversalQuantificationNDE
  include ?UniversalQuantificationNDI
theory ExistentialQuantification =
  include ?UntypedLogic
  exists : (term → prop) → prop | # ∃ 1 | ## ∃ V1
↪ 2
theory ExistentialQuantificationNDI =
  include ?ExistentialQuantification
  existsI  : {P,X} ⊢ P X → ⊢ ∃ P | # existsI 2 3
theory ExistentialQuantificationNDE =
  include ?ExistentialQuantification
  existsE  : {P,C} ⊢ ∃ P → ({x} ⊢ (P x) → ⊢ C)
↪ → ⊢ C | # 3 existsE 4 | role
↪ EliminationRule
theory ExistentialQuantificationND =
  include ?ExistentialQuantificationNDI
  include ?ExistentialQuantificationNDE
theory IFOL =
  include ?IPL
  include ?UniversalQuantification
  include ?ExistentialQuantification
theory IFOLND =
  include ?IFOL
  include ?IPLND
  include ?UniversalQuantificationND
  include ?ExistentialQuantificationND
theory IFOLEQ =
  include ?IPL
  include ?Equality
  include ?UniversalQuantification
  include ?ExistentialQuantification
theory IFOLEQND =
  include ?IFOLEQ
  include ?IPLND
  include ?EqualityND
  include ?UniversalQuantificationND
  include ?ExistentialQuantificationND
theory FOL =
  include ?PL
  include ?IFOL
theory FOLND =
  include ?FOL
  include ?PLND
  include ?IFOLND
theory FOLEQ =
  include ?FOL
  include ?Equality
theory FOLEQND =
  include ?FOLEQ
  include ?FOLND
  include ?EqualityND
theory RelativizedUniversalQuantification =
  include ?SoftTypedTerms
  include ?FOLEQ
  rforall : tp → (term → prop) → prop | =
↪ [A,P] ∀[x] x:A ⇒ P x | # ∀' 1 2
theory RelativizedUniversalQuantificationND =
  include ?RelativizedUniversalQuantification
  include ?FOLEQND
  rforallI : {A,F} ({x} ⊢x:A → ⊢ F x) → ⊢ ∀' A
↪ F | = [A,F,P] forallI [x] impI [xa] P x xa
  rforallE : {A,F} ⊢ ∀' A F → {x} ⊢x:A → ⊢ F
↪ x | = [A,F,p,x,xa] p forallE x impE xa
theory RelativizedExistentialQuantification =
  include ?SoftTypedTerms
  include ?FOLEQ
  rexists : tp → (term → prop) → prop | =
↪ [A,P] ∃[x] x:A ∧ P x | # ∃' 1 2
theory RelativizedExistentialQuantificationND =
  include
↪ ?RelativizedExistentialQuantification
  include ?FOLEQND
  rexistsI : {A,F} {x} ⊢x:A → ⊢ F x → ⊢ ∃' A
↪ F | = [A,F,x,xa,p] existsI x andI xa p
↪ rexistsE : {A,F,C} ⊢ ∃' A F → ({x} ⊢x:A → ⊢ F
↪ x → ⊢ C) → ⊢ C | = [A,F,C,p,P] p existsE
↪ [x,q] P x (q andEl) (q andEr)
theory UniqueExistentialQuantification =
  include ?ExistentialQuantification
  include ?Equality
  existsUnique  : (term → prop) → prop | # ∃¹
↪ 1
theory UniqueExistentialQuantificationND =
  include ?UniqueExistentialQuantification
  existsUniqueI : {P} {x} ⊢ P x → ({y} ⊢ P y →
↪ ⊢ x ≐ y) → ⊢ ∃¹ P | # existsUniqueI 3 4 5
↪ existsUniqueE : {P,C} ⊢ ∃¹ P → ({x} ⊢ P x →
↪ ({y} ⊢ P y → ⊢ x ≐ y) → ⊢ C) → ⊢ C | # 4
↪ existsUniqueE 5
```

Before: manually kept in sync, in fact `TFOL` was out-of-sync

After:

- simplified previous work
- enabled more work

previously above human tediousness threshold

# Type Theory Features in Church & Curry

$$\texttt{ChurchProd} = \left\{ \begin{array}{ll} \textbf{include}\ \texttt{TypedLogic} & \\ \_ \times \_ & : \texttt{tp} \to \texttt{tp} \to \texttt{tp} \\ (\_, \_) & : \Pi A\, B \colon \texttt{tp}.\ \ \texttt{tm}\, A \to \texttt{tm}\, B \to \texttt{tm}\, A \times B \end{array} \right\}$$

$$\texttt{CurryProd} = \left\{ \begin{array}{ll} \textbf{include}\ \texttt{UntypedLogic} & \\ \texttt{tp} & : \texttt{type} \\ \_ :: \_ & : \texttt{tm} \to \texttt{tp} \to \texttt{type} \\ \_ \times \_ & : \texttt{tp} \to \texttt{tp} \to \texttt{tp} \\ (\_, \_) & : \texttt{tm} \to \texttt{tm} \to \texttt{tm} \\ (\_, \_)^T & : \Pi A\, B \colon \texttt{tp}.\ \Pi a\, b \colon \texttt{tm}. \\ & \qquad a :: A \to b :: B \to (a, b) :: A \times B \end{array} \right\}$$

Non-compositional redundancy!

# Church To Curry in Practice

```
namespace latin:/
fixmeta ur:?LF
theory SimpleProductTypes =
  include ?Types
  simpprod : tp ⟶ tp ⟶ tp | # 1 × 2 prec 50

theory SimpleProducts =
  include ?SimpleProductTypes
  include ?TypedEquality
  simppair: {A,B} tm A ⟶ tm B ⟶ tm A × B | # 3 ,
↪ 4 prec 50
  simppi1 : {A,B} tm A × B ⟶ tm A | # 3 ₁ prec
↪ 60
  simppi2 : {A,B} tm A × B ⟶ tm B | # 3 ₂ prec
↪ 60

  compute1 : {A,B,a:tm A,b:tm B} ⊢ (a,b)₁ ≐
↪ a | role Simplify
  compute2 : {A,B,a:tm A,b:tm B} ⊢ (a,b)₂ ≐
↪ b | role Simplify

theory DependentProductTypes =
  include ?TypedTerms
  depprod : {A} (tm A ⟶ tp) ⟶ tp | # Σ 2 prec
↪ 40
  realize ?SimpleProductTypes
  simpprod = [A,B] Σ [x: tm A] B

theory DependentProducts =
  include ?DependentProductTypes
  include ?TypedEquality
  include ?Transport
  deppair: {A,B, a: tm A} tm B a ⟶ tm Σ B | # 3 ,
↪ 4 prec 50

theory SimpleProductsExpand =
  include ?SimpleProducts
  expand: {A,B,u:tm A × B} ⊢ (u₁,u₂) ≐ u

theory DependentProductsExpand =
  include ?DependentProducts
  expand: {A,B, u:tm Σ[x:tm A] B x} ⊢ (u₁,u₂) ≐
↪ u

theory SimpleProductsExtensionality =
  include ?SimpleProducts
  exten: {A,B,u,v:tm A × B} ⊢ u₁≐v₁ ⟶ ⊢ u₂≐v₂ ⟶
↪ ⊢ u≐v

theory DependentProductsExtensionality =
  include ?DependentProducts
  exten: {A,B,u,v:tm Σ[x:tm A] B x} {p: ⊢ u₁≐v₁}
↪ ⊢ u₂⟆(p congTp B)≐v₂ ⟶ ⊢ u≐v

theory SimpleFunctionTypes =
  include ?Types
  simpfun : tp ⟶ tp ⟶ tp | # 1 %R→ 2 prec 50
  test = [a:tp] a → a → a

theory DependentFunctionTypes =
  include ?TypedTerms
  depfun : {A} (tm A ⟶ tp) ⟶ tp | # Π 2 prec 40
  realize ?SimpleFunctionTypes
  simpfun = [A,B] Π [x: tm A] B

theory SimpleFunctions =
  include ?SimpleFunctionTypes
  include ?TypedEquality
  simplambda: {A,B} (tm A ⟶ tm B) ⟶ tm A → B | #
↪ λ 3 prec 40
  simpapply: {A,B} tm A → B ⟶ tm A ⟶ tm B | # 3
↪ @ 4 prec 50
  simpbeta : {A,B,F: tm A ⟶ tm B, X} ⊢ (λ F) @ X
↪ ≐ F X

theory DependentFunctions =
  include ?DependentFunctionTypes
  include ?TypedEquality
  deplambda: {A,B} ({x: tm A} tm B x) ⟶ tm Π
↪ B | # λ 3 prec 40
  depapply: {A,B} tm Π B ⟶ {x: tm A} tm B x | #
↪ 3 @ 4 prec 50
  depbeta : {A,B,F:{x:tm A} tm B x, X} ⊢ (λ F) @
↪ X ≐ F X

theory SimpleFunctionsEta =
  include ?SimpleFunctions
  eta: {A,B,F:tm A → B} ⊢ (λ[x] F @ x) ≐ F

theory SimpleFunctionsExtensionality =
  include ?SimpleFunctions
  exten: {A,B,F,G:tm A → B} ({x} ⊢ F@x ≐ G@x) ⟶
↪ ⊢ F≐G
```

```
namespace latin:/
fixmeta ur:?LF
theory SimpleProductTypes =
  include ?Types
  simpprod : tp ⟶ tp ⟶ tp | # 1 × 2 prec 50

theory SimpleProducts =
  include ?SimpleProductTypes
  include ?TypedEquality
  simppair: {A,B} tm A ⟶ tm B ⟶ tm A × B | # 3 ,
↪ 4 prec 50
  simppi1 : {A,B} tm A × B ⟶ tm A | # 3 ₁ prec
↪ 60
  simppi2 : {A,B} tm A × B ⟶ tm B | # 3 ₂ prec
↪ 60

  compute1 : {A,B,a:tm A,b:tm B} ⊢ (a,b)₁ ≐
↪ a | role Simplify
  compute2 : {A,B,a:tm A,b:tm B} ⊢ (a,b)₂ ≐
↪ b | role Simplify

theory DependentProductTypes =
  include ?TypedTerms
  depprod : {A} (tm A ⟶ tp) ⟶ tp | # Σ 2 prec
↪ 40
  realize ?SimpleProductTypes
  simpprod = [A,B] Σ [x: tm A] B

theory DependentProducts =
  include ?DependentProductTypes
  include ?TypedEquality
  include ?Transport
  deppair: {A,B, a: tm A} tm B a ⟶ tm Σ B | # 3 ,
↪ 4 prec 50

theory SimpleProductsExpand =
  include ?SimpleProducts
  expand: {A,B,u:tm A × B} ⊢ (u₁,u₂) ≐ u

theory DependentProductsExpand =
  include ?DependentProducts
  expand: {A,B, u:tm Σ[x:tm A] B x} ⊢ (u₁,u₂) ≐
↪ u

theory SimpleProductsExtensionality =
  include ?SimpleProducts
  exten: {A,B,u,v:tm A × B} ⊢ u₁≐v₁ ⟶ ⊢ u₂≐v₂ ⟶
↪ ⊢ u≐v

theory DependentProductsExtensionality =
  include ?DependentProducts
  exten: {A,B,u,v:tm Σ[x:tm A] B x} {p: ⊢ u₁≐v₁}
↪ ⊢ u₂⟆(p congTp B)≐v₂ ⟶ ⊢ u≐v

theory SimpleFunctionTypes =
  include ?Types
  simpfun : tp ⟶ tp ⟶ tp | # 1 %R→ 2 prec 50
  test = [a:tp] a → a → a

theory DependentFunctionTypes =
  include ?TypedTerms
  depfun : {A} (tm A ⟶ tp) ⟶ tp | # Π 2 prec 40
  realize ?SimpleFunctionTypes
  simpfun = [A,B] Π [x: tm A] B

theory SimpleFunctions =
  include ?SimpleFunctionTypes
  include ?TypedEquality
  simplambda: {A,B} (tm A ⟶ tm B) ⟶ tm A → B | #
↪ λ 3 prec 40
  simpapply: {A,B} tm A → B ⟶ tm A ⟶ tm B | # 3
↪ @ 4 prec 50
  simpbeta : {A,B,F: tm A ⟶ tm B, X} ⊢ (λ F) @ X
↪ ≐ F X

theory DependentFunctions =
  include ?DependentFunctionTypes
  include ?TypedEquality
  deplambda: {A,B} ({x: tm A} tm B x) ⟶ tm Π
↪ B | # λ 3 prec 40
  depapply: {A,B} tm Π B ⟶ {x: tm A} tm B x | #
↪ 3 @ 4 prec 50
  depbeta : {A,B,F:{x:tm A} tm B x, X} ⊢ (λ F) @
↪ X ≐ F X

theory SimpleFunctionsEta =
  include ?SimpleFunctions
  eta: {A,B,F:tm A → B} ⊢ (λ[x] F @ x) ≐ F

theory SimpleFunctionsExtensionality =
  include ?SimpleFunctions
  exten: {A,B,F,G:tm A → B} ({x} ⊢ F@x ≐ G@x) ⟶
↪ ⊢ F≐G
```

Before: again manually bookkept and out-of-sync

# Church To Curry in Practice

```
namespace latin:/
fixmeta ur:?LF
theory SimpleProductTypes =
  include ?Types
  simpprod : tp → tp → tp | # 1 × 2 prec 50

theory SimpleProducts =
  include ?SimpleProductTypes
  include ?TypedEquality
  simppair: {A,B} tm A → tm B → tm A × B | # 3 ,
↪ 4 prec 50
  simppi1 : {A,B} tm A × B → tm A | # 3 ₁ prec
↪ 60
  simppi2 : {A,B} tm A × B → tm B | # 3 ₂ prec
↪ 60

  compute1 : {A,B,a:tm A,b:tm B} ⊢ (a,b)₁ ≐
↪ a | role Simplify
  compute2 : {A,B,a:tm A,b:tm B} ⊢ (a,b)₂ ≐
↪ b | role Simplify

theory DependentProductTypes =
  include ?TypedTerms
  depprod : {A} (tm A → tp) → tp | # Σ 2 prec
↪ 40
  realize ?SimpleProductTypes
  simpprod = [A,B] Σ [x: tm A] B

theory DependentProducts =
  include ?DependentProductTypes
  include ?TypedEquality
  include ?Transport
  deppair: {A,B, a: tm A} tm B a → tm Σ B | # 3 ,
↪ 4 prec 50
```

```
theory SimpleProductsExpand =
  include ?SimpleProducts
  expand: {A,B,u:tm A × B} ⊢ (u₁,u₂) ≐ u

theory DependentProductsExpand =
  include ?DependentProducts
  expand: {A,B, u:tm Σ[x:tm A] B x} ⊢ (u₁,u₂) ≐
↪ u

theory SimpleProductsExtensionality =
  include ?SimpleProducts
  exten: {A,B,u,v:tm A × B} ⊢ u₁≐v₁ → ⊢ u₂≐v₂ →
↪ ⊢ u≐v

theory DependentProductsExtensionality =
  include ?DependentProducts
  exten: {A,B,u,v:tm Σ[x:tm A] B x} {p: ⊢ u₁≐v₁}
↪ ⊢ u₂⟅(p congTp B)≐v₂ → ⊢ u≐v

theory SimpleFunctionTypes =
  include ?Types
  simpfun : tp → tp → tp | # 1 %R→ 2 prec 50
  test = [a:tp] a → a → a

theory DependentFunctionTypes =
  include ?TypedTerms
  depfun : {A} (tm A → tp) → tp | # Π 2 prec 40
  realize ?SimpleFunctionTypes
  simpfun = [A,B] Π [x: tm A] B

theory SimpleFunctions =
  include ?SimpleFunctionTypes
  include ?TypedEquality
  simplambda: {A,B} (tm A → tm B) → tm A → B | #
↪ λ 3 prec 40
```

```
  simpapply: {A,B} tm A → B → tm A → tm B | # 3
↪ @ 4 prec 50
  simpbeta : {A,B,F: tm A → tm B, X} ⊢ (λ F) @ X
↪ ≐ F X

theory DependentFunctions =
  include ?DependentFunctionTypes
  include ?TypedEquality
  deplambda: {A,B} ({x: tm A} tm B x) → tm Π
↪ B | # λ 3 prec 40
  depapply: {A,B} tm Π B → {x: tm A} tm B x | #
↪ 3 @ 4 prec 50
  depbeta : {A,B,F:{x:tm A} tm B x, X} ⊢ (λ F) @
↪ X ≐ F X

theory SimpleFunctionsEta =
  include ?SimpleFunctions
  eta: {A,B,F:tm A → B} ⊢ (λ[x] F @ x) ≐ F

theory SimpleFunctionsExtensionality =
  include ?SimpleFunctions
  exten: {A,B,F,G:tm A → B} ({x} ⊢ F@x ≐ G@x) →
↪ ⊢ F≐G
```

```
namespace latin:/
fixmeta ur:?LF
theory SoftTypedProducts =
  include ?Terms
  include ?SoftTypedEquality
  pair: term → term → term | # 1 , 2 prec 50
  pi1 : term → term | # 1 ₁ prec 60
  pi2 : term → term | # 1 ₂ prec 60
  compute1 : {a,b} ⊢ (a,b)₁ ≐ a | role Simplify
  compute2 : {a,b} ⊢ (a,b)₂ ≐ b | role Simplify

theory SoftTypedSimpleProducts =
  include ?SoftTypedTerms
  include ?SoftTypedProducts
  include ?SimpleProductTypes
  fun_typing : {A,B,a,b} ⊢ a⁝A → ⊢ b⁝B → ⊢
↪ (a,b) ⁝ A × B

theory SoftDependentProductTypes =
  include ?SoftTypedTerms
  softprod : tp → (term → tp) → tp | # Σ 1 2
↪ prec 40

theory SoftTypedDependentProducts =
  include ?SoftTypedTerms
  include ?SoftTypedProducts
  include ?SoftDependentProductTypes
  fun_typing : {A,B,a,b} ⊢ a⁝A → ⊢ b⁝(B a) → ⊢
↪ (a,b) ⁝ Σ A B

theory SoftTypedProductsExpand =
  include ?SoftTypedProducts
  expand: {u} ⊢ (u₁,u₂) ≐ u

theory SoftTypedProductsExtensionality =
  include ?SoftTypedProducts
  exten: {u,v} ⊢ u₁ ≐ v₁ → ⊢ u₂≐v₂ → ⊢ u≐v

theory SoftDependentFunctionTypes =
  include ?SoftTypedTerms
```

```
  softfun : tp → (term → tp) → tp | # Π 1 2
↪ prec 40

theory SoftTypedFunctions =
  include ?Terms
  include ?SoftTypedEquality
  lambda: (term → term) → term | # λ 1 prec 40
  apply : term → term → term | # 1 @ 2 prec 50
  beta  : {F,X} ⊢ (λ F) @ X ≐ F X

theory SoftTypedSimpleFunctions =
  include ?SoftTypedTerms
  include ?SoftTypedFunctions
  include ?SimpleFunctionTypes
  fun_typing : {A,B,F} ({x} ⊢ x⁝A → ⊢ (F x)⁝B)
↪ → ⊢ λ F ⁝ A → B

theory SoftTypedDependentFunctions =
  include ?SoftTypedTerms
  include ?SoftTypedFunctions
  include ?SoftDependentFunctionTypes
  fun_typing : {A,B,F} ({x} ⊢ x⁝A → ⊢ (F x)⁝(B
↪ x)) → ⊢ λ F ⁝ Π A B

theory SoftTypedFunctionsEta =
  include ?SoftTypedFunctions
  eta: {F} ⊢ (λ[x] F @ x) ≐ F

theory SofttypedFunctionsExtensionality =
  include ?SoftTypedFunctions
  eta: {F,G} ({x} ⊢ F@x ≐ G@x) → ⊢ F≐G
```

Before: again manually bookkept and out-of-sync

# Church To Curry in Practice

```
namespace latin:/
fixmeta ur:?LF
theory SimpleProductTypes =
  include ?Types
  simpprod : tp → tp → tp | # 1 × 2 prec 50

theory SimpleProducts =
  include ?SimpleProductTypes
  include ?TypedEquality
  simppair: {A,B} tm A → tm B → tm A × B | # 3 ,
↪ 4 prec 50
  simppi1 : {A,B} tm A × B → tm A | # 3 ₁ prec
↪ 60
  simppi2 : {A,B} tm A × B → tm B | # 3 ₂ prec
↪ 60

  compute1 : {A,B,a:tm A,b:tm B} ⊢ (a,b)₁ ≐
↪ a | role Simplify
  compute2 : {A,B,a:tm A,b:tm B} ⊢ (a,b)₂ ≐
↪ b | role Simplify

theory DependentProductTypes =
  include ?TypedTerms
  depprod : {A} (tm A → tp) → tp | # Σ 2 prec
↪ 40
  realize ?SimpleProductTypes
  simpprod = [A,B] Σ [x: tm A] B

theory DependentProducts =
  include ?DependentProductTypes
  include ?TypedEquality
  include ?Transport
  deppair: {A,B, a: tm A} tm B a → tm Σ B | # 3 ,
↪ 4 prec 50
```

```
theory SimpleProductsExpand =
  include ?SimpleProducts
  expand: {A,B,u:tm A × B} ⊢ (u₁,u₂) ≐ u

theory DependentProductsExpand =
  include ?DependentProducts
  expand: {A,B, u:tm Σ[x:tm A] B x} ⊢ (u₁,u₂) ≐
↪ u

theory SimpleProductsExtensionality =
  include ?SimpleProducts
  exten: {A,B,u,v:tm A × B} ⊢ u₁≐v₁ → ⊢ u₂≐v₂ →
↪ ⊢ u≐v

theory DependentProductsExtensionality =
  include ?DependentProducts
  exten: {A,B,u,v:tm Σ[x:tm A] B x} {p: ⊢ u₁≐v₁}
↪ ⊢ u₂↑(p congTp B)≐v₂ → ⊢ u≐v

theory SimpleFunctionTypes =
  include ?Types
  simpfun : tp → tp → tp | # 1 %R→ 2 prec 50
  test = [a:tp] a → a → a

theory DependentFunctionTypes =
  include ?TypedTerms
  depfun : {A} (tm A → tp) → tp | # Π 2 prec 40
  realize ?SimpleFunctionTypes
  simpfun = [A,B] Π [x: tm A] B

theory SimpleFunctions =
  include ?SimpleFunctionTypes
  include ?TypedEquality
  simplambda: {A,B} (tm A → tm B) → tm A → B | #
↪ λ 3 prec 40
```

```
  simpapply: {A,B} tm A → B → tm A → tm B | # 3
↪ @ 4 prec 50
  simpbeta : {A,B,F: tm A → tm B, X} ⊢ (λ F) @ X
↪ ≐ F X

theory DependentFunctions =
  include ?DependentFunctionTypes
  include ?TypedEquality
  deplambda: {A,B} ({x: tm A} tm B x) → tm Π
↪ B | # λ 3 prec 40
  depapply: {A,B} tm Π B → {x: tm A} tm B x  | #
↪ 3 @ 4 prec 50
  depbeta : {A,B,F:{x:tm A} tm B x, X} ⊢ (λ F) @
↪ X ≐ F X

theory SimpleFunctionsEta =
  include ?SimpleFunctions
  eta: {A,B,F:tm A → B} ⊢ (λ[x] F @ x) ≐ F

theory SimpleFunctionsExtensionality =
  include ?SimpleFunctions
  exten: {A,B,F,G:tm A → B} ({x} ⊢ F@x ≐ G@x) →
↪ ⊢ F≐G
```

Before: again manually bookkept and out-of-sync

After: consistent knowledge

# More Examples

Generalizations:

- variants of TYPIFY
- transforming models into Kripke models

Logic Translations: e.g. modal logic with axioms in Sahlqvist form to FOL

first talk today

Elimination of...

- ...lambdas when translating HOL to FOL
- ...subtyping when translating between type theories
- ...in general of features occurring in non-compositional logic translations

# More Examples

Generalizations:

- variants of TYPIFY
- transforming models into Kripke models

Logic Translations: e.g. modal logic with axioms in Sahlqvist form to FOL

first talk today

Elimination of...

- ...lambdas when translating HOL to FOL
- ...subtyping when translating between type theories
- ...in general of features occurring in non-compositional logic translations

Universal Algebra: compute generic constructions like

- homomorphisms — e.g. `Group` $\mapsto$ `GroupHom`
- substructures and more

# Conclusion: Foster Modularity by Meta-Programming

## Summary

- Diagram operators serve as meta-programming facilities
  - decreasing redundancy
  - increasing coherence across related formalizations
- Linear operators are tradeoff between unrestricted computation and compositionality
- Lifting ensures scalability to large diagrams
- Many examples in logic translations

## Future Work

- implementation of more operators
- feasibility of static typechecking of operators

# Further Pointers I

## to our systems and libraries

- Extended abstract for these slides: [RR20]
- OMDoc/Mmt Language and Representation: [Rab17; RM18]
- Mmt System: [MR19; SR19; Rab13]
- Our archive for case studies: available at [LATIN]; description of previous version at [Cod+11]


[Cod+11] Mihai Codescu et al. "Project Abstract: Logic Atlas and Integrator (LATIN)". In: *Intelligent Computer Mathematics*. Ed. by James Davenport et al. LNAI 6824. Springer Verlag, 2011, pp. 289–291. ISBN: 978-3-642-22672-4. URL: https://kwarc.info/people/frabe/Research/CHKMR_latinabs_11.pdf.

[LATIN] *LATIN2 – Logic Atlas Version 2*. URL: https://gl.mathhub.info/MMT/LATIN2 (visited on 06/02/2017).

[MR19] Dennis Müller and Florian Rabe. "Rapid Prototyping Formal Systems in MMT: 5 Case Studies". In: *LFMTP 2019*. Electronic Proceedings in Theoretical Computer Science (EPTCS), 2019. URL: https://kwarc.info/people/frabe/Research/MR_prototyping_19.pdf.

# Further Pointers II

to our systems and libraries


[Rab13] Florian Rabe. "The MMT API: A Generic MKM System". In: *Intelligent Computer Mathematics*. Ed. by Jacques Carette et al. Lecture Notes in Computer Science 7961. Springer, 2013, pp. 339–343. ISBN: 978-3-642-39319-8. DOI: 10.1007/978-3-642-39320-4.

[Rab17] Florian Rabe. "How to Identify, Translate, and Combine Logics?" In: *Journal of Logic and Computation* 27.6 (2017), pp. 1753–1798.

[RM18] Florian Rabe and Dennis Müller. "Structuring Theories with Implicit Morphisms". In: *24th International Workshop on Algebraic Development Techniques 2018*. 2018. URL: https://kwarc.info/people/frabe/Research/RM_implicit_18.pdf.

[RR20] Navid Roux and Florian Rabe. "Diagram Operators in a Logical Framework". Extended abstract. 2020. URL: https://lfmtp.org/workshops/2020/inc/papers/LFMTP_2020_paper_9.pdf.

[SR19] Yasmine Sharoda and Florian Rabe. "Diagram Operators in MMT". In: *Intelligent Computer Mathematics*. Ed. by Cezary Kaliszyck et al. LNAI 11617. Springer, 2019, pp. 211–226. DOI: 10.1007/978-3-030-23250-4.