Functorial Diagram Operators

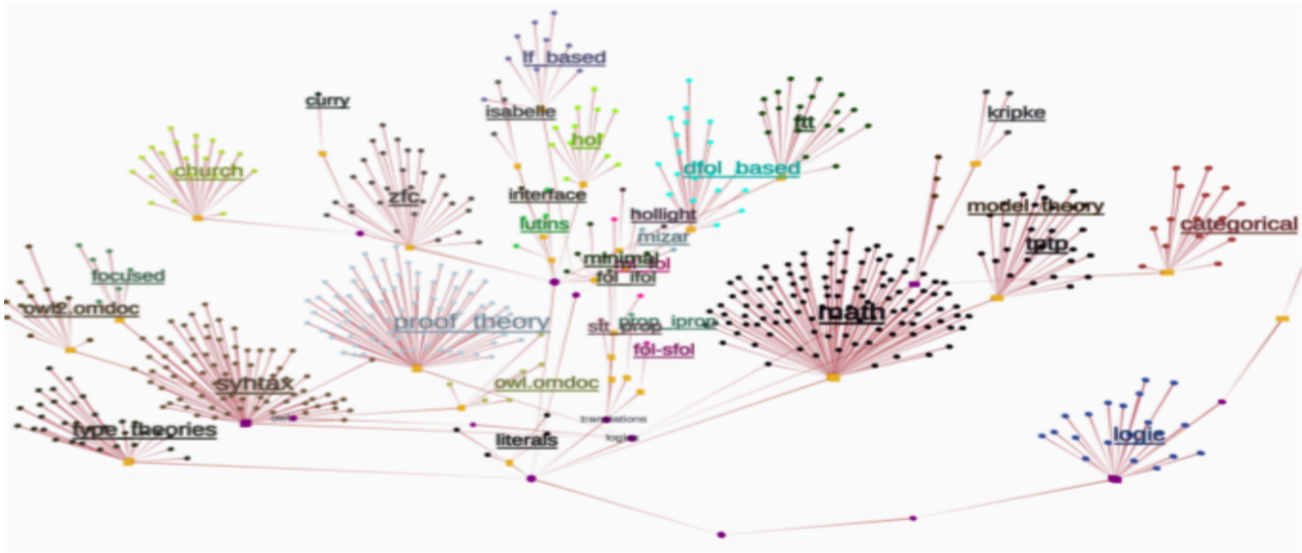
Navid Roux[1] and Florian Rabe[2]

General Stream track at WADT 2020

Computer Science, FAU Erlangen-Nürnberg

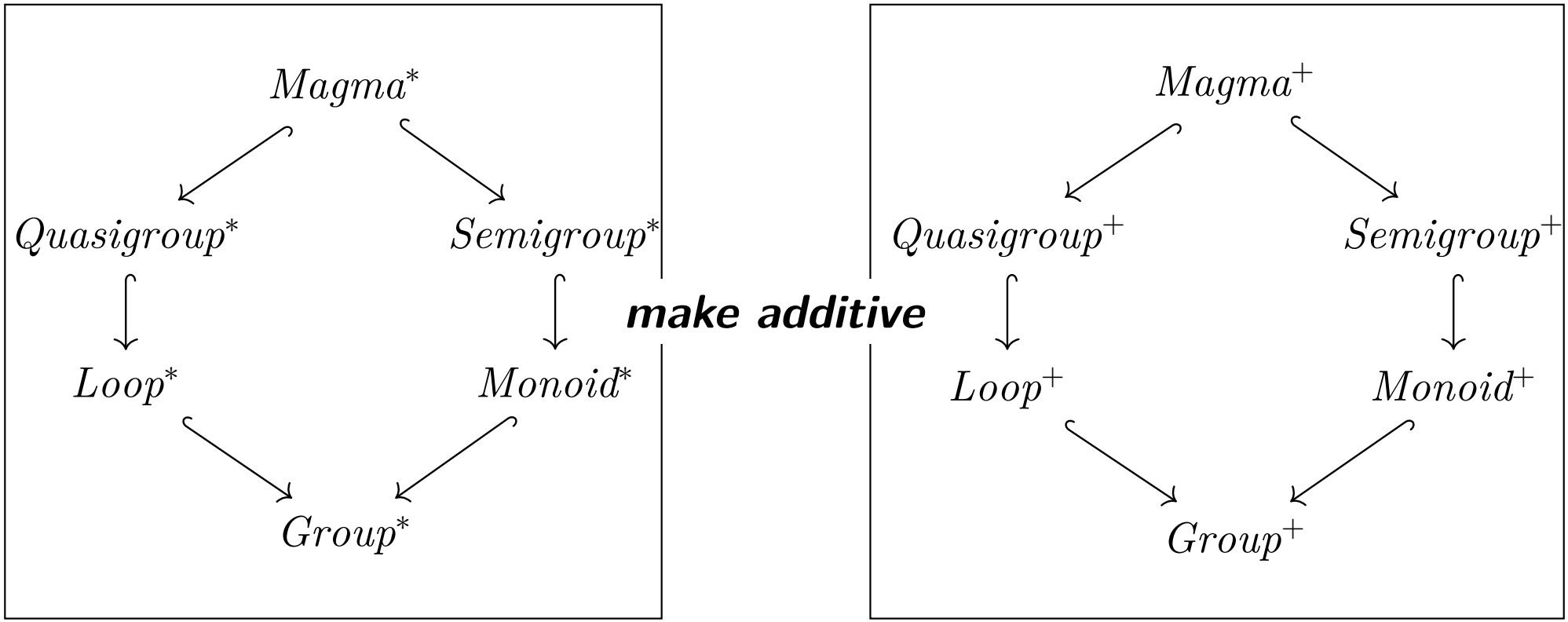
[1] https://orcid.org/0000-0002-8348-2441
[2] https://orcid.org/0000-0003-3040-3655

Modularity in Theory Graphs

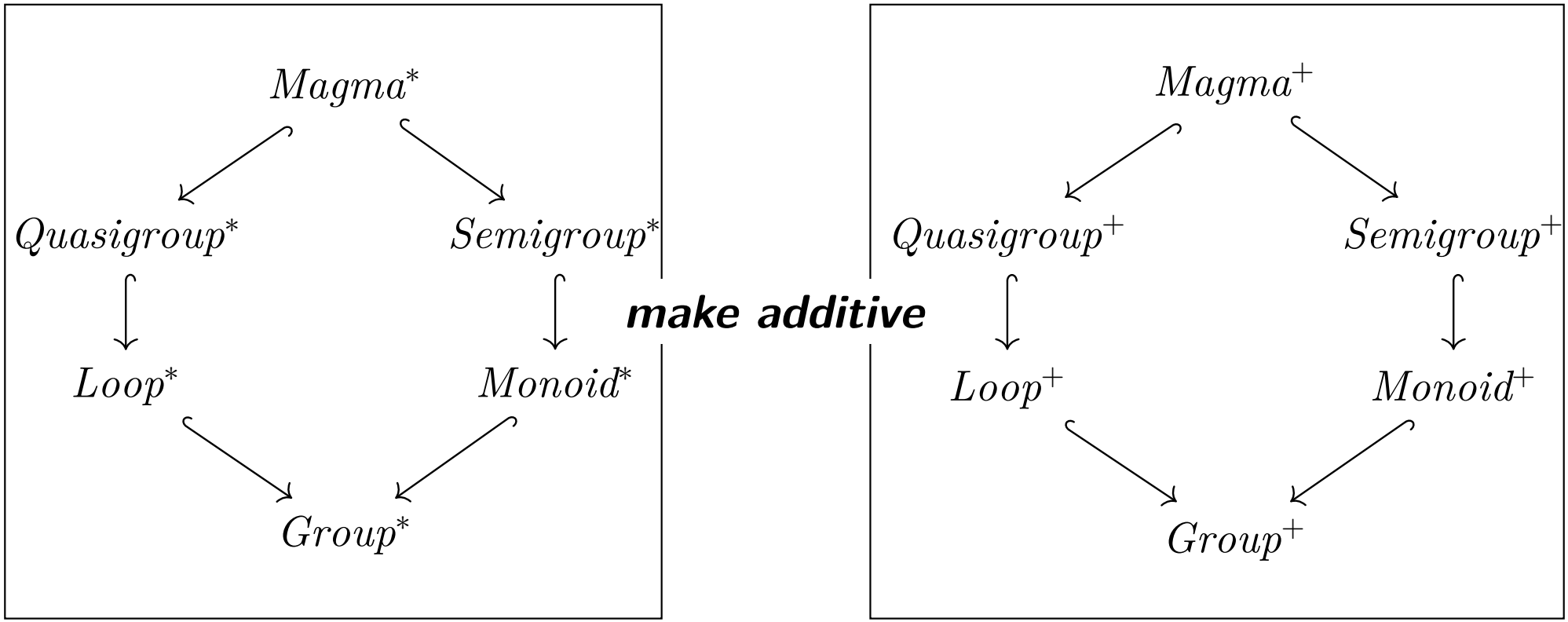


- Large theory graphs successful in
  - algebraic specification languages
  - deduction systems
- Modularity essential at large scale

Meta-Programming with Diagram Operators



Meta-Programming with Diagram Operators




```
MagmaHierarchy := DIAG(Magma, Quasigroup, ...)
AdditiveMagmas = makeAdditive(MagmaHierarchy)
```

Spectrum of Diagram Operators

- fixed theory operators [**casl_syntactic_theory_functors**; Mos+13; CO12]

  inclusions, colimits etc.

- operators to form named diagrams [Mos+13]

  `S union T`, `net remove T`

- extended to user-programmable diagram operators [SR19]

  `AdditiveMagmas := makeAdditive(MagmaHierarchy)`
  $\approx \mathrm{Diag} \to \mathrm{Diag}$

- ...in a structured and modularity-preserving way this talk

  $\approx$ subclasses of $\mathrm{Diag} \to \mathrm{Diag}$

OMDoc/Mmt Language

Syntax

$$
\begin{array}{lcl}
\mathit{Diag} & ::= & (\mathit{Thy} \mid \mathit{View})^* \\
\\
\mathit{Thy} & ::= & T = \{\mathit{Decl}^*\} \\
\mathit{Decl} & ::= & c\colon A\,[= A] \\
\\
\mathit{View} & ::= & v\colon S \to T = \{\mathit{Ass}^*\} \\
\mathit{Ass} & ::= & c := A \\
\\
A & ::= & \mathrm{type} \mid c \mid x \mid A\ A \mid A \to A \mid \\
 & & \lambda x\!:\!A.\ A \mid \Pi x\!:\!A.\ A
\end{array}
$$

OMDoc/Mmt Language

Syntax

$$
\begin{array}{lcl}
\mathit{Diag} & ::= & (\mathit{Thy} \mid \mathit{View})^* \\
\\
\mathit{Thy} & ::= & T = \{\mathit{Decl}^*\} \\
\mathit{Decl} & ::= & c\colon A\,[= A] \\
\\
\mathit{View} & ::= & v\colon S \to T = \{\mathit{Ass}^*\} \\
\mathit{Ass} & ::= & c := A \\
\\
A & ::= & \mathrm{type} \mid c \mid x \mid A\ A \mid A \to A \mid \\
 & & \lambda x\!:\!A.\ A \mid \Pi x\!:\!A.\ A
\end{array}
$$

Semantics

Category $\mathbb{Thy}$ with

- objects being well-typed flat theories

$$(c\colon A[= A])^*$$

- morphisms being well-typed flat views

$$(c := A)^*$$

OMDoc/Mmt Language

Syntax

$$
\begin{array}{lcl}
\mathit{Diag} & ::= & (\mathit{Thy} \mid \mathit{View})^* \\
\\
\mathit{Thy} & ::= & T = \{\mathit{Decl}^*\} \\
\mathit{Decl} & ::= & c\colon A\,[= A] \mid \textbf{include}\ T \\
\\
\mathit{View} & ::= & v\colon S \to T = \{\mathit{Ass}^*\} \\
\mathit{Ass} & ::= & c := A \mid \textbf{include}\ v \\
\\
A & ::= & \mathrm{type} \mid c \mid x \mid A\ A \mid A{\to}A \mid \\
 & & \lambda x{:}A.\ A \mid \Pi x{:}A.\ A
\end{array}
$$

Semantics

Category $\mathbb{Thy}$ with

- objects being well-typed flat theories

$$(c\colon A[= A])^*$$

- morphisms being well-typed flat views

$$(c := A)^*$$

Flattening $\cdot^\flat$ from syntax to semantics

e.g. $(S = \{\textbf{include}\ T, c, d, e\})^\flat = T^\flat \cup \{c, d, e\}$

Operators in Detail

Definition

A **syntactic diagram operator** is a partial function $\mathrm{Diag} \to \mathrm{Diag}$.
A **flat diagram operator** is a partial functor $\Delta \colon \mathbb{T}\mathrm{hy} \to \mathbb{T}\mathrm{hy}$.

Operators in Detail

Definition

A **syntactic diagram operator** is a partial function $\mathrm{Diag} \to \mathrm{Diag}$.
A **flat diagram operator** is a partial functor $\Delta \colon \mathbb{T}\mathrm{hy} \to \mathbb{T}\mathrm{hy}$.

Definition

For a theory T we denote the **subcategory of T-extensions** by $\mathbb{T}\mathrm{hy}^T$.

Operators in Detail

Definition

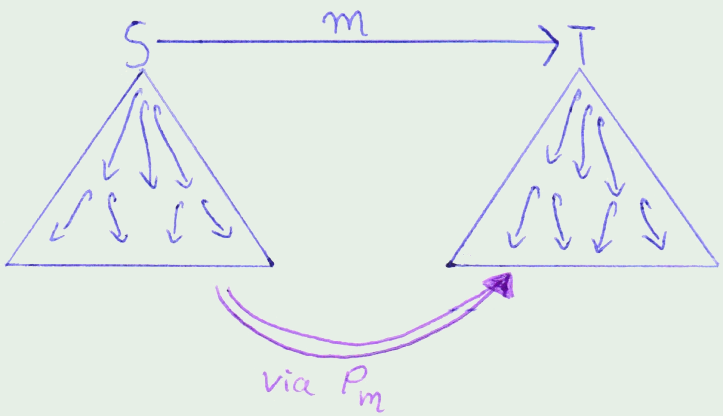
A **syntactic diagram operator** is a partial function $\mathrm{Diag} \to \mathrm{Diag}$.
A **flat diagram operator** is a partial functor $\Delta \colon \mathbb{T}\mathrm{hy} \to \mathbb{T}\mathrm{hy}$.

Definition

For a theory T we denote the **subcategory of T-extensions** by $\mathbb{T}\mathrm{hy}^T$.

Example (Pushout)

For a view $m \colon S \to T$ we have $\mathrm{P}_m \colon \mathbb{T}\mathrm{hy}^S \to \mathbb{T}\mathrm{hy}^T$



Pushout on Flat Theories

$$\begin{array}{ccc} S & \xrightarrow{m} & T \\ \downarrow & & \downarrow \\ A & \xrightarrow{m^{\mathcal{A}}} & \mathrm{P}_m A \end{array}$$

$$\mathrm{P}_m((c_i : E_i = e_i)_{i=1\,\ldots\,n}) =$$

Pushout on Flat Theories

$$\begin{array}{ccc} S & \xrightarrow{m} & T \\ \downarrow & & \downarrow \\ A & \xrightarrow{m^A} & \mathrm{P}_m A \end{array}$$

$$\mathrm{P}_m((c_i : E_i = e_i)_{i=1\,\ldots\,n}) = \begin{cases} \text{all declarations of } T \\ \\ \textit{and} \\ \\ \textbf{for all } c : E\,[= e] \textbf{ in } A \quad S\textbf{:} \\ \quad c : \overline{m^A}(E)\,[= \overline{m^A}(e)] \end{cases}$$

$$m^A = c \mapsto \begin{cases} m(c) & c \in S \\ c & \text{else} \end{cases}$$

Pushout on Flat Theories

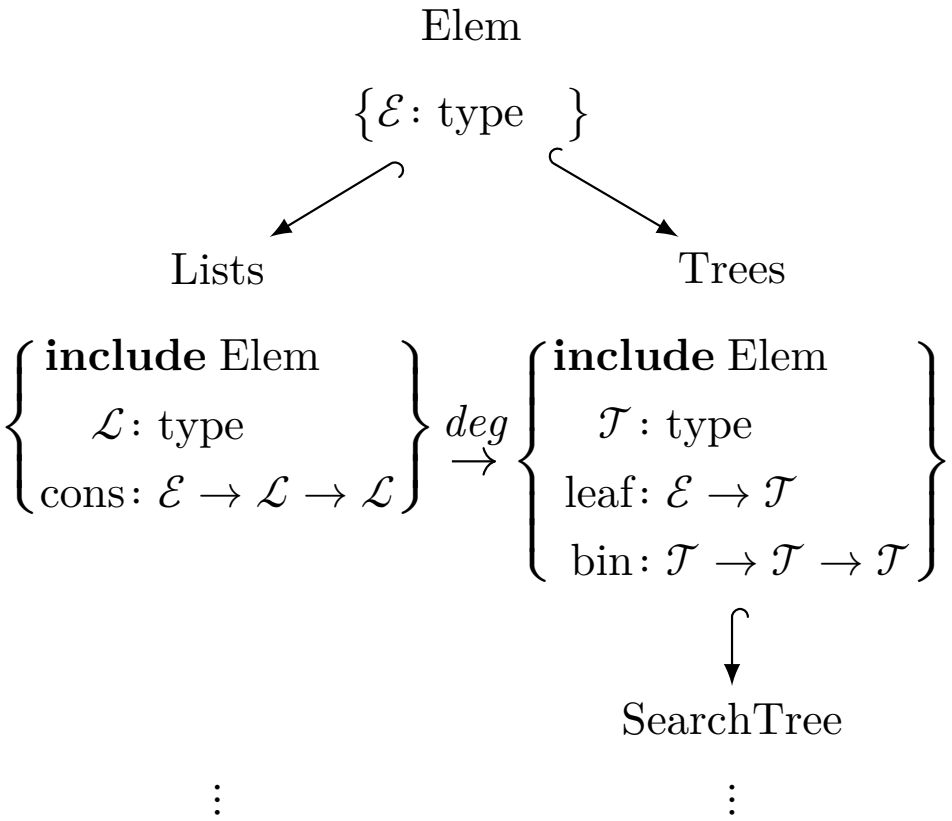
$$\begin{array}{ccc} S & \xrightarrow{m} & T \\ \downarrow & & \downarrow \\ A & \xrightarrow{m^A} & \mathrm{P}_m A \end{array}$$

$$\mathrm{P}_m((c_i : E_i = e_i)_{i=1\,\ldots\,n}) = \begin{cases} \text{all declarations of } T \\ \\ \textit{and} \\ \\ \textbf{for all } c : E\,[= e] \textbf{ in } A \quad S\textbf{:} \\ \quad c : \overline{m^A}(E)\,[= \overline{m^A}(e)] \end{cases}$$

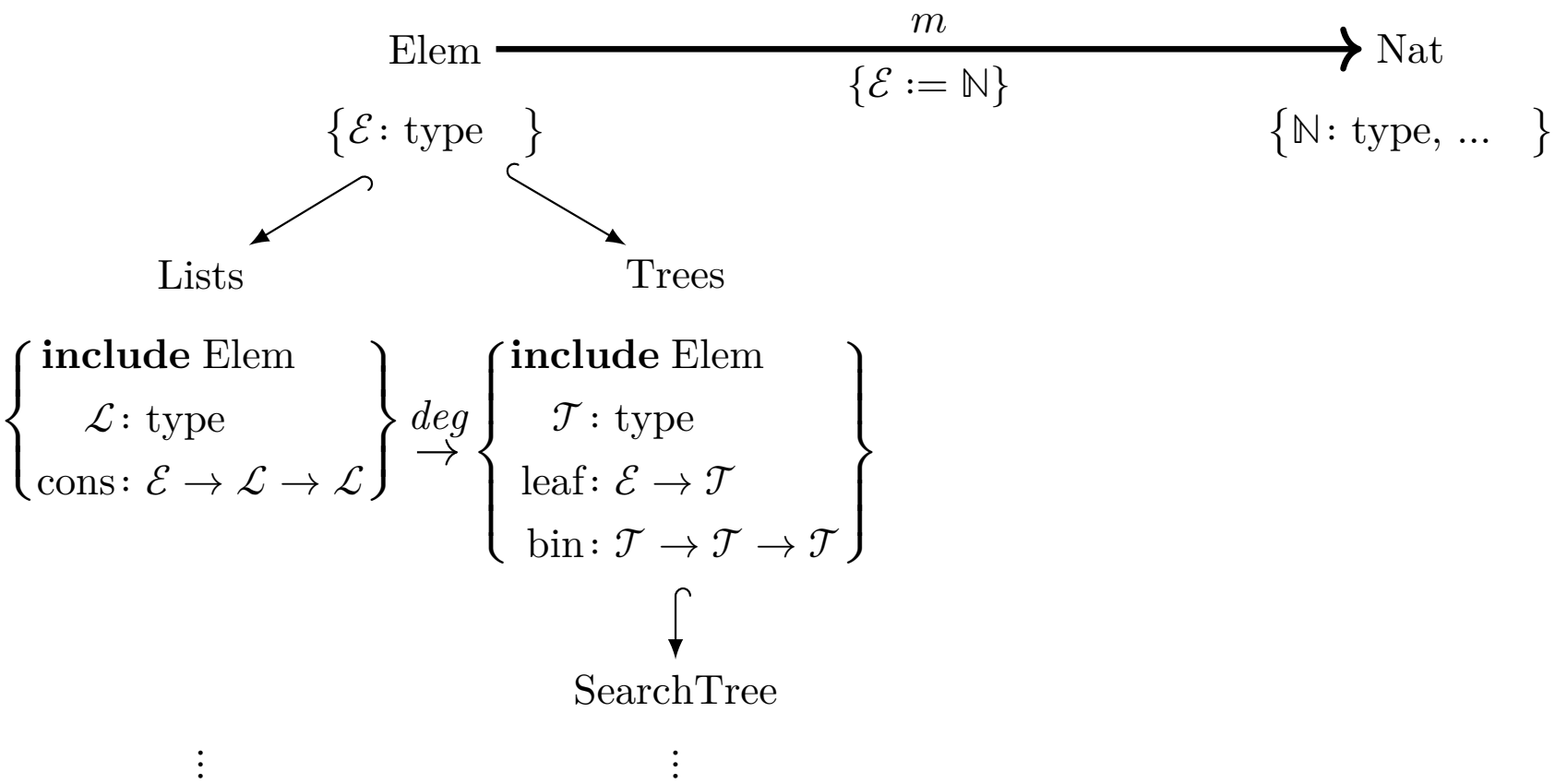
$$m^A = c \mapsto \begin{cases} m(c) & c \in S \\ c & \text{else} \end{cases}$$

$\Rightarrow P_m$ works declaration-wise; it is *sublinear*

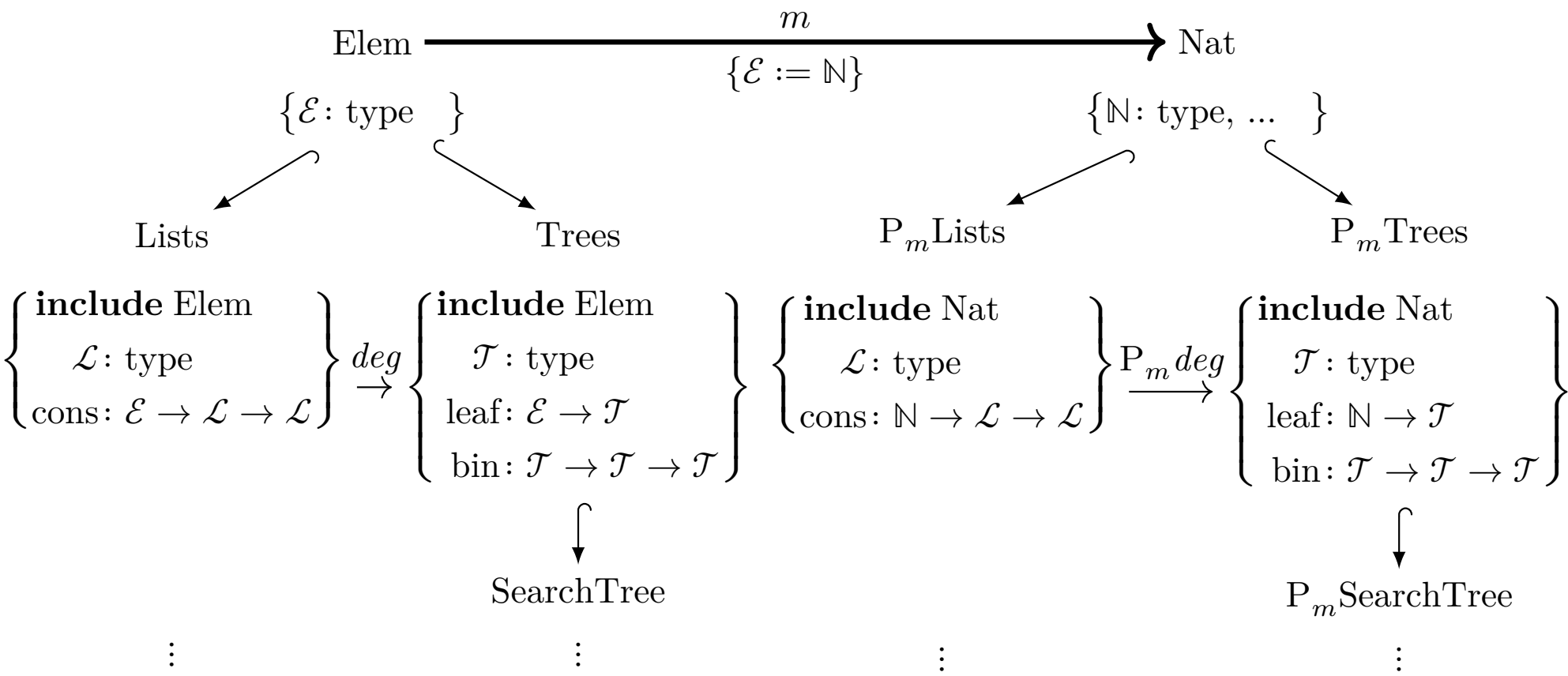
Pushout on Syntactic Diagrams



Pushout on Syntactic Diagrams



Pushout on Syntactic Diagrams



$\Rightarrow \mathrm{P}_m$ is *include-preserving*

Properties of P_m

Goal: Identify useful properties of flat operators
e.g. to lift their definitions to syntax

Overview: P_m is

- functorial
- *sublinear*
- *include-preserving*
- *homogeneous*

Sublinearity

Definition

We call Δ **sublinear** if there is a partial function $\Delta\_(-)$ such that for every theory Σ and declaration D as well as view σ and assignment a:

$$\Delta(\Sigma, D) = \Delta(\Sigma), \Delta_\Sigma(D)$$
$$\Delta(\sigma, a) = \Delta(\sigma), \Delta_\sigma(a).$$

Upon existence, we call $\Delta\_(-)$ the **context-aware action** of Δ.

Lemma

Sublinear operators

- *uniquely determine* $\Delta\_(-)$,
- *and are uniquely determined by* $\Delta(\varnothing)$ *and* $\Delta\_(-)$.

Sublinearity of P_m

Example

P_m is sublinear with

$$\Delta(\varnothing) = \mathrm{cod}(\mathrm{m})$$

$$\Delta_T(c\colon E = e) = \begin{cases} c\colon \overline{m_T}(E) = \overline{m_T}(e) & c \notin \mathrm{dom}(m) \\ \varnothing & \text{else} \end{cases}$$

Sublinearity of P_m

Example

P_m is sublinear with

$$\Delta(\varnothing) = \mathrm{cod}(\mathrm{m})$$

$$\Delta_T(c\colon E = e) = \begin{cases} c\colon \overline{m_T}(E) = \overline{m_T}(e) & c \notin \mathrm{dom}(m) \\ \varnothing & \text{else} \end{cases}$$

Specification

"Let P_m be the sublinear operator with $\Delta(\varnothing) = \ldots$ and $\Delta_T(c) = \ldots$"

Implementation

```
def transformDeclaration(declaration: Declaration, ctx:
        HelperContext): List[Declaration]
```

Include Preservation

Definition

We call Δ **include-preserving** if

$$S \subseteq T \Rightarrow \Delta(S) \subseteq \Delta(T)$$
$$\varphi \subseteq \psi \Rightarrow \Delta(\varphi) \subseteq \Delta(\psi)$$

Example

P_m is include-preserving.

Include Preservation

Definition

We call Δ **include-preserving** if

$$S \subseteq T \Rightarrow \Delta(S) \subseteq \Delta(T)$$
$$\varphi \subseteq \psi \Rightarrow \Delta(\varphi) \subseteq \Delta(\psi)$$

Example

P_m is include-preserving.

Theorem

Operators are sublinear iff. include-preserving.

Implementation: Take sublinear specification and lift to syntax with inclusions

Semantics-Preserving Lifting

Theorem (lifting preserves semantics)

Let Δ be a sublinear operator and define its lifting $\hat{\Delta}$ via

$$\hat{\Delta}\left(T = \left\{\begin{array}{l}\textbf{include } S_1, \dots, S_n \\ c_1, \dots, c_n\end{array}\right\}\right) := T' = \left\{\begin{array}{l}\textbf{include } S'_1, \dots, S'_n \\ \Delta_{S_1^\flat, \dots, S_n^\flat}(c_1, \dots, c_n)\end{array}\right\}$$

and analogously for views and entire diagrams. Then

$$\begin{array}{ccc} \mathit{Diag} & \xrightarrow{\hat{\Delta}} & \mathit{Diag} \\ \downarrow{\scriptstyle .^\flat} & & \downarrow{\scriptstyle .^\flat} \\ \mathbb{T}\mathrm{hy} & \xrightarrow{\Delta} & \mathbb{T}\mathrm{hy} \end{array}$$

commutes.

More Examples

- Church To Curry

  Convert Church type theories

$$\begin{aligned} \_ \times \_ &: tp \to tp \to tp \\ (\_, \_) &: \{A, B\}\ tm\ A \to tm\ B \to tm\ A \times B \end{aligned}$$

  to their Curry equivalents:

$$\begin{aligned} \_ \times \_ &: tp \to tp \to tp \\ (\_, \_) &: tm \to tm \to tm \\ (\_, \_)^T &: \{A, B, a : tm, b : tm\}\ a :: A \to b :: B \to (a, b) :: (A \times B) \end{aligned}$$

Case Study: Church To Curry in LATIN2

```
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:
  - 180 LOC for Church and Curry
  - manually kept in sync, in fact Curry was out-of-sync

Case Study: Church To Curry in LATIN2

```
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:
  - 180 LOC for Church and Curry
  - manually kept in sync, in fact Curry was out-of-sync

Case Study: Church To Curry in LATIN2

```
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:
  - 180 LOC for Church and Curry
  - manually kept in sync, in fact Curry was out-of-sync
- After: 120 LOC, *consistent knowledge*

More Examples (cont.)

- Typify Fol

  Convert declarations of $\forall\exists\ldots$ from FOL

  $$\begin{array}{ll} \forall & : (tm \to prop) \to prop \\ \forall I & : \{P : tm \to prop\}(\{x : tm\} \vdash P\,x) \to \vdash \forall P \end{array}$$

  to SFOL:

  $$\begin{array}{ll} \forall & : \{A : tp\}\ (tm\ A \to prop) \to prop \\ \forall I & : \{A : tp, P : tm\ A \to prop\}(\{x : tm\ A\} \vdash P\,x) \to \vdash \forall P \end{array}$$

Case Study: Typify Fol in LATIN2

```
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:
  - 170 LOC for FOL and SFOL
  - manually kept in sync

(now really)

Case Study: Typify Fol in LATIN2

```
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 andE1) (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:
  - 170 LOC for FOL and SFOL
  - manually kept in sync

(now really)

Case Study: Typify Fol in LATIN2

```
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:
  - 170 LOC for FOL and SFOL
  - manually kept in sync (now really)
- After: 110 LOC

More Examples (cont.)

General: Compose, Union, AbstractDecl, ...

Algebra:

- Additive, Multiplicative
- Homomorphisms
- Substructures
- Quotients
- Products

Logic:

- Polymorphify
- Kripkefy

Conclusion: Foster Modularity by Meta-Programming

- Diagram operators serve as meta-programming facilities
  - to lessen redundancy
  - to increase consistency
- Classifying diagram operators eases
  - specification
  - implementation
- Semantics-preserving lifting ensures applicability on real-world theory graphs

Bibliography I

Jacques Carette and Russell O'Connor. "Theory Presentation Combinators". In: *Intelligent Computer Mathematics*. Ed. by Johan Jeuring et al. LNAI 7362. Berlin and Heidelberg: Springer Verlag, 2012, pp. 202–215. ISBN: 978-3-642-31373-8. URL: http://www.cas.mcmaster.ca/~carette/publications/tpc.pdf.

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.

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

Bibliography II

Till Mossakowski et al. "The Distributed Ontology, Modeling and Specification Language". In: *Modular Ontologies*. Ed. by Chiara Del Vescovo et al. CEUR Workshop Proceedings 1081. Invited paper. Aachen, 2013. URL: http://CEUR-WS.org/Vol-1081.

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.

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

Bibliography III

Florian Rabe and Michael Kohlhase. "A Scalable Module System". In: *Information & Computation* 0.230 (2013), pp. 1–54. URL: https://kwarc.info/frabe/Research/mmt.pdf.

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.

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.