Systematic Translation of Formalizations of Type Theory from Intrinsic to Extrinsic Style

Florian Rabe[1] Navid Roux[2]

FAU Erlangen-Nuremberg
LFMTP Workshop 2021 (virtual)

2021-07-16

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

Hard vs. Soft Typing

Hard Typing (intrinsic, Church)

- terms carry their types
- typing is a function from terms to types

$$\begin{aligned}&\text{theory HTyped} =\\ &\quad \text{tp} : \text{type}\\ &\quad \text{tm} : \text{tp} \to \text{type}\end{aligned}$$

Soft Typing (extrinsic, Curry)

- terms and types are independent
- typing is a predicate on terms and types

$$\begin{aligned}&\text{theory STyped} =\\ &\quad \text{tp} : \text{type}\\ &\quad \text{term} : \text{type}\\ &\quad :: : \text{term} \to \text{tp} \to \text{type}\end{aligned}$$

Both flavors needed: some languages even mix them

Hard vs. Soft Function Types

theory HTyped =
  tp : type
  tm : tp → type

theory HFun =
  include HTyped
  fun : tp → tp → tp
  lam : Π a b. (tm a → tm b) → tm fun a b

  app : Π a b. tm (fun a b) → tm a → tm b

Hard vs. Soft Function Types

theory HTyped =
  tp : type
  tm : tp → type

theory HFun =
  include HTyped
  fun : tp → tp → tp
  lam : Π a b. (tm a → tm b) → tm fun a b
  app : Π a b. tm (fun a b) → tm a → tm b

theory STyped =
  tp : type
  term : type
  :: : term → tp → type

theory SFun =
  include STyped
  fun : tp → tp → tp
  lam : Π a. (term → term) → term
  lam* : Π a b. Π F : term → term.
    (Π x. x :: a → (F x) :: b) →
    (lam a F) :: (fun a b)
  app : term → term → term
  app* : Π a b. Π f. f :: (fun a b) → Π x. x :: a →
    (app f x) :: b

Bigger Picture: LATIN2

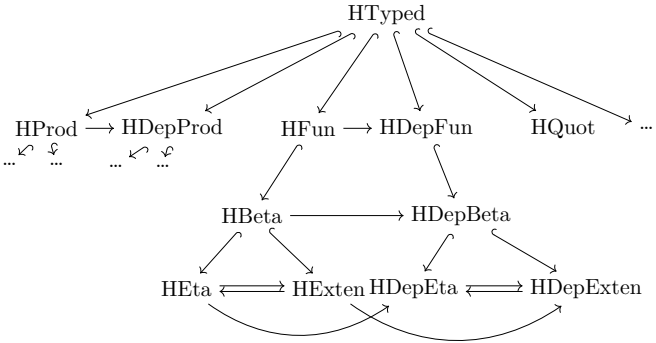
Building an atlas of formal systems in LF

Florian pointing at first-order logic

Going back to Logosphere 1999, ongoing since 2009
Highly modular network of little logic formalizations

- state every feature once and only in smallest possible context
- separate theory for each connective, quantifier, type operator, controversial axiom
- reference catalog of standardized logics

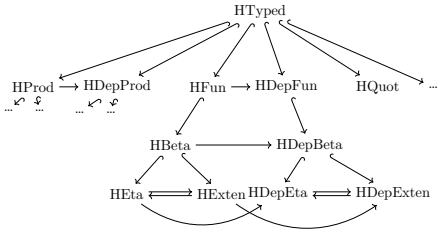
A Hard-Typed Library of Type-Theoretical Features

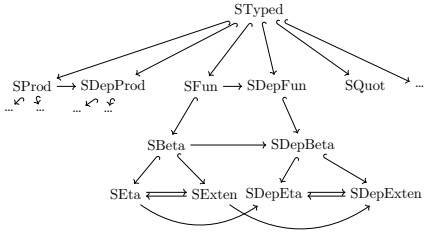


Analogously: a soft-typed library

Hard- and Soft-Typed Libraries

Challenges in Library Management

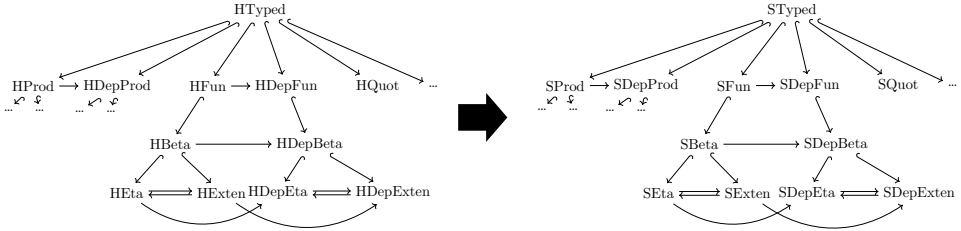




Strong desire to keep things consistent

- same diagrammatic structure — same granularity of theories
- same theory names — HFun and SFun etc.
- same and predictable constant names — $\mathsf{lam} \in \mathsf{HFun}$ and $\mathsf{lam}, \mathsf{lam}^* \in \mathsf{SFun}$
- same orders of parameters — $\mathsf{app} \in \mathsf{HFun}$ takes f, x and $\mathsf{app}^* \in \mathsf{SFun}$ takes $\mathsf{f}, \mathsf{x}, \mathsf{f}^*, \mathsf{x}^*$
- same notations — significant in practice

Central Idea: Maintain One, Generate the Other



Central Idea

Provide an automatic and systematic translation

$$\text{HTyped-library} \mapsto \text{STyped-library}$$

this is trickier than it sounds!

Naive Approach: Ad-hoc Translation

Idea: ad-hoc specification of inductive translation HTyped-library $\mapsto$ STyped-library

Difficulties

- hard to specify & implement all details
- hard to verify/review
- hard to even state the meta theorem
- hard to ascertain whether it scales to all hard-typed/LF features

Naive Approach: Ad-hoc Translation

Idea: ad-hoc specification of inductive translation HTyped-library $\mapsto$ STyped-library

Difficulties

- hard to specify & implement all details
- hard to verify/review
- hard to even state the meta theorem
- hard to ascertain whether it scales to all hard-typed/LF features

Better: instantiate a general theory with the right parameters/few base cases

Naive Approach: Ad-hoc Translation

Idea: ad-hoc specification of inductive translation HTyped-library $\mapsto$ STyped-library

Difficulties

- hard to specify & implement all details — vs. provide only few base cases & code reuse
- hard to verify/review — vs. mechanically verifiable
- hard to even state the meta theorem — vs. read off from base cases
- hard to ascertain whether it scales to all hard-typed/LF features — vs. guaranteed

Better: instantiate a general theory with the right parameters/few base cases

namely, "Logical Relations for a Logical Framework" (Rabe and Sojakova 2013)

Systematic Approach: Intuition

theory HFun =
  include HTyped
  fun : tp → tp → tp

theory SFun =
  include STyped
  fun : tp → tp → tp

- map include HTyped to include STyped
- map constants and parameters as follows:

HTyped decl.	STyped decls.
a : tp	a : tp
x : tm a	x : term, x^* : x :: a

Systematic Approach: Intuition

theory HFun =
  include HTyped

app: Π a b: tp. Π f: tm (fun a b).
  Π x: tm a. tm b

theory SFun =
  include STyped

app : term → term → term
app*: Π a b: tp. Π f: term. Π f*: f :: (fun a b).
  Π x: term. Π x*: x :: a.
  app f x :: b

- map include HTyped to include STyped
- map constants and parameters as follows:

HTyped decl.	STyped decls.
a: tp	a: tp
x: tm a	x: term, x*: x :: a

Systematic Approach: Intuition

theory HFun =
  include HTyped

  lam : Π a b : tp. Π F : (Π x : tm a. tm b).
    tm fun a b

theory SFun =
  include STyped

  lam : Π a : tp. Π F : term → term. term
  lam* : Π a b : tp. Π F : term → term.
    Π F* : (Π x : term. Π x* : x :: a. F x :: b).
    (lam a F) :: (fun a b)

- map include HTyped to include STyped
- map constants and parameters as follows:

HTyped decl.	STyped decls.
a : tp	a : tp
x : tm a	x : term, x* : x :: a

Systematic Approach: Intuition

theory HFun =
  include HTyped
  fun : tp → tp → tp
  lam : Π a b: tp. Π F: (Π x: tm a. tm b).
        tm fun a b
  app : Π a b: tp. Π f: tm (fun a b).
        Π x: tm a. tm b

theory SFun =
  include STyped
  fun : tp → tp → tp
  lam : Π a: tp. Π F: term → term. term
  lam* : Π a b: tp. Π F: term → term.
         Π F*: (Π x: term. Π x*: x :: a. F x :: b).
         (lam a F) :: (fun a b)
  app : term → term → term
  app* : Π a b: tp. Π f: term. Π f*: f :: (fun a b).
         Π x: term. Π x*: x :: a.
         app f x :: b

- map include HTyped to include STyped
- map constants and parameters as follows:

HTyped decl.	STyped decls.
a: tp	a: tp
x: tm a	x: term, x*: x :: a

Systematic Approach: Details

HTyped decl.	STyped decls.
$a : tp$	$a : tp$
$x : tm\ a$	$x : term,\ x^* : x :: a$

Systematic Approach: Details

HTyped decl.	STyped decls.
$a : tp$	$a : tp$
$x : tm\ a$	$x : term,\ x^* : x :: a$

HTyped expressions	STyped expressions
$a : tp$	$TE(a) : TE(tp)$
$x : tm\ a$	$TE(x) : \underbrace{TE(tm)}_{=term},\ TP(x) : \underbrace{TP(tm\ a)\ TE(x)}_{=TE(x)\ ::\ TE(a)}$

Two compositional translations:

- type erasure TE maps LF types to LF types:
  $$\begin{aligned} TE(tp) &= tp \\ TE(tm) &= \lambda a : tp.\ term \end{aligned}$$
- type preservation TP maps LF types A to LF judgements on TE(A)

$$TP(tm) = \lambda a : tp. \quad \underbrace{\lambda x : term.\ x :: a}_{\text{relation that tm a is mapped to}}$$

Recap: “Logical Relations for a Logical Framework”

by Rabe and Sojakova 2013

A theory of logical relations formulated for LF

- formalize logical relation arguments on LF signatures in LF ⇒ mechanically verifiable
- for proof: state meta theorems on LF signature translations e.g., TP on TE

Recap: “Logical Relations for a Logical Framework”

by Rabe and Sojakova 2013

A theory of logical relations formulated for LF

- formalize logical relation arguments on LF signatures in LF ⇒ mechanically verifiable
- for proof: state meta theorems on LF signature translations e.g., TP on TE
- for data: generate new content

Recap: "Logical Relations for a Logical Framework"

by Rabe and Sojakova 2013

A theory of logical relations formulated for LF

- formalize logical relation arguments on LF signatures in LF ⇒ mechanically verifiable
- for proof: state meta theorems on LF signature translations e.g., TP on TE
- for data: generate new content

$$\begin{array}{ll} \mathrm{TE}(\mathrm{tp}) = \mathrm{tp} & \mathrm{TP}(\mathrm{tp}) = \text{undefined} \\ \mathrm{TE}(\mathrm{tm}) = \lambda \mathrm{a}\colon \mathrm{tp}.\ \mathrm{term} & \mathrm{TP}(\mathrm{tm}) = \lambda \mathrm{a}\colon \mathrm{tp}.\ \lambda \mathrm{x}\colon \mathrm{term}.\ \mathrm{x} :: \mathrm{a} \end{array}$$

Via the logical relation TP,

- LF types $\mathrm{A}\colon \mathrm{type}$ are mapped to unary predicates $\mathrm{TP}(\mathrm{A})\colon \mathrm{TE}(\mathrm{A}) \to \mathrm{type}$
- LF terms $\mathrm{t}\colon \mathrm{A}$ are mapped to proofs $\mathrm{TP}(\mathrm{t})\colon \mathrm{TP}(\mathrm{A})\ \mathrm{TE}(\mathrm{t})$

Softening Hard-Typed Theories

Definition (Soften on Theories)

Soften translates HTyped-theories to STyped-theories by

- mapping include HTyped to include STyped
- mapping every constant $c : A$ to $c : \mathrm{TE}(A)$ and $c^* : \mathrm{TP}(A)\ c$ whichever are defined

$$\begin{aligned} \mathrm{TE}(\mathrm{tp}) &= \mathrm{tp} & \mathrm{TP}(\mathrm{tp}) &= \text{undefined} \\ \mathrm{TE}(\mathrm{tm}) &= \lambda a : \mathrm{tp}.\ \mathrm{term} & \mathrm{TP}(\mathrm{tm}) &= \lambda a : \mathrm{tp}.\ \lambda x : \mathrm{term}.\ x :: a \end{aligned}$$

Softening Hard-Typed Theories

Definition (Soften on Theories)

Soften translates HTyped-theories to STyped-theories by

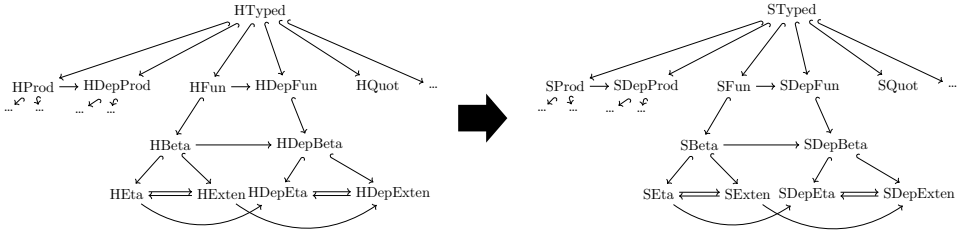
- mapping include HTyped to include STyped
- mapping every constant $c : A$ to $c : \mathrm{TE}(A)$ and $c^* : \mathrm{TP}(A)\ c$ whichever are defined

$$\begin{array}{ll} \mathrm{TE}(\mathrm{tp}) = \mathrm{tp} & \mathrm{TP}(\mathrm{tp}) = \text{undefined} \\ \mathrm{TE}(\mathrm{tm}) = \lambda a : \mathrm{tp}.\ \mathrm{term} & \mathrm{TP}(\mathrm{tm}) = \lambda a : \mathrm{tp}.\ \lambda x : \mathrm{term}.\ x :: a \end{array}$$

theory HFun =
  include HTyped
  fun : tp → tp → tp
  ...
  app: Π a b: tp. Π f: tm (fun a b).
    Π x: tm a. tm b

theory SFun =
  include STyped
  fun : tp → tp → tp
  ...
  app : term → term → term
  app*: Π a b: tp. Π f: term. Π f*: f :: (fun a b).
    Π x: term. Π x*: x :: a. app f x :: b

Coming back: Translating HTyped Libraries



We wanted to autogenerate the STyped-library

$\Rightarrow$ lift softening on theories to whole diagrams

Softening of Libraries

Definition (Soften)

The functor Soften translates HTyped-diagrams D to STyped-diagrams D' by

- mapping every theory $X \in D$ to $X^{\text{Soften}} \in D'$
- mapping

$$\begin{aligned} \text{include HTyped} &\mapsto \text{include STyped} \\ \text{include X} &\mapsto \text{include } X^{\text{Soften}} \end{aligned}$$

- mapping every constant $c\colon A$ to $c\colon \text{TE}(A)$ and $c^*\colon \text{TP}(A)\ c$

$$\begin{aligned} \text{TE}(\text{tp}) &= \text{tp} & \text{TP}(\text{tp}) &= \text{undefined} \\ \text{TE}(\text{tm}) &= \lambda a\colon \text{tp. term} & \text{TP}(\text{tm}) &= \lambda a\colon \text{tp. } \lambda x\colon \text{term. } x :: a \end{aligned}$$

Preserves includes, extends easily to theory morphisms

"Structure-Preserving Diagram Operators" by Roux and Rabe

Case Study: HTyped-library $\mapsto$ STyped-library in LATIN2

```
diagram HLibrary := {HProd, HDepProd, HFun, ...}
diagram SLibrary := SOFTEN HLibrary
```

Advantages

- $>$ 300 LOC autogenerated
- even better coverage & consistency than before
- more than a functor: Soften also relates HLibrary and SLibrary

  e.g., morphism $\mathrm{TE}^{\mathrm{HFun}} : \mathrm{HFun} \to \mathrm{SFun}$

  logical relation $\mathrm{TP}^{\mathrm{HFun}} : \mathrm{HFun} \to \mathrm{SFun}$

  allows us to type-erase all HLibrary-programs & have type preservation property

Subtleties I

- Need to have partial logical relation for type preservation

$$\begin{aligned} \mathrm{TP}(\mathrm{tp}) &= \mathrm{Unit} \\ \mathrm{TP}(\mathrm{tm}) &= \lambda \mathrm{a}\colon \mathrm{tp}.\ \lambda \mathrm{x}\colon \mathrm{term}.\ \mathrm{x} :: \mathrm{a} \end{aligned}$$

For object types $\mathrm{a}\colon \mathrm{tp}$ there is nothing we need to prove, thus we assign Unit. But that leads to undesired output:

in HFun:	app:	$\Pi\, \mathrm{a}\ \mathrm{b}\colon \mathrm{tp}.\ \Pi\, \mathrm{f}\colon \mathrm{tm}\ (\mathrm{fun}\, \mathrm{a}\, \mathrm{b}).\ \Pi\, \mathrm{x}\colon \mathrm{tm}\ \mathrm{a}.\ \mathrm{tm}\ \mathrm{b}$
in SFun:	app:	$\ldots$
	app^*:	$\Pi\, \mathrm{a}\ \mathrm{b}\colon \mathrm{tp}.\ \Pi\, \mathrm{a}^*\ \mathrm{b}^*\colon \mathrm{Unit}.\ \Pi\, \mathrm{f}\colon \mathrm{term}.\ \Pi\, \mathrm{f}^*\colon \mathrm{f} :: (\mathrm{fun}\, \mathrm{a}\, \mathrm{b}).$ $\Pi\, \mathrm{x}\colon \mathrm{term}.\ \Pi\, \mathrm{x}^*\colon \mathrm{x} :: \mathrm{a}.\ \mathrm{app}\, \mathrm{f}\, \mathrm{x} :: \mathrm{b}$

Subtleties I

- Need to have partial logical relation for type preservation

$$\begin{aligned} \mathrm{TP}(\mathrm{tp}) &= \sout{\mathrm{Unit}} \\ \mathrm{TP}(\mathrm{tm}) &= \lambda \mathrm{a}\colon \mathrm{tp}.\ \lambda \mathrm{x}\colon \mathrm{term}.\ \mathrm{x} :: \mathrm{a} \end{aligned}$$

For object types $\mathrm{a}\colon \mathrm{tp}$ there is nothing we need to prove, thus we assign Unit. But that leads to undesired output:

$$\begin{array}{lll} \text{in HFun:} & \mathrm{app}\colon & \Pi\, \mathrm{a}\ \mathrm{b}\colon \mathrm{tp}.\ \Pi\, \mathrm{f}\colon \mathrm{tm}\ (\mathrm{fun}\ \mathrm{a}\ \mathrm{b}).\ \Pi\, \mathrm{x}\colon \mathrm{tm}\ \mathrm{a}.\ \mathrm{tm}\ \mathrm{b} \\ \text{in SFun:} & \mathrm{app}\colon & \ldots \\ & \mathrm{app}^*\colon & \Pi\, \mathrm{a}\ \mathrm{b}\colon \mathrm{tp}.\ \sout{\Pi\, \mathrm{a}^*\ \mathrm{b}^*\colon \mathrm{Unit}.}\ \Pi\, \mathrm{f}\colon \mathrm{term}.\ \Pi\, \mathrm{f}^*\colon \mathrm{f}\ ::\ (\mathrm{fun}\ \mathrm{a}\ \mathrm{b}). \\ & & \Pi\, \mathrm{x}\colon \mathrm{term}.\ \Pi\, \mathrm{x}^*\colon \mathrm{x} :: \mathrm{a}.\ \mathrm{app}\ \mathrm{f}\ \mathrm{x} :: \mathrm{b} \end{array}$$

Subtleties I

- Need to have partial logical relation for type preservation

$$\begin{aligned} \mathrm{TP}(\mathrm{tp}) &= \sout{\mathrm{Unit}} \\ \mathrm{TP}(\mathrm{tm}) &= \lambda \mathrm{a} : \mathrm{tp}.\ \lambda \mathrm{x} : \mathrm{term}.\ \mathrm{x} :: \mathrm{a} \end{aligned}$$

For object types $\mathrm{a} : \mathrm{tp}$ there is nothing we need to prove, thus we assign Unit.
But that leads to undesired output:

in HFun:	app:	$\Pi\, \mathrm{a}\, \mathrm{b} : \mathrm{tp}.\ \Pi\, \mathrm{f} : \mathrm{tm}\ (\mathrm{fun}\, \mathrm{a}\, \mathrm{b}).\ \Pi\, \mathrm{x} : \mathrm{tm}\ \mathrm{a}.\ \mathrm{tm}\ \mathrm{b}$
in SFun:	app:	$\ldots$
	app^*:	$\Pi\, \mathrm{a}\, \mathrm{b} : \mathrm{tp}.\ \sout{\Pi\, \mathrm{a}^*\, \mathrm{b}^* : \mathrm{Unit}.}\ \Pi\, \mathrm{f} : \mathrm{term}.\ \Pi\, \mathrm{f}^* : \mathrm{f} :: (\mathrm{fun}\, \mathrm{a}\, \mathrm{b}).$ $\Pi\, \mathrm{x} : \mathrm{term}.\ \Pi\, \mathrm{x}^* : \mathrm{x} :: \mathrm{a}.\ \mathrm{app}\, \mathrm{f}\, \mathrm{x} :: \mathrm{b}$

$\Rightarrow$ generalize "Logical Relations for a Logical Framework" (Rabe, Sojakova 2013) to allow for partiality

Subtleties II

- Need to remove parameters that became unnecessary after type erasure

in HFun:	app:	$\Pi\,a\,b\colon \mathrm{tp}.\ \Pi\,f\colon \mathrm{tm}\ (\mathrm{fun}\,a\,b).\ \Pi\,x\colon \mathrm{tm}\ a.\ \mathrm{tm}\ b$
in SFun:	app:	~~$\Pi\,a\,b\colon \mathrm{tp}.$~~ $\mathrm{term} \to \mathrm{term} \to \mathrm{term}$
	app*:	...
in HFun:	lam:	$\Pi\,a\,b\colon \mathrm{tp}.\ \Pi\,F\colon \mathrm{tm}\ a \to \mathrm{tm}\ b.\ \mathrm{tm}\ \mathrm{fun}\ a\ b$
in SFun:	lam:	$\Pi\,a$ ~~b~~$\colon \mathrm{tp}.\ \Pi\,F\colon \mathrm{term} \to \mathrm{term}.\ \mathrm{term}$
	lam*:	...
in HFun:	dfun:	$\Pi\,a\colon \mathrm{tp}.\ (\mathrm{tm}\ a \to \mathrm{tp}) \to \mathrm{tp}$
in SFun:	dfun:	$\Pi\,a\colon \mathrm{tp}.\ (\mathrm{term} \to \mathrm{tp}) \to \mathrm{tp}$
	dfun*:	...

Removal may be required (app), optional (lam), or forbidden (dfun)

Subtleties II

- Need to remove parameters that became unnecessary after type erasure

in HFun:	app:	$\Pi\,a\;b\colon \mathrm{tp}.\ \Pi\,f\colon \mathrm{tm}\ (\mathrm{fun}\ a\ b).\ \Pi\,x\colon \mathrm{tm}\ a.\ \mathrm{tm}\ b$
in SFun:	app:	~~$\Pi\,a\;b\colon \mathrm{tp}.$~~ $\mathrm{term} \to \mathrm{term} \to \mathrm{term}$
	app^*:	...
in HFun:	lam:	$\Pi\,a\;b\colon \mathrm{tp}.\ \Pi\,F\colon \mathrm{tm}\ a \to \mathrm{tm}\ b.\ \mathrm{tm}\ \mathrm{fun}\ a\ b$
in SFun:	lam:	$\Pi\,a$ ~~b~~$\colon \mathrm{tp}.\ \Pi\,F\colon \mathrm{term} \to \mathrm{term}.\ \mathrm{term}$
	lam^*:	...
in HFun:	dfun:	$\Pi\,a\colon \mathrm{tp}.\ (\mathrm{tm}\ a \to \mathrm{tp}) \to \mathrm{tp}$
in SFun:	dfun:	$\Pi\,a\colon \mathrm{tp}.\ (\mathrm{term} \to \mathrm{tp}) \to \mathrm{tp}$
	dfun^*:	...

Removal may be required (app), optional (lam), or forbidden (dfun)
$\Rightarrow$ use some heuristic and allow user annotations to deviate from it

Conclusion

- Two flavors of formalizing type theories

  hard-typed (intrisic) vs. soft-typed (extrinsic)

- Maintenance of both is too costly, hence generate one from the other

  $\texttt{diagram HLibrary} := \{\texttt{HProd}, \texttt{HDepProd}, \texttt{HFun}, \ldots\}$

  $\texttt{diagram SLibrary} := \texttt{SOFTEN HLibrary}$

Conclusion

- Two flavors of formalizing type theories

  hard-typed (intrisic) vs. soft-typed (extrinsic)

- Maintenance of both is too costly, hence generate one from the other

  $\texttt{diagram HLibrary} := \{\texttt{HProd}, \texttt{HDepProd}, \texttt{HFun}, \ldots\}$
  $\texttt{diagram SLibrary} := \texttt{SOFTEN HLibrary}$

- Key Idea: systematically translate constants $c\colon A$ to
  1. a type-erased variant $c : \mathrm{TE}(A)$ via an LF signature morphism
  2. a type-preservation property $c^*\colon \mathrm{TP}(A)\ c$ via an LF logical relation

  At LF types tm a (of hard-typed terms) TP gives the LF relation $\lambda x\colon \mathrm{term}.\ x :: \mathrm{TE}(a)$.