# A Generalized Logical Framework

**András Kovács**[1], Christian Sattler[1]

[1]University of Gothenburg & Chalmers University of Technology

18 Apr 2025, EuroProofNet WG6 meeting, Genoa

# Overview

1. Two-level type theories (2LTT):
   - metaprogramming over a **single model** of a **single type theory**.

# Overview

1. Two-level type theories (2LTT):
    - metaprogramming over a **single model** of a **single type theory**.
    - the chosen model is defined **outside the system**.

# Overview

1. Two-level type theories (2LTT):
   - metaprogramming over a **single model** of a **single type theory**.
   - the chosen model is defined **outside the system**.
   - **only a second-order ("internal")** view on the model.

# Overview

1. Two-level type theories (2LTT):
   - metaprogramming over a **single model** of a **single type theory**.
   - the chosen model is defined **outside the system**.
   - **only a second-order ("internal")** view on the model.
2. Generalized logical framework (GLF):
   - metaprogramming over **any number of models** of **any number of type theories**.

# Overview

1. Two-level type theories (2LTT):
   - metaprogramming over a **single model** of a **single type theory**.
   - the chosen model is defined **outside the system**.
   - **only a second-order ("internal")** view on the model.
2. Generalized logical framework (GLF):
   - metaprogramming over **any number of models** of **any number of type theories**.
   - models are defined **inside the system**.

# Overview

1. Two-level type theories (2LTT):
   - metaprogramming over a **single model** of a **single type theory**.
   - the chosen model is defined **outside the system**.
   - **only a second-order ("internal")** view on the model.
2. Generalized logical framework (GLF):
   - metaprogramming over **any number of models** of **any number of type theories**.
   - models are defined **inside the system**.
   - both a **first-order/external** and a **second-order/internal** view on each model.

# Overview

1. Two-level type theories (2LTT):
    - metaprogramming over a **single model** of a **single type theory**.
    - the chosen model is defined **outside the system**.
    - **only a second-order ("internal")** view on the model.
2. Generalized logical framework (GLF):
    - metaprogramming over **any number of models** of **any number of type theories**.
    - models are defined **inside the system**.
    - both a **first-order/external** and a **second-order/internal** view on each model.

*In this talk*:

1. A syntax of GLF + examples + increasing amount of syntactic sugar.
2. A short overview of semantics.

# GLF basic universes & type formers

| | | |
|---|---|---|
| $\mathbf{U}$ | $: \mathbf{U}$ | A universe of that supports ETT. |
| $\mathbf{Base}$ | $: \mathbf{U}$ | Type of "base categories". |
| $\mathbf{1}$ | $: \mathbf{Base}$ | The terminal category as a base category. |
| $\mathbf{PSh}$ | $: \mathbf{Base} \to \mathbf{U}$ | Universes of presheaves. Cumulativity: $\mathsf{PSh}_i \subseteq \mathsf{U}$. Supports ETT. |
| | | We can only eliminate from $\mathsf{PSh}_i$ to $\mathsf{PSh}_i$. |
| $\mathbf{Cat}_i$ | $: \mathbf{PSh}_i$ | $:=$ *type of categories in* $\mathsf{PSh}_i$ |
| $\mathbf{In}$ | $: \mathbf{Cat}_i \to \mathbf{U}$ | "Permission token" for working in presheaves over some $\mathbb{C} : \mathsf{Cat}_i$. |
| $\mathbf{base}$ | $: \mathbf{In}\,\mathbb{C} \to \mathbf{Base}$ | "Using the permission". |

We use type-in-type everywhere for simplicity, i.e. $\mathsf{U} : \mathsf{U}$ and $\mathsf{PSh}_i : \mathsf{PSh}_i$.

# Basic things we can do

$$\mathsf{U} : \mathsf{U} \qquad \mathsf{Base} : \mathsf{U} \qquad \mathbf{1} : \mathsf{Base} \qquad \mathsf{PSh} : \mathsf{Base} \to \mathsf{U}$$

$$\mathsf{Cat}_i : \mathsf{PSh}_i := \textit{type of cats in } \mathsf{PSh}_i \qquad \mathsf{In} : \mathsf{Cat}_i \to \mathsf{U} \qquad \mathsf{base} : \mathsf{In}\,\mathbb{C} \to \mathsf{Base}$$

$\mathsf{PSh}_1$ is a universe supporting ETT (semantically: universe of sets).

# Basic things we can do

$$\mathsf{U} : \mathsf{U} \qquad \mathsf{Base} : \mathsf{U} \qquad \mathbf{1} : \mathsf{Base} \qquad \mathsf{PSh} : \mathsf{Base} \to \mathsf{U}$$

$$\mathsf{Cat}_i : \mathsf{PSh}_i := \textit{type of cats in } \mathsf{PSh}_i \qquad \mathsf{In} : \mathsf{Cat}_i \to \mathsf{U} \qquad \mathsf{base} : \mathsf{In}\,\mathbb{C} \to \mathsf{Base}$$

$\mathsf{PSh}_1$ is a universe supporting ETT (semantically: universe of sets).

We can define some $\mathbb{C} : \mathsf{Cat}_1$, where $\mathsf{Obj}(\mathbb{C}) : \mathsf{PSh}_1$.

# Basic things we can do

$$\mathsf{U} : \mathsf{U} \qquad \mathsf{Base} : \mathsf{U} \qquad \mathbf{1} : \mathsf{Base} \qquad \mathsf{PSh} : \mathsf{Base} \to \mathsf{U}$$

$$\mathsf{Cat}_i : \mathsf{PSh}_i := \textit{type of cats in } \mathsf{PSh}_i \qquad \mathsf{In} : \mathsf{Cat}_i \to \mathsf{U} \qquad \mathsf{base} : \mathsf{In}\,\mathbb{C} \to \mathsf{Base}$$

$\mathsf{PSh}_1$ is a universe supporting ETT (semantically: universe of sets).

We can define some $\mathbb{C} : \mathsf{Cat}_1$, where $\mathsf{Obj}(\mathbb{C}) : \mathsf{PSh}_1$.

Now, under the assumption of $i : \mathsf{In}\,\mathbb{C}$, we can form the universe $\mathsf{PSh}_{(\mathsf{base}\,i)}$, which is semantically the universe of presheaves over $\mathbb{C}$.

# Basic things we can do

$$\mathsf{U} : \mathsf{U} \qquad \mathsf{Base} : \mathsf{U} \qquad \mathbf{1} : \mathsf{Base} \qquad \mathsf{PSh} : \mathsf{Base} \to \mathsf{U}$$

$$\mathsf{Cat}_i : \mathsf{PSh}_i := \textit{type of cats in } \mathsf{PSh}_i \qquad \mathsf{In} : \mathsf{Cat}_i \to \mathsf{U} \qquad \mathsf{base} : \mathsf{In}\,\mathbb{C} \to \mathsf{Base}$$

$\mathsf{PSh}_1$ is a universe supporting ETT (semantically: universe of sets).

We can define some $\mathbb{C} : \mathsf{Cat}_1$, where $\mathsf{Obj}(\mathbb{C}) : \mathsf{PSh}_1$.

Now, under the assumption of $i : \mathsf{In}\,\mathbb{C}$, we can form the universe $\mathsf{PSh}_{(\mathsf{base}\,i)}$, which is semantically the universe of presheaves over $\mathbb{C}$.

*Syntax sugar:* we'll omit base in the following.

At this point, we have no interesting interaction between $\mathsf{PSh}_1$ and $\mathsf{PSh}_i$.

# Example: embedding pure lambda calculus

A **second-order model of pure LC** in $\mathsf{PSh}_i$ consists of:

$$
\begin{aligned}
\mathsf{Tm} &: \mathsf{PSh}_i \\
\mathsf{lam} &: (\mathsf{Tm} \to \mathsf{Tm}) \to \mathsf{Tm} \\
–\$– &: \mathsf{Tm} \to \mathsf{Tm} \to \mathsf{Tm} \\
\beta \quad &: \mathsf{lam}\, f \,\$\, t = f\ t \\
\eta \quad &: \mathsf{lam}\,(\lambda x.\, t \,\$\, x) = t
\end{aligned}
$$

We define $\mathsf{SMod}_i : \mathsf{PSh}_i$ as the above $\Sigma$-type.

# Example: embedding pure lambda calculus

A **first-order model of pure LC** consists of:

- A category of contexts and substitutions with $\mathsf{Con} : \mathsf{PSh}_i$, $\mathsf{Sub} : \mathsf{Con} \to \mathsf{Con} \to \mathsf{PSh}_i$ and terminal object $\bullet$.
- $\mathsf{Tm} : \mathsf{Con} \to \mathsf{PSh}_i$, plus a term substitution operation.
- A context extension operation $-\triangleright : \mathsf{Con} \to \mathsf{Con}$ such that $\mathsf{Sub}\,\Gamma\,(\Delta\triangleright) \simeq \mathsf{Sub}\,\Gamma\,\Delta \times \mathsf{Tm}\,\Gamma$.
- A natural isomorphism $\mathsf{Tm}\,(\Gamma\triangleright) \simeq \mathsf{Tm}\,\Gamma$ whose components are $\lambda$ and application.

We define $\mathsf{FMod}_i : \mathsf{PSh}_i$ as the above $\Sigma$-type.

FMod is mechanically derivable from SMod.[1]

[1] Ambrus Kaposi & Szumi Xie: *Second-Order Generalised Algebraic Theories*.

# Example: embedding pure lambda calculus

**GLF Axiom 1**

Assuming $M : \mathsf{FMod}_i$ and $j : \mathsf{In}\, M$, we have $\mathsf{S}_j : \mathsf{SMod}_j$.
(In "In $M$" we implicitly convert $M$ to its underlying category.)

Now we have 2LTT over $M$ inside $\mathsf{PSh}_j$:

- ETT type formers in $\mathsf{PSh}_j$ comprise the outer level.
- $S_j$ comprises the inner level.

Y-combinator as example:

$$\begin{aligned} &\mathsf{YC} : \mathsf{Tm}_{\mathsf{S}_j} \\ &\mathsf{YC} := (\mathsf{lam}_{\mathsf{S}_j}(\lambda x.\, x \,\$_{\mathsf{S}_j}\, x))\,\$_{\mathsf{S}_j}\,(\mathsf{lam}_{\mathsf{S}_j}(\lambda f.\, \mathsf{lam}_{\mathsf{S}_j}(\lambda x.\, f \,\$_{\mathsf{S}_j}\,(x \,\$_{\mathsf{S}_j}\, x)))) \end{aligned}$$

With a reasonable amount of sugar:

$$\begin{aligned} &\mathsf{YC} : \mathsf{Tm}_{\mathsf{S}_j} \\ &\mathsf{YC} := (\mathsf{lam}\, x.\, x\, x)\,(\mathsf{lam}\, f.\, \mathsf{lam}\, x.\, f\,(x\, x)) \end{aligned}$$

- More generally, we have the previous axiom for every second-order generalized algebraic theory.
- Hence: all 2LTTs are syntactic fragments of GLF.
- Moreover, for each 2LTT, the semantics of GLF restricts to the standard presheaf semantics of the 2LTT.

# Yoneda embedding

## GLF Axiom: Yoneda embedding for pure LC

Assuming $M : \mathsf{FMod}_i$, we have

$$
\begin{aligned}
\mathsf{Y} : \mathsf{Con}_M &\to ((j : \mathsf{In}_M) \to \mathsf{PSh}_j) \\
\mathsf{Y} : \mathsf{Sub}_M\,\Gamma\,\Delta &\simeq ((j : \mathsf{In}_M) \to \mathsf{Y}\,\Gamma\,j \to \mathsf{Y}\,\Delta\,j) \\
\mathsf{Y} : \mathsf{Tm}_M\,\Gamma &\simeq ((j : \mathsf{In}_M) \to \mathsf{Y}\,\Gamma\,j \to \mathsf{Tm}_{\mathsf{S}_j})
\end{aligned}
$$

such that Y preserves empty context and context extension up to iso:

$$
\begin{aligned}
\mathsf{Y}\bullet j &\simeq \top \\
\mathsf{Y}\,(\Gamma \triangleright)\,j &\simeq \mathsf{Y}\,\Gamma\,j \times \mathsf{Tm}_{\mathsf{S}_j}
\end{aligned}
$$

and Y preserves all other structure strictly.

*Notation*: we write $\Lambda$ for inverses of Y.

# LC examples, sugar

$\mathsf{Y}$ and $\Lambda$ allow ad-hoc switching between first-order and second-order notation. Let's redefine some operations using second-order notation:

$$\begin{array}{ll} \mathsf{id} : \mathsf{Sub}_M\,\Gamma\,\Gamma & \mathsf{comp} : \mathsf{Sub}_M\,\Delta\,\Theta \to \mathsf{Sub}_M\,\Gamma\,\Delta \to \mathsf{Sub}_M\,\Gamma\,\Theta \\ \mathsf{id} := \Lambda\,(\lambda\,j\,\gamma.\,\gamma) & \mathsf{comp}\,\sigma\,\delta := \Lambda\,(\lambda\,j\,\gamma.\,\mathsf{Y}\,\sigma\,(\mathsf{Y}\,\delta\,\gamma\,j)\,j \end{array}$$

With reasonable amount of sugar:

$$\mathsf{id} := \Lambda\,\gamma.\,\gamma \qquad \mathsf{comp}\,\sigma\,\delta := \Lambda\,\gamma.\,\mathsf{Y}\,\sigma\,(\mathsf{Y}\,\delta\,\gamma)$$

Or even:

$$\mathsf{comp}\,\sigma\,\delta := \Lambda\,\gamma.\,\sigma\,(\delta\,\gamma)$$

Example for "pattern matching" notation:

$$\begin{array}{l} \mathsf{wk} : \mathsf{Sub}_M\,(\Gamma \triangleright)\,\Gamma \\ \mathsf{wk} := \Lambda\,(\gamma,\,\alpha).\,\gamma \end{array}$$

# Second-order named notation

- When working with CwF-s, De Bruijn indices and substitutions can be hard to read.
- Handwaved "named" binders in CwFs have been used a couple of times.
- GLF provides a rigorous implementation of such notation.

# Example: type theoretic gluing

This is a model construction which looks awful in explicit CwF notation.[2] We assume a weak CwF morphism $F : S \to M$ between two models of a type theory. We define a displayed model $P$ lying over $S$.

[2]Kaposi, Huber, Sattler: *Gluing for Type Theory*.