# THE ESSENCE OF ELABORATION

**joint work with Andrej Bauer**

MEETING ON SYNTAX
AND SEMANTICS OF TYPE THEORY

**Anja Petković Komel**

STOCKHOLM, 20 MAY 2022

# WHAT IS ELABORATION?

**Saroupille** @Saroupille · 7 Mar

Replying to @andrejbauer

Going from the user syntax to the kernel syntax? A bit similar to the compilation process actually.

1 2

**Bernardo Toninho** @bernpton · 7 Mar

Replying to @andrejbauer

Don't know if I qualify, but elaboration to me is going from a surface "practical" syntax (implicits / omitting type annotations in binders / universe levels) to some core fully explicit representation, which usually involves higher-order unification (but not necessarily).

1 9

**Henri Tuhola** @HenriTuhola · 7 Mar

Replying to @andrejbauer

It's a form of type checking where you also "elaborate" the term from one syntax into another syntax.

**Brendan Zabarauskas (@brendan@types.pl)** @brendanzab · 7 Mar

Replying to @andrejbauer

It's the process of expressing implicit, overloaded, context sensitive information in a surface language a simpler, more explicit core language, while checking for any mistakes in the process. Kind of like type checking, type inference, and compilation pass rolled into one.

1 9

**Elaboration =**

**Type checking + compilation ?**

DISCLAIMER: NOT ALL ANSWERS ARE INCLUDED

**Thorsten Altenkirch** @taooftypes · 9 Mar
Replying to @andrejbauer
Translating from a human readable notation into a core calculus by filling in inferable parts.
3

**Martin Escardo** @EscardoMartin · 7 Mar
Replying to @andrejbauer
Elaboration is the process of trying to automatically figure out the information that the mathematician using the proof checker left out deliberately because it is considered culturally obvious in the wide mathematical community.
14

**Jon Sterling** @jonmsterling · 7 Mar
Replying to @andrejbauer
It is the process of transforming things that we write down or type in (e.g. raw syntax or code) into (representations of) the mathematical objects we are talking about.
1 9

**Elaboration =**

**Transformation into mathematical objects?**

**Figuring out missing (mathematical) context information?**

DISCLAIMER: NOT ALL ANSWERS ARE INCLUDED

**Esa Pulkkinen** @PulkkinenEsa · 7 Mar

Replying to @andrejbauer

My understanding is elaboration refers to transition from high level of abstraction to lower one by incrementally adding missing details.
But my impression on this could be mistaken, I do still remember I was confused when I first heard of the concept.

4

**???**

**Confused, frustrated.**

**We don’t really know.**

**andgate (paranormal software detective)** @the_andgate · 7 Mar

Reading papers on type theory is very frustrating, because type theorists almost never provide definitions for their esoteric terms and phrases. “Elaboration” being the most damaging example.

1

# IDEA OF ELABORATION:

## Adding missing information

- **Adding missing types**

$$\frac{\vdash A\ \text{type} \qquad \vdash B\ \text{type} \qquad x : A \vdash e : B}{\vdash \lambda(x.e) : A \to B} \quad \Rightarrow \quad \frac{\vdash A\ \text{type} \qquad \vdash B\ \text{type} \qquad x : A \vdash e : B}{\vdash \lambda(A, B, x.e) : A \to B}$$

- **Adding missing evidence**

**(termination checker, universe levels)**

# TERMINOLOGY

- **Elaboration map** - a map $\ell : T \to S$ from an economic type theory to a fully annotated type theory
- **Elaborator** - an algorithm performing adding information (related to type-checking)

# QUESTIONS TO ANSWER

- What are the properties of the economic syntax/type theory?
- What are the properties of the fully annotated (kernel) syntax/type theory?
- What is an elaboration map?
- What is the input and output of elaborator?
- How does elaborator relate to type checking?

# THE ESSENCE OF ELABORATION

retrogression
transformation
(conservative)

$$\begin{tikzcd} S \arrow[r, bend left, "r"] & T \arrow[l, bend left, "\ell"] \end{tikzcd}$$

standard type theory
(kernel)

finitary type theory
(economic)

elaboration map

# FINITARY TYPE THEORY

A **finitary type theory** [HB21] is a formal deductive system that consists of:

- A **signature** of symbols.
- 4 kinds of judgements.

$A\ \mathsf{type} \qquad a : A \qquad A \equiv B \text{ by } \star \qquad a \equiv b : A \text{ by } \star$

- **Boundaries** (for every judgement kind).

$\square\ \mathsf{type} \qquad \square : A \qquad A \equiv B \text{ by } \square \qquad a \equiv b : A \text{ by } \square$

- Hypothetical judgements and boundaries.

variable context +
metavariable context

$\Gamma \vdash \mathcal{J} \qquad \Gamma \vdash \mathcal{B}$

# RULES OF FINITARY TYPE THEORY

A **finitary type theory** [HB21] is a formal deductive system that consists of:

- **Structural rules**: Variable rule, reflexivity, symmetry and transitivity of equations etc.
- **Specific rules**:

**Object rules**

$$\frac{\vdash A \text{ type} \qquad \vdash B \text{ type}}{\vdash A \to B \text{ type}}$$

$$\frac{\vdash A \text{ type} \qquad \vdash B \text{ type} \qquad x : A \vdash e : B}{\vdash \lambda(x.e) : A \to B}$$

**Equality rules**

$$\frac{}{\vdash \mathrm{N} \equiv \mathbb{N}}$$

$$\frac{\vdash A \text{ type} \qquad \vdash B \text{ type} \qquad \vdash a : A \qquad \vdash b : B}{\vdash \mathrm{fst}(\mathrm{pair}(a,b)) \equiv a : A}$$

- **Congruence rules** (for every object rule).

Such that the rules have well-formed boundaries (presuppositivity).

# SYMBOL RULES

## Compare the two rules.

$$\frac{\vdash A \ \mathsf{type} \qquad \vdash B \ \mathsf{type} \qquad \vdash p : A \times B}{\vdash \mathrm{fst}(p) : A}$$

Better for user input.

$$\frac{\vdash A \ \mathsf{type} \qquad \vdash B \ \mathsf{type} \qquad \vdash p : A \times B}{\vdash \mathrm{fst}(A, B, p) : A}$$

Faithfully records the (proof-relevant parts of) the premises.

# STANDARD TYPE THEORY

A type theory is **standard** if every object rule is a symbol rule and every symbol has exactly one symbol rule.

Standard type theories are **well behaved**:

- inversion
- uniqueness of typing

# THE ESSENCE OF ELABORATION

retrogression
transformation
(conservative)

$$\begin{tikzcd} S \arrow[rr, bend left, "r"] & & T \arrow[ll, bend left, "\ell"] \end{tikzcd}$$

standard type theory
(kernel)

elaboration map

finitary type theory
(economic)

# SYNTACTIC TRANSFORMATION

**Recall from Andrej Bauer's talk:**

$$
\begin{array}{cccc}
f : & U & \longrightarrow & T \\
 & S & \longmapsto & e
\end{array}
$$

a symbol from signature of U

a well-formed expression in T

**The syntactic transformation f acts on expressions e' in U to produce expressions $f_*e'$ in T.**

**Syntactic transformations form a relative monad for syntax .**

# TYPE-THEORETIC TRANSFORMATION

A **type-theoretic transformation** is a syntactic transformation $f : U \to T$ such that for every **specific rule**

$$\frac{P_1 \quad \cdots \quad P_n}{\vdash \mathcal{J}}$$

in U there is a derivation of

$$\frac{f_* P_1 \quad \cdots \quad f_* P_n}{\vdash f_* \mathcal{J}}$$

this is data
of the transformation

in T.

Type-theoretic transformations **preserve derivability**.

# CONSERVATIVE TRANSFORMATION

start with a derivable boundary

act on it with f to get a derivable boundary

$$
\begin{tikzcd}
U \arrow[r, "f"] & T \\
\mathcal{B} \arrow[r, dashed] & f_* \mathcal{B} \arrow[d, dashed] \\
\mathcal{B}\,\boxed{e'} & (f_* \mathcal{B})\,\boxed{e} \arrow[l, dashed]
\end{tikzcd}
$$

if it can be (derivably) filled with a head

The original could also be filled (derivably)

# THE ESSENCE OF ELABORATION

retrogression
transformation
(conservative)

$$\begin{tikzcd} S \arrow[r, bend left, "r"] & T \arrow[l, bend left, "\ell"] \end{tikzcd}$$

standard type theory
(kernel)

elaboration map

finitary type theory
(economic)

# ELABORATION MAP

$$\begin{tikzcd} S \arrow[r, bend left, "r"] & T \arrow[l, bend left, "\ell"] \end{tikzcd}$$

Side note:
elaboration map
works uniformly
on contexts
and boundaries
$\ell(\Gamma', \mathscr{B}', \mathscr{D})$

Elaboration map
takes a derivation!

$$\Gamma' \vdash \mathscr{J}' \longleftarrow\!\shortmid \quad \frac{\mathscr{D}}{\Gamma \vdash \mathscr{J}}$$

**... such that $r_*(\Gamma' \vdash \mathscr{J}') = \Gamma \vdash \mathscr{J}$.**

**$\ell$ is a section of r.**

Elaboration map
preserves derivability.

# THE ESSENCE OF ELABORATION

retrogression
transformation
(conservative)

$$
\begin{tikzcd}
S \arrow[r, bend left, "r"] & T \arrow[l, bend left, "\ell"]
\end{tikzcd}
$$

standard type theory
(kernel)

finitary type theory
(economic)

elaboration map

**Retrogression transformation is surjective on derivable judgements.**

$$\begin{tikzcd} S \arrow[r, bend left, "r"] & T \arrow[l, bend left, "\ell"] \end{tikzcd}$$

**Elaboration map is unique**

**up-to judgemental equality.**

$$\Gamma' \vdash e' \equiv \ell(\mathscr{D}) : A'$$

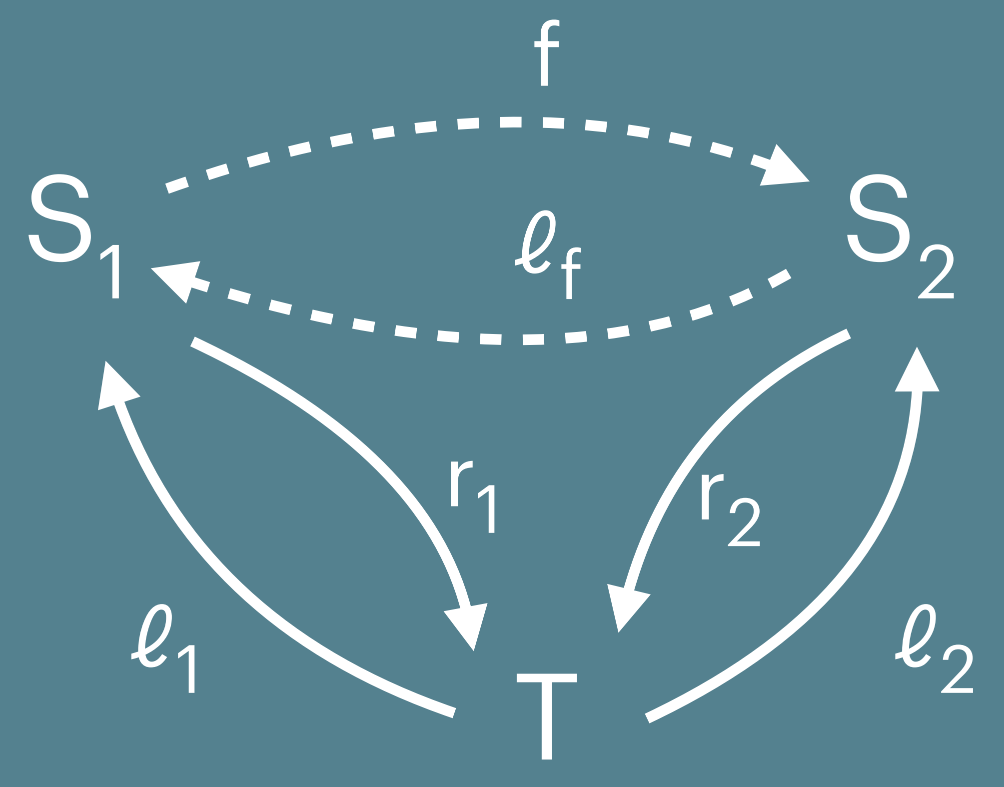
Another
elaboration
candidate

$$\begin{tikzcd} S \arrow[rr, bend left, "r"] & & T \arrow[ll, bend left, "\ell"] \end{tikzcd}$$

$$\frac{\mathscr{D}}{\Gamma \vdash e : A}$$

# UNIVERSAL PROPERTY

**Elaboration map satisfies the following universal property:**




$$r_2 \circ f = r_1$$

**f is conservative and unique up-to judgemental equality.**

# AN ELABORATION THEOREM

Every finitary type theory has "an elaboration".

**For every finitary type theory T there exists a standard type theory S with a retrogression transformation $r : S \to T$ and elaboration map $\ell : T \to S$.**

# PROOF IDEA

## (Elaboration theorem)

For every **specific object rule** $Ri$ = $\dfrac{p_1 \quad \cdots \quad p_n}{\vdash \mathscr{B}\ \boxed{e}}$

introduce a symbol S_(i, $Ri$).

i is the index of the rule
in the signature

**Retrogression transformation** **(syntactic part)**    $r :\ S\_(i, Ri) \longmapsto e$

# PROOF IDEA

## (Elaboration theorem)

Specific rules of standard type theory S:

Specific object rules of T ➡ Symbol rules in S

Specific equality rules of T ➡ Specific equality rules in S

+ close under derivability

Do this inductively on the ordering of rules.

# PROOF IDEA

## (Elaboration theorem)

Define elaboration map inductively.

Prove desired properties of retrogression transformation (conservativity, type-theoretic transformation) and of elaboration map (section, preserves derivability).

# AN ELABORATION THEOREM

**For every finitary type theory T there exists a standard type theory S with a retrogression transformation r : S → T and elaboration map ℓ : T → S .**

But S has a looooot of specific equality rules!

Recall: universal property

# ELABORATOR: ALGORITHM

# ELABORATOR

in finitary type theory

* in equation-free meta context

**Elaborator: an algorithm**

takes : judgement J

outputs: a derivable elaborated judgement J′ if it exists, or reports there is none

* strongly derivable

in standard type theory

**An elaborator, if it exists, is *computable* for our chosen type theory.**

# CHECKING

**Type-checking:**

Derivable boundary!

* in equation-free meta context

Check that a term $\mathbf{a}$ has type $\mathbf{A}$.

Check that the head $\mathbf{a}$ fits the boundary $\boxed{\phantom{x}} : \mathbf{A}$.

**Equality-checking:**

Check that $\mathbf{A} \equiv \mathbf{B}$ (or $\mathbf{a} \equiv \mathbf{b} : \mathbf{A}$ ).

Check that the head $\bigstar$ fits the boundary $\mathbf{A} \equiv \mathbf{B}$ by $\boxed{\phantom{x}}$ (or $\mathbf{a} \equiv \mathbf{b} : \mathbf{A}$ by $\boxed{\phantom{x}}$ ).

**Checking:**

Check that the head $\mathbf{e}$ fits the boundary $\mathscr{B}$ to get the judgement $\mathscr{B}\boxed{\mathbf{e}}$.

# EXISTENCE OF ELABORATOR

**T has an elaborator if and only if T has decidable (judgement) checking.**

Elaborator is the most general checking algorithm for T,
if any exists.

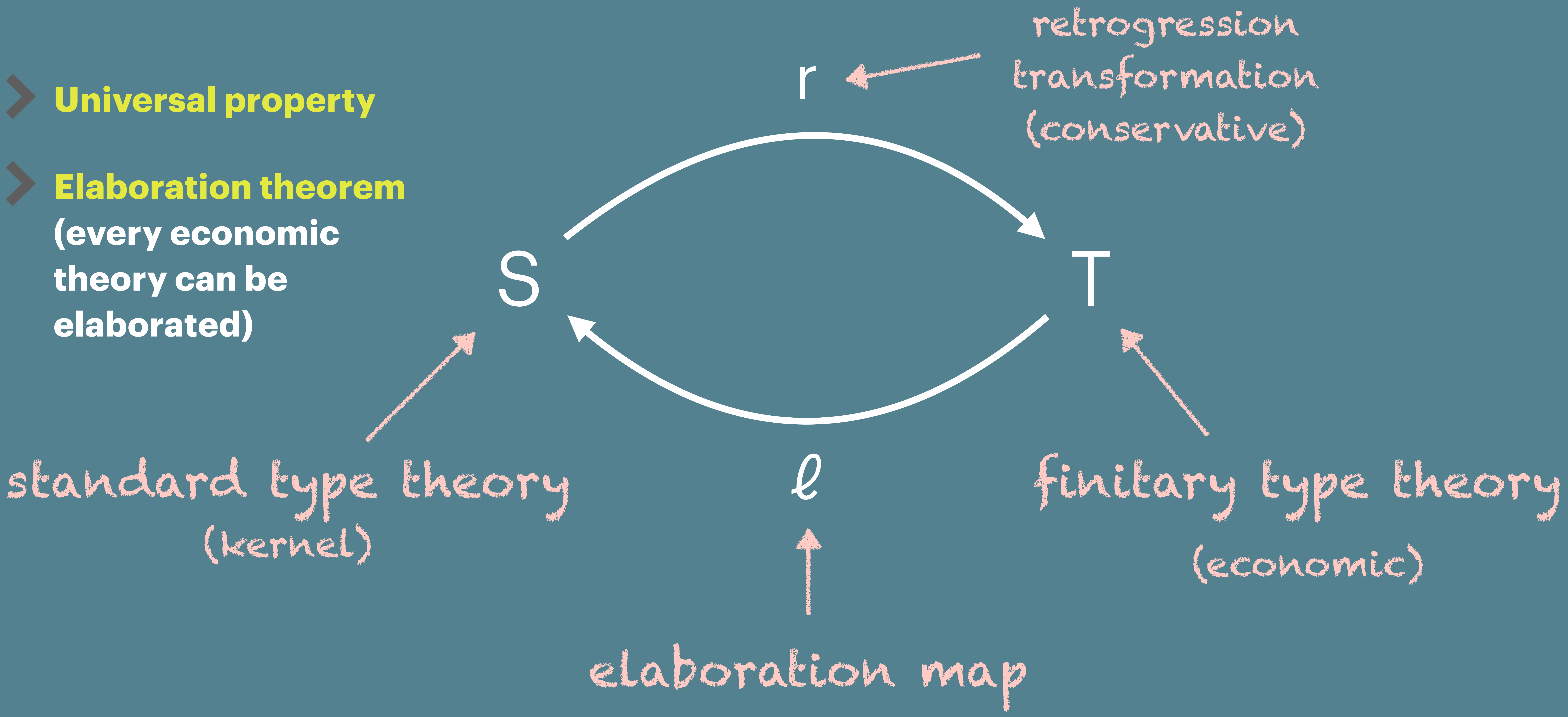
**If a standard type theory has decidable equality checking, then it has decidable checking.**

**If a finitary type theory has decidable (equality) checking, so does its elaborated standard type theory.**

Note: the converse does not hold!

# THE ESSENCE OF ELABORATION

- **Universal property**
- **Elaboration theorem** **(every economic theory can be elaborated)**



We have an example of a type theory such that:

(a) Checking is semidecidable.

(b) Equality checking is decidable.

(c) Checking is not decidable.

$D \subseteq \mathbb{N} \times \mathbb{N}$ a computable subset, such that

$\pi_1(D) = \{ n \in \mathbb{N} \mid \exists\, m \in \mathbb{N}.\ (n, m) \in D\}$ is semidecidable, not computable.

A signature is given by $(A\_n : \mathrm{Type})\_n \in \mathbb{N}$

Rules: for every $(n, m) \in D$ $\quad$ $R\_(n, m) = \dfrac{}{\vdash A\_n \ \mathrm{type}}$

Derivable boundary $\quad [\,] \vdash \square$ type

Check if $\quad [\,] \vdash A\_n$ type $\quad$ is derivable?