# Eliminating **reflection** through *reflection*

Théo **Winterhalter**

joint work with

Matthieu **Sozeau**
Nicolas **Tabareau**

# Different notions of equality

## Conversion

Extends the notion of β-equality

```
(λx.t) u ≡ t[x← u]
```

## Identity types

To handle equalities within type theory

```
refl u : u = u
```

# Different notions of equality

## Conversion

Extends the notion of β-equality

```
(λx.t) u ≡ t[x← u]
```

## Identity types

To handle equalities within type theory

```
refl u : u = u
```

$\boxed{\text{If } u \equiv v \text{ then } \mathbf{refl}\ u : u = v}$

# Reflection

## Conversion

Extends the notion of β-equality

## Identity types

To handle equalities within type theory

# Reflection

## Conversion

Extends the notion of β-equality

## Identity types

To handle equalities within type theory

$$\frac{p : u = v}{u \equiv v}$$

# Example

# Example

$\mathtt{vec}_A : \mathtt{nat} \to \mathbf{Type}$

# Example

$\mathrm{vec}_A : \mathrm{nat} \to \mathbf{Type}$

$[] : \mathrm{vec}_A\ 0$

# Example

```
vecA : nat → Type

[] : vecA 0

cons : ∀ n, A → vecA n → vecA (S n)
```

# Example

```
vecA : nat → Type

[] : vecA 0

cons : ∀ n, A → vecA n → vecA (S n)

rev : ∀ {n m}, vecA n → vecA m → vecA (n + m)
```

# Example

```
vecA : nat → Type
[] : vecA 0
cons : ∀ n, A → vecA n → vecA (S n)

rev : ∀ {n m}, vecA n → vecA m → vecA (n + m)
rev [] acc ≐ acc
```

# Example

```
vecA : nat → Type

[] : vecA 0

cons : ∀ n, A → vecA n → vecA (S n)
```

```
rev : ∀ {n m}, vecA n → vecA m → vecA (n + m)

rev [] acc ≐ acc

rev (cons n a v) acc ≐ rev v (cons m a acc)
```

# Example

```
vecA : nat → Type

[] : vecA 0

cons : ∀ n, A → vecA n → vecA (S n)

rev : ∀ {n m}, vecA n → vecA m → vecA (n + m)

rev [] acc ≐ acc

rev (cons n a v) acc ≐ rev v (cons m a acc)
                                      vecA m
```

# Example

```
vecA : nat → Type

[] : vecA 0

cons : ∀ n, A → vecA n → vecA (S n)


rev : ∀ {n m}, vecA n → vecA m → vecA (n + m)

rev [] acc ≐ acc

rev (cons n a v) acc ≐ rev v (cons m a acc)
                             vecA (S m)
```

# Example

```
vecA : nat → Type

[] : vecA 0

cons : ∀ n, A → vecA n → vecA (S n)


rev : ∀ {n m}, vecA n → vecA m → vecA (n + m)

rev [] acc ≐ acc

rev (cons n a v) acc ≐ rev v (cons m a acc)

                       vecA (n + S m)
```

# Example

```
vecA : nat → Type

[] : vecA 0

cons : ∀ n, A → vecA n → vecA (S n)


rev : ∀ {n m}, vecA n → vecA m → vecA (n + m)

rev [] acc ≐ acc

rev (cons n a v) acc ≐ rev v (cons m a acc)
```

expected: $\mathrm{vec}_A\ (S\ n + m) \neq \mathrm{vec}_A\ (n + S\ m)$

# Example

```
vec_A : nat → Type

[] : vec_A 0

cons : ∀ n, A → vec_A n → vec_A (S n)


rev : ∀ {n m}, vec_A n → vec_A m → vec_A (n + m)

rev [] acc ≐ acc

rev (cons n a v) acc ≐ rev v (cons m a acc)
```

reflection ⇒ `vec_A (S n + m) ≡ vec_A (n + S m)`

# Intensional VS Extensional

$$\frac{p : u = v}{u \equiv v}$$

ETT = ITT + reflection

# Intensional VS Extensional

$$\frac{p : u = v}{u \equiv v}$$

ETT = ITT + reflection

What is the relation between the two?

# Intensional VS Extensional

What is the relation between the two?

**ETT** is *conservative* over **ITT + K + funext**

1995 Martin Hofmann

# Intensional VS Extensional

What is the relation between the two?

**ETT** is *conservative* over **ITT + K + funext**

1995 Martin Hofmann

# Intensional VS Extensional

What is the relation between the two?

**ETT** is *conservative* over **ITT + K + funext**

1995 Martin Hofmann

```
K : ∀ A (x : A) (e : x = x), e = refl x
```

# Intensional VS Extensional

What is the relation between the two?

**ETT** is *conservative* over **ITT + K + funext**

1995 Martin Hofmann

```
K : ∀ A (x : A) (e : x = x), e = refl x

funext : ∀ A B (f g : A → B), (∀ (x : A), f x = g x) → f = g
```

# Intensional VS Extensional

What is the relation between the two?

**ETT** can be *translated* to **ITT + K + funext + ?**

2005 Nicolas Oury

```
K : ∀ A (x : A) (e : x = x), e = refl x

funext : ∀ A B (f g : A → B), (∀ (x : A), f x = g x) → f = g
```

# Intensional VS Extensional

What is the relation between the two?

**ETT** can be *translated* to **ITT + K + funext + ?**

2005 Nicolas Oury

```
K : ∀ A (x : A) (e : x = x), e = refl x
funext : ∀ A B (f g : A → B), (∀ (x : A), f x = g x) → f = g
‘?’ : « heterogenous equality is a congruence for application »
```

# Intensional VS Extensional

What is the relation between the two?

**ETT** can be *translated* to **ITT + K + funext**

TODAY

```
K : ∀ A (x : A) (e : x = x), e = refl x

funext : ∀ A B (f g : A → B), (∀ (x : A), f x = g x) → f = g

'?' : « heterogenous equality is a congruence for application »
```

# Intensional VS Extensional

What is the relation between the two?

**ETT** can be *translated* to **ITT + K + funext**

TODAY

```
K : ∀ A (x : A) (e : x = x), e = refl x
funext : ∀ A B (f g : A → B), (∀ (x : A), f x = g x) → f = g
```

~~‘?’ : « heterogenous equality is a congruence for application »~~

# Intensional VS Extensional

What is the relation between the two?

**ETT** can be *translated* to **ITT + K + funext**

TODAY

# Intensional VS Extensional

## What is the relation between the two?

**ETT** can be *translated* to **ITT + K + funext**

TODAY

- **Minimal** (axiom-wise)
- **Constructive** (formalised in Coq)
- **Computes** (produces Coq terms)

# Fundamental difference

Oury

Hofmann / us

# Fundamental difference

## Oury

Minimal annotations

```
λ(x : A).t

t u
```

## Hofmann / us

# Fundamental difference

## Oury

Minimal annotations

$\lambda(x : A).t$

$t\ u$

## Hofmann / us

Fully annotated terms

$\lambda(x : A).B.t$

$t\ @^{(x:A).B}\ u$

# Fundamental difference

## Oury

Minimal annotations

$$\lambda(x : A).t$$

$$t\ u$$

## Hofmann / us

Fully annotated terms

$$\lambda(x : A).B.t$$

$$t\ @^{(x:A).B}\ u$$

Blocked β-reduction

$$(\lambda(x : A).B.t)\ @^{(x:A).B}\ u$$
$$\equiv t[x := u]$$

# Fundamental difference

## Oury

Minimal annotations

$$\lambda(x : A).t$$

$$t\ u$$

Free β-reduction

$$(\lambda(x : A).t)\ u$$
$$\equiv t[x := u]$$

## Hofmann / us

Fully annotated terms

$$\lambda(x : A).B.t$$

$$t\ @^{(x:A).B}\ u$$

Blocked β-reduction

$$(\lambda(x : A).B.t)\ @^{(x:A).B}\ u$$
$$\equiv t[x := u]$$

# Fundamental difference

## Oury

Free β-reduction

$$(\lambda(x : A).t)\ u \equiv t[x := u]$$

## Hofmann / us

Blocked β-reduction

$$(\lambda(x : A).B.t)\ @^{(x:A).B}\ u \equiv t[x := u]$$

# Fundamental difference

## Oury

Free β-reduction

$$(\lambda(x : A).x)\ u \equiv x[x := u]$$

## Hofmann / us

Blocked β-reduction

$$(\lambda(x : A).B.t)\ @^{(x:A).B}\ u \equiv t[x := u]$$

# Fundamental difference

## Oury

Free β-reduction

$$(\lambda(x : A).x)\ u \equiv u$$

## Hofmann / us

Blocked β-reduction

$$(\lambda(x : A).B.t)\ @^{(x:A).B}\ u \equiv t[x := u]$$

# Fundamental difference

## Oury

Free β-reduction

$$(\lambda(x : \mathtt{nat}).x)\ 0 \equiv 0$$

## Hofmann / us

Blocked β-reduction

$$(\lambda(x : A).B.t)\ @^{(x:A).B}\ u$$
$$\equiv t[x := u]$$

# Fundamental difference

## Oury

Free β-reduction

$$\frac{(\lambda(x : \mathtt{nat}).x)\ 0 \equiv 0}{\mathtt{nat} \to \mathtt{nat}}$$

## Hofmann / us

Blocked β-reduction

$$(\lambda(x : A).B.t)\ @^{(x:A).B}\ u$$

$$\equiv t[x := u]$$

# Fundamental difference

## Oury

Free β-reduction

$$\frac{(\lambda(x : \mathtt{nat}).x)\ 0 \equiv 0}{\begin{array}{c}\mathtt{nat} \to \mathtt{nat} \\ \equiv\ \mathtt{nat} \to \mathtt{bool}\end{array}}$$

## Hofmann / us

Blocked β-reduction

$$(\lambda(x : A).B.t)\ @^{(x:A).B}\ u$$
$$\equiv t[x := u]$$

# Fundamental difference

## Oury

Free β-reduction

$$\frac{(\lambda(x : \mathtt{nat}).x)\ 0 \equiv 0}{\mathtt{bool}}$$

## Hofmann / us

Blocked β-reduction

$$(\lambda(x : A).B.t)\ @^{(x:A).B}\ u$$
$$\equiv t[x := u]$$

# Fundamental difference

## Oury

Free β-reduction

$$\underbrace{(\lambda(x : \mathtt{nat}).x)\ 0}_{\mathtt{bool}} \equiv \underbrace{0}_{\mathtt{nat}}$$

## Hofmann / us

Blocked β-reduction

$$(\lambda(x : A).B.t)\ @^{(x:A).B}\ u \equiv t[x := u]$$

# Fundamental difference

## Oury

Free β-reduction

$$\underbrace{(\lambda(x : \mathtt{nat}).x)\ 0}_{\mathtt{bool}} \equiv \underbrace{0}_{\mathtt{nat}}$$

$\mathtt{bool} \neq \mathtt{nat}$

## Hofmann / us

Blocked β-reduction

$$(\lambda(x : A).B.t) @^{(x:A).B} u$$
$$\equiv t[x := u]$$

# Fundamental difference

## Oury

Free β-reduction

$$\underbrace{(\lambda(x : \mathtt{nat}).x)\ 0}_{\mathtt{bool}} \equiv \underbrace{0}_{\mathtt{nat}}$$

$$\mathtt{bool} \neq \mathtt{nat}$$

**No** Uniqueness of type
**OR**
**No** Subject reduction

## Hofmann / us

Blocked β-reduction

$$(\lambda(x : A).B.t)\ @^{(x:A).B}\ u$$
$$\equiv t[x := u]$$

# Fundamental difference

## Oury

Free β-reduction

$$\underbrace{(\lambda(x : \mathtt{nat}).x)\ 0}_{\mathtt{bool}} \equiv \underbrace{0}_{\mathtt{nat}}$$

$\mathtt{bool} \neq \mathtt{nat}$

**No** Uniqueness of type

**OR**

**No** Subject reduction

## Hofmann / us

Blocked β-reduction

$$(\lambda(x : A).B.t)\ @^{(x:A).B}\ u \equiv t[x := u]$$

**Uniqueness of type**

$\Gamma \vdash t : A$ and $\Gamma \vdash t : B$

$\Rightarrow \Gamma \vdash A \equiv B$

# Principle of the translation

ETT

ITT

# Principle of the translation

ETT

Typing derivation

ITT

Well typed term

# Principle of the translation

ETT

Typing derivation

$$\dfrac{\dfrac{\begin{array}{c}\vdots \\ \cdots\end{array} \quad \begin{array}{c}\vdots \\ \cdots\end{array}}{\cdots} \quad \begin{array}{c}\vdots \\ \cdots\end{array}}{\Gamma \vdash_x t : A}$$

$\longrightarrow$

ITT

Well typed term

$$\Gamma' \vdash_i t' : A'$$

# Principle of the translation

## ETT

Typing derivation

$$\dfrac{\dfrac{\begin{matrix}\vdots & \vdots\\ \dots & \dots\end{matrix}}{\dots} \qquad \begin{matrix}\vdots\\ \dots\end{matrix}}{\Gamma \vdash_x t : A}$$

$$\longrightarrow$$

## ITT

Well typed term

$$\Gamma' \vdash_i t' : A'$$

**Idea:** *Conversion* is translated to *transport*.

# Principle of the translation

## ETT

Typing derivation

$$\dfrac{\begin{array}{c}\vdots \\ \Gamma \vdash_x t : B\end{array} \qquad \begin{array}{c}\vdots \\ \Gamma \vdash_x B \equiv A\end{array}}{\Gamma \vdash_x t : A} \longrightarrow$$

## ITT

Well typed term

$$\Gamma' \vdash_i t' : B'$$

$$\Gamma' \vdash_i e : B' = A'$$

$$\Downarrow$$

$$\Gamma' \vdash_i \mathbf{transp}(e, t') : A'$$

**Idea:** *Conversion* is translated to *transport*.

# Principle of the translation

**ETT**

Typing derivation

$$\dfrac{\begin{matrix}\vdots & & \vdots\\ \Gamma \vdash_x t : B & & \Gamma \vdash_x B \equiv A\end{matrix}}{\Gamma \vdash_x t : A} \longrightarrow$$

**ITT**

Well typed term

$$\Gamma' \vdash_i t' : B'$$
$$\Gamma' \vdash_i e : B' = A'$$
$$\Downarrow$$
$$\Gamma' \vdash_i \mathbf{transp}(e, t') : A'$$

**Idea:** *Conversion* is translated to *transport*.

⇒ Coherence problems

# Heterogenous equality

$$a \;\; {}_{A}{=}_{B} \;\; b$$

# Heterogenous equality

$$a \ {}_{A}{=}_{B} \ b$$

$$\doteq \sum \ (p : A = B), \ \mathbf{transp}(p, a) = b$$

# Terms up to transport

$$\frac{\texttt{t} \sim \texttt{t'}}{\texttt{t} \sim \textbf{\texttt{transp}}\texttt{(e,t')}}$$

# Terms up to transport

$$\frac{\texttt{t} \sim \texttt{t'}}{\texttt{t} \sim \textbf{transp}\texttt{(e,t')}} \qquad \frac{\texttt{t} \sim \texttt{t'} \quad \texttt{A} \sim \texttt{A'} \quad \texttt{B} \sim \texttt{B'} \quad \texttt{u} \sim \texttt{u'}}{\texttt{t} \ @^{\texttt{(x:A).B}}\ \texttt{u} \sim \texttt{t'} \ @^{\texttt{(x:A').B'}}\ \texttt{u'}}$$

# Terms up to transport

$$\frac{\mathtt{t} \sim \mathtt{t'}}{\mathtt{t} \sim \mathbf{transp}(\mathtt{e},\mathtt{t'})} \qquad \frac{\mathtt{t} \sim \mathtt{t'} \quad \mathtt{A} \sim \mathtt{A'} \quad \mathtt{B} \sim \mathtt{B'} \quad \mathtt{u} \sim \mathtt{u'}}{\mathtt{t} \ @^{(\mathtt{x}:\mathtt{A}).\mathtt{B}}\ \mathtt{u} \sim \mathtt{t'} \ @^{(\mathtt{x}:\mathtt{A'}).\mathtt{B'}}\ \mathtt{u'}} \qquad \ldots$$

# Terms up to transport

$$\frac{t \sim t'}{t \sim \mathbf{transp}(e, t')} \qquad \frac{t \sim t' \quad A \sim A' \quad B \sim B' \quad u \sim u'}{t \ @^{(x:A).B}\ u \sim t' \ @^{(x:A').B'}\ u'} \qquad \ldots$$

**Invariant**

t is **translated** to t' with t ~ t'

# Terms up to transport

$$\frac{t \sim t'}{t \sim \mathbf{transp}(e, t')} \qquad \frac{t \sim t' \quad A \sim A' \quad B \sim B' \quad u \sim u'}{t \ @^{(x:A).B}\ u \sim t' \ @^{(x:A').B'}\ u'} \qquad \ldots$$

**Invariant**

t is **translated** to t' with $t \sim t'$

**Fundamental lemma**

Given Γ and $t \sim t'$, there exists a term p such that if $\Gamma \vdash_i t : A$ and $\Gamma \vdash_i t' : B$ then $\Gamma \vdash_x p : t \ {}_A{=}_B\ t'$.

# Translation

```
def Γ’ ∈ ⟦⊢x Γ⟧ ≐ ⊢i Γ’ × Γ ~ Γ’

def (Γ’,t’,A’) ∈ ⟦Γ ⊢x t : A⟧ ≐
Γ’ ⊢i t’ : A’ × Γ ~ Γ’ × A ~ A’ × t ~ t’
```

# Translation

```
def Γ' ∈ ⟦⊢x Γ⟧ ≐ ⊢i Γ' × Γ ~ Γ'

def (Γ',t',A') ∈ ⟦Γ ⊢x t : A⟧ ≐
Γ' ⊢i t' : A' × Γ ~ Γ' × A ~ A' × t ~ t'
```

```
if ⊢x Γ then ∑ Γ', Γ' ∈ ⟦⊢x Γ⟧
```

# Translation

$\mathsf{def}\ \Gamma' \in [\![\vdash_x \Gamma]\!] \doteq\ \vdash_i \Gamma' \times \Gamma \sim \Gamma'$

$\mathsf{def}\ (\Gamma',t',A') \in [\![\Gamma \vdash_x t : A]\!] \doteq$
$\Gamma' \vdash_i t' : A' \times \Gamma \sim \Gamma' \times A \sim A' \times t \sim t'$

$\mathsf{if}\ \vdash_x \Gamma\ \mathsf{then}\ \sum \Gamma',\ \Gamma' \in [\![\vdash_x \Gamma]\!]$

$\mathsf{if}\ \Gamma \vdash_x t : A\ \mathsf{then}$
$\forall\ \Gamma',\ \Gamma' \in [\![\vdash_x \Gamma]\!] \to \sum t'\ A',\ (\Gamma',t',A') \in [\![\Gamma \vdash_x t : A]\!]$

# Translation

$$\mathsf{def}\ \Gamma' \in [\![\vdash_x \Gamma]\!] \doteq\ \vdash_i \Gamma' \times \Gamma \sim \Gamma'$$

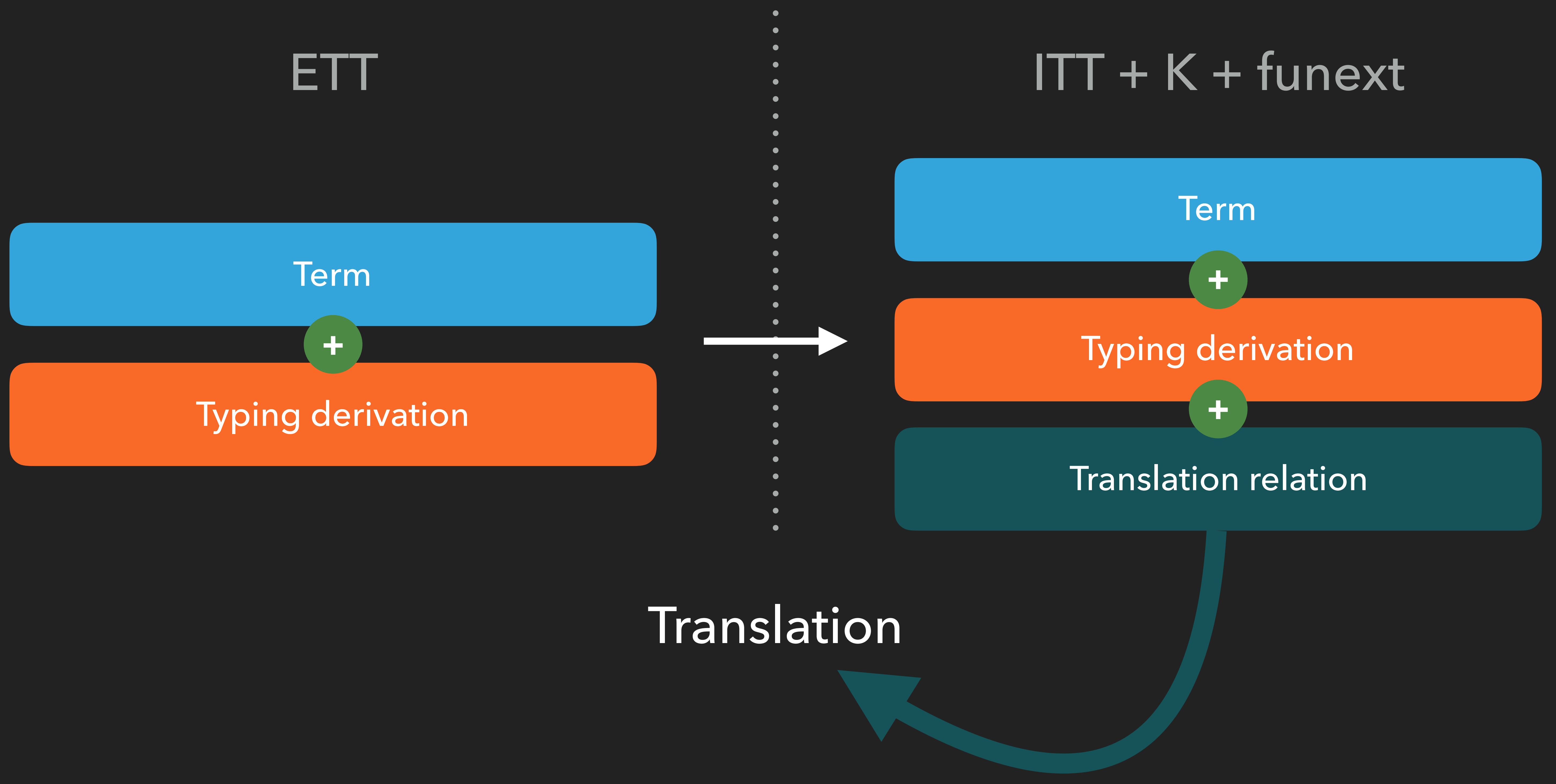
$$\mathsf{def}\ (\Gamma',t',A') \in [\![\Gamma \vdash_x t : A]\!] \doteq \\ \Gamma' \vdash_i t' : A' \times \Gamma \sim \Gamma' \times A \sim A' \times t \sim t'$$

$$\mathsf{if}\ \vdash_x \Gamma\ \mathsf{then}\ \sum \Gamma',\ \Gamma' \in [\![\vdash_x \Gamma]\!]$$

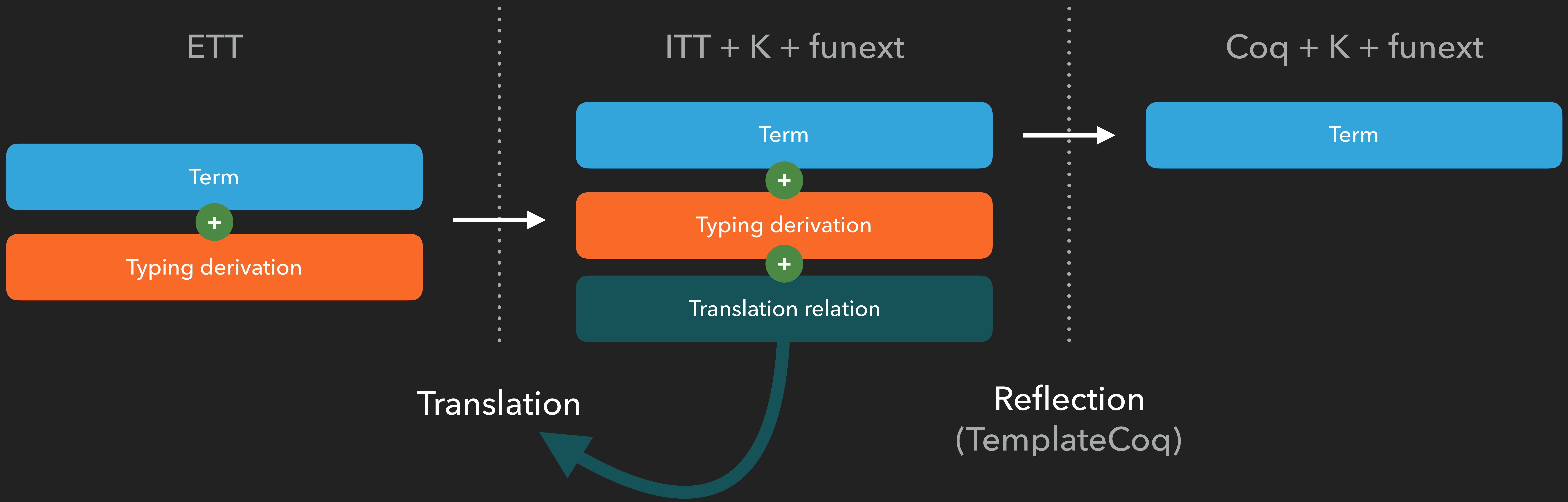
$$\mathsf{if}\ \Gamma \vdash_x t : A\ \mathsf{then} \\ \forall\ \Gamma',\ \Gamma' \in [\![\vdash_x \Gamma]\!] \to \sum t'\ A',\ (\Gamma',t',A') \in [\![\Gamma \vdash_x t : A]\!]$$

$$\mathsf{if}\ \Gamma \vdash_x t \equiv u : A\ \mathsf{then} \\ \forall\ \Gamma',\ \Gamma' \in [\![\vdash_x \Gamma]\!] \to \sum t'\ A'\ u'\ A''\ p,\ \Gamma' \vdash_i p : t'\ {}_{A'}{=}_{A''}\ u'$$

# Conclusion

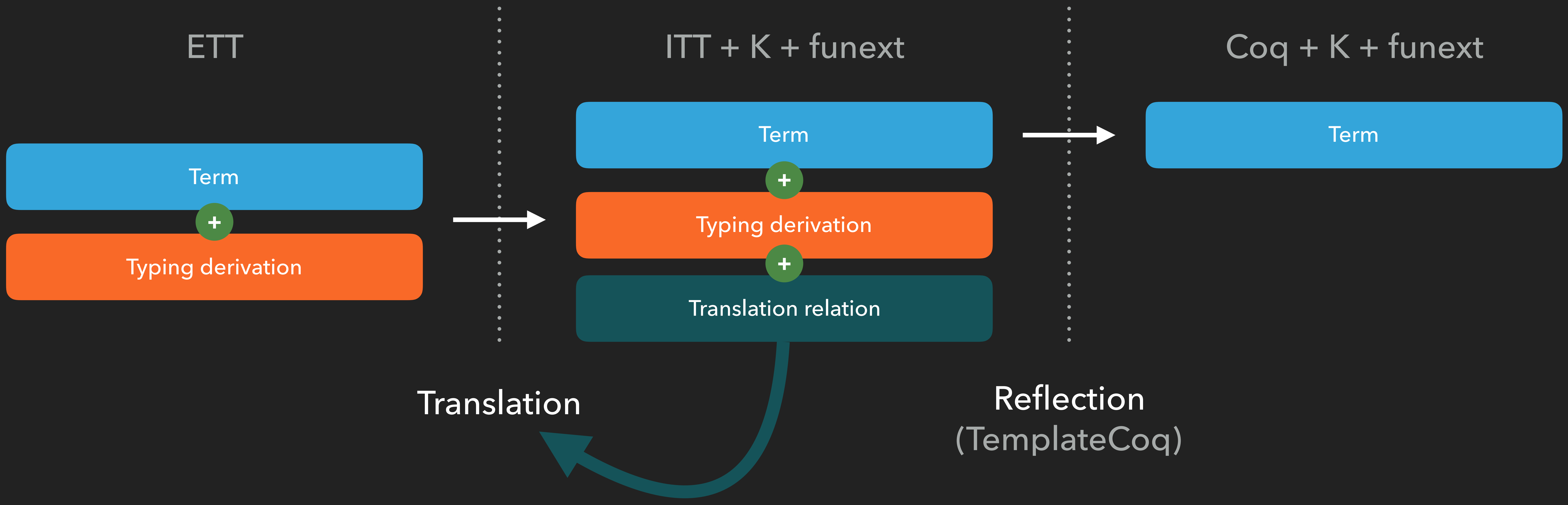

ETT
Term
+
Typing derivation
ITT + K + funext
Term
+
Typing derivation
+
Translation relation
Translation

# Conclusion



ETT
ITT + K + funext
Coq + K + funext
Term
+
Typing derivation
Term
+
Typing derivation
+
Translation relation
Term
Translation
Reflection
(TemplateCoq)

# Conclusion




https://github.com/TheoWinterhalter/**ett-to-itt**