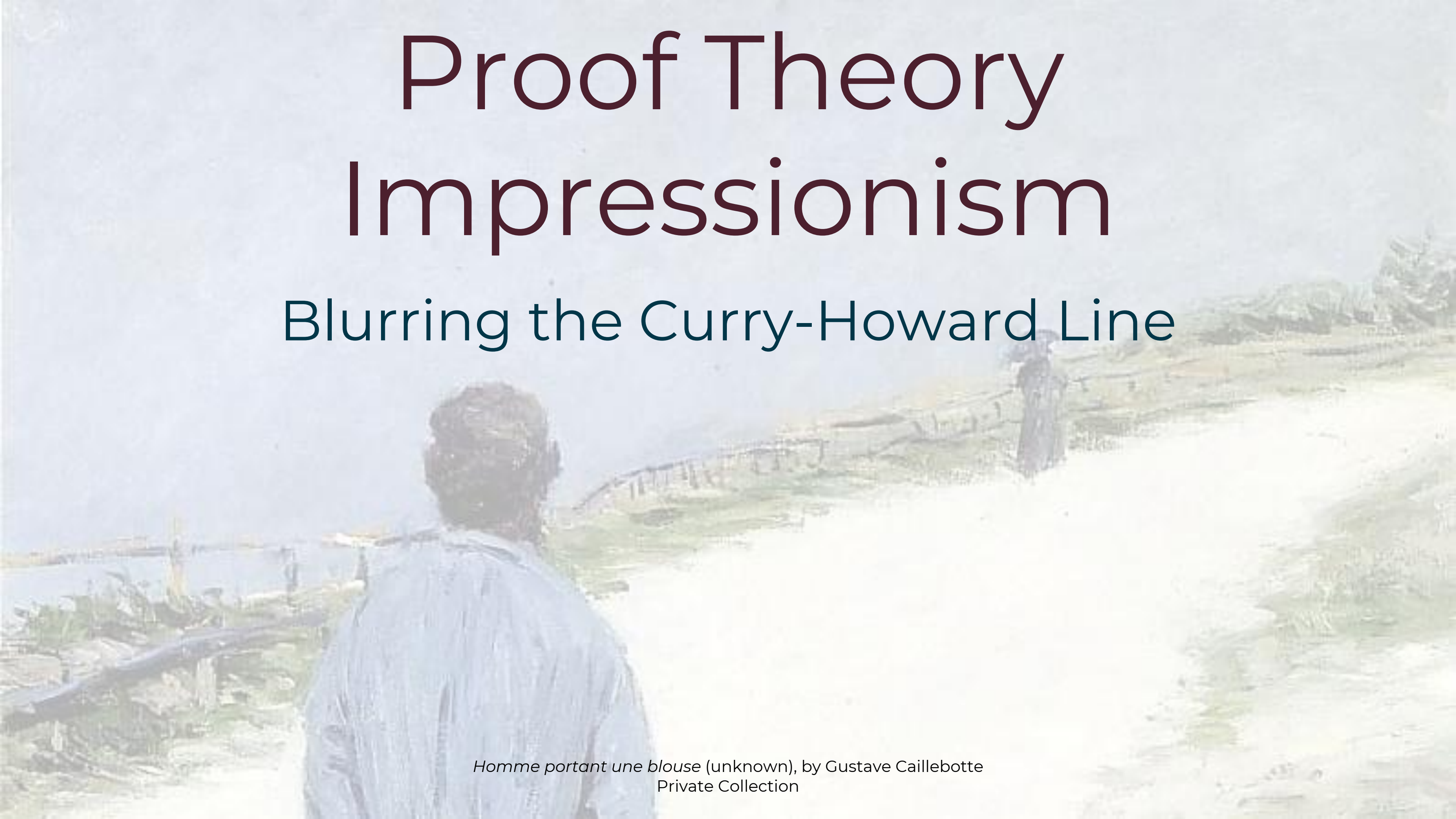
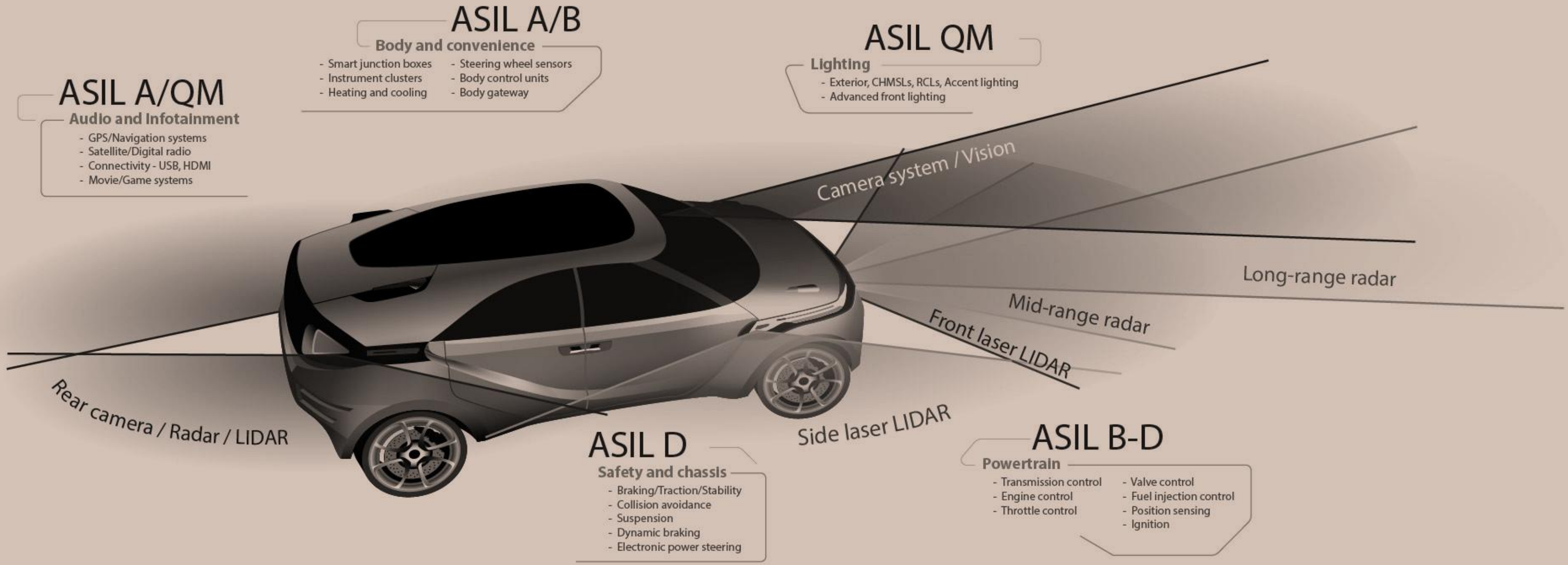


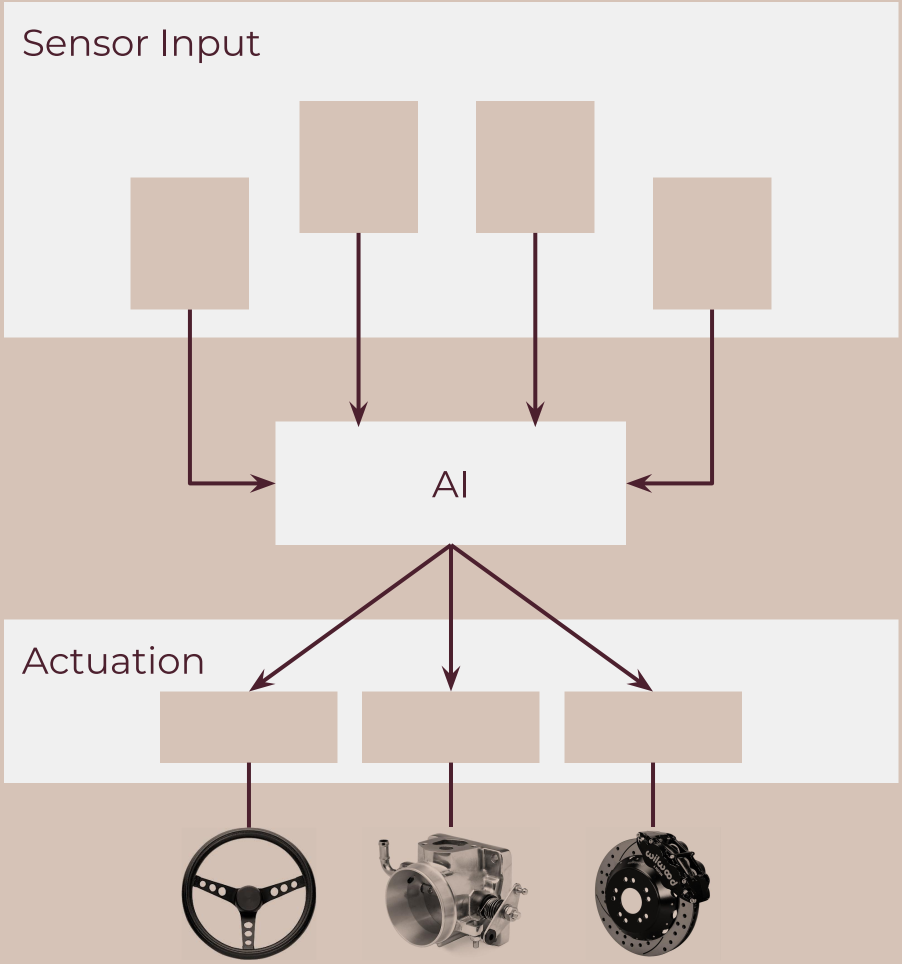
Proof Theory Impressionism

Blurring the Curry-Howard Line

Homme portant une blouse (unknown), by Gustave Caillebotte
Private Collection

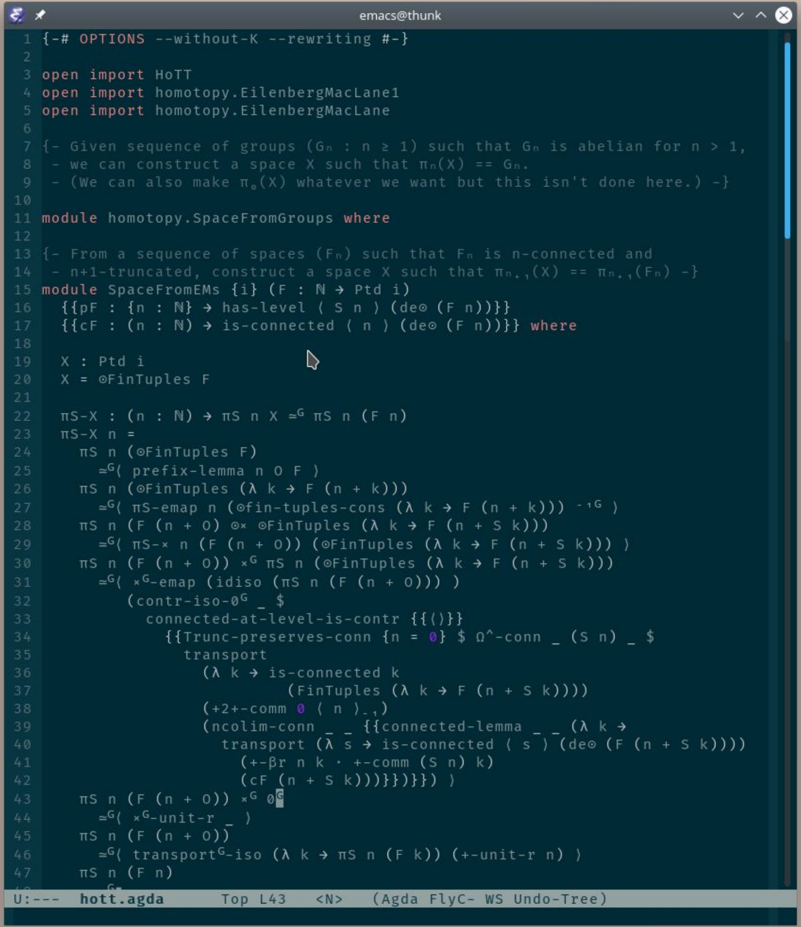
Program Proof


ASIL A/B
Body and convenience
- Smart junction boxes
- Instrument clusters
- Heating and cooling
- Steering wheel sensors
- Body control units
- Body gateway
ASIL QM
Lighting
- Exterior, CHMSLs, RCLs, Accent lighting
- Advanced front lighting
ASIL A/QM
Audio and Infotainment
- GPS/Navigation systems
- Satellite/Digital radio
- Connectivity - USB, HDMI
- Movie/Game systems
Camera system / Vision
Long-range radar
Mid-range radar
Front laser LIDAR
Rear camera / Radar / LIDAR
Side laser LIDAR
ASIL D
Safety and chassis
- Braking/Traction/Stability
- Collision avoidance
- Suspension
- Dynamic braking
- Electronic power steering
ASIL B-D
Powertrain
- Transmission control
- Engine control
- Throttle control
- Valve control
- Fuel injection control
- Position sensing
- Ignition


Sensor Input
AI
Actuation

Program Proof

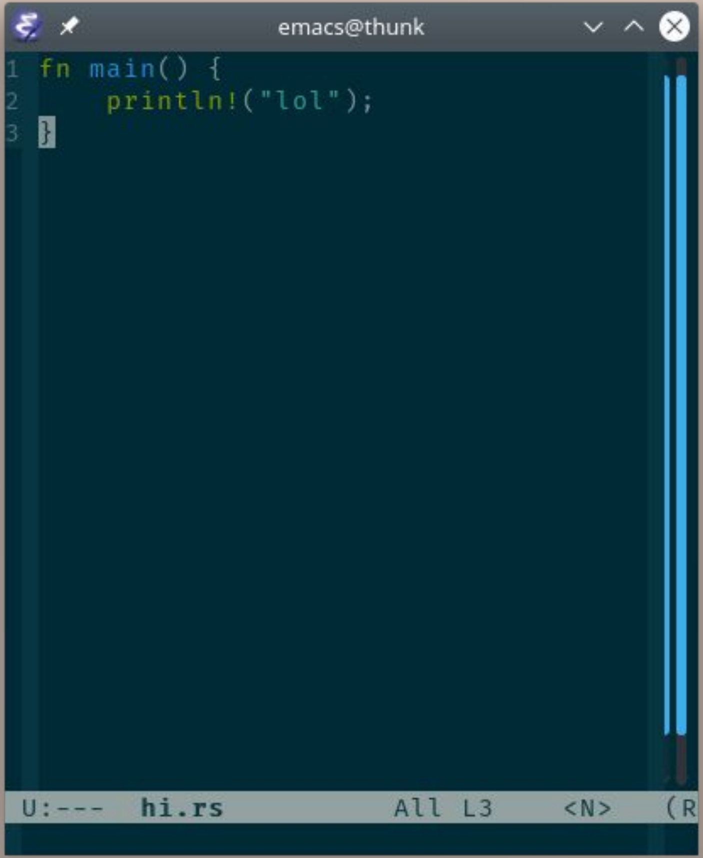
Program Me Proof



emacs@thunk

```
{-# OPTIONS --without-K --rewriting #-}

open import HoTT
open import homotopy.EilenbergMacLane1
open import homotopy.EilenbergMacLane

{- Given sequence of groups (Gₙ : n ≥ 1) such that Gₙ is abelian for n > 1,
 - we can construct a space X such that πₙ(X) == Gₙ.
 - (We can also make π₀(X) whatever we want but this isn't done here.) -}

module homotopy.SpaceFromGroups where

{- From a sequence of spaces (Fₙ) such that Fₙ is n-connected and
 - n+1-truncated, construct a space X such that πₙ₊₁(X) == πₙ₊₁(Fₙ) -}
module SpaceFromEMs {i} (F : ℕ → Ptd i)
  {{pF : {n : ℕ} → has-level ⟨ S n ⟩ (de⊙ (F n))}}
  {{cF : (n : ℕ) → is-connected ⟨ n ⟩ (de⊙ (F n))}} where

  X : Ptd i
  X = ⊙FinTuples F

  πS-X : (n : ℕ) → πS n X ≃ᴳ πS n (F n)
  πS-X n =
    πS n (⊙FinTuples F)
      ≃ᴳ⟨ prefix-lemma n O F ⟩
    πS n (⊙FinTuples (λ k → F (n + k)))
      ≃ᴳ⟨ πS-emap n (⊙fin-tuples-cons (λ k → F (n + k))) ⁻¹ᴳ ⟩
    πS n (F (n + O) ⊙× ⊙FinTuples (λ k → F (n + S k)))
      ≃ᴳ⟨ πS-× n (F (n + O)) (⊙FinTuples (λ k → F (n + S k))) ⟩
    πS n (F (n + O)) ×ᴳ πS n (⊙FinTuples (λ k → F (n + S k)))
      ≃ᴳ⟨ ×ᴳ-emap (idiso (πS n (F (n + O))) )
          (contr-iso-0ᴳ _ $
            connected-at-level-is-contr {{⟨⟩}}
              {{Trunc-preserves-conn {n = 0} $ Ω^-conn _ (S n) _ $
                transport
                  (λ k → is-connected k
                           (FinTuples (λ k → F (n + S k))))
                  (+2+-comm 0 ⟨ n ⟩₋₁)
                  (ncolim-conn _ _ {{connected-lemma _ _ (λ k →
                    transport (λ s → is-connected ⟨ s ⟩ (de⊙ (F (n + S k))))
                      (+-βr n k ∙ +-comm (S n) k)
                      (cF (n + S k)))}})}}) ⟩
    πS n (F (n + O)) ×ᴳ 0ᴳ
      ≃ᴳ⟨ ×ᴳ-unit-r _ ⟩
    πS n (F (n + O))
      ≃ᴳ⟨ transportᴳ-iso (λ k → πS n (F k)) (+-unit-r n) ⟩
    πS n (F n)
```

```
U:---  hott.agda     Top L43   <N>   (Agda FlyC- WS Undo-Tree)
```

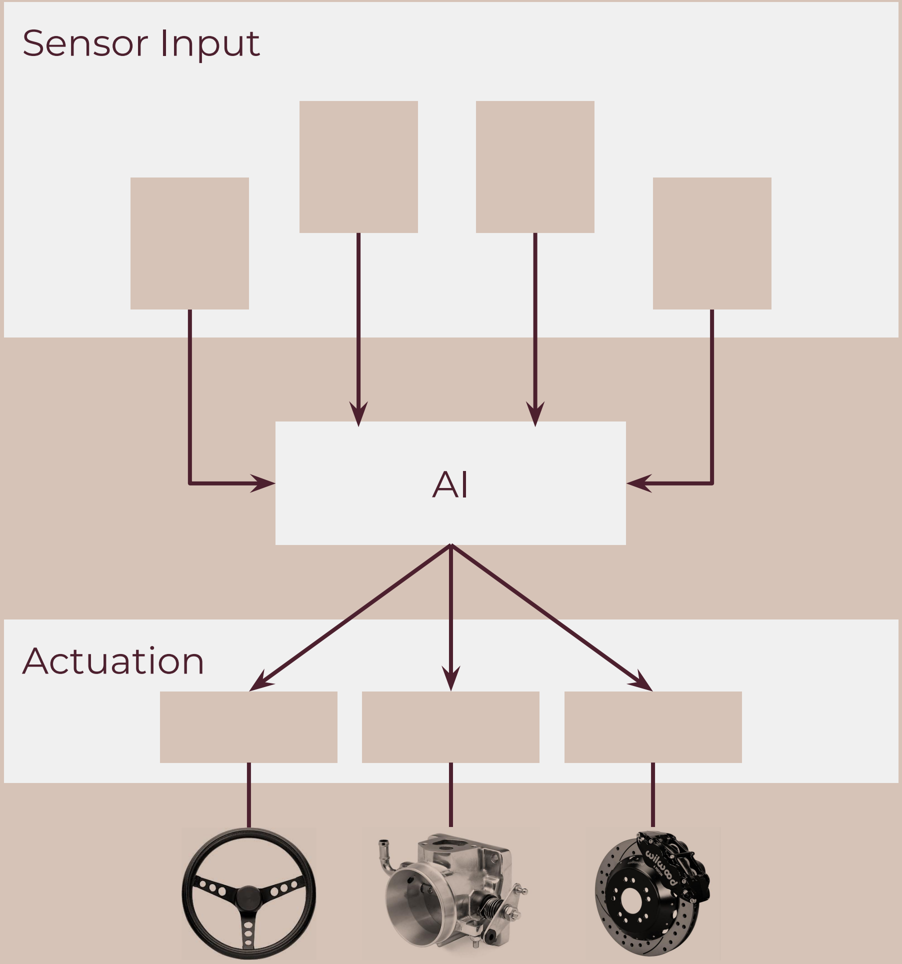


```
fn main() {
    println!("lol");
}
```

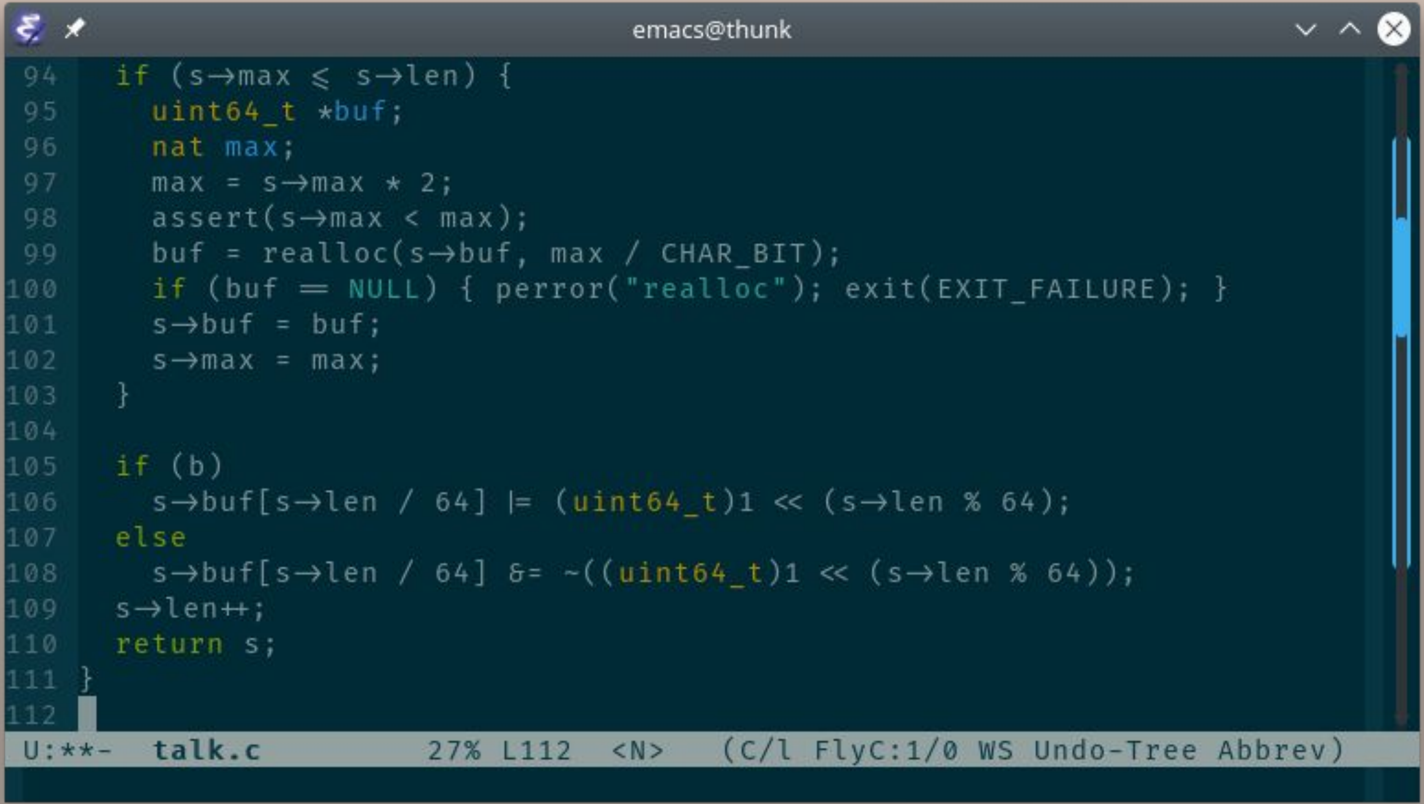
```
U:---  hi.rs          All L3     <N>    (R
```

Program Extraction Proof

Proof Assistant	Extraction Target
Coq	Haskell, OCaml, Scheme
Agda	Haskell
Isabelle / HOL	SML, Haskell, OCaml, Scala


Sensor Input
AI
Actuation

Proof Assistant	Extraction Target
???	C / C++

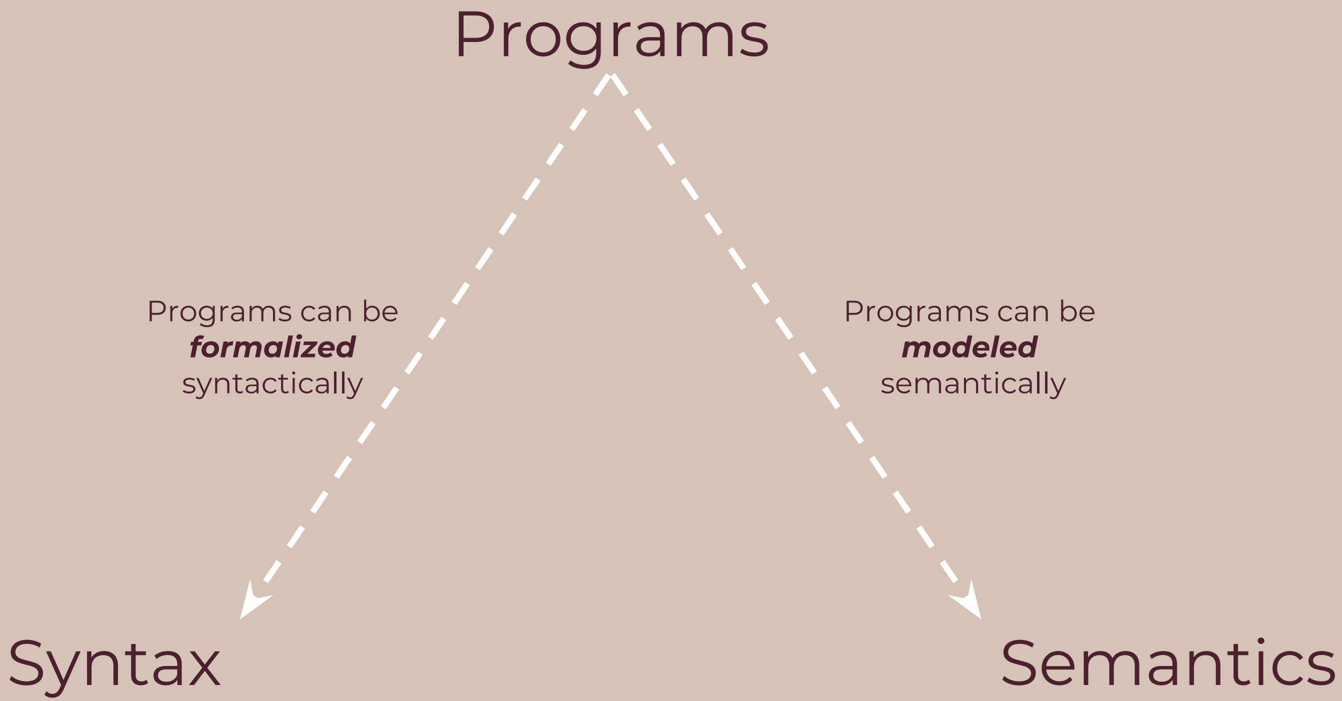


```c
  if (s->max <= s->len) {
    uint64_t *buf;
    nat max;
    max = s->max * 2;
    assert(s->max < max);
    buf = realloc(s->buf, max / CHAR_BIT);
    if (buf == NULL) { perror("realloc"); exit(EXIT_FAILURE); }
    s->buf = buf;
    s->max = max;
  }

  if (b)
    s->buf[s->len / 64] |= (uint64_t)1 << (s->len % 64);
  else
    s->buf[s->len / 64] &= ~((uint64_t)1 << (s->len % 64));
  s->len++;
  return s;
}

```

```
U:**-  talk.c        27% L112  <N>   (C/l FlyC:1/0 WS Undo-Tree Abbrev)
```

Program Proof

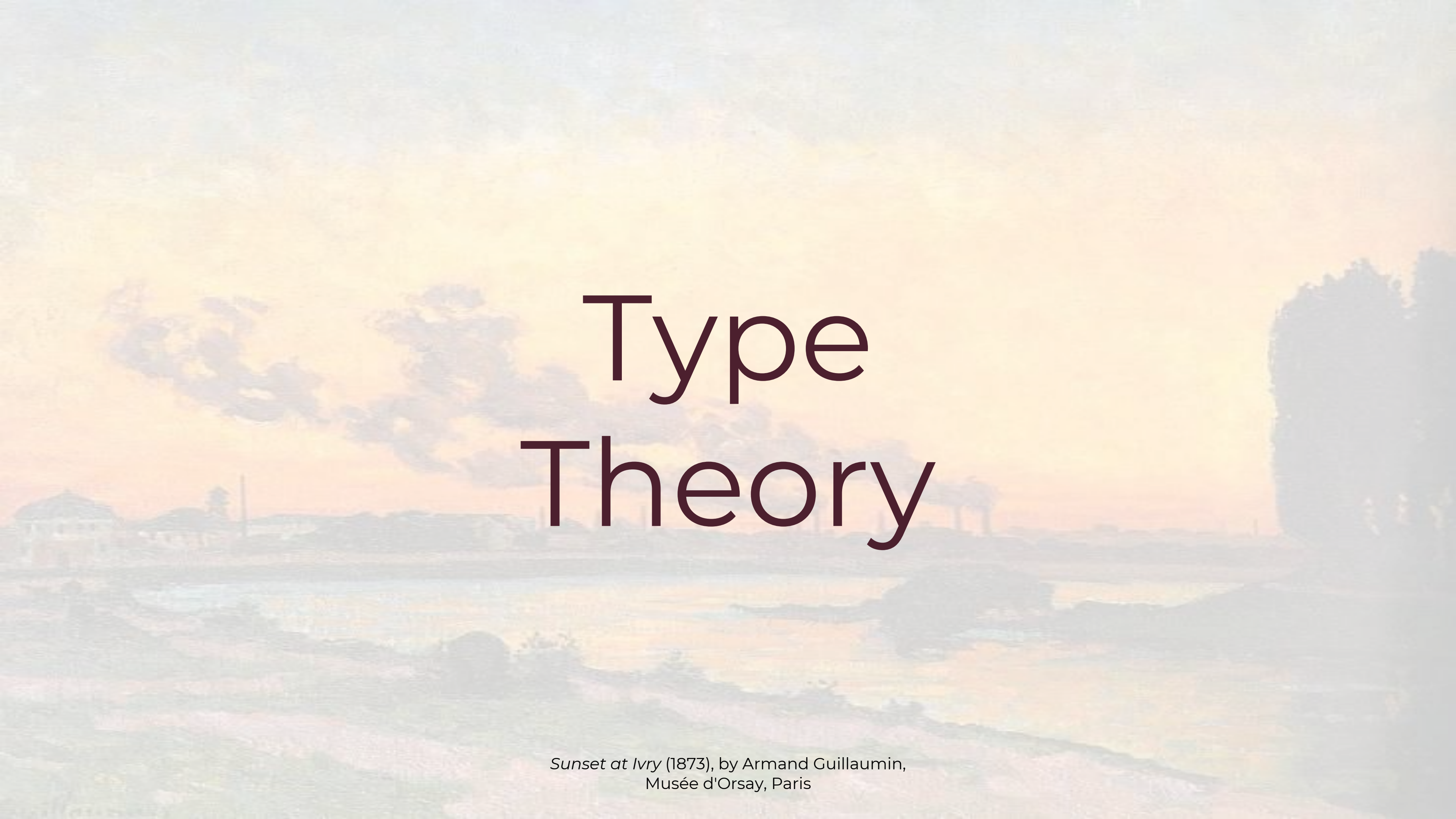
Program Semantics Proof


Programs
Programs can be *formalized* syntactically
Programs can be *modeled* semantically
Syntax
Semantics

$$\begin{tikzcd}
& \text{Programs} \arrow[dl, dashed, "\text{Programs can be }\textbf{\textit{formalized}}\text{ syntactically}"] \arrow[dr, dashed, "\text{Programs can be }\textbf{\textit{modeled}}\text{ semantically}"'] & \\
\text{Syntax} \arrow[rr, dashed, "\textbf{Denotational Semantics}"'] & & \text{Semantics}
\end{tikzcd}$$

$$\begin{tikzcd} \texttt{u8} \arrow[r, dashed] & \{x \mid x \in \mathbb{N} \land x \leq 255\} \\ \texttt{u16} \arrow[r, dashed] & \{x \mid x \in \mathbb{N} \land x \leq 65{,}535\} \end{tikzcd}$$

$$\begin{tikzcd}\text{Program} \arrow[r, dashed] & \text{Mathematical Meaning}\end{tikzcd}$$



Type Theory

Sunset at Ivry (1873), by Armand Guillaumin,
Musée d'Orsay, Paris

$\mathit{term} : \mathrm{Type}$

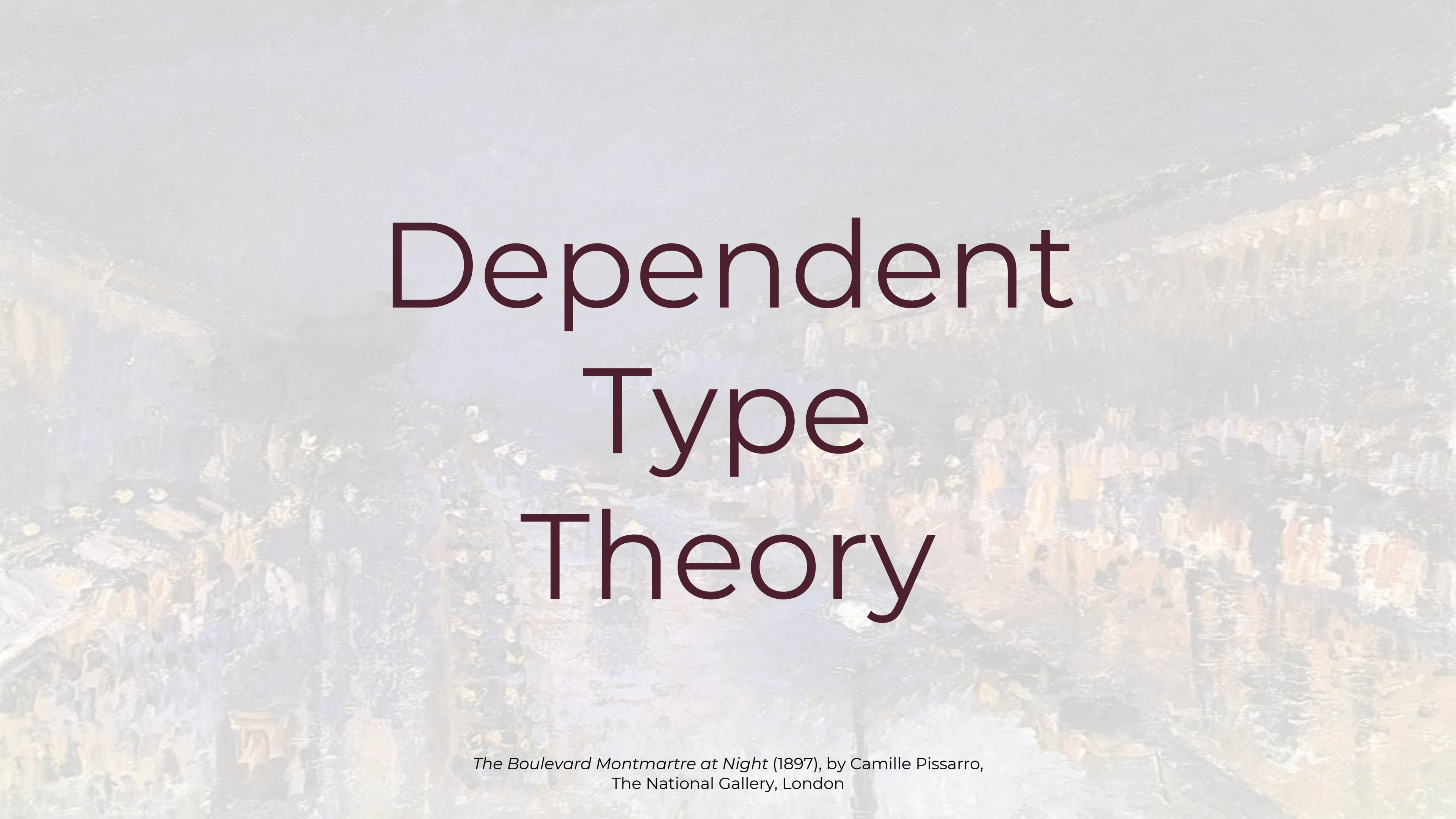
$$\mathit{true} : \mathrm{Bool}$$

$$\frac{\Gamma \vdash a : \mathrm{A}, f : \mathrm{A} \to \mathrm{B}}{\Gamma \vdash f\,a : \mathrm{B}}\;[\mathrm{App}]$$

Program

Type
Theory

Dependent Type Theory

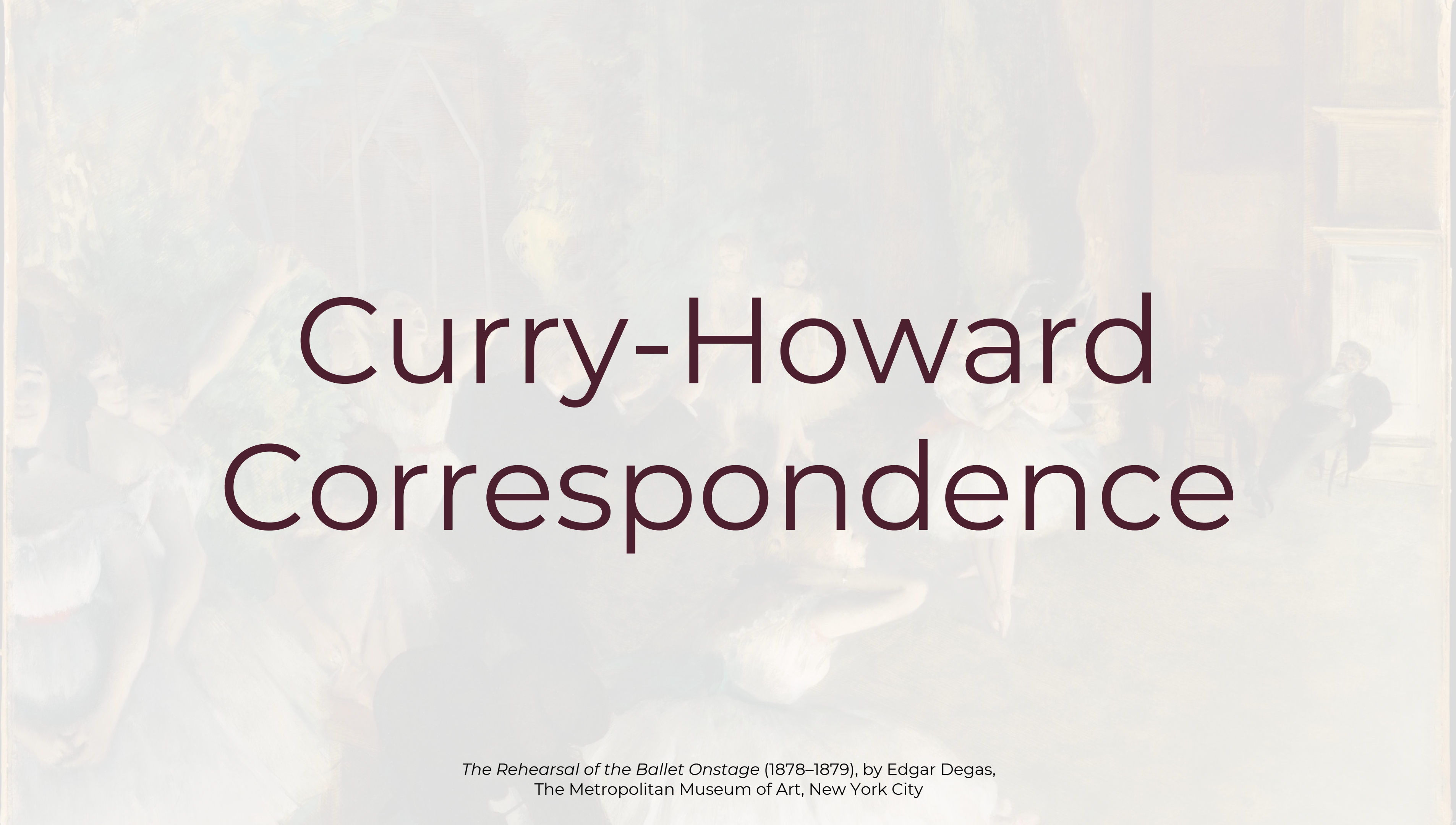

The Boulevard Montmartre at Night (1897), by Camille Pissarro,
The National Gallery, London

$$\mathbf{\Pi}_{(x\,:\,\mathrm{A})}\mathrm{B}(x)$$

$$\frac{\Gamma \vdash a : \mathrm{A}, f : \mathbf{\Pi}_{(x\,:\,\mathrm{A})} \to \mathrm{B}(x)}{\Gamma \vdash f\,a : \mathrm{B}[a/x]}\ [\mathbf{\Pi}\text{-App}]$$

“Dependent types realize a continuum of precision up to a complete specification of the program’s *behaviour*.”

- Altenkirch, McBride, & McKinna, *Why Dependent Types Matter*



Curry-Howard Correspondence

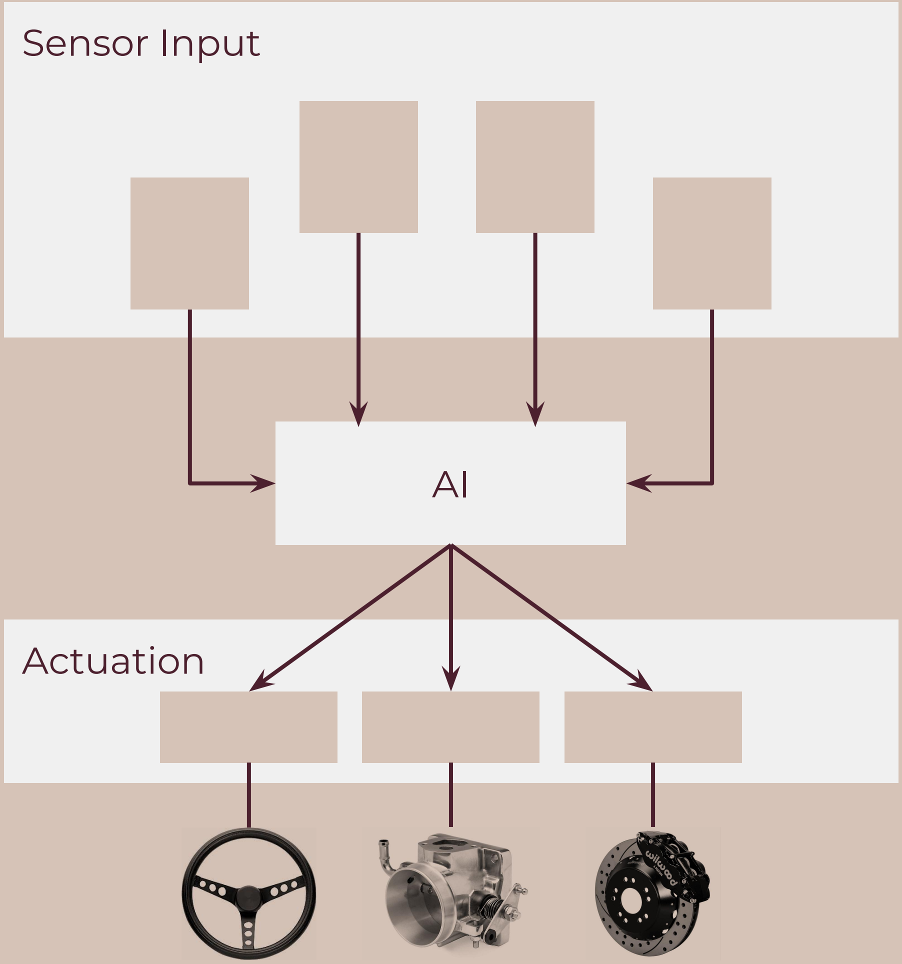
The Rehearsal of the Ballet Onstage (1878–1879), by Edgar Degas,
The Metropolitan Museum of Art, New York City

Program Proof

“Dependently typed programs are, by their nature, proof carrying code.”

- Altenkirch, McBride, & McKinna, *Why Dependent Types Matter*

Language	Dependent Types
Agda	✓
Idris	✓


Sensor Input
AI
Actuation

The Old Garden (1912), by Edmund W. Greacen,
Florence Griswold Museum, Connecticut

Dependent
Type
Theory

```rust
struct Zero;
struct Succ<N: Nat>(PhantomData<N>);

trait Nat {}

impl Nat for Zero {}
impl<N: Nat> Nat for Succ<N> {}

fn main() {
    let _zero: Zero;
    let _one: Succ<Zero>;
}
```

$\mathbb{N} : \mathrm{Type}$

$\mathit{zero} : \mathbb{N}$

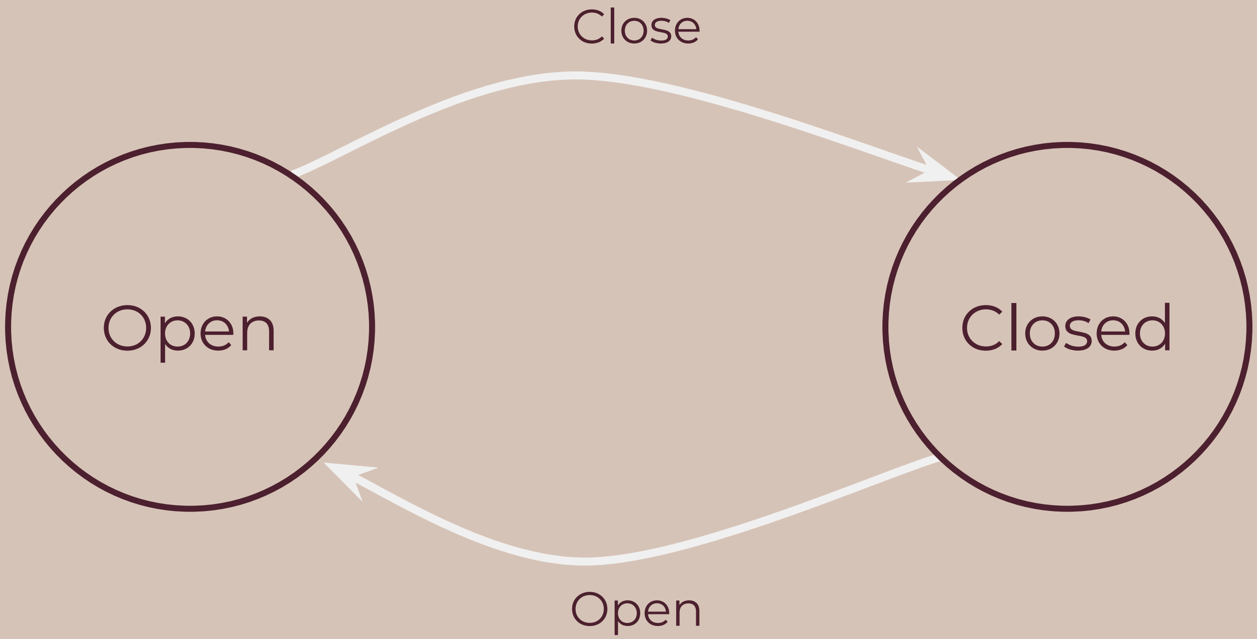
$\mathit{succ} : \mathbb{N} \longrightarrow \mathbb{N}$

$\mathit{one} : \mathbb{N}$

$\mathit{one} = \mathit{succ}\ \mathit{zero}$

```rust
struct Zero;
struct Succ<N: Nat>(PhantomData<N>);

trait Nat {}

impl Nat for Zero {}
impl<N: Nat> Nat for Succ<N> {}

struct Vector<N: Nat, A>
    (Vec<A>, PhantomData<N>);

fn main() {
    let v: Vector<Zero, u8> =
        Vector::<Zero, u8>::new();
    let _v_prime: Vector<Succ<Zero>, u8> =
        v.cons(1);
}
```

$$\mathrm{Vec}\ (\mathrm{X} : \mathrm{Type}) : \mathbb{N} \rightarrow \mathrm{Type}$$
$$\mathit{empty} : \mathrm{Vec}\ \mathrm{X}\ \mathit{zero}$$
$$\mathit{cons} : \mathbf{\Pi}_{(n\,:\,\mathbb{N})} \rightarrow \mathrm{X} \rightarrow \mathrm{Vec}\ \mathrm{X}\ n \rightarrow \mathrm{Vec}\ \mathrm{X}\ (\mathit{succ}\ n)$$


Close
Open
Closed
Open

```rust
trait Trans<S> {}

struct Open;
struct Closed;

trait OpenAdj {}
trait ClosedAdj {}

impl ClosedAdj for Open {}
impl OpenAdj for Closed {}

impl<N> Trans<N> for Open
where N: OpenAdj {}

impl<N> Trans<N> for Closed
where N: ClosedAdj {}
```

$$\text{State} : \text{Type}$$
$$\text{State} = \text{OpenS} + \text{ClosedS}$$

$$\mathbf{\Pi}_{(s\,:\,\text{State})}\ \text{Adj}(s)$$

$$\text{Closed} : \text{Adj}(\text{OpenS})$$
$$\text{Open} : \text{Adj}(\text{ClosedS})$$

$$\mathit{trans} : \mathbf{\Pi}_{(s\,:\,\text{State})}\ s \to \text{Adj}(s)$$

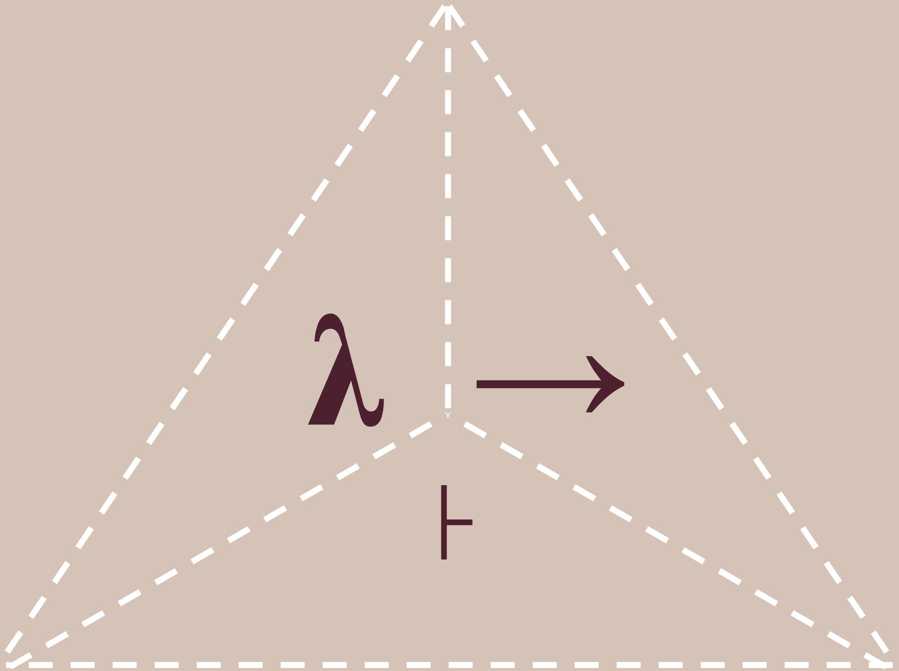
Dependent
Type
Theory

Proof Assistant	Dependent Types
Coq	✓
Agda	✓
Isabelle / HOL	

$$\begin{tikzcd}\text{Rust} \arrow[r, dashed, "\text{Denotational Semantics}"] & \text{Agda}\end{tikzcd}$$

$$\begin{tikzcd} \text{Rust} \arrow[r, dashed, "\text{Extraction}"] & \text{Agda} \end{tikzcd}$$

$$\begin{tikzcd}\text{Program} \arrow[r, dashed] & \text{Proof}\end{tikzcd}$$


λ
→
⊢